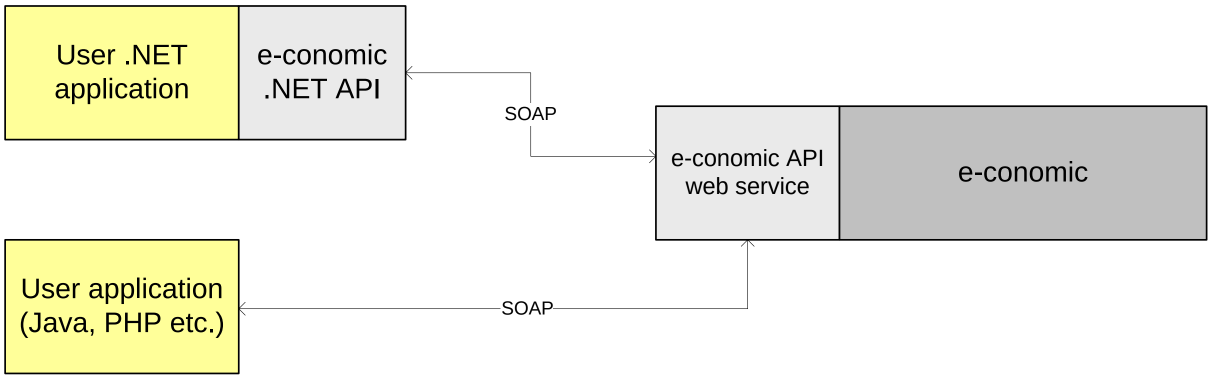
Table of contents

Introduction

The e-conomic API[1] makes it possible to integrate other systems with e-conomic. The API is an interface between e-conomic and other systems. Through the API other systems can easily create, manipulate and delete entities in e-conomic.

The API is based on web services using SOAP 1.2 which is supported by all modern programming languages and development tools. Furthermore, we provide a special assembly for .NET developers making it even more elegant to integrate other systems with e-conomic. We call this assembly the *e-conomic .NET API*.

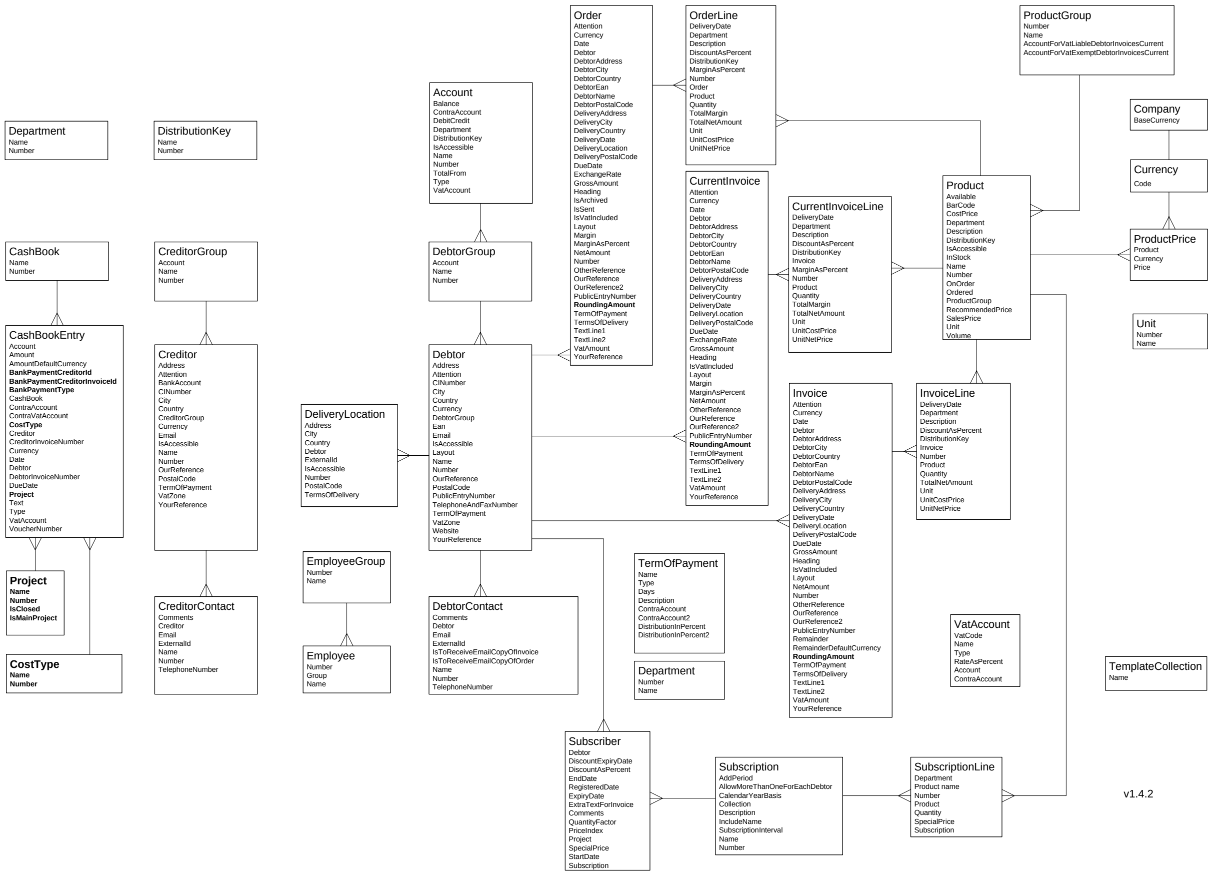
As shown in the flowchart below it is possible to communicate with the e-conomic API in two ways: by using the e-conomic .NET API or by invoking web service methods directly through the *e-conomic API web service*.



