

enaio[®]

Software Documentation

enaio[®] webservice

Version 8.10

All software products and all related programs and functions are registered and/or used trademarks of OPTIMAL SYSTEMS GmbH, Berlin or one of its companies. They may only be used with a valid license agreement. The software and its associated documentation are protected by German and international copyright law. Unauthorized duplication and sales is plagiarism and subject to criminal prosecution. All rights reserved, including reproduction, transfer, translation, and storing on any media. Any names of companies and persons appearing in examples (screenshots) in preconfigured test scenarios or demo presentations are purely fictitious. Any resemblance to existing companies or persons is purely coincidental and unintentional.

Copyright 1992 – 2015 by OPTIMAL SYSTEMS GmbH
 Cicerostraße 26
 D-10709 Berlin

10/30/2015
Version 8.10

Contents

Introduction	5
About the Interface.....	5
Installation	5
Server-side Requirements	5
Client-side Requirements.....	5
Web Service Installation.....	5
Installing a Hotfix or Patch.....	6
Core Service Update.....	6
Configuration	7
Configuration Using the Administration Page.....	7
Interface Description	10
General.....	10
Execute Method.....	10
Method Parameter	10
Job Parameter	11
Structure of the ExecuteParameter Type	11
CreateServerFile Method	11
Method Parameter	11
Structure of the CreateServerFile Parameter	12
Example for Microsoft .NET with Visual C#.....	12
GetServerFileInfo Method	12
Method Parameter	12
Structure of the GetServerFileInfo Parameter	13
Example for Microsoft .NET with Visual C#.....	13
AppendChunk Method	13
Method Parameter	13
Structure of the AppendChunk Parameter.....	13
Example for Microsoft .NET with Visual C#.....	14
GetChunk Method	14
Method Parameter	14
Structure of the GetChunk Parameter	14
Example for Microsoft .NET with Visual C#.....	15
DeleteServerFile Method.....	15
Method Parameter	15
Structure of the DeleteServerFile Parameter	15
Example for Microsoft .NET with Visual C#.....	16
Authentication.....	16
Apache CXF – WS Security.....	16
Alternative Authentication	16
Session Management.....	17
Content Interface Type	18

Usage	18
SoapUI (2.5)	20
General.....	20
Examples.....	21
Microsoft .NET with Microsoft Visual Studio 2008 and Visual C#	24
General.....	24
Examples.....	25
Java with NetBeans 6.7	29
General.....	29
Examples.....	30
Annex	33
Links	33
Legend of Structure Elements.....	34

Introduction

About the Interface

The Web service interface enaio® webservice is a core service of the enaio® document management, workflow and archiving system.

With enaio® webservice, external applications can be integrated into the enaio® system.

A special feature of enaio® webservice compared to other interface libraries is that it can be accessed from any platform and is also available for 64-bit systems.

The interface was tested for Microsoft .NET clients (3.5), SOAP-UI 2.5 and Java clients (1.6).

Installation

Server-side Requirements

The Web service interface is one of the enaio® core services and can be installed with the standard setup. The runtime environment (JDK and Tomcat Web application server) required for enaio® webservice is included in the setup and will be installed automatically.

The core services are default components of enaio® and are required for operating the enaio® platform and a proper functioning of the individual enaio® components.

Client-side Requirements

No specific components must be installed on the client side. The application libraries must fulfill all requirements for the integration of the interface.

Web Service Installation

enaio® webservice is included in the enaio® setup as a core service. To install it, select the core service enaio® webservice in the setup.

The runtime environment (JDK and Web application server) will be installed automatically.

The installed runtime environment should be used only for this core service, because when updating the core service the runtime environment is updated as well. If other enaio® or third-party components are run in the runtime environment, update errors may occur or the components may not be run after an update anymore.

This completes the installation of enaio® webservice.

enaio® webservice registers its service endpoint with enaio® server during installation. The service endpoint can be opened and changed in enaio® enterprise-manager from the **Server properties > Category: Services > 'Core Service' > Service endpoint** menu. The registry key for the service endpoint of the core service is transferred to the client computer and can be read by enaio® client and other components.

To test whether enaio® webservice was installed correctly, it is recommended to start a first running test of the application as soon as the console output is visible. The console allows you to follow the progress of the application and check whether the application can be started without any errors.

The console mode can be activated by starting the program `tomcat6.exe`. The file can be found in the `\bin` directory which, just as `\webapps`, is located at the top level of the Tomcat program directory. If the `startup.bat` file is in the `\bin` directory, start this file instead of the program `tomcat6.exe`.

Information output to the Tomcat console should look like this:

```
INFO: Deploying web application archive EcmWS.war
INFO: Creating Service
{http://schemas.optimalsystems.de/OsEcm/Ws}EcmWsMtomWsSecuritySoapService
INFO: Creating Service {http://schemas.optimalsystems.de/OsEcm/Ws}EcmWsMtomSoapService
INFO: Setting the server's publish address to be /EcmWsMtomWsSecurity
INFO: Setting the server's publish address to be /EcmWsMtom
INFO: Server startup in 4029 ms
```

Alternatively, you can install enaio® webservice manually:

- Manually integrate the `osws.war` archive file with the application server. The file is part of the `...Services\enaio® webservices\osws.zip` directory of the installation data.

Expert knowledge is required for the manual installation. The configuration effort is significantly higher than with the recommended installation through the enaio® setup, because more parameters have to be entered and the service must be registered manually at enaio® server.

If the interface was installed correctly, the main page of enaio® webservice is now available (see chapter 'Configuration').

Note that enaio® webservice only supports the basic authentication type and user name/password authentication. NTLM authentication is not supported.

Installing a Hotfix or Patch

A hotfix or patch installation will only replace the obsolete files of the existing version with new, modified files. Updating your installation to a newer version is not possible using a hotfix or a patch.

A hotfix or patch does not back up the existing enaio® webservice installation.

Before replacing files, it is checked whether adequate file versions are available at hotfix or patch installation. If this is not the case or a newer hotfix or patch was already installed, no files will be replaced. The hotfix and patch installations will be canceled reporting a message that the version of the installed service is wrong.

Hotfixes are located in the `SP` directory of the installation data.

Patches can be downloaded from the OPTIMAL SYSTEMS [partner portal](#), the service portal for partners and customers of the OPTIMAL SYSTEMS group.

Core Service Update

A core service update can be done from the enaio® setup.

A backup of the current configuration file versions is performed automatically. After an update, the backed up configuration files can be found in the subdirectory `backup-(timestamp)` of the core service.

Configuration

Configuration Using the Administration Page

enaio® webservice has a built-in Web interface for its configuration. To open the configuration page, start Tomcat and open the following URL in a browser window:

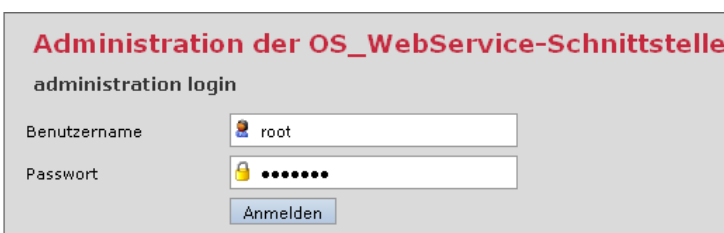
`http://<Web server name/IP>:<Tomcat port (usually 8050)>/OSWS/`

The main page of enaio® webservice will be displayed first:



Screenshot 1: Main page of enaio® webservice

Click the **Administration** link on the main page. The user information for the first login is `root` and `optimal`.



Screenshot 2: Login of the enaio® webservice administration page

The following settings can be changed on the administration page:

- **Name (Administrator name)**
Login name of the enaio® webservice administrator. The default name is `root`.
- **Passwort (Administrator password)**
Password of the enaio® webservice administrator. The default password is `optimal`. It is recommended to change the password on the initial configuration.
- **Verbindungsdaten (Server connection data)**

Every server can be defined using the following pattern: [Name|IP]:[Port]:[Weighting]. Individual patterns must be separated with a '#' character. The server's weighting indicates the possibility to establish a connection to the server.

A configuration may look like this: localhost:4000:50#127.0.0.1:4600:50

- Verbindungsdaten auswerten (Test server connection data)

If this check box is activated, all connections to specified servers are tested. If this is not the case, no data will be saved.

- Temp-Verzeichnis (Temp directory)

The path of the temporary working directory can be specified here. Files that have been transferred with the interface function 'CreateServerFile' will be stored in this directory. It is recommended to set up this directory on the same drive as the temp directory of the operating system to avoid unnecessary data transfer.

