OPTIMAL SYSTEMS

Softwaredokumentation

enaio® webservice

Version 8.50

02.02.2017
Version 8.50

Inhalt

Zur Einführung

Über die Schnittstelle

Die Webservice-Schnittstelle enaio® webservice ist ein Kerndienst des Dokumentenmanagement-, Workflow- und Archivsystems enaio®.

enaio® webservice dient zur Anbindung externer Anwendungen an das enaio®-System.

Die Besonderheit gegenüber anderen Schnittstellen-Bibliotheken ist, dass enaio® webservice plattformunabhängig angesteuert werden kann und auch für 64-Bit-Systeme zur Verfügung steht.

Installation

Voraussetzungen auf der Serverseite

Die Webservice-Schnittstelle ist ein enaio®-Kerndienst, der über das Standardsetup installiert wird. Die für enaio® webservice notwenige Laufzeitumgebung (JDK und Tomcat Webanwendungsserver) ist im Setup integriert und wird automatisch mitinstalliert.

Die Kerndienste sind Standardkomponenten von enaio®, die für den Betrieb der enaio®-Plattform und dem reibungslosen Zusammenspiel der einzelnen enaio®-Komponenten erforderlich sind.

Voraussetzungen auf der Clientseite

Auf der Clientseite sind keine spezifischen Komponenten zu installieren. Die Voraussetzungen für die Anbindung an die Schnittstelle müssen die Anwendungs-Bibliotheken mitbringen.

Installation des Webservices

enaio® webservice wird als Kerndienst über das enaio®-Setup installiert. Wählen Sie dazu beim Setup den Kerndienst enaio® webservice.

Die Laufzeitumgebung (JDK und Webanwendungsserver) wird automatisch mitinstalliert.

> Die installierte Laufzeitumgebung sollte nur von diesem Kerndienst verwendet werden, da bei der Aktualisierung des Kerndiensts über das Setup auch die Laufzeitumgebung aktualisiert wird. Wenn mit der Laufzeitumgebung andere enaio®- oder Fremdkomponenten betrieben werden, kann möglicherweise die Aktualisierung nicht korrekt durchgeführt werden oder die anderen Komponenten funktionieren dann nicht mehr.

Damit ist die Installation von enaio® webservice abgeschlossen.

Bei der Installation registriert enaio® webservice seinen Service-Endpoint bei enaio® server. Einsehen und ändern können Sie den Service-Endpoint in enaio® enterprise-manager unter **Servereigenschaften > Kategorie: Services > 'Kerndienst' > Service-Endpoint.** Der Registrierungsschlüssel für den Service-Endpoint des

Kerndiensts wird an die Clientrechner übertragen, sodass er von enaio® client und anderen Komponenten ausgelesen werden kann.

Um zu testen, ob enaio® webservice erfolgreich installiert wurde, empfiehlt es sich den ersten Lauftest der Anwendung zu machen, sobald die Konsolenausgabe sichtbar ist. Mit der Konsole ist es möglich, den Anwendungsablauf mitzuverfolgen und zu prüfen, ob die Anwendung fehlerfrei gestartet werden kann.

Um den Konsolenmodus zu aktivieren, muss das Programm `tomcat6.exe` gestartet werden. Dieses befindet sich im Verzeichnis `\bin`, dass wie `\webapps` auf der obersten Ebene des Programmverzeichnisses des Tomcat liegt. Befindet sich im Verzeichnis `\bin` die Datei `startup.bat`, so ist diese an Stelle das Programm `tomcat6.exe` zu starten.

Auf der Konsole des Tomcat muss in etwa folgendes ausgegeben werden:

```
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
```

Beachten Sie, dass enaio® webservice lediglich die Authentifizierungsmethoden Basic Authentication und Benutzername/Passwort unterstützt. Die NTLM-Authentifizierung nicht unterstützt.

Installation eines Hotfix oder Patches

Bei der Installation eines Hotfix oder eines Patches werden nur die Dateien ersetzt, die sich gegenüber der bestehenden Version verändert haben. Das Aktualisieren auf eine neue Version ist mit der Installation eines Hotfix oder eines Patches nicht möglich.

Eine Sicherung der bestehenden enaio® webservice-Installation wird nicht durchgeführt.

Bei der Installation eines Hotfix oder eines Patches wird vor dem Ersetzen von Dateien geprüft, ob die Version der bestehenden Installation zu der des Hotfix oder Patch passt. Ist das nicht der Fall oder ist bereits ein neuerer Hotfix oder Patch installiert, werden die Dateien nicht ersetzt. Der Hotfix-/Patch-Installer bricht dann mit der Meldung ab, dass der installierte Service die falsche Version trägt.

Hotfixe befinden sich im `SP`-Verzeichnis der Installationsdaten.

Patches erhalten Sie von OPTIMAL SYSTEMS und sind als Download im Partnerportal, dem Serviceportal für Partner und Kunden der OPTIMAL SYSTEMS Gruppe, verfügbar.

Aktualisierung des Kerndiensts

Die Aktualisierung des Kerndiensts wird über das enaio®-Setup vorgenommen.

Dabei wird der aktuelle Stand der Konfigurationsdateien automatisch gesichert. Die gesicherten Konfigurationsdateien befinden sich nach einer Aktualisierung im Programmunterverzeichnis `backup-(Zeitstempel)`des Kerndiensts.

Konfiguration

Konfiguration über die Administrationsseite

enaio® webservice verfügt über eine Webschnittstelle für die Konfiguration. Um die Konfigurationsseite zu öffnen, muss Tomcat gestartet und die folgende URL im Browser geöffnet werden:

`http://<Name/IP des Webservers>:<Port zum Tomcat (meist 8050)>/OSWS/`

Zunächst wird die Hauptseite von enaio® webservice angezeigt:

Abbildung 1: Hauptseite von enaio® webservice

Auf der Hauptseite muss der Link zur **Administration** aktiviert werden. Die Angaben für den ersten Zugang sind `root` und `optimal`.

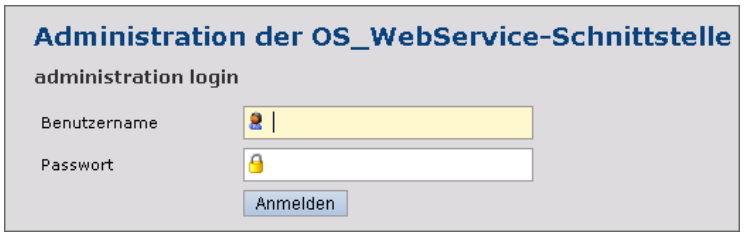


Abbildung 2: Anmeldung an die Administrationsseite von enaio® webservice

Auf der Administrationsseite können folgende Einstellungen gemacht werden:

§ Name des Administrators

  Anmeldename für den Administrator von enaio® webservice. Standardmäßig `root`.

§ Passwort des Administrators

  Passwort für den Administrator von enaio® webservice. Standardmäßig `optimal`. Es wird empfohlen, dass Passwort bei der Erstkonfiguration zu ändern.

§ Verbindungsdaten zu den Servern

Jeder Server wird mit dem Muster '[Name|IP]:[Port]:[Wichtung]' definiert. Die einzelnen Muster werden mit einem #-Zeichen voneinander getrennt. Die Wichtung eines Servers legt fest, mit welcher Wahrscheinlichkeit eine Verbindung zu diesem Server aufgebaut wird.

Die Konfiguration könnte wie folgt aussehen: `localhost:4000:50#127.0.0.1:4600:50`

§ Verbindungsdaten zu den Servern auswerten

Wenn dieses Kontrollkästchen aktiviert ist, wird beim Speichern getestet, ob alle angegebenen Server erreichbar sind. Ist das nicht der Fall, werden keine Daten gespeichert.

§ Temp-Verzeichnis

Hier kann der Pfad zum temporären Arbeitsverzeichnis angegeben werden. In dieses Verzeichnis werden Dateien abgelegt, die mit der Schnittstellenfunktion 'CreateServerFile' übertragen wurden. Es wird empfohlen, dieses Verzeichnis auf dasselbe Laufwerk wie das temp-Verzeichnis des Betriebssystems zu legen, um unnötige Kopieraktionen zu vermeiden.