[1] API is an abbreviation for *Application Programming Interface*.

The e-conomic model

The e-conomic model consists of different entity types such as **Debtor**, **Order** and **Invoice**. Each entity type has a number of properties and operations. The diagram below shows the entity types in the model with their properties and the most important relations between them.



The type of a property can be a value type (such as string, enumeration type, integer or decimal) or a reference type (which refers to other entities). Example: The property **Name** on entity type **Debtor** is a string, the property **Price** on entity type **ProductPrice** is a decimal and the property **Debtor** on entity type **Invoice** refers to an entity of type **Debtor**.

Using the API

e-conomic .NET API

The e-conomic .NET API is a special .NET assembly that encapsulates all communication with the e-conomic web service. When using the .NET API, developers can manipulate entities as if they where normal local objects. There is a one-to-one mapping between the e-conomic entities and the available entity classes in the .NET API. The e-conomic .NET API requires *Microsoft .NET Framework 2.0*.

To compile a user-application that uses the .NET API, the application must refer to the assembly `Economic.Api.dll`. Example using Microsoft C# compiler:

```
csc UserApp.cs /reference:Economic.Api.dll
```

The e-conomic .NET API is divided into three namespaces:

Namespace	Description
Economic.Api	Contains interfaces for all e-conomic entities and their utility classes.
Economic.Api.Data	Contains data interfaces for all e-conomic entities. See the section 'Data classes' later in this document.
Economic.Api.Exceptions	Contains e-conomic exception classes.

When using the API, all three namespaces must be brought into scope by the user application. In C# this is done by using the **`using`** keyword.

Connecting to e-conomic

To connect to e-conomic, create a new instance of the **`EconomicSession`** class and invoke the method **`Connect`**. The **`Connect`** method takes three arguments: *agreement number*, *user name* and *password*. If connecting fails an **`AuthenticationException`** is thrown.

To disconnect, invoke the method **`Disconnect`** with no arguments.

Example

```
using System;
using Economic.Api;
using Economic.Api.Data;
using Economic.Api.Exceptions;

class UserApp
{
    public static void Main(string[] args)
    {
        IEconomicSession session = new EconomicSession();
        session.Connect(123456, ”username”, ”password”);

        // Application code

        session.Disconnect();
    }
}
```

Creating entities

The session class contains a property for each entity type. Each of these properties refers to an instance of a utility class that contains methods relevant for the entity type. For example, the session contains the property **Debtor** of type **`IDebtorUtil`** for the entity type **Debtor**.

Example

To create a new debtor, invoke **`Create`** through the property **`EconomicSession.Debtor`**. The following example shows how to create a debtor.

```
IDebtorGroup debtorGroup = (...)

// Create new debtor.
Debtor debtor = session.Debtor.Create("100", debtorGroup, "Debtor name",
                                       VatZone.HomeCountry);
```

Manipulating entities

To manipulate entities, use properties on entity classes.