- Pfad zum Temp-Verzeichnis auswerten (Test temp directory path)

If this check box is activated, the existence of the temp directory is checked and it is verified whether enaio® webservice has write permission to this directory.

- Session-Validierung vor Verwendung (Validate session before use)

If this check box is activated, existing sessions in the session pool are validated before use. This will ensure that a new server session is established automatically whenever an enaio® server session fails (restart or server failure in a multi-server system).

- Sessionpool-Kapazität (Session pool capacity)

The session pool capacity defines the maximum amount of sessions per logged-in user. If a session is used while its owner tries to execute another job, an additional session will be established if there is enough capacity for it. Otherwise the job call will be deferred until another session is available. A more precise description of the session management can be found in the section 'Authentication' of the chapter 'Interface Description'. A number higher than 1 should only be used for integration scenarios with a technical user.

- Session-Validierung während der Wartezeit (Validate session while waiting)

The session validation defines whether sessions will be validated when they are not in use (see option 'Validate session before use').

- Validierungs-Intervall (Validation interval)

If sessions are not in use, they can be validated (depending on the option 'Validate session while waiting'). The validation interval is used to specify the frequency of this validation. If the validation result shows that a session expired, this session is then removed from the pool.

- Session-Timeout (Session timeout)

Unused sessions will be closed automatically. The session timeout value defines how long sessions will be available after they were used last.

Administration der OS_WebService-Schnittstelle



Administrator	
Name	root
Passwort
Serveranbindung	
Verbindungsdaten	localhost:4000:100
Verbindungsdaten auswerten	<input checked="" type="checkbox"/>
Arbeitsverzeichnisse	
Temp-Verzeichnis	C:\wstemp
Pfad zum Temp-Verzeichnis auswerten	<input checked="" type="checkbox"/>
Allgemeine Konfiguration	
Session-Validierung vor Verwendung	<input checked="" type="checkbox"/>
Sessionpool-Kapazität	1
Session-Validierung während der Wartezeit	<input checked="" type="checkbox"/>
Validierungs-Intervall	30000
Session-Timeout	60000

Screenshot 3: Login of the enaio® webservice administration page

Click the **Speichern** (Save) button to finish the configuration. In case of errors occurring while saving the configuration, error messages will be displayed next to the respective fields and none of the configuration fields will be saved.

If an error occurs in the **Verbindungsdaten** (Server connection data) field, because, e.g. one of the servers is unavailable at the moment of configuration, the **Verbindungsdaten auswerten** (Test server connection data) checkbox can be deactivated to avoid the error. Only correct configurations will be saved.

The **Abmelden** (Logout) button will lead back to the login page (see Screenshot 2).

Interface Description

General

The Simple Object Access Protocol (SOAP) is used to establish the communication between client applications and enaio® webservice. Basically, two approaches for the use of SOAP can be distinguished. On the one hand, the RPC approach allows passing multiple parameters per call. On the other hand, a document-oriented approach features the passing of exactly one parameter that contains objects of complex types into each method. enaio® webservice follows the latter.

The structure and the contents of these parameters, i.e. type descriptions, as well as the signatures of Web service methods provided by enaio® webservice are described with the Web Service Description Language (WSDL).

enaio® webservice provides two of these interface descriptions. They only differ on their authentication procedures. Detailed information can be found in the section 'Authentication'.

The available methods are described below.

Execute Method

The `Execute` method is used to invoke functions of the enaio® server API. Further information on the functions provided by the enaio® server can be found in the enaio® serverapi handbook.

Method Parameter

The `inParameter` parameter of the `ExecuteParameter` type is passed into the `Execute` method. This parameter must contain a job object and can contain an authentication object and multiple property objects (`Properties`).

The structure of an authentication object is described in detail in the section 'Authentication'.

Properties can be identified with a key. This key will be set as the `key` attribute of the property. Currently, only two properties are supported: `useBase64AsString` and `maxCountSize`. `useBase64AsString` can take the values `true` and `false` and controls character encoding of file attachments of the server response, independent of the return types defined in the query. If the value is `true`, the file attachment is sent as part of the SOAP message in base64 code. It will be readable in the body of the response message. If the value is `false`, the file will be attached to the SOAP message in binary code. This applies to data of all parameters with the type `Base64` (see also the section 'Job Parameter'). `maxCountSize` defines a size limit in byte. If a requested file exceeds this limit, it will not be transferred. Instead, a `contentIdentifier` is returned that can be used to download the file with the `getChunk()` method.

When using `getChunk()` and `maxCountSize` it must be ensured that files are requested as an attachment of the SOAP message, not as `Base64` parameters. For some server jobs, such as `GetResultList` or `GetObjDef`, the return type will be controlled with the `Flag` attribute. Further information on the values of the `Flag` attribute can be found in the server API.

Job Parameter

The `Execute` method must include a job parameter and also returns such a parameter. The returned parameter is used to perform an enaio® server query, hereafter referred to as server job.

The job parameter must contain exactly one `Name` parameter and can contain multiple or no `Parameter` objects and `FileParameter` objects. Usually the `returnCode` parameter is only contained in server responses and produces a return value specific to the server job. The `Name` parameter is used to define the server job to be called.

The parameter list is contained in the job parameter and represents the parameters of the respective server job. All parameter types of the server job will be assigned in the SOAP message using the `type` attribute.

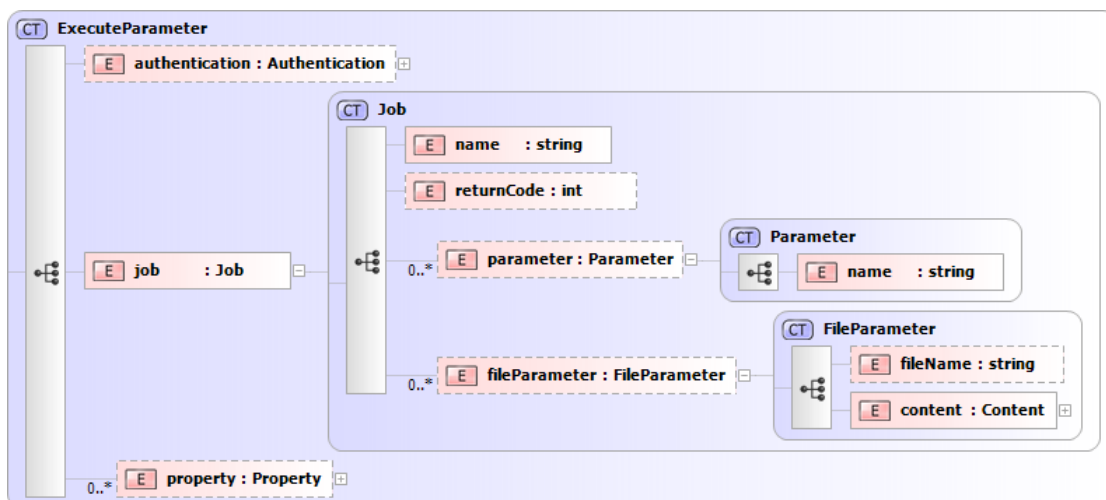
Possible types are:

- `StringParameter`
- `Base64AsStringParameter`
- `Base64Parameter`
- `BooleanParameter`
- `IntegerParameter`

Files can be attached to the call with the `FileParameter` list. Every `FileParameter` consists of two elements: the `fileName` that can be set optionally to name the attached file and exactly one `content` element described in detail in the section 'Content Interface Type'.

Structure of the ExecuteParameter Type

The structure of the `Execute` parameter can be described schematically as follows:



Screenshot 4: Structure of the `ExecuteParameter` type with the type structure of the job parameter (see 'Annex' for the legend)

CreateServerFile Method

The `CreateServerFile` method can be used to transfer files to the `Temp` directory of enaio® webservice or create new files. Further configuration information can be found in the chapter 'Configuration'.

Method Parameter

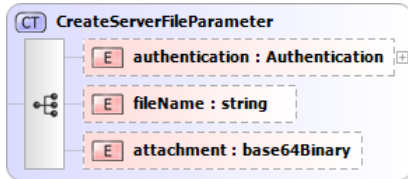
As with all other interface methods, an authentication object can be passed to the `CreateServerFile` method. Detailed information on this topic can be found in the section 'Authentication'. When you create a file and also want to fill it with data, you can write data to the attachment element. In the Microsoft .NET

environment, the element must be initialized using a byte array in both cases. A detailed description of this element can be found in the section 'Content Interface Type'. The name of the file that can be entered in the optional `fileName` element is irrelevant for the server business logic, but it should be set to facilitate retracing.