§ Pfad zum Temp-Verzeichnis auswerten

Wenn dieses Kontrollkästchen aktiviert wird, wird geprüft, ob das angegebene Temp-Verzeichnis existiert und ob enaio® webservice Schreibrechte dafür hat.

§ Session-Validierung vor Verwendung

Mit diesem Kontrollkästchen wird festgelegt, ob vorhandene Sessions im Sessionpool vor ihrer Verwendung auf Gültigkeit geprüft werden sollen. Dies sorgt dafür, dass bei einer ausgefallen Session am enaio®-Server (Neustart, oder ausgefallener Server im Mehrserverbetrieb) automatisch eine neue Session erzeugt wird.

§ Sessionpool-Kapazität

Mit der Sessionpool-Kapazität kann festgelegt werden, wie viele Sessions pro angemeldeten Benutzer existieren können. Ist eine Session in Benutzung während ihr Besitzer einen weiteren Job ausführen möchte, kann, wenn es die Kapazität zulässt, eine weitere Session angelegt werden. Andernfalls wird der Aufruf des Jobs zurückgehalten bis eine andere Session verfügbar ist. Die Sitzungsverwaltung wird im Abschnitt 'Authentifikation' des Kapitels 'Beschreibung der Schnittstellendefinition' genauer beschrieben. Eine Anzahl von mehr als 1 sollte nur in Integrationsszenarien mit einem technischen Benutzer verwendet werden.

§ Session-Validierung während der Wartezeit

Die Session-Validierung legt fest, ob Sessions auf Gültigkeit geprüft werden sollen, wenn sie nicht verwendet werden (siehe Option 'Session-Validierung vor Verwendung').

§ Validierungs-Intervall

Sessions können, wenn sie nicht verwendet werden, auf Gültigkeit geprüft werden (abhängig vom Feld 'Session-Validierung während der Wartezeit'). Mit dem Validierungs-Intervall kann eingestellt werden, wie oft diese Prüfung durchgeführt wird. Ergibt sich aus der Prüfung, dass die Session abgelaufen ist, dann wird die Session aus dem Pool entfernt.

§ Session-Timeout

Nicht verwendete Sessions werden automatisch geschlossen. Der Session-Timeout legt fest, wie lange die Sessions nach ihrer letzten Verwendung offen gehalten werden sollen.

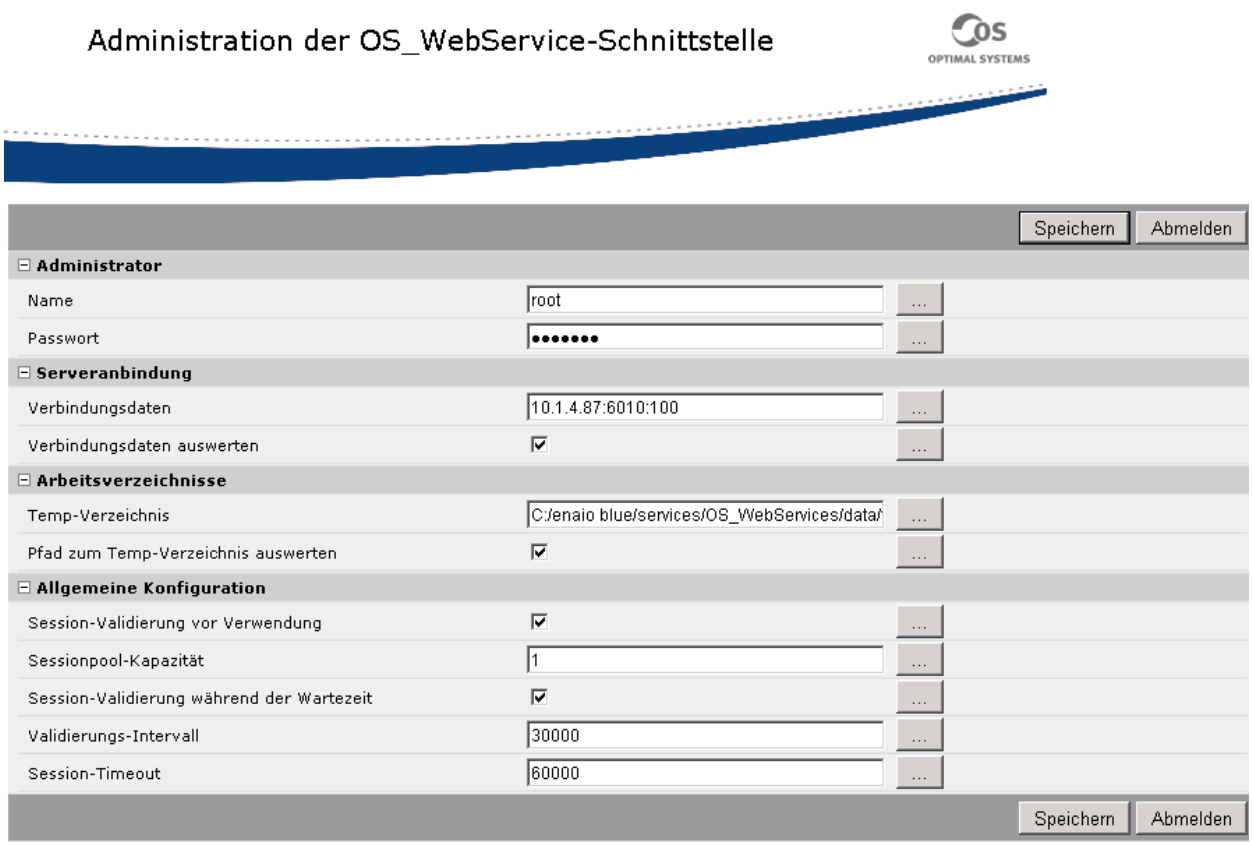


Abbildung 3: Die Administrationsseite von enaio® webservice

Mit einem Klick auf die Schaltfläche **Speichern** wird die Konfiguration beendet. Wenn Fehler beim Speichern der Konfiguration auftreten, werden die Fehlermeldungen neben den betreffenden Feldern ausgegeben und keines der Konfigurationsfelder wird gespeichert.

Tritt ein Fehler etwa beim Feld **Verbindungsdaten** auf, weil beispielsweise einer der Server zur Konfigurationszeit nicht erreichbar ist, dann kann die Markierung des Kontrollkästchens **Verbindungsdaten auswerten** entfernt werden, um den Fehler zu umgehen. Erst eine fehlerfreie Konfiguration wird gespeichert.

Die Schaltfläche **Abmelden** führt wieder zur Anmeldeseite zurück (siehe Abbildung 2).

Für SAP-Systeme, die nicht unicode-fähig sind, ist es notwendig, über die Konfigurationsdatei `config.properties` aus dem Verzeichnis `..\webapps\osws\WEB-INF\config\` den Wert des Parameters `messages.incoming.forceEncoding` von `off` auf `UTF-8` zu ändern.

Beschreibung der Schnittstellendefinition

Allgemeines

Für die Kommunikation zwischen Client-Anwendungen und enaio® webservice wird das Simple Object Access Protocol verwendet, kurz SOAP. Im Wesentlichen gibt es zwei Herangehensweisen für die Verwendung von SOAP. Zum einen den RPC-Ansatz, der sich in SOAP dadurch auszeichnet, dass mehrere Parameter für einen Aufruf übergeben werden. Zum anderen den dokumentorientierten Ansatz, der sich dadurch auszeichnet, dass jeder Methode genau ein Parameter übergeben wird, der Objekte komplexer Typen enthält. enaio® webservice folgt dem dokumentorientierten Ansatz.

Sowohl der Aufbau und die Inhalte dieser Parameter, also die Typbeschreibungen, als auch die Signaturen, der Webservice-Methoden, die enaio® webservice zur Verfügung stellt, werden mir der Web Service Description Language (WSDL) beschrieben.

enaio® webservice bietet zwei solcher Schnittstellenbeschreibungen an. Diese unterscheiden sich lediglich durch deren Authentifikationsverfahren. Eine genauere Beschreibung dazu befindet sich im Abschnitt 'Authentifikation'.

Die zur Verfügung gestellten Methoden werden im Folgenden beschrieben.