Example

```
IDebtor debtor = (...)

// Fetch debtor info and write it to the console.
Console.Write("Debtor number: " + debtor.Number);
Console.Write("Debtor name: " + debtor.Name);

// Modify debtor info.
debtor.Name = "New debtor name";
debtor.Address = "New address";
...
```

Invoking methods

Methods that apply to one specific entity type are placed in the entity class. These types of methods are referred to as *entity methods.*

Example

To book a current invoice, invoke method **`Book`** on the entity instance.

```
ICurrentInvoice currentInvoice = (...)

// Book current invoice.
currentInvoice.Book();
```

Methods that apply to many entities of the same entity type are placed in the corresponding *`Util`-class and can be accessed through the active **`IEconomicSession`** instance. These types of methods are referred to as *session methods.*

Example

To fetch all orders, invoke method **`GetAll()`** on **`IOrderUtil`** through the **`IEconomicSession.Order`** property.

```
// Fetch all orders.
IOrder[] orders = session.Order.GetAll();
```

Searching for entities

To search for entities, use **`Find*`** session methods.

Example

Find debtors by name and an invoice by number.

```
// Find debtors by name.
IDebtor[] debtors = session.Debtor.FindByName(”e-conomic International A/S”);
if(debtors.Length > 0)
{
    Console.WriteLine(”Debtors found.”);
}
else
{
    Console.WriteLine(”No debtors found.”);
}

// Find invoice by number.
IInvoice invoice = session.Invoice.FindByNumber(123);
if(invoice != null)
{
    Console.WriteLine(”Invoice found”);
}
else
{
    Console.WriteLine(”Invoice not found”);
}
```

Deleting entities

To delete an entity, invoke the **`Delete`** entity method on the entity.

Example

Delete a unit:

```
IUnit unit = (...)

// Delete unit.
unit.Delete();
```

Exceptions

To enable API users to handle errors in an elegant way, the e-conomic .NET API uses exceptions. The following table shows the different exceptions:

Exception	Description
`AccessException`	Base class for all exceptions which are caused by access violations.
`AuthenticationException`	This is the exception thrown when the client has not been authenticated on the server.
`AuthorizationException`	This exception is thrown when an API user tries to use parts of the API that he/she does not have access rights to.
`EconomicApiException`	Base class for all exceptions thrown by the e-conomic API.
`IntegrityException`	This exception is thrown if an operation violates integrity constraints of the e-conomic model. For example it is not legal to delete a debtor, if it has current invoices, because all current invoices must have an existing debtor.
`ServerException`	Base class for all exceptions which are caused by an internal server error.
`UserException`	Base class for all exceptions which are caused by the user.
`ValidationException`	This exception is thrown if an operation tries to set data to an invalid state.

Sample application

To give an example of how to use the .NET API, e-conomic provides a sample application written in C#.

To compile the sample application, run the compiler with the following arguments:

```
C:\...\dotnet>csc BasicSample.cs /reference:Economic.Api.dll
```

To run the sample application, execute BasicSample.exe:

```
C:\...\dotnet>BasicSample
```

When the application starts it asks for the agreement number, the user name and the password. Enter the credentials to go to the main menu.

e-conomic API web service (SOAP)

To access the e-conomic API from non-.NET languages, it is possible to use the e-conomic web service directly using SOAP 1.2[2]. The documented WSDL[3] for the e-conomic web service can be found at https://www.e-conomic.com/secure/api1/EconomicWebService.asmx?wsdl.

A complete overview of the e-conomic web service operations can be found at https://www.e-conomic.com/secure/api1/EconomicWebService.asmx.

There exist many SOAP client frameworks that generate stubs based on WSDLs. For instance, to access the e-conomic API from *PHP*, use the *PEAR SOAP* package[4]. However, in this section the *Apache Axis 1.4* framework[5] for the *Java* programming language is used. For other SOAP frameworks for other programming languages most of the concepts are similar to Axis.

To generate stubs with the Axis framework, run the WSDL2Java-program:

```
C:\...\java> java -cp axis.jar;commons-discovery-0.2.jar;commons-logging-
1.0.4.jar;jaxrpc.jar;saaj.jar;log4j-1.2.8.jar;wsdl4j-1.5.1.jar
org.apache.axis.wsdl.WSDL2Java
https://www.e-conomic.com/secure/api1/EconomicWebService.asmx?WSDL
```

The WSDL2Java-program generates a class for each *complex type* specified in the WSDL specification and a class **`EconomicWebServiceSoap`** that contains all web service operations. These classes will be put in the **`com.e_conomic`** package.

Connecting to e-conomic

To connect to e-conomic, create an instance of the default web service locator and get an instance of the **`EconomicWebServiceSoap`** by invoking the **`getEconomicWebServiceSoap`** method and invoke **`connect`**. The **`connect`** method takes three arguments: *agreement number*, *user name* and *password* respectively.

To disconnect, invoke the method **`disconnect`** with no arguments.