The method returns a `contentIdentifier` element that is necessary to identify the file created in enaio® webservice.

Structure of the CreateServerFile Parameter

The structure of the `CreateServerFile` parameter can be described schematically as follows:



Screenshot 5: Structure of the `CreateServerFileParameter` type (see 'Annex' for the legend)

Example for Microsoft .NET with Visual C#

A description of how to integrate OS_WebService with Microsoft Visual Studio can be found in the chapter 'Usage'. In the following example, a new file is filled with the content of a 'Pattern.txt'.

```
Authentication auth = new Authentication();
auth.user = "root";
auth.password = "optimal";
auth.applicationName = "C# example";
auth.sessionIdentifier = "12345";

StreamReader inputFile = File.OpenText(@"C:\Pattern.txt");
byte[] data = (new
    ASCIIEncoding()).GetBytes(inputFile.ReadToEnd());

CreateServerFileParameter param = new
    CreateServerFileParameter();
param.attachment = data;
param.authentication = auth;
param.fileName = "Pattern.txt";

OsecmWsPortTypeClient client = new OsecmWsPortTypeClient();
string contentIdentifier = client.createServerFile(param);
```

GetServerFileInfo Method

This method allows you to request information about a file that has been created with the `CreateServerFile` parameter. The current implementation returns the file size in bytes only.

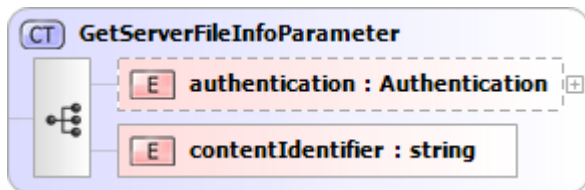
Method Parameter

In addition to the optional authentication object, the `GetServerFileInfo` method has a `contentIdentifier` element that labels the file of which the status is to be requested.

At the moment, the method returns the `size` element that contains the size of the requested file (in bytes).

Structure of the GetServerFileInfo Parameter

The structure of the `GetServerFile` parameter can be described schematically as follows:



Screenshot 6: Structure of the `GetServerFileInfoParameter` type (see 'Annex' for the legend)

Example for Microsoft .NET with Visual C#

A description of how to integrate `OS_WebService` with Microsoft Visual Studio can be found in the chapter 'Usage'.

```

Authentication auth = new Authentication();
auth.user = "root";
auth.password = "optimal";
auth.applicationName = "C# example";
auth.sessionIdentifier = "12345";

GetServerFileInfoParameter param = new
    GetServerFileInfoParameter();
param.authentication = auth;
param.contentIdentifier =
    "4437d76e-1e43-4e2a-ba49-ddd8c54060df";

OsecmWsPortTypeClient client = new OsecmWsPortTypeClient();
GetServerFileInfoResponse response =
    client.getServerFileInfo(param);
Console.WriteLine(response.size);
  
```

AppendChunk Method

The `AppendChunk` method can be used to attach additional data to a file that has been created with the `CreateServerFile` method.

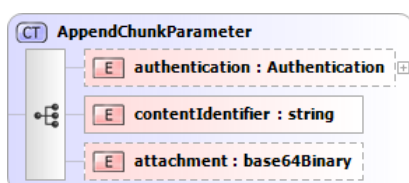
Method Parameter

An authentication object can be passed to this method, too. The `contentIdentifier` element must be used to specify the file to which you want to add data. The added data is contained in the `attachment` element. Detailed instructions can be found in the section 'Content Interface Type'.

After adding the data, the method returns the size of the file.

Structure of the AppendChunk Parameter

The structure of the `AppendChunk` parameter can be described schematically as follows:



Screenshot 7: Structure of the `AppendChunkParameter` type (see 'Annex' for the legend)

Example for Microsoft .NET with Visual C#

A description of how to integrate `OS_WebService` with Microsoft Visual Studio can be found in the chapter 'Usage'. In this example, data from the file 'Pattern2.txt' are added to the server file.

```
Authentication auth = new Authentication();
auth.user = "root";
auth.password = "optimal";
auth.applicationName = "C# example";
auth.sessionIdentifier = "12345";

StreamReader inputFile = File.OpenText(@"C:\Pattern2.txt");
byte[] data = (new
    ASCIIEncoding()).GetBytes(inputFile.ReadToEnd());

AppendChunkParameter param = new AppendChunkParameter();
param.authentication = auth;
param.attachment = data;
param.contentIdentifier =
    "4437d76e-1e43-4e2a-ba49-ddd8c54060df";

OsecmWsPortTypeClient client = new OsecmWsPortTypeClient();
Nullable<long> response = client.appendChunk(param);
Console.WriteLine(response);
```

GetChunk Method

This method allows you to transfer files that have been created with the `CreateServerFile` parameter completely or partially from enaio® webservice to the client application.

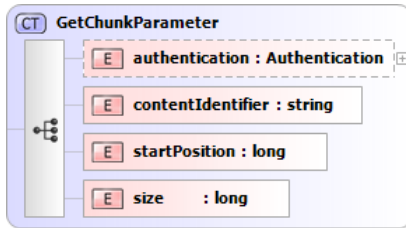
Method Parameter

As with all other interface methods, an authentication object can be passed to the `GetChunk` method. The `contentIdentifier` element must be set in order to identify the requested file. It is possible to request a fragmented file, therefore it must be specified where the fragment begins and what its maximum size can be. The `startPosition` parameter and the `size` parameter are available for this purpose. The details of these parameters are stated in bytes.

Two elements are returned with the `GetChunk` method: the `attachment` and `size` elements. The first element is used to pass a file parameter (see the section 'Content Interface Type' for details). The second is used to specify the actual amount read from the file. If a smaller amount of data is read than specified with the maximum size, the end of the file is reached. If the total file size can be divided by the specified maximum size without a remainder, the value of the `size` element is set to -1 on the initial read access outside the files area.

Structure of the GetChunk Parameter

The structure of the `GetChunk` parameter can be described schematically as follows:



Screenshot 8: Structure of the GetChunkParameter type (see 'Annex' for the legend)

Example for Microsoft .NET with Visual C#

A description of how to integrate OS_WebService with Microsoft Visual Studio can be found in the chapter 'Usage'. In the following example, a chunk is called at the starting position 0 with the maximum size 2.300 and is written in the 'Pattern.txt' file.

```
Authentication auth = new Authentication();
auth.user = "root";
auth.password = "optimal";
auth.applicationName = "C# example";
auth.sessionIdentifier = "12345";

GetChunkParameter param = new GetChunkParameter();
param.authentication = auth;
param.contentIdentifier =
    "4437d76e-1e43-4e2a-ba49-ddd8c54060df";
param.startPosition = 0;
param.size = 2300;

OsecmWsPortTypeClient client = new OsecmWsPortTypeClient();
GetChunkResponse response = client.getChunk(param);
Console.WriteLine(response.size);
byte[] output = response.attachment;
FileStream outputStream =
    new FileStream(@"C:\Pattern.txt", FileMode.Create);
BinaryWriter outFile = new BinaryWriter(outputStream);
outFile.Write(output);
outFile.Close();
```

DeleteServerFile Method

This method deletes a file that has been created using the CreateServerFile parameter or the Execute method. It is recommended to delete every file with this method after processing because no automatic operation exists for this purpose.

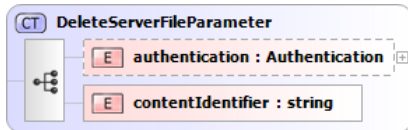
Method Parameter

The DeleteServerFile parameter only gets a contentIdentifier element to define the file to delete. Optionally, an authentication object can be added.

The return value of this method has the type Boolean and provides information whether the deletion was successful.