Die Methode Execute

Mit der Methode `Execute` können Funktionen der enaio®-Server-API aufgerufen werden. Welche Funktionen vom enaio® server angeboten werden, kann im Handbuch enaio® serverapi nachgelesen werden.

Methodenparameter

Der Methode `Execute` wird der Parameter `inParameter` vom Typ `ExecuteParameter` übergeben. Dieser Parameter muss ein Jobobjekt enthalten und kann ein Authentifikationsobjekt und mehrere Eigenschaftsobjekte (`Properties`) enthalten.

Der Aufbau des Authentifikationsobjekts wird im Abschnitt 'Authentifikation' näher beschrieben.

Eigenschaften werden über einen Schlüssel identifiziert. Dieser Schlüssel wird als `key`-Attribut der Eigenschaft gesetzt. Derzeit werden nur zwei Eigenschaften unterstützt, zum einen `useBase64AsString` und zum anderen `maxCountSize`. `useBase64AsString` kann die Werte `true` oder `false` annehmen und steuert die Zeichenkodierung von Dateianhängen der Serverantwort, und zwar unabhängig von den in der Anfrage definierten Rückgabetypen. Im Falle von `true` wird der Dateianhang, als Teil der SOAP-Nachricht, Base64-kodiert übertragen. Dann kann er aus dem Body der Antwortnachricht gelesen werden. Im Falle von `false` wird der Anhang binärkodiert und als Anlage an die SOAP-Nachricht gehängt. Dies gilt für die Daten aller Parameter vom Typ `Base64` (siehe hierzu auch den Abschnitt 'Der Job-Parameter'). `maxCountSize` legt eine Größenbegrenzung in Byte fest. Sprengt eine angefragte Datei die Begrenzung, so wird diese nicht übertragen. Stattdessen wird ein `contentIdentifier` zurückgeliefert, über den man die Datei mit der Methode `getChunk()` herunterladen kann.

Bei der Verwendung von `getChunk()` und `maxCountSize` ist zu beachten, dass die Dateien als Anhang der SOAP-Nachricht und nicht als Base64-Parameter angefragt werden. Bei einigen Serverjobs wie `GetResultList` oder `GetObjDef` wird der Rückgabetyp über das Flags-Attribut gesteuert. Welcher Wert für das Flag-Attribut gesetzt werden muss kann der ServerAPI nachgelesen werden.

Der Job-Parameter

Die Methode `Execute` muss zwingend einen Job-Parameter beinhalten und liefert einen solchen auch wieder zurück. Mit diesem Parameter wird die Anfrage an den enaio®-Server beschrieben, die im folgendem als Serverjob bezeichnet wird.

Der Job-Parameter muss genau einen `Name`-Parameter enthalten und kann mehrere oder keine Parameter- und `FileParameter`-Objekte enthalten. Der `returnCode`-Parameter ist üblicherweise nur in Antworten vom Server enthalten und liefert einen serverjobspezifischen Rückgabewert. Mit dem `Name`-Parameter wird definiert, welcher Serverjob aufgerufen werden soll.

Die Parameterliste, die der Job-Parameter enthält, steht für die Parameter der jeweiligen Serverjobs. Die Parametertypen des Serverjobs werden in der SOAP-Nachricht über das `type`-Attribut zugeordnet. Die möglichen Typen sind:

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

Mit Liste der `FileParameter` können Dateien an den Aufruf angehängt werden. Jeder `FileParameter` setzt sich aus zwei Elementen zusammen. Dem `fileName`, der wahlweise gesetzt werden kann um die angehängte Datei zu benennen, und genau einem `content`-Element, das im Abschnitt 'Der Schnittstellentyp Content' beschrieben wird.

Aufbau des Typs ExecuteParameter

Der schematische Aufbau des `Execute`-Parameters kann wie folgt dargestellt werden:

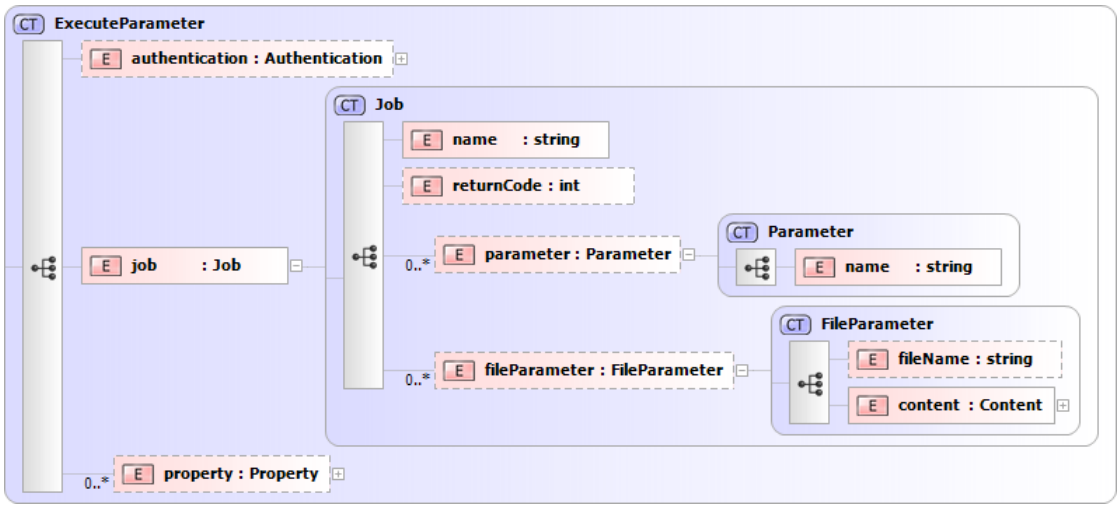


Abbildung 4: Aufbau des Typs `ExecuteParameter` mit der Typstruktur des Job-Parameters (Legende siehe 'Anhang')

Die Methode CreateServerFile

Mit der Methode `CreateServerFile` können Dateien in das `Temp`-Verzeichnis von enaio® webservice übertragen bzw. neu angelegt werden. Konfigurationshinweise hierzu befinden sich im Kapitel 'Konfiguration'.

Methodenparameter

Wie allen Schnittstellenmethoden kann auch der `CreateServerFile`-Methode ein Authentifikationsobjekt übergeben werden. Näheres dazu kann im Abschnitt 'Authentifikation' nachgelesen werden. Wenn nicht nur eine Datei angelegt, sondern auch mit Daten gefüllt werden soll, kann man diese in das Element `attachment` schreiben. In der Umgebung Microsoft .NET muss das Element in beiden Fällen mit einem Byte-Array initialisiert werden! Eine genauere Beschreibung dieses Elements befindet sich im Abschnitt 'Der Schnittstellentyp Content'. Der Dateiname, der für das optionale Element `fileName` anzugeben ist, ist für die serverseitige Geschäftslogik nicht relevant, sollte dennoch gesetzt werden, um die Rückverfolgung zu vereinfachen.

Die Methode gibt ein `contentIdentifier`-Element zurück, der die in enaio® webservice angelegte Datei eindeutig identifiziert.

Aufbau des CreateServerFile-Parameters

Der schematische Aufbau des `CreateServerFile`-Parameters kann wie folgt dargestellt werden:

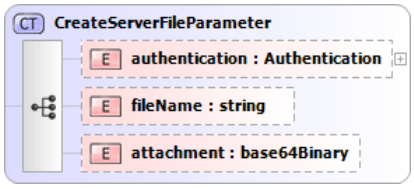


Abbildung 5: Aufbau des Typs `CreateServerFileParameter` (Legende siehe 'Anhang')

Beispiel unter Microsoft .NET mit Visual C#

Wie der EcmWS mit dem Microsoft Visual Studio eingebunden werden kann, lesen Sie im Kapitel 'Verwendung' nach. In diesem Beispiel wird die neue Datei mit dem Inhalt einer Muster.txt gefüllt.

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

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

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

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

Die Methode GetServerFileInfo

Mit dieser Methode ist es möglich, Informationen einer Datei anzufragen, die mit dem Parameter `CreateServerFile` angelegt wurde. In der derzeitigen Implementierung wird nur die Dateigröße in Byte zurückgegeben.