Example

```
import javax.xml.rpc.ServiceException;
import com.e_conomic.*;

public class UserApp {
    public static void main(String[] args){

        EconomicWebServiceLocator locator = new EconomicWebServiceLocator();
        locator.setMaintainSession(true);

        EconomicWebServiceSoap session = locator.getEconomicWebServiceSoap();
        session.connect(123456, ”username”, ”password”);

        // Application code.

        session.disconnect();
    }
}
```

[2] The SOAP specification is currently maintained by the *XML Protocol Working Group*. See http://www.w3.org/2000/xp/Group/ for more information.
[3] WSDL (Web Services Description Language) is a standardized language to describe web services. See http://www.w3.org/TR/wsdl for more info.
[4] More information is available at http://pear.php.net/package/SOAP.
[5] See http://ws.apache.org/axis/ for more info.

Creating entities

To create a new entity of an entity type, invoke the corresponding **`<entity type>_Create`** web service operation. The create operation returns *a handle* to the new entity.

Example

```
DebtorGroupHandle debtorGroup = (...)

// Create new debtor.
DebtorHandle debtor = session.debtor_Create("100", debtorGroup, "Debtor name",
                                            VatZone.HomeCountry);
```

Manipulating entities

To manipulate the properties of entities, use the **`<Entity type>_Get<Property name>`** and **`<Entity type>_Set<Property name>`** web service operations. The first argument of the get and set operations is a handle for the entity.

Example

```
DebtorHandle debtor = (...)

// Fetch debtor info and write it to the console.
Console.Write("Debtor number: " + session.debtor_GetNumber(debtor));
Console.Write("Debtor name: " + session.debtor_GetName(debtor));

// Modify debtor info.
session.debtor_SetName(debtor, "New debtor name");
session.debtor_SetAddress(debtor, "New address");
...
```

Invoking methods

Methods for a specific entity type have the name prefix "**`<Entity type>_`**". The first argument of these methods is often a handle for a certain entity.

Example

Book a current invoice.

```
CurrentInvoiceHandle currentInvoice = (...)

// Book current invoice.
session.currentInvoice_Book(currentInvoice);
```

Example

Fetch all orders:

```
// Fetch all orders.
OrderHandle[] orders = session.order_GetAll();
```

Searching for entities

To search for entities, use the "**`<Entity type>_Find*`**" methods on the session object.

Example

Find debtors by name and an invoice by number:

```
// Find debtors by name.
DebtorHandle[] debtors =
    session.debtor_FindByName("e-conomic International A/S");

if(debtors.length > 0)
{
    Console.WriteLine("Debtors found.");
}
else
{
    Console.WriteLine("No debtors found.");
}

// Find invoice by number.
InvoiceHandle invoice = session.invoice_FindByNumber(123);
if(invoice != null)
{
    Console.WriteLine("Invoice found");
}
else
{
    Console.WriteLine("Invoice not found");
}
```

Deleting entities

To delete an entity, invoke the **`<Entity name>_Delete`** web service operation.

Example

Delete a unit.

```
UnitHandle unit = (...)

// Delete unit.
session.unit_Delete(unitHandle);
```

SOAP faults

If an error occurs, then the e-conomic web service will return a SOAP fault. The SOAP fault contains a **`faultactor`** and a **`faultstring`** field.

If the error is caused by the user, then the **`faultactor`** is set to "**`{http://schemas.xmlsoap.org/soap/envelope/}Client`**". Otherwise, if the error is caused by the e-conomic server, then **`faultactor`** is set to "**`{http://schemas.xmlsoap.org/soap/envelope/}Server`**".

The **`faultstring`** field contains an error description, which has the following form:

`System.Web.Services.Protocols.SoapException:` [*e-conomic .net exception*]**:** [*error message*]

where [*e-conomic .net exception*] is the qualified name of the exception (see the section about .NET API exceptions) and [*error message*] is a human readable error message.

Data classes

When manipulating properties on entities each small change will result in one SOAP-message sent to the e-conomic server. The following code creates a debtor via .NET API and SOAP.

.NET API:

```
IDebtor debtor = session.Debtor.Create(...);
debtor.Name = (...);
debtor.Address = (...);
debtor.PostalCode = (...);
debtor.City = (...);
debtor.Country = (...);
debtor.Currency = (...);
```

SOAP (Axis):

```
DebtorHandle debtor = session.debtor_Create(...);
session.debtor_SetName(debtor, ...);
session.debtor_SetAddress(debtor, ...);
session.debtor_SetPostalCode(debtor, ...);
session.debtor_SetCity(debtor, ...);
session.debtor_SetCountry(debtor, ...);
session.debtor_SetCurrency(debtor, ...);
```

Each code line will result in one network round-trip. In most cases the round-trip time is relatively long compared to the actual request processing time. Too many round-trips in an API application can lead to poor performance.