Structure of the DeleteServerFile Parameter

The structure of the DeleteServerFile parameter can be described schematically as follows:



Screenshot 9: Structure of the DeleteServerFileParameter type (see 'Annex' for the legend)

Example for Microsoft .NET with Visual C#

A description of how to integrate OS_WebService with Microsoft Visual Studio can be found in the chapter 'Usage'.

```
Authentication auth = new Authentication();
auth.user = "root";
auth.password = "optimal";
auth.applicationName = "C# example";
auth.sessionIdentifier = "12345";

DeleteServerFileParameter param =
    new DeleteServerFileParameter();
param.authentication = auth;
param.contentIdentifier =
    "4437d76e-1e43-4e2a-ba49-ddd8c54060df";

OsecmWsPortTypeClient client = new OsecmWsPortTypeClient();
Nullable<bool> successful = client.deleteServerFile(param);
Console.WriteLine(successful);
```

Authentication

Two WSDL interface definitions were implemented for enaio® webservice which differ from one another in the authentication procedure.

Apache CXF – WS Security

This interface definition allows logging in by using the standard procedure WS security (Web Services Security).

For this purpose, the authentication parameters, i.e. user name and password, are passed into the header of the SOAP messages. The password is transferred in the PasswordText type. For a higher level of data security it is recommended to encrypt the entire communication with SSL/TLS.

The interface definition can be accessed from the following URL:

<http://<Server>:<Port>/EcmWS/services/EcmWsMtomWsSecurity?wsdl>

The interface was tested successfully on the client side for a Java environment using NetBeans IDE.

Alternative Authentication

Using the alternative interface definition, the authentication parameters, i.e. user name and password, can be passed into the body of the SOAP message.

For this purpose, WSDL provides an authentication type of which an instance can be set up in the input parameters of every interface method. This type includes an optional applicationName element that is used to pass the name of the calling application. It is recommended to set this value in order to identify server calls of the respective application at the enaio® server. Authentication objects can have a list of properties included for future use. The remaining elements can be divided into two groups of which at least one must be

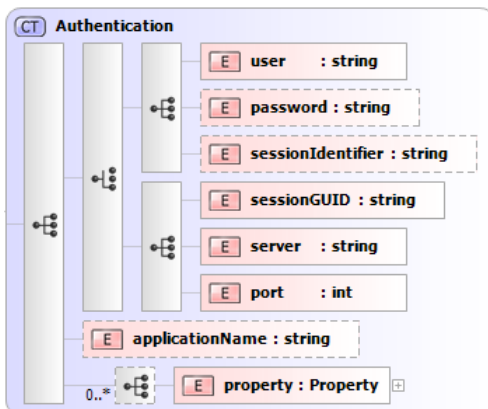
passed. The first group consists of the elements `user`, `password` and `sessionIdentifier`. The `user` element is mandatory, all other elements are optional. These elements must be passed into every method call because they are required at the server side to assign previously opened sessions to the respective owners. More details on this topic can be found in the section 'Session Management'. The second group consists of the following mandatory elements: `sessionGUID`, `server` and `port`. Note that the server doesn't support this group yet.

The interface definition can be accessed from the following URL:

`http://<Server>:<Port>/EcmWS/services/EcmWsMtom?wsdl`

The interface was tested successfully on the client side for Java and Microsoft .NET. Examples are listed in the section 'Usage'.

Structure of the authentication object:



Screenshot 10: Structure of authentication objects (see 'Annex' for the legend)

Session Management

Session objects, also called sessions, must be created to send requests to the enaio® server. A session establishes and maintains the connection to the enaio® server and is required to send server jobs.

enaio® webservice supports complete server-side session management.

At every call an authentication object used by enaio® webservice to establish a connection at the enaio® server must be passed to every interface method. After its creation, the session can be clearly assigned to the authentication object of the client application. In other words, upon the first request of a client a new session is created in enaio® webservice and can also be used for future requests of the same client. In order to reuse the session, the client must pass the same authentication object. Sessions of individual clients are kept in a session pool. Since the server cannot predict when a session is no longer required, a timeout can be set up on the administration page of enaio® webservice. The session will be closed after the timeout has expired.

The session management is optimized for multi-threading. If a client sends a second request after initially sending a request that didn't receive a response yet, the second request can be executed concurrently depending on the session capacity. With a session capacity of 1, all additional calls will be deferred until the previous calls have been executed. Furthermore, it is possible to validate sessions automatically with enaio® webservice. When validating, it is checked whether the connection of the session to the enaio® server still exists. A validation can be executed after a configurable time interval or before using a session. Detailed information on this topic can be found in the chapter 'Configuration'.

The mapping of sessions and authentication objects is done with the help of the elements `user`, `password`, `sessionIdentifier`, `sessionGUID`, `server` and `port`. If these elements have the same values repetitively as at the session creation time, the session can be mapped to the call again. All mandatory elements are described in detail in the section 'Alternative Authentication'.

It is possible to log in two clients at the same time using the same user account and therefore send the same authentication object. In this case, both clients would share a single session. If an individual session is required for each client, for example in case of successive requests that are interdependent, clients can write any value to the `sessionIdIdentifier` element. The value will be used to map the authentication object and the session.

Content Interface Type

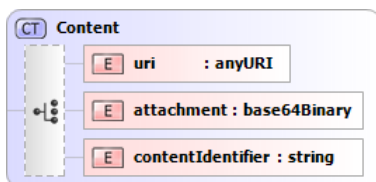
The `Content` type is used to transfer files to enaio® webservice. It is an integral part of the `Execute` parameter of the `Execute` method and can both be passed to the method as well as returned. The type consists of three optional elements.

The `URI` element defines a location where the file to be transferred can be found. This `URI` (Uniform Resource Identifier) must be resolved by enaio® webservice and must be accessible. File system links follow the pattern `file:/// <Path>`.

Additionally, files that were created in the `Temp` directory of enaio® webservice using the `CreateServerFile` method can be processed. For this purpose, the `Content` type provides the `contentIdentifier` element for which the value originally returned by the `CreateServerFile` method must be configured.

Furthermore it is possible to transfer MTOM encoded data. The availability of these data depends on the respective environment and can be found in the corresponding specifications. The `attachment` element is available to implement this procedure. In the SoapUI environment, for example, a `cid` that is linked to a file attachment is entered in this element. For other environments, such as Microsoft .NET or Java, binary data can be written directly to the attribute of the generated class. Note that for Microsoft .NET, binary data are stored as arrays. It is recommended to use the `Chunk` methods for the processing of larger files. In addition to the `Execute` parameter, the `attachment` element is also part of several `Chunk` method parameters or return values. Examples on how to use this element can be found in the chapter 'Usage'.

Structure of the `Content` interface type:



Screenshot 11: Structure of `Content` objects (see 'Annex' for the legend)

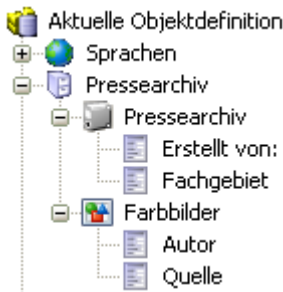
Usage

Several exemplary job calls for the SoapUI, Microsoft .NET and Java environments are described in this section.

All environments must be configured for the use with enaio® webservice. To do so, the URL of the interface definition is required. In order to determine this URL, the link **Service-Schnittstellen** (Service interfaces) must be activated on the main page of enaio® webservice and the link

`{http://schemas.optimalsystems.de/OsEcm/Ws}EcmWsMtomSoapService` must be activated on the following page. Subsequently, the technical interface definition will be displayed and the URL can be copied from the address line of the browser. It must also be ensured, that enaio® webservice is running while configuring the environment.

Some examples require a suitable object definition on the target server. These examples must be adapted for specific test systems. Minimum requirements are: a `Pressearchiv` (press archive) cabinet with a folder where objects of the `Farbbilder` (color images) image document type can be created and an `Autor` (author) index data field for this document type.



Screenshot 12: Example of an object definition

- `krn.GetServerInfo`

With this call, the server status can be requested. The return values of the server can be controlled using the `Info` parameter. In the respective examples, the server IP is sent as a response ensured by the value 4 of the `Info` parameter.

- `dms.GetResultList`

This call sends a request to the server and returns a suitable result table. The request has been adapted to the previously described object definition and queries all documents of the `Farbbilder` type in all folders of the `Pressearchiv` cabinet. Detailed information on the structure of server calls can be found in the respective server interface handbook. The request is passed as a value of the `XML` parameter. The result table of the call can be read from the return parameters and can be displayed.

- `dms.XMLInsert`

With `dms.XMLInsert`, new objects can be created in the enaio® archive system. The object description or indexing of the object to be created and its location are passed in XML notation using the `XML` parameter. In the examples a document of the `Farbbilder` type is created in a specific folder of the `Pressearchiv` cabinet. To which folder it belongs is specified in the `folder_id` field. The object ID of the target folder must be entered in this field to match the archive system. Detailed information on the structure of the object description can be found in the server interface handbook. In addition, a file can be sent that is contained as a document file in the object to create. This file is linked to a file parameter as an attachment. In the examples, the file name of the parameter is `Muster` (pattern). The object ID of the created object can be read from the returned parameters of the call.