Methodenparameter

Die `GetServerFileInfo`-Methode bekommt neben einem optionalen Authentifikationsobjekt ein `contentIdentifier`-Element, das die Datei auszeichnet, von der der Status abgefragt werden soll.

Die Methode liefert derzeit das Element `size` zurück, in dem die Größe, der angefragten Datei steht (in der Maßeinheit Byte).

Aufbau des GetServerFileInfo-Parameters

Der schematische Aufbau des `GetServerFile`-Parameters kann wie folgt dargestellt werden:

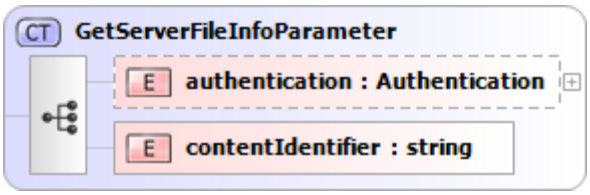


Abbildung 6: Aufbau des Typs `GetServerFileInfoParameter` (Legende siehe 'Anhang')

Beispiel unter Microsoft .NET mit Visual C#

Wie der EcmWS mit dem Microsoft Visual Studio eingebunden werden kann, lesen Sie lesen Sie im Kapitel 'Verwendung' nach.

```
Authentication auth = new Authentication();
auth.user = "root";
auth.password = "optimal";
auth.applicationName = "C# - Beispiel";
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);
```

Die Methode AppendChunk

Die Methode `AppendChunk` dient dazu, zusätzliche Daten an eine Datei anzuhängen, die mit der Methode `CreateServerFile` bereits angelegt wurde.

Methodenparameter

Auch dieser Methode kann ein Authentifikationsobjekt übergeben werden. Mit dem Element `contentIdentifier` muss spezifiziert werden, welcher Datei die Daten hinzugefügt werden sollen. Das Element `attachment` enthält die hinzuzufügenden Daten. Genaue Verwendungshinweise stehen im Abschnitt 'Der Schnittstellentyp Content'.

Nachdem ihr die zusätzlichen Daten hinzugefügt wurden, liefert die Methode die Größe der Datei zurück.

Aufbau des AppendChunk-Parameters

Der schematische Aufbau des `AppendChunk`-Parameters kann wie folgt dargestellt werden:

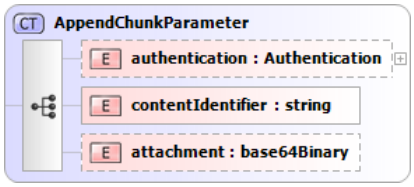


Abbildung 7: Aufbau des Typs `AppendChunkParameter` (Legende siehe 'Anhang')

Beispiel unter Microsoft .NET mit Visual C#

Wie der EcmWS mit dem Microsoft Visual Studio eingebunden werden kann, lesen Sie lesen Sie im Kapitel 'Verwendung' nach. In diesem Beispiel werden der Serverdatei die Daten aus der Datei 'Muster2.txt' hinzugefügt.

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

StreamReader inputFile = File.OpenText(@"C:\Muster2.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);
```

Die Methode GetChunk

Mit dieser Methode können Dateien, die mit dem Parameter `CreateServerFile` angelegt wurden, vollständig oder teilweise von enaio® webservice zur Client-Anwendung übertragen werden.

Methodenparameter

Wie auch bei den anderen Schnittstellenmethoden kann `GetChunk` ein Authentifikationsobjekt übergeben werden. Um die Datei zu identifizieren, die ausgelesen werden sollen, muss das Element `contentIdentifier` gesetzt werden. Da es möglich ist, die Datei fragmentiert abzurufen, muss angegeben werden, wo das Fragment anfangen soll und wie groß es maximal werden darf. Hierfür stehen die Parameter `startPosition` und `size` zur Verfügung. Die Angaben für diese Parameter werden in der Einheit Byte vorgenommen.

Es werden zwei Elemente von der `GetChunk`-Methode zurückgegeben. Zum einen das Element `attachment` und zum anderen das Element `size`. Mit ersterem wird ein Dateiparameter übergeben (näheres dazu steht im Abschnitt 'Der Schnittstellentyp Content'). Mit letzterem wird angegeben, wie viel tatsächlich aus der Datei gelesen wurde. Sollte weniger ausgelesen worden sein als mit der Maximalgröße

festgelegt, dann ist das Ende der Datei erreicht. Ist die Gesamtgröße der Datei restlos durch die spezifizierte Maximalgröße teilbar, so wird beim ersten Lesezugriff außerhalb des Dateibereichs der Wert des Elements `size` auf `-1` gesetzt.

Aufbau des GetChunk-Parameters

Der schematische Aufbau des `GetChunk`-Parameters kann wie folgt dargestellt werden:

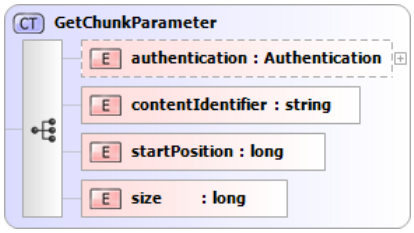


Abbildung 8: Aufbau des Typs `GetChunkParameter` (Legende siehe 'Anhang')

Beispiel unter Microsoft .NET mit Visual C#

Wie der EcmWS mit dem Microsoft Visual Studio eingebunden werden kann, lesen Sie lesen Sie im Kapitel 'Verwendung' nach. In diesem Beispiel wird ein Chunk von der Startposition 0 mit der maximalen Größe 2300 abgerufen und in eine Datei 'Muster.txt' geschrieben.

```
Authentication auth = new Authentication();
auth.user = "root";
auth.password = "optimal";
auth.applicationName = "C# - Beispiel";
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 outStream =
      new FileStream(@"C:\Muster.txt", FileMode.Create);
BinaryWriter outFile = new BinaryWriter(outStream);
outFile.Write(output);
outFile.Close();
```

Die Methode DeleteServerFile

Diese Methode löscht eine zuvor mit dem Parameter `CreateServerFile` oder durch die `Execute`-Methode erzeugte Datei. Es wird empfohlen jede Datei nach ihrer Verarbeitung mit Hilfe dieser Methode zu löschen, da es dafür keinen Automatismus gibt.

Methodenparameter

Der `DeleteServerFile`-Parameter bekommt lediglich ein `contentIdentifier`-Element, um die zu löschende Datei zu spezifizieren. Optional kann ein Authentifikationsobjekt hinzugefügt werden.

Der Rückgabewert der Methode ist vom Typ `Boolean` und gibt Auskunft darüber, ob das Löschen erfolgreich verlief.

Aufbau des DeleteServerFile-Parameters

Der schematische Aufbau des `DeleteServerFile`-Parameters kann wie folgt dargestellt werden:

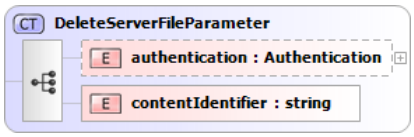


Abbildung 9: Aufbau des Typs `DeleteServerFileParameter` (Legende siehe 'Anhang')

Beispiel unter Microsoft .NET mit Visual C#

Wie der EcmWS mit dem Microsoft Visual Studio eingebunden werden kann, lesen Sie lesen Sie im Kapitel 'Verwendung' nach.