This can be avoided by using *data classes*. A data class is a lightweight class, which contains all data of an entity type. Data classes enable API users to create or fetch an entity using only one round-trip.

Creating entities

Each data class object must adhere to the constraints of the entity type, because all the properties of the entity are set simultaneously. The following examples show how to create a debtor using data classes.

.NET API:

```
IDebtorData debtorData = session.DebtorData.Create(...);
debtorData.Currency = (...); // The currency must be set.
debtorData.Name = (...);
debtorData.Address = (...);
debtorData.PostalCode = (...);
debtorData.City = (...);
debtorData.Country = (...);
session.Debtor.CreateFromData(debtorData);
```

SOAP (Axis):

```
DebtorData debtorData = new DebtorData();
debtorData.setCurrency(...); // The currency must be set.
debtorData.setName(...);
debtorData.setAddress(...);
debtorData.setPostalCode(...);
debtorData.setCity(...);
debtorData.setCountry(...);
session.debtor_CreateFromData(debtorData);
```

The first line of each example creates an instance of a debtor data class. The next five lines are filling the data class object with data. All this is done without the program communicating with the server. Finally, the debtor data class object is sent to the e-conomic server by invoking the appropriate **CreateFromData** method. **CreateFromData** methods exist for all entity types that have a **Create** method. To create entities based on more than one data class object, use the **CreateFromDataArray** methods which takes an array of data class objects.

Fetching entities

Data classes can be used when fetching data. To fetch a single data class, use the **GetData** method. The following examples show how to fetch a current invoice using data classes.

.NET API:

```
ICurrentInvoice currentInvoice = (...);
ICurrentInvoiceData currentInvoiceData =
                                  session.CurrentInvoiceData.GetData(currentInvoice);
Console.WriteLine(currentInvoiceData.Date);
Console.WriteLine(currentInvoiceData.DueDate);
Console.WriteLine(currentInvoiceData.Heading);
Console.WriteLine(currentInvoiceData.DebtorName);
```

SOAP (Axis):

```
CurrentInvoiceHandle currentInvoice = (...);
CurrentInvoiceData currentInvoiceData =
                                  session.currentInvoice_GetData(currentInvoice);
Console.WriteLine(currentInvoiceData.getDate());
Console.WriteLine(currentInvoiceData.getDueDate());
Console.WriteLine(currentInvoiceData.getHeading());
Console.WriteLine(currentInvoiceData.getDebtorName());
```

The call to the **GetData** method (in the second line) fetches the current invoice data class object. When the data class object has been fetched, all its data can be accessed without communicating with the server.

To fetch more than one data class, use the **GetDataArray** methods. The following examples show how to fetch all current invoices.

NET API:

```
ICurrentInvoice[] currentInvoices = session.CurrentInvoice.GetAll();
ICurrentInvoiceData[] currentInvoicesData =
                              session.CurrentInvoiceData.GetDataArray(currentInvoices);
```

SOAP (Axis):

```
CurrentInvoiceHandle[] currentInvoices = session.currentInvoice_GetAll();
CurrentInvoiceData[] currentInvoicesData =
                                  session.currentInvoice_GetDataArray(currentInvoices);
```

Updating entities

Data classes can also be used when updating data. This is done by using the **UpdateFromData** methods. As with the **CreateFromData** methods each data class object must adhere to the constraints of the entity type. The following examples show how to update a current invoice line using data classes.

.NET API:

```
ICurrentInvoiceLine line = (...);
ICurrentInvoiceLineData lineData = session.CurrentInvoiceLineData.GetData(line);
lineData.Product = (...);
lineData.Quantity = (...);
lineData.UnitNetPrice = (...);
session.CurrentInvoiceLine.UpdataFromData(lineData);
```

SOAP (Axis):

```
CurrentInvoiceLineHandle line = (...);
CurrentInvoiceLineData lineData = session.currentInvoiceLine_GetData(line);
lineData.setProduct(...);
lineData.setQuantity(...);
lineData.setUnitNetPrice(...);
session.currentInvoiceLine_UpdateFromData(lineData);
```

The call to the **`UpdateFromData`** method (in the last line) updates all the changed properties in a single round-trip.

To update more than one data class, use the **`UpdateFromDataArray`** methods.