- `std.StoreInCache`

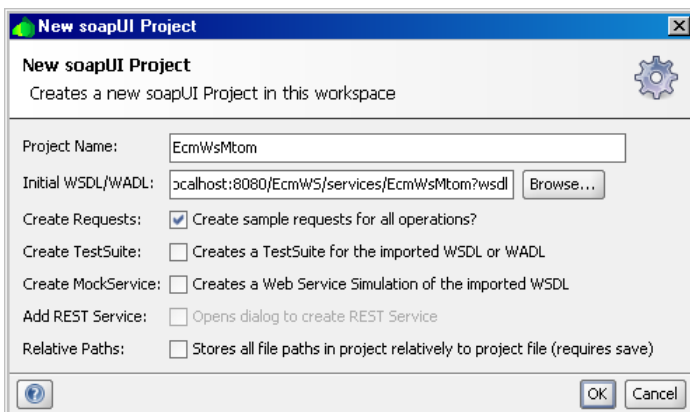
`std.StoreInCache` allows you to download files from the enaio® archive system. The object from which the file will be downloaded is specified with the parameters `dwObjectID` and `dwObjectType`. The first parameter must be linked to the object ID and the second to the type ID of the target object. The file can be read and saved from the file return parameters of the call.

SoapUI (2.5)

General

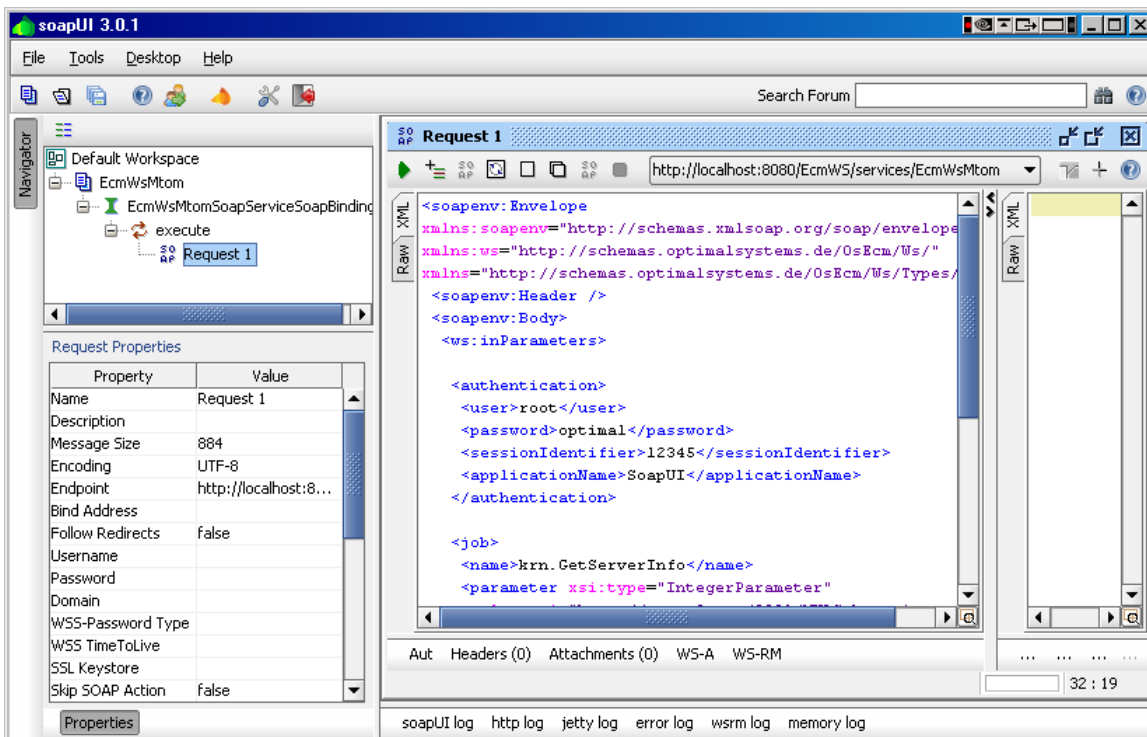
SoapUI is a software specifically designed for the testing of Web services. An advantage of SoapUI is the ability to place server requests directly on the SOAP level. In addition, it is the easiest and fastest possibility to test whether enaio® webservice can be called. A link to download SoapUI can be found in the section 'Links'.

After its installation, SoapUI can be started from the Programs Menu of Microsoft Windows. Subsequently, you must press **Ctrl+N** to open the dialog for the creation of a new project. After choosing a project name, the interface definition URL must be entered in the **Initial WSDL/WADL** field.



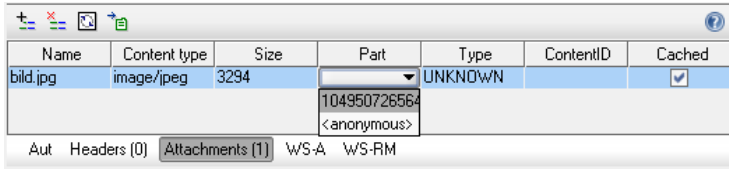
Screenshot 13: Dialog for creation a new project

If the creation of the project was successful, it will be displayed in the SoapUI navigator. SoapUI creates a request for every service function automatically. In order to test the following examples, you must copy the example source text to Request 1 of the execute function. By clicking the arrow button on top of the window, the request is passed into enaio® webservice.



Screenshot 14: Fully configured SoapUI

The 'dms.XMLInsert' job transfers a file attachment to the server. The file must be linked to the `cid` from the SOAP code example. In the lower part of the request you can find the **Attachments** button. By clicking this button a list of attachments of the request and further buttons can be shown. With the **Aut** button, a new file can be attached to the message. After adding a file, the `cid` from the request must be selected from the `Part` column.



Screenshot 15: Dialog for file attachments

The 'std.StoreInCache' job requests a file from the archive system. This file is contained in the response. In the lower part of the response, the **Attachments** button can be found. This button can be used to show attachments in the response. The **Exports the selected attachment to a file** button is used to save attachments as files.

Examples

krn.GetServerInfo

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ws="http://schemas.optimalsystems.de/OsEcm/Ws/"
xmlns="http://schemas.optimalsystems.de/OsEcm/Ws/Types/" >
  <soapenv:Header />
  <soapenv:Body>
    <ws:inParameters>

      <authentication>
        <user>root</user>
        <password>optimal</password>
        <sessionIdentifier>12345</sessionIdentifier>
        <applicationName>SoapUI</applicationName>
      </authentication>

      <job>
        <name>krn.GetServerInfo</name>
        <parameter xsi:type="IntegerParameter"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
          <name>Flags</name>
          <value>0</value>
        </parameter>
        <parameter xsi:type="IntegerParameter"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
          <name>Info</name>
          <value>4</value>
        </parameter>
      </job>

    </ws:inParameters>
  </soapenv:Body>
</soapenv:Envelope >
```

dms.GetResultList

```
<soapenv:Envelope
```

```

xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ws="http://schemas.optimalsystems.de/OsEcm/Ws/"
xmlns="http://schemas.optimalsystems.de/OsEcm/Ws/Types/">
<soapenv:Header />
<soapenv:Body>
  <ws:inParameters>

    <authentication>
      <user>root</user>
      <password>optimal</password>
      <sessionIdentifier>12345</sessionIdentifier>
      <applicationName>SoapUI</applicationName>
    </authentication>

    <job>
      <name>dms.GetResultList</name>
      <parameter xsi:type="IntegerParameter"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <name>Flags</name>
        <value>10</value>
      </parameter>
      <parameter xsi:type="StringParameter"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <name>Encoding</name>
        <value>UTF-8</value>
      </parameter>
      <parameter xsi:type="Base64AsStringParameter"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <name>XML</name>
        <value><![CDATA[<?xml version="1.0" encoding="UTF-8" ?>
          <DMSQuery>
            <Archive name="Press archive">
              <ObjectType name="Color pictures" type="DOCUMENT" >
                <Fields field_schema="ALL"/>
              </ObjectType>
            </Archive>
          </DMSQuery>]]>
        </value>
      </parameter>
    </job>