```
Authentication auth = new Authentication();
auth.user = "root";
auth.password = "optimal";
auth.applicationName = "C# - Beispiel";
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);
```

Authentifikation

Für enaio® webservice wurden zwei WSDL-Schnittstellendefinitionen implementiert, die sich in ihren Authentifikationsverfahren unterscheiden.

Apache CXF – WS-Security

Diese Schnittstellendefinition ermöglicht die Anmeldung mit dem Standardverfahren WS-Security (Web Services Security).

Hierfür werden die Authentifikationsparameter, sprich Benutzername und Passwort, im Header der SOAP-Nachrichten eingetragen. Das Passwort wird im Typ `PasswordText` übertragen. Für eine höhere Sicherheit wird empfohlen, die gesamte Kommunikation mit SSL/TLS zu verschlüsseln.

Die Schnittstellendefinition kann mit der folgenden URL erreicht werden:
`http://<Server>:<Port>/EcmWS/services/EcmWsMtomWsSecurity?wsdl`

Die Schnittstelle wurde clientseitig erfolgreich für die Umgebung Java, unter Verwendung der NetBeans-IDE, getestet.

Alternative Authentifikation

Mit der alternativen Schnittstellendefinition können die Authentifikationsparameter, also der Benutzername und das Passwort, im Body der SOAP-Nachricht eingetragen werden.

Die WSDL stellt hierfür einen Authentifikationstyp zu Verfügung, von dem eine Instanz in den Eingabeparametern jeder Schnittstellenmethode gesetzt werden kann. Der Typ sieht ein optionales Element `applicationName` vor, in dem der Name der aufrufenden Anwendung übergeben wird. Es wird empfohlen, den Wert zu setzen, um die Serveraufrufe der betreffenden Anwendung am enaio®-Server identifizieren zu können. Die Authentifikationsobjekte können eine Liste von Eigenschaften beinhalten, welche für zukünftige Verwendung vorgesehen ist. Die restlichen Elemente sind in zwei Gruppen geteilt, von denen mindestens eine übergeben werden muss. Die erste Gruppe besteht aus den Elementen `user`, `password` und `sessionIdentifier`. Dabei muss das Element `user` zwingend vorhanden sein, während die anderen Elemente optional angegeben werden können. Diese Elemente müssen bei jedem Methodenaufruf übergeben werden, denn sie dienen auf der Serverseite dazu, zuvor geöffnete Sessions den zugehörigen Aufrufern zuzuordnen. Genaueres zu diesem Thema findet sich im Abschnitt 'Sitzungsverwaltung'. Die zweite Gruppe setzt sich aus den Pflichtelementen `sessionGUID`, `server` und `port` zusammen. Diese Gruppe wird derzeit noch nicht vom Server unterstützt.

Die Schnittstellendefinition kann mit der folgenden URL erreicht werden:
`http://<Server>:<Port>/EcmWS/services/EcmWsMtom?wsdl`

Die Schnittstelle wurde clientseitig für die Umgebungen Java und Microsoft .NET erfolgreich getestet. Beispiele hierfür werden im Abschnitt 'Verwendung' aufgeführt.

Aufbau des Authentifikationsobjekts:

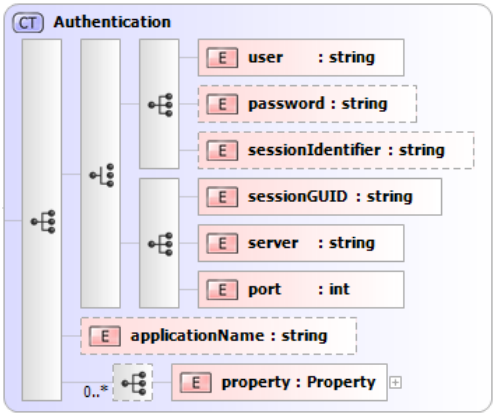


Abbildung 10: Struktur von Authentifikationsobjekten (Legende siehe 'Anhang')

Sitzungsverwaltung

Um Anfragen an den enaio®-Server zu senden, müssen Sitzungsobjekte, Sessions erstellt werden. Eine Session hält die Verbindung zum enaio®-Server offen und über sie können Serverjobs abgesetzt werden.

enaio® webservice unterstützt eine vollständige serverseitige Sitzungsverwaltung.

Jeder Schnittstellenmethode muss bei jedem Aufruf ein Authentifikationsobjekt übergeben werden, mit dem enaio® webservice eine Session am enaio®-Server anmeldet. Diese Session kann nach ihrer Erzeugung eindeutig dem Authentifizierungsobjekt der Client-Anwendung zugeordnet werden. Das heißt, dass bei der ersten Anfrage eines Clients eine neue Session bei enaio® webservice erstellt und verwendet wird und für folgende Anfragen desselben Clients wiederverwendet werden kann. Für die Wiederverwendung muss der Client stets dasselbe Authentifizierungsobjekt übergeben. Die Sessions der einzelnen Clients werden in einem Sessionpool gehalten. Da von Serverseite aus nicht absehbar ist, wann eine Session nicht mehr benötigt wird,

kann auf der Administrationsseite von enaio® webservice ein Timeout eingestellt werden, nach dessen Ablauf die Session geschlossen werden.

Die Sitzungsverwaltung wurde für Mehrläufigkeit optimiert. Sollte ein Client versuchen, eine Anfrage ausführen zu lassen, obwohl er zuvor eine Anfrage gesendet hat, die noch nicht beantwortet wurde, kann abhängig von der Sitzungskapazität, der zweite Aufruf nebenläufig bearbeitet werden. Beträgt die Sitzungskapazität `1`, werden zusätzliche Aufrufe zurückgestellt, bis ihre Vorgänger abgearbeitet wurden. Des Weiteren ist es möglich, Sessions automatisch von enaio® webservice validieren zu lassen. Bei der Validierung wird geprüft, ob die Verbindung der Session zum enaio®-Server noch besteht. Die Validierung kann in konfigurierbaren Zeitabständen und vor der Verwendung einer Session durchgeführt werden. Nähere Informationen hierzu finden sich im Kapitel 'Konfiguration'.