    <property key="useBase64AsString">false</property>

  </ws:inParameters>
</soapenv:Body>
</soapenv:Envelope >

```

dms.XMLInsert

```

<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ws="http://schemas.optimalsystems.de/OsEcm/Ws/"
  xmlns="http://schemas.optimalsystems.de/OsEcm/Ws/Types/">
  <soapenv:Header />
  <soapenv:Body>
    <ws:inParameters>

      <authentication>
        <user>root</user>
        <password>optimal</password>

```

```

    <sessionIdentifier>12345</sessionIdentifier>
    <applicationName>SoapUI</applicationName>
  </authentication>

  <job>
    <name>dms.XMLInsert</name>
    <parameter xsi:type="IntegerParameter"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <name>Flags</name>
      <value>0</value>
    </parameter>
    <parameter xsi:type="StringParameter"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <name>Encoding</name>
      <value>UTF-8</value>
    </parameter>
    <parameter xsi:type="Base64AsStringParameter"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <name>XML</name>
      <value><![CDATA[<?xml version="1.0" encoding="UTF-8" ?>
        <DMSData>
          <Archive name="Press archive">
            <ObjectType name="Color pictures" type="DOCUMENT">
              <Object folder_id="17">
                <Fields>
                  <Field name="Author">Jon Doe</Field>
                </Fields>
              </Object>
            </ObjectType>
          </Archive>
        </DMSData>]]>
      </value>
    </parameter>
    <fileParameter>
      <fileName>Pattern</fileName>
      <content>
        <attachment>cid:1049507265646</attachment>
      </content>
    </fileParameter>
  </job>

</ws:inParameters>
</soapenv:Body>
</soapenv:Envelope >

```

std.StoreInCache

```

<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ws="http://schemas.optimalsystems.de/OsEcm/Ws/"
  xmlns="http://schemas.optimalsystems.de/OsEcm/Ws/Types/">
  <soapenv:Header />
  <soapenv:Body>
    <ws:inParameters>

      <authentication>
        <user>root</user>
        <password>optimal</password>
        <sessionIdentifier>12345</sessionIdentifier>
        <applicationName>SoapUI</applicationName>
      </authentication>
    </ws:inParameters>
  </soapenv:Body>
</soapenv:Envelope>

```

```
</authentication>

<job>
  <name>std.StoreInCache</name>
  <parameter xsi:type="IntegerParameter"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <name>Flags</name>
    <value>1</value>
  </parameter>
  <parameter xsi:type="IntegerParameter"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <name>dwObjectID</name>
    <value>1254</value>
  </parameter>
  <parameter xsi:type="IntegerParameter"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <name>dwObjectType</name>
    <value>196608</value>
  </parameter>
  <parameter xsi:type="IntegerParameter"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <name>DocState</name>
    <value>0</value>
  </parameter>
  <parameter xsi:type="IntegerParameter"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <name>FileCount</name>
    <value>1</value>
  </parameter>
</job>

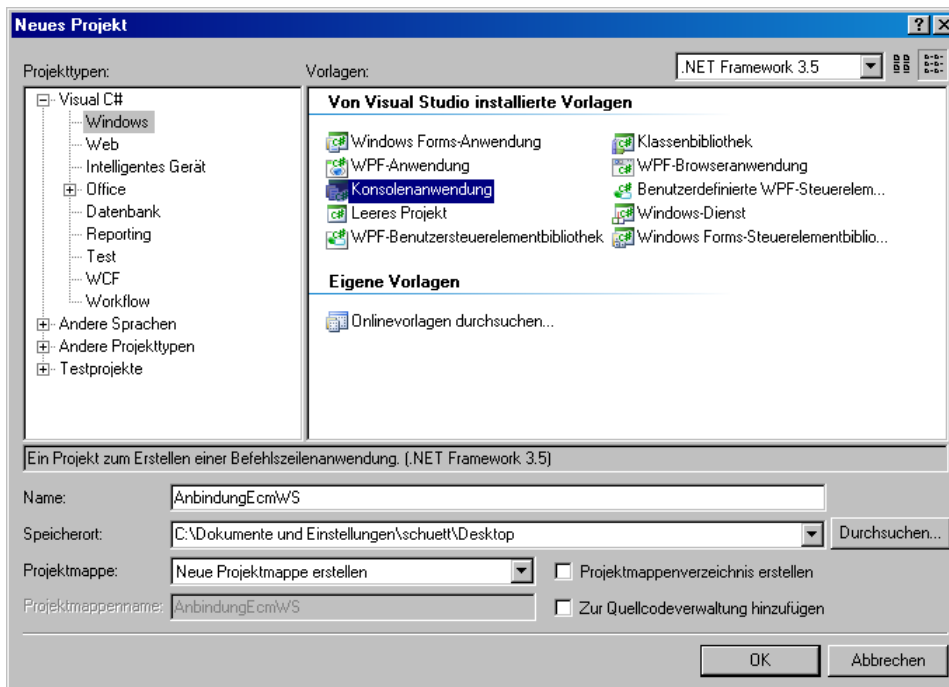
</ws:inParameters>
</soapenv:Body>
</soapenv:Envelope >
```

Microsoft .NET with Microsoft Visual Studio 2008 and Visual C#

General

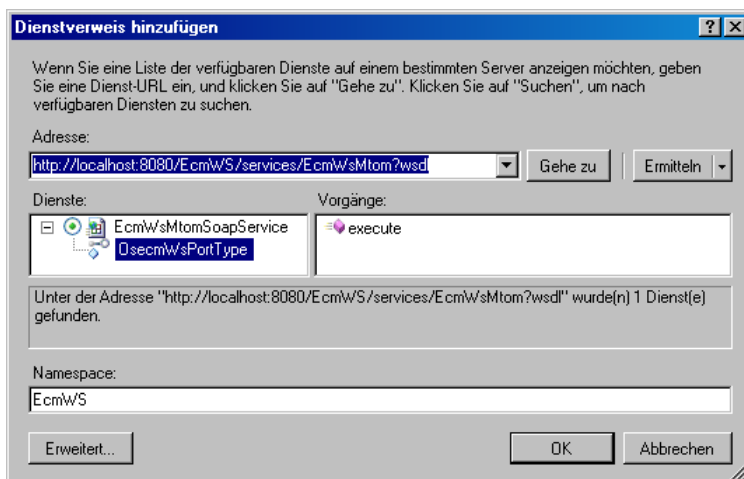
Microsoft .NET enables the integration with enaio® webservice. The minimum requirement for this purpose is version 3.0. Unlike SoapUI, Microsoft .NET generates class files from the WSDL enabling object-oriented programming for the service. After generating the classes, objects can be used for the login and for job calls. The development environment Microsoft Visual Studio 2008 is used in the examples. A link to download Visual Studio 2008 Express can be found in the section 'Links'.

To execute the code example, a new project must be created in Microsoft Visual Studio 2008 as a console application in the language C#. To do so, select the **Datei** (File), **Neu** (New) and **Projekt...** (Project...) entries in the Main menu. In the following dialog the required configurations can be done.



Screenshot 16: Dialog for creation a new project

After its creation, the project will be shown in the **Solution** explorer. In order to integrate the project with enaio® webservice, select the **Projekt** (Project) and **Dienstverweis hinzufügen...** (Add Service Reference...) entries from the Main menu. In the following dialog, enter the interface definition URL in the Adresse (Address) field. The name of the namespace that is used to find future classes can be entered in the Namespace field.



Screenshot 17: Integration of the project with enaio® webservice

After saving the settings, the classes will be generated automatically. The import directive using `<project name> . <namespace> ;` ensures access to the generated classes in the main program. Microsoft Visual Studio 2008 is now fully configured and the code example can be copied to the main program and executed.

Examples

krn.GetServerInfo

```
Authentication auth = new Authentication();
auth.user = "root";
auth.password = "optimal";
```

```

IntegerParameter flags = new IntegerParameter();
flags.name = "Flags";
flags.value = 0;
flags.valueSpecified = true;

IntegerParameter info = new IntegerParameter();
info.name = "Info";
info.value = 4;
info.valueSpecified = true;

Job job = new Job();
job.name = "krn.GetServerInfo";
job.parameter = new Parameter[2];
job.parameter[0] = flags;
job.parameter[1] = info;

ExecuteParameter execPara = new ExecuteParameter();
execPara.authentication = auth;
execPara.job = job;

OsecmWsPortTypeClient client = new OsecmWsPortTypeClient();
ExecuteParameter response = client.execute(execPara);
StringParameter comString =
    (StringParameter)response.job.parameter[1];

System.Console.WriteLine(comString.value);

```

dms.GetResultList

```

Authentication auth = new Authentication();
auth.user = "root";
auth.password = "optimal";

IntegerParameter flags = new IntegerParameter();
flags.name = "Flags";
flags.value = 10;
flags.valueSpecified = true;

StringParameter encoding = new StringParameter();
encoding.name = "Encoding";
encoding.value = "UTF-8";

Base64AsStringParameter xml = new Base64AsStringParameter();
xml.name = "XML";
xml.value = "<?xml version=\"1.0\" encoding=\"UTF-8\" ?>\" +
    "<DMSQuery>\" +
    "<Archive name=\"Press archive\">\" +
    "<ObjectType name=\"Color pictures\" \" \" +
    "type=\"DOCUMENT\" \" >\" +
    "<Fields field_schema=\"ALL\"/>\" +
    "</ObjectType>\" +
    "</Archive>\" +
    "</DMSQuery>\";

Job job = new Job();
job.name = "dms.GetResultList";
job.parameter = new Parameter[3];
job.parameter[0] = flags;
job.parameter[1] = encoding;

```

```

job.parameter[2] = xml;

Property useBase64AsString = new Property();
useBase64AsString.key = "useBase64AsString";
useBase64AsString.Value = "true";

ExecuteParameter execPara = new ExecuteParameter();
execPara.authentication = auth;
execPara.job = job;
execPara.property = new Property[1];
execPara.property[0] = useBase64AsString;

OsecmWsPortTypeClient client = new OsecmWsPortTypeClient();
ExecuteParameter response = client.execute(execPara);
Base64AsStringParameter resultList =
    (Base64AsStringParameter)response.job.parameter[1];
System.Console.WriteLine(resultList.value);

```

dms.XMLInsert

```

Authentication auth = new Authentication();
auth.user = "root";
auth.password = "optimal";

IntegerParameter flags = new IntegerParameter();
flags.name = "Flags";
flags.value = 0;
flags.valueSpecified = true;

StringParameter encoding = new StringParameter();
encoding.name = "Encoding";
encoding.value = "UTF-8";

Base64AsStringParameter xml = new Base64AsStringParameter();
xml.name = "XML";
xml.value = "<?xml version='1.0' encoding='UTF-8' ?>" +
    "<DMSData>" +
    "  <Archive name='Press archive'>" +
    "    <ObjectType name='Color pictures' " +
    "      type='DOCUMENT'>" +
    "        <Object folder_id='17'>" +
    "          <Fields>" +
    "            <Field name='Author'>Jon Doe</Field>" +
    "          </Fields>" +
    "        </Object>" +
    "      </ObjectType>" +
    "    </Archive>" +
    "  </DMSData>";

StreamReader inputFile = File.OpenText(@"C:\Muster.jpg");
byte[] data = (new
    ASCIIEncoding()).GetBytes(inputFile.ReadToEnd());
inputFile.Close();

Content content = new Content();
content.attachment = data;

FileParameter file = new FileParameter();
file.fileName = "Pattern";
file.content = content;

```

```
Job job = new Job();
job.name = "dms.XMLInsert";
job.parameter = new Parameter[3];
job.parameter[0] = flags;
job.parameter[1] = encoding;
job.parameter[2] = xml;
job.fileParameter = new FileParameter[1];
job.fileParameter[0] = file;

Property useBase64AsString = new Property();
useBase64AsString.key = "useBase64AsString";
useBase64AsString.Value = "true";

ExecuteParameter execPara = new ExecuteParameter();
execPara.authentication = auth;
execPara.job = job;
execPara.property = new Property[1];
execPara.property[0] = useBase64AsString;

OsecmWsPortTypeClient client = new OsecmWsPortTypeClient();
client.execute(execPara);
```

std.StoreInCache

```
Authentication auth = new Authentication();
auth.user = "root";
auth.password = "optimal";

IntegerParameter flags = new IntegerParameter();
flags.name = "Flags";
flags.value = 0;
flags.valueSpecified = true;

IntegerParameter dwObjectID = new IntegerParameter();
dwObjectID.name = "dwObjectID";
dwObjectID.value = 2355;
dwObjectID.valueSpecified = true;

IntegerParameter dwObjectType = new IntegerParameter();
dwObjectType.name = "dwObjectType";
dwObjectType.value = 196608;
dwObjectType.valueSpecified = true;

IntegerParameter docState = new IntegerParameter();
docState.name = "DocState";
docState.value = 0;
docState.valueSpecified = true;

IntegerParameter fileCount = new IntegerParameter();
fileCount.name = "FileCount";
fileCount.value = 1;
fileCount.valueSpecified = true;

Job job = new Job();
job.name = "std.StoreInCache";
job.parameter = new Parameter[5];
job.parameter[0] = flags;
job.parameter[1] = dwObjectID;
job.parameter[2] = dwObjectType;
```

```

job.parameter[3] = docState;
job.parameter[4] = fileCount;

ExecuteParameter execPara = new ExecuteParameter();
execPara.authentication = auth;
execPara.job = job;

OsecmWsPortTypeClient client = new OsecmWsPortTypeClient();
ExecuteParameter response = client.execute(execPara);
byte[] output =
    response.job.fileParameter[0].content.attachment;
String extension =
    ((StringParameter)response.job.parameter[0]).value;

FileStream outStream = new FileStream(@"C:\Muster." +
    extension, FileMode.Create);
BinaryWriter outFile = new BinaryWriter(outStream);
outFile.Write(output);
outFile.Close();

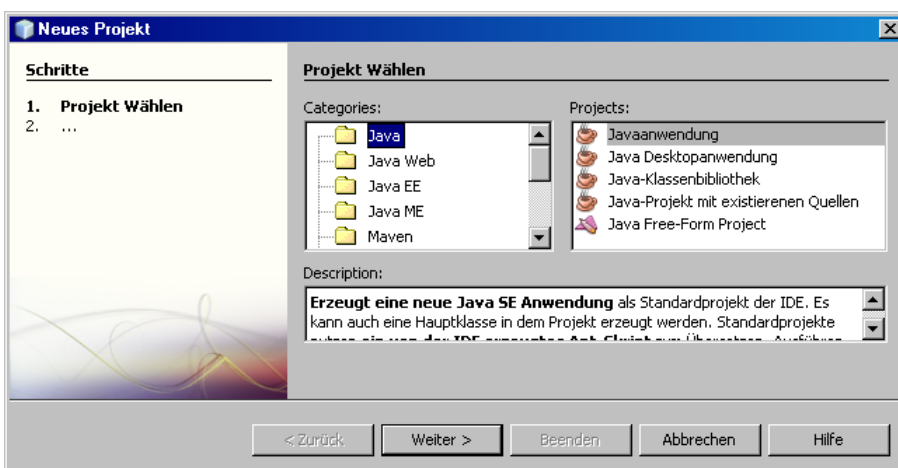
```

Java with NetBeans 6.7

General

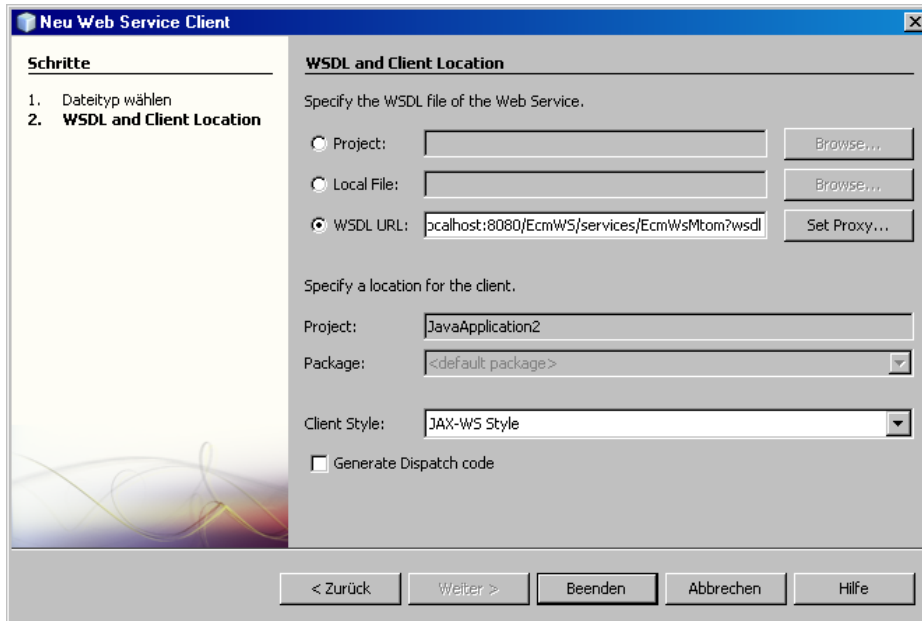
Similar to Microsoft .NET, WSDL class files are created in Java in order to enable object-oriented programming. These objects can also be used for the login and the configuration. The examples have been tested for the NetBeans 6.7 development environment.

For testing the examples, a new project must be created. To do so, select the **Datei** (File) and **Neues Projekt...** (New project...) entries in the Main menu. In the following dialog, select the **Java** category and the **Javaanwendung** (Java application) project type. In the next dialog specify the **Projektname** (project name) and the **Zielort** (target location).



Screenshot 18: Dialog for creation a new project

After confirming by clicking **Beenden** (Exit) the project will be created and will be shown in the **Projekte** (projects) window. Select the **Neu** (New) and **Web Service Client...** entries from the context menu of the project. In the following dialog, select the **WSDL URL** radio button and enter the interface definition URL in the respective field.



Screenshot 19: Dialog for creation a new project

After confirming by clicking the **Beenden** (Exit) button, the interface classes will be generated and can be made available to the main program by using the `import de.optimalsystems.schemas.osecm.ws.*;` and `import de.optimalsystems.schemas.osecm.ws.types.*;` import directives. In doing so, the test project is fully configured and the code example can be copied and executed.

Examples

krn.GetServerInfo