Die Zuordnung von Sessions und Authentifikationsobjekten wird mit Hilfe der Elemente `user`, `password`, `sessionIdentifier`, `sessionGUID`, `server` und `port` realisiert. Wenn diese Elemente bei wiederholten Anfragen dieselben Werte haben wie zur Zeit der Erstellung der Session, dann kann die Session der Anfrage neu zugeordnet werden. Welche dieser Elemente zwingend angegeben werden müssen, wird im Abschnitt 'Alternative Authentifikation' genauer beschrieben.

Es besteht die Möglichkeit, dass sich zwei Clients mit demselben Benutzerkonto anmelden und somit dasselbe Authentifikationsobjekt senden. In diesem Fall würden die beiden Clients dieselbe Session benutzen. Sollte es für solche Anwendungsfälle nötig sein, dass jeder Client seine eigene Session verwendet, etwa weil aufeinanderfolgende Anfragen voneinander abhängig sind, so können die Clients das Element `sessionIdentifier` mit einem beliebigen Wert belegen, der für die Zuordnung von Authentifikationsobjekt und Session verwendet wird.

Der Schnittstellentyp Content

Der Typ `Content` dient zur Übertragung von Dateien zu enaio® webservice. Er ist Bestandteil des `Execute`-Parameters der `Execute`-Methode und kann sowohl an sie übergeben werden, als auch von ihr zurückgegeben werden. Der Typ setzt sich aus drei optionalen Elementen zusammen.

Das `URI`-Element spezifiziert einen Ort, an dem die zu übertragende Datei zu finden ist. Dieser URI (Uniform Resource Identifier) muss von enaio® webservice aufgelöst werden können und erreichbar sein. Verweise ins Dateisystem entsprechen dem Schema `file:///<Pfad>`.

Es können außerdem Dateien verarbeitet werden, die zuvor von der Methode `CreateServerFile` im `Temp`-Verzeichnis von enaio® webservice angelegt wurden. Hierzu bietet der Typ `Content` das Element `contentIdentifier` an, dass dafür auf den Wert gesetzt werden muss, den die `CreateServerFile`-Methode ursprünglich als Rückgabewert geliefert hat.

Des Weiteren besteht die Möglichkeit, MTOM-codierte Daten zu übertragen. Wie diese Daten zur Verfügung gestellt werden hängt von der jeweiligen Umgebung ab und muss in den entsprechenden Spezifikationen nachgelesen werden. Für die Realisierung dieses Verfahrens steht das Element `attachment` zur Verfügung. In dieses wird beispielsweise in der Umgebung SoapUI eine `cid` eingetragen, die mit einem Dateianhang verbunden ist. In anderen Umgebungen wie Microsoft .NET oder Java können die Binärdaten direkt in das Attribut der generierten Klasse geschrieben werden. Bei Microsoft .NET ist zu beachten, dass die Binärdaten als Arrays im Speicher gehalten werden, daher wird empfohlen, für die Verarbeitung großer Dateien die `Chunk`-Methoden zu verwenden. Das Element `attachment` ist neben dem `Execute`-Parameter auch Bestandteil einiger `Chunk`-Methodenparameter bzw. Rückgabewerte. Beispiele für die Verwendung dieses Elements können im Kapitel 'Verwendung' nachvollzogen werden.

Grafische Darstellung des Schnittstellentyps `Content`:

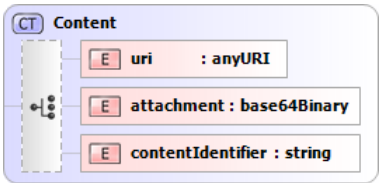


Abbildung 11: Struktur von `Content`-Objekten (Legende siehe 'Anhang')

Verwendung

In diesem Abschnitt werden einige beispielhafte Jobaufrufe für die Umgebungen SoapUI, Microsoft .NET und Java vorgestellt.

Alle Umgebungen müssen für enaio® webservice konfiguriert werden, wofür die URL zur Schnittstellendefinition gebraucht wird. Um diese URL zu ermitteln, muss auf der Hauptseite von enaio® webservice der Link **Service-Schnittstellen** und auf der Folgeseite der Link `{http://schemas.optimalsystems.de/OsEcm/Ws} EcmWsMtomSoapService` aktiviert werden. Anschließend wird die technische Schnittstellendefinition angezeigt und ihre URL kann aus der Adresszeile des Browsers kopiert werden. Außerdem muss sichergestellt sein, dass enaio® webservice während der Umgebungskonfiguration hochgefahren ist.

Einige der Beispiele setzen eine passende Objektdefinition auf dem Zielserver voraus. Diese müssen für spezielle Testsysteme angepasst werden. Die Mindestvoraussetzungen sind: ein Schrank `Pressearchiv` in dessen Ordner Objekte eines Bilddokumenttyps `Farbbilder` angelegt werden können und ein Indexdatenfeld `Autor` für diesen Dokumenttyp.

Abbildung 12: Mögliche Objektdefinition

§ krn.GetServerInfo

  Mit diesem Aufruf, kann der Zustand des Servers angefragt werden. Was der Server zurücksendet, kann über den Parameter `Info` gesteuert werden. In den zugehörigen Beispielen wird die IP des Servers als Antwort gesendet, wofür der Wert `4` des Parameters `Info` sorgt.

§ dms.GetResultList

  Dieser Aufruf überträgt eine Anfrage zum Server und liefert eine passende Ergebnisliste zurück. Die Anfrage ist an die oben beschriebene Objektdefinition angepasst und fragt alle Dokumente des Typs `Farbbilder` in allen Ordnern des Schrankes `Pressearchiv` an. Genaue Informationen zum Aufbau von Serveranfragen können im Handbuch für die Serverschnittstelle nachgelesen werden. Übergeben wird die Anfrage als Wert des Parameters `XML`. Die Ergebnisliste des Aufrufs kann aus den Parametern der Rückgabe ausgelesen und angezeigt werden.

§ dms.XMLInsert