```
Authentication auth = new Authentication();
auth.setUser("root");
auth.setPassword("optimal");

IntegerParameter flags = new IntegerParameter();
flags.setName("Flags");
flags.setValue(0);

IntegerParameter info = new IntegerParameter();
info.setName("Info");
info.setValue(4);

Job job = new Job();
job.setName("krn.GetServerInfo");

List<Parameter> parameter = job.getParameter();
parameter.add(flags);
parameter.add(info);

ExecuteParameter execPara = new ExecuteParameter();
execPara.setAuthentication(auth);
execPara.setJob(job);

EcmWsMtomSoapService service = new EcmWsMtomSoapService();
OsecmWsPortType port =
    service.getPort(OsecmWsPortType.class);
ExecuteParameter response = port.execute(execPara);
```

```
StringParameter comString =
    (StringParameter)response.getJob().getParameter().get(1);

System.out.println(comString.getValue());
```

dms.GetResultList

```
Authentication auth = new Authentication();
auth.setUser("root");
auth.setPassword("optimal");

IntegerParameter flags = new IntegerParameter();
flags.setName("Flags");
flags.setValue(0);

StringParameter encoding = new StringParameter();
encoding.setName("Encoding");
encoding.setValue("UTF-8");

Base64AsStringParameter xml = new Base64AsStringParameter();
xml.setName("XML");
String query = "<?xml version='1.0' encoding='UTF-8' ?>" +
    "<DMSQuery>" +
        "<Archive name='Press archive'>" +
            "<ObjectType name='Color pictures' " +
                "type='DOCUMENT' >" +
                    "<Fields field_schema='ALL' />" +
            "</ObjectType>" +
        "</Archive>" +
    "</DMSQuery>";
xml.setValue(query);

Job job = new Job();
job.setName("dms.GetResultList");

List<Parameter> parameter = job.getParameter();
parameter.add(flags);
parameter.add(encoding);
parameter.add(xml);

ExecuteParameter execPara = new ExecuteParameter();
execPara.setAuthentication(auth);
execPara.setJob(job);

EcmWsMtomSoapService service = new EcmWsMtomSoapService();
OsecmWsPortType port =
    service.getPort(OsecmWsPortType.class);
ExecuteParameter response = port.execute(execPara);

Base64AsStringParameter list = (Base64AsStringParameter)
    response.getJob().getParameter().get(1);
System.out.println(list.getValue());
```

dms.XMLInsert

```
Authentication auth = new Authentication();
auth.setUser("root");
auth.setPassword("optimal");
```

```

IntegerParameter flags = new IntegerParameter();
flags.setName("Flags");
flags.setValue(0);

StringParameter encoding = new StringParameter();
encoding.setName("Encoding");
encoding.setValue("UTF-8");

Base64AsStringParameter xml = new Base64AsStringParameter();
xml.setName("XML");
String query = "<?xml version='1.0' encoding='UTF-8' ?>" +
    "<DMSData>" +
    "    <Archive name='Press archive'>" +
    "        <ObjectType name='Color pictures' " +
    "            type='DOCUMENT'>" +
    "                <Object folder_id='17'>" +
    "                    <Fields>" +
    "                        <Field name='Author'>Jon Doe</Field>" +
    "                    </Fields>" +
    "                </Object>" +
    "            </ObjectType>" +
    "        </Archive>" +
    "    </DMSData>";
xml.setValue(query);

DataHandler data = new DataHandler(new
    FileDataSource("C:\\Pattern.jpg"));

Content content = new Content();
content.setAttachment(data);

FileParameter filePara = new FileParameter();
filePara.setFileName("Pattern");
filePara.setContent(content);

Job job = new Job();
job.setName("dms.XMLInsert");

List<Parameter> parameter = job.getParameter();
parameter.add(flags);
parameter.add(encoding);
parameter.add(xml);

List<FileParameter> fileList = job.getFileParameter();
fileList.add(filePara);

ExecuteParameter execPara = new ExecuteParameter();
execPara.setAuthentication(auth);
execPara.setJob(job);

EcmWsMtomSoapService service = new EcmWsMtomSoapService();
OsecmWsPortType port =
    service.getPort(OsecmWsPortType.class);
ExecuteParameter response = port.execute(execPara);

```

std.StoreInCache

```

Authentication auth = new Authentication();
auth.setUser("root");
auth.setPassword("optimal");

```



```
IntegerParameter flags = new IntegerParameter();
flags.setName("Flags");
flags.setValue(0);

IntegerParameter dwObjectID = new IntegerParameter();
dwObjectID.setName("dwObjectID");
dwObjectID.setValue(2355);

IntegerParameter dwObjectType = new IntegerParameter();
dwObjectType.setName("dwObjectType");
dwObjectType.setValue(196608);

IntegerParameter DocState = new IntegerParameter();
DocState.setName("DocState");
DocState.setValue(0);

IntegerParameter FileCount = new IntegerParameter();
FileCount.setName("FileCount");
FileCount.setValue(1);

Job job = new Job();
job.setName("std.StoreInCache");

List<Parameter> parameter = job.getParameter();
parameter.add(flags);
parameter.add(dwObjectID);
parameter.add(dwObjectType);
parameter.add(DocState);
parameter.add(FileCount);

ExecuteParameter execPara = new ExecuteParameter();
execPara.setAuthentication(auth);
execPara.setJob(job);

EcmWsMtomSoapService service = new EcmWsMtomSoapService();
OsecmWsPortType port =
    service.getPort(OsecmWsPortType.class);
ExecuteParameter response = port.execute(execPara);

Parameter extensionPara =
    response.getJob().getParameter().get(0);
String extension =
    ((StringParameter)extensionPara).getValue();

FileParameter file =
    response.getJob().getFileParameter().get(0);
DataHandler data = file.getContent().getAttachment();
data.writeTo(new FileOutputStream("C:\\Muster." +
    extension));
```

Annex

Links

Download: SoapUI

<http://sourceforge.net/projects/soapui/files/soapui/>



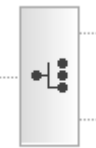
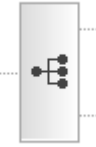

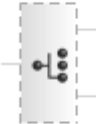

Download: Visual Studio 2008 Express

<http://www.microsoft.com/germany/express/download/downloadaddetails.aspx?p=iso>

Download: NetBeans IDE (Java)

<http://www.netbeans.org/downloads/>

Legend of Structure Elements

	Interface element: Instance of an interface type.
	Interface type: Complex type composed of elements.
	Selection element: Exactly one of the available elements must exist.
	Sequence element: All available elements must exist in the shown sequence.
	Optional interface element: This element is not mandatory and no more than exactly one element must exist.
	Optional selection: It must contain exactly one of the available elements or may be empty.
	Random element: Optional element, any desired number of this element can exist. This element is not mandatory.