Mit dms.XMLInsert können neue Objekte im Archivsystem enaio® angelegt werden. Die Objektbeschreibung bzw. die Verschlagwortung des anzulegenden Objekts und sein Standort, werden in XML-Notation im Parameter `XML` übergeben. In den Beispielen wird ein Dokument des Typs `Farbbilder` in einem bestimmten Ordern des Schrankes `Pressearchiv` angelegt. Um welchen Ordner es sich handelt, wird mit dem Feld `folder_id` festgelegt. In dieses muss passend zum Archivsystem die Objekt-ID des Zielordners eingetragen werden. Detaillierte Informationen zum Aufbau der Objektbeschreibung befinden sich im Handbuch für die Serverschnittstelle. Darüber hinaus kann eine Datei mit gesendet werden, die als Dokumentdatei im zu erstellendem Objekt hinterlegt wird. Diese Datei wird als Anhang mit einem Dateiparameter verbunden. Der Dateiname des Parameters wurde in den Beispielen auf `Muster` gesetzt. Die Objekt-ID des erzeugten Objekts kann aus den Rückgabeparametern des Aufrufs gelesen werden.

§ std.StoreInCache

std.StoreInCache ermöglicht es, Dateien aus dem Archivsystem enaio® herunterzuladen. Von welchem Objekt die Datei heruntergeladen wird, wird über die Parameter `dwObjectID` und `dwObjectType` festgelegt. Dabei muss ersterer auf die Objekt-ID und letzterer auf die Typ-ID des Zielobjekts gesetzt werden. Die Datei kann aus den Dateirückgabeparametern des Aufrufs gelesen und gespeichert werden.

SoapUI (2.5)

Allgemein

SoapUI ist eine Software, die speziell auf das Testen von Webservices ausgelegt ist. Vorteil von SoapUI ist, dass Serveranfragen direkt auf SOAP-Ebene abgesetzt werden können. Außerdem ist es die einfachste und schnellste Variante um zu testen, ob eine Anfrage an enaio® webservice funktioniert. Im Abschnitt 'Verweise' ist ein Link zu finden, mit dem SoapUI heruntergeladen werden kann.

Nach der Installation kann SoapUI im Programmmenü von Microsoft Windows aktiviert werden. Anschließend muss `STRG+N` gedrückt werden, um den Dialog zum Anlegen eines neuen Projekts zu öffnen. Nach Wahl eines beliebigen Projektnamens, muss im Feld **Initial WSDL/WADL** die URL zur Schnittstellendefinition angegeben werden.

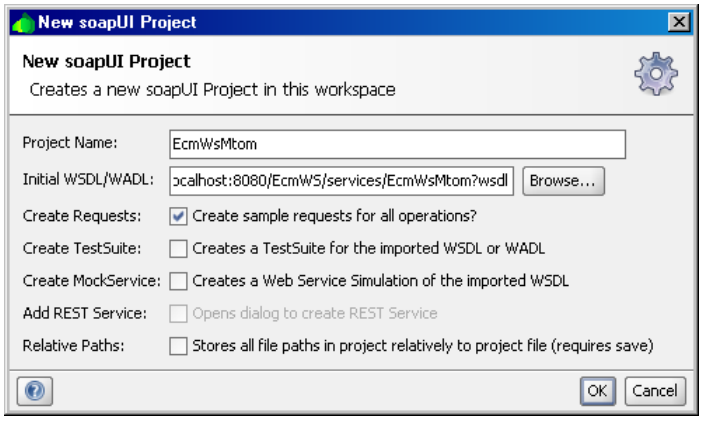


Abbildung 13: Dialog zum Anlegen eines neuen Projekts

Wurde das Projekt erfolgreich angelegt, wird es im Navigator von SoapUI angezeigt. SoapUI fertigt automatisch eine Anfrage für jede Dienstfunktion an. Um die nachfolgenden Beispiele zu testen, werden der Beispielquelltext in die Anfrage `Request 1` der Funktion `execute` kopiert. Mit der Pfeilschaltfläche im oberen Teil der Anfrage, kann die Anfrage zu enaio® webservice übertragen werden.

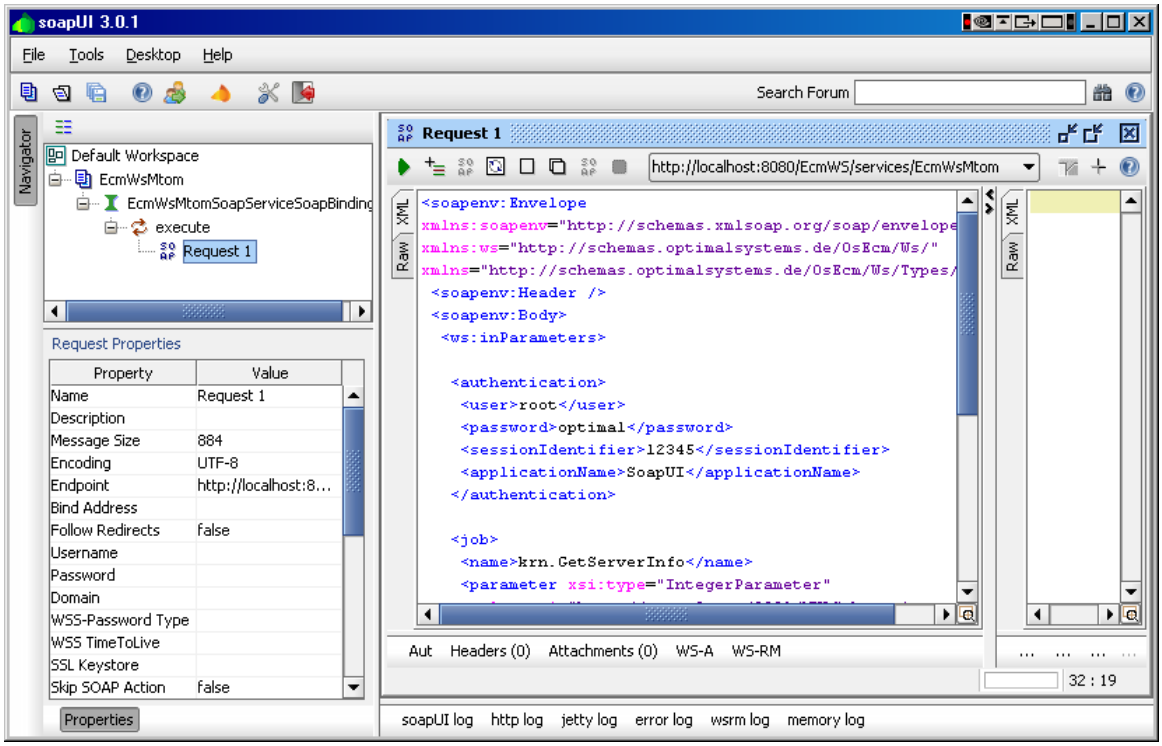


Abbildung 14: Fertig konfiguriertes SoapUI

Der Job 'dms.XMLInsert' überträgt eine Datei als Anhang zum Server. Die Datei muss mit der `cid` aus dem SOAP-Beispielcode verknüpft werden. Im unteren Teil der Anfrage gibt es eine Schaltfläche **Attachments**. Darüber können eine Liste mit Anhängen an die Anfrage und weitere Schaltflächen sichtbar gemacht werden. Über die Schaltfläche **Aut** kann eine neue Datei an die Nachricht angehängt werden. Nach dem Hinzufügen einer Datei, muss in der Spalte `Part` die `cid` aus der Anfrage ausgewählt werden.

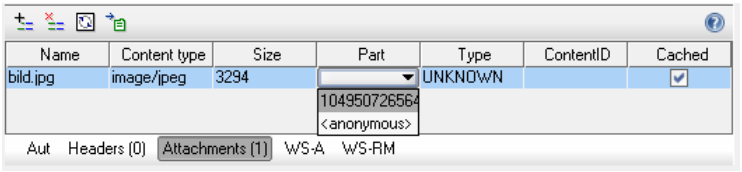


Abbildung 15: Dialog für Dateianhänge

Der Job 'std.StoreInCache' forder eine Datei vom Archivsystem an. Diese Datei ist als Anhang in der Antwort enthalten. Im unteren Teil der Antwort ist eine weitere Schaltfläche **Attachments** enthalten. Mit ihr können Anhänge in der Antwort angezeigt werden. Mit der Schaltfläche **Exports the selected attachment to a file** können Anhänge in Dateien gespeichert werden.

Beispiele

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="Pressearchiv">
        <ObjectType name="Farbbilder" 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="Pressearchiv">
        <ObjectType name="Farbbilder" type="DOCUMENT">
         <Object folder_id="17">
          <Fields>
           <Field name="Autor">Max Muster</Field>
          </Fields>
         </Object>
        </ObjectType>
       </Archive>
      </DMSData>]]>
     </value>
    </parameter>
```

```
    <fileParameter>
     <fileName>Muster</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>

   <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 mit Microsoft Visual Studio 2008 und Visual C#

Allgemein

Mit Microsoft .NET ist es möglich, sich an enaio® webservice anzubinden. Voraussetzung hierfür ist mindestens die Version 3.0. Anders als bei SoapUI werden unter Microsoft .NET Klassendateien aus der WSDL generiert, was ein objektorientiertes Programmieren gegen den Dienst ermöglicht. Nach dem Generieren der Klassen, können Objekte zur Anmeldung und zum Aufrufen von Jobs verwendet werden. Als Entwicklungsumgebung für die Beispiele wird Microsoft Visual Studio 2008 verwendet. Im Abschnitt 'Verweise' steht ein Link zur Verfügung, mit dem Visual Studio 2008 Express heruntergeladen werden kann.

Um den Beispielcode auszuführen, muss im Microsoft Visual Studio 2008 eine neues Projekt als Konsolenanwendung der Sprache C# angelegt werden. Dafür müssen im Hauptmenü die Einträge **Datei**, **Neu** und **Projekt…** ausgewählt werden. Im folgenden Dialog können die entsprechenden Einstellungen gemacht werden.

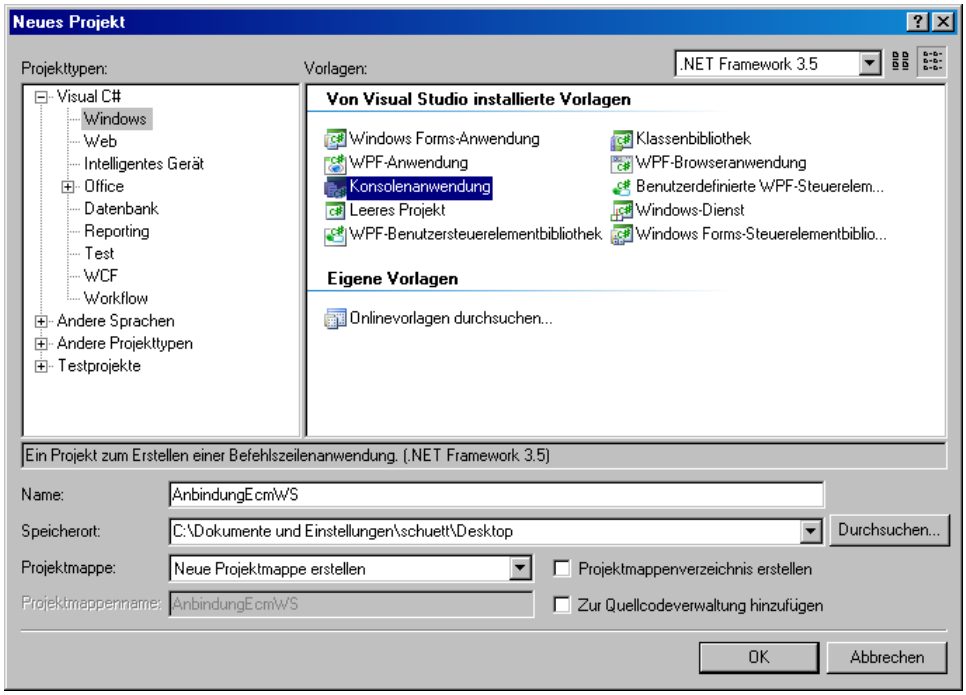


Abbildung 16: Dialog zum Anlegen eines neuen Projekts

Nach dem Anlegen wird das Projekt im Projektmappen-Explorer angezeigt. Um das Projekt an enaio® webservice anzubinden, müssen im Hauptmenü die Einträge **Projekt** und **Dienstverweis hinzufügen…** ausgewählt werden. Im nachfolgenden Dialog, muss im Feld **Adresse** die URL zur Schnittstellendefinition angegeben werden. Im Feld **Namespace** kann der Name des Namensraums eingetragen werden, unter dem später die generierten Klassen zu finden sind.

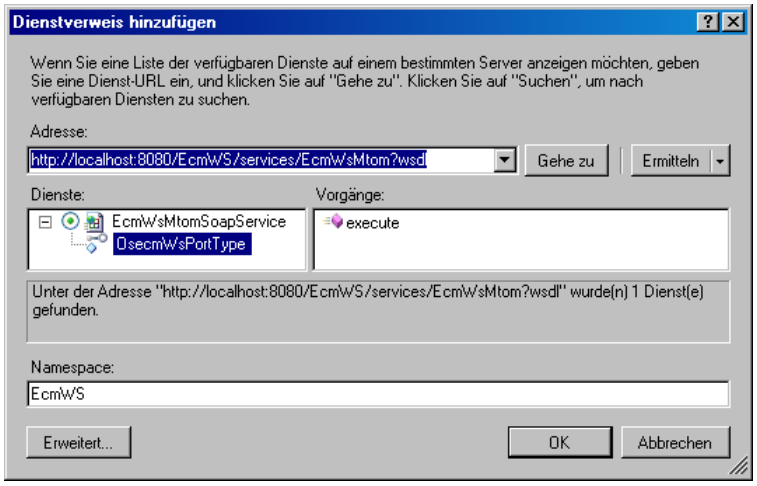


Abbildung 17: Anbindung des Projekts an enaio® webservice

Nach Bestätigung der Einstellungen werden die Klassen automatisch generiert. Mit der Importdirektive `using <Projektname>.<Namensraum>;` wird der Zugriff auf die generierten Klassen im Hauptprogramm sichergestellt. An dieser Stelle ist Microsoft Visual Studio 2008 fertig konfiguriert und der Beispielcode kann in das Hauptprogramm kopiert und ausgeführt werden.

Beispiele

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=\"Pressearchiv\">" +
              "<ObjectType name=\"Farbbilder\" " +
               "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='Pressearchiv'>" +
              "<ObjectType name='Farbbilder' " +
               "type='DOCUMENT'>" +
               "<Object folder_id='17'>" +
                "<Fields>" +
                 "<Field name='Autor'>Max Muster</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 = "Muster";
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 mit NetBeans 6.7

Allgemein

Ähnlich wie unter Microsoft .NET werden auch mit Java aus der WSLD Klassendateien erzeugt, um objektorientiert programmieren zu können. Auch hier können Objekte für die Anmeldung und

Konfiguration verwendet werden. Die Beispiele wurden für die Entwicklungsumgebung NetBeans 6.7 getestet.

Um die Beispiele auszuprobieren, muss ein neues Projekt erstellt werden. Hierfür müssen im Hauptmenü die Einträge **Datei** und **Neues Projekt…** ausgewählt werden. Im folgenden Dialog werden dann die Kategorie **Java** und der Projekttyp **Javaanwendung** markiert. Auf der nächsten Dialogseite können **Projektname** und **Zielort** festgelegt werden.

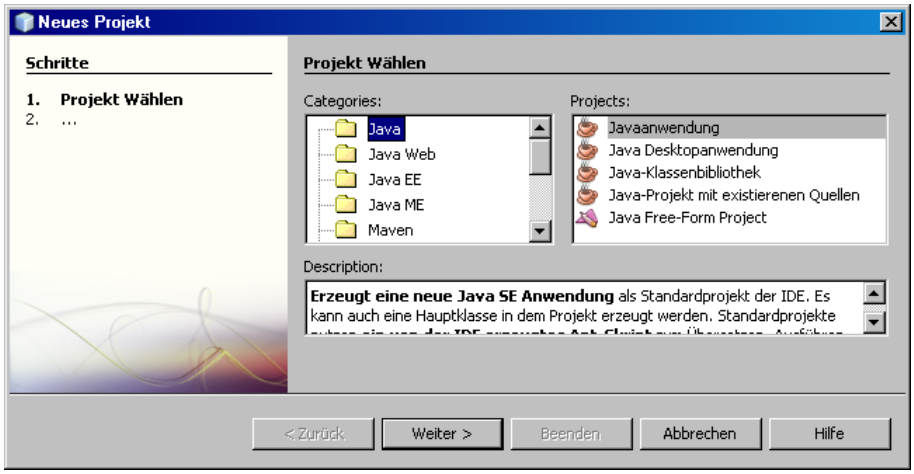


Abbildung 18: Dialog zum Anlegen eines neuen Projekts

Nach dem Bestätigen mit der Schaltfläche **Beenden** wird das Projekt angelegt und im Fenster **Projekte** angezeigt. Im Kontextmenü des Projekts müssen nun die Einträge **Neu** und **Web Service Client…** ausgewählt werden. Im anschließenden Dialog wird die Optionsschaltfläche **WSDL URL** gewählt und im zugehörigen Feld die URL zu Schnittstellendefinition angegeben.

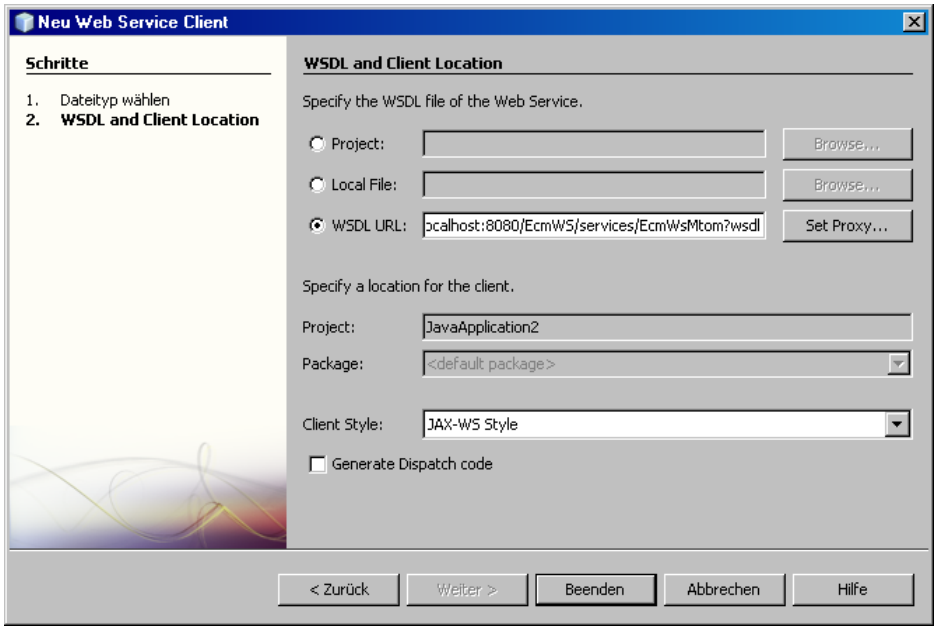


Abbildung 19: Dialog zum Anlegen eines neuen Projekts

Nach dem Bestätigen über die Schaltfläche **Beenden** werden die Schnittstellenklassen generiert und können über die Importdirektiven `import de.optimalsystems.schemas.osecm.ws.*;` und `import de.optimalsystems.schemas.osecm.ws.types.*;` im Hauptprogramm verfügbar gemacht werden. Damit ist das Testprojekt fertig konfiguriert und der Beispielcode kann eingefügt und ausgeführt werden.

Beispiele

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='Pressearchiv'>" +
                        "<ObjectType name='Farbbilder' " +
                            "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='Pressearchiv'>" +
                 "<ObjectType name='Farbbilder' " +
                  "type='DOCUMENT'>" +
                  "<Object folder_id='17'>" +
                   "<Fields>" +
                    "<Field name='Autor'>Max Muster</Field>" +
                   "</Fields>" +
                  "</Object>" +
                 "</ObjectType>" +
                "</Archive>" +
               "</DMSData>";
xml.setValue(query);

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

Content content = new Content();
```

```
content.setAttachment(data);

FileParameter filePara = new FileParameter();
filePara.setFileName("Muster");
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));
```

Anhang

Verweise

Download: SoapUI

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

Download: Visual Studio 2008 Express

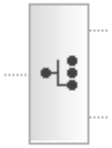
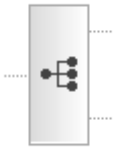
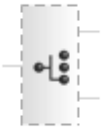
http://www.microsoft.com/germany/express/download/downloaddetails.aspx?p=iso

Download: NetBeans IDE (Java)

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

Legende für Aufbaubilder

	Schnittstellenelement: Die Instanz eines Schnittstellentyps.
	Schnittstellentyp: Komplexer Typ, der sich aus Elementen zusammensetzt.

	Auswahlelement: Es muss genau eines der möglichen Elemente vorhanden sein.
	Sequenzelement: Es müssen alle möglichen Elemente in dargestellter Reihenfolge vorhanden sein.
	Optionales Schnittstellenelement: Diese Element muss nicht vorhanden sein und darf nicht mehr als einmal vorkommen.
	Optionale Auswahl: Muss genau eines der möglichen Elemente beinhalten oder kann leer sein.
	Variantes Element: Optionales Element, das beliebig oft vorhanden sein kann. Das Element muss nicht zwingend vorhanden sein.