Sem.Sync.

Synchronization Library

[Sem.Sync 3](#_Toc235801396)

[What’s the goal? 3](#_Toc235801397)

[What’s in the package? 3](#_Toc235801398)

[Architecture thoughts 4](#_Toc235801399)

[The engine 5](#_Toc235801400)

[Interacting with the user 5](#_Toc235801401)

[The connectors 5](#_Toc235801402)

[The File System connectors 5](#_Toc235801403)

[The Online Storage connector 7](#_Toc235801404)

[The Xing connector 7](#_Toc235801405)

[The Active Directory Connector 8](#_Toc235801406)

[The Facebook connector 8](#_Toc235801407)

[The Wer-Kennt-Wen connector 8](#_Toc235801408)

[Authoring connectors 8](#_Toc235801409)

[Authoring commands 9](#_Toc235801410)

[Auto-Update-Check 9](#_Toc235801411)

[Working with contacts 9](#_Toc235801412)

[Importing data 9](#_Toc235801413)

[Sem.Sync.LocalSyncManager 10](#_Toc235801414)

[Configuration 11](#_Toc235801415)

[Working folder 11](#_Toc235801416)

[Sample scripts 11](#_Toc235801417)

[Sync with Xing 11](#_Toc235801418)

[Sync with WCF (Simple) 11](#_Toc235801419)

[Sync TESTS 12](#_Toc235801420)

[Sync Outlook with Xing 12](#_Toc235801421)

[Planned things 12](#_Toc235801422)

[FAQ 12](#_Toc235801423)

# Sem.Sync

The reason for me to start and continue this project is not to build up a super user friendly end user product, but more to have some playground to experiment with. You will also find some things I’m not sure of if they are good practice: E.g. this project does make massive use of the “var” keyword, just to see, if it will do harm to the readability of the code. Also you will find a lot of reflection code which will make the library not so good for high volume processing.

## What’s the goal?

Sem.Sync is a project for synchronizing entities between different object stores. Currently this is a ***project***, not a product. If you trust in my capabilities to write nice programs and want to try synchronizing without installing Visual Studio, then download the file “**Sync Outlook with Xing**” – this is a compiled version with a setup and a very simple user interface.

The current implementation of the library is done for synchronizing contact entities and comes with “Connectors” to

1. the file system
   * one big
   * many small xml files
   * vCards
   * CSV files
2. Microsoft Outlook 2007
3. Microsoft Outlook 2003 (also compatible with Outlook 2007)
4. Xing contacts using web scraping technology (read only)
5. Active Directory via LDAP (read only)
6. Facebook (read only and only with a personal Facebook application key)
7. Wer-Kennt-Wen.de (read only)
8. A simple WCF online storage
9. A simple write only statistic module (the XML generated by the module can be read by Microsoft Excel)

***Please read this document carefully,   
because this project may alter your native data under unexpected conditions!***

## What’s in the package?

The project consists of a base library which contains the entity, helper classes and the execution engine. The engine will execute commands which contain parameters and up to three “connectors” (source, target and baseline). You can think of a command as processing data streaming from one connector (source) to another (target) – actually the data is copied without any “streaming”, but for the expected amount of data this is “good enough” … may be I’ll change this behavior later. Some commands also involve a third data “stream” (baseline), like the merge command which can detect changes of a source and a target stream by comparing both to the baseline.

The internal data representation is a proprietary class. The class might change in the future (because the current implementation does not follow xNL/xAL), but currently there are more attractive goals with this project.

## Architecture thoughts

First: the libraries are NOT intended to be used on servers or high volume environments – although I will do everything that they could. Why should you not use them in high volume environments?

Well, first of all: all the data is loaded into memory at once. To fix that I will have to change some basic things in the way the connectors do work.

Second: I do make use of reflection to solve some of the tasks of this library – and this might be not fast enough in high volume environments. If you have implemented some improvements, mail me to include them in the next release ;-).

The assembly Sem.Sync.SyncBase does not contain any UI interaction – this is delegated to other assemblies. This way one aspect of not using this library on a server is eliminated.

The factory is in the project to be able to generate objects from class names read from xml. The factory is a very simple one that is not tuned in any way (but that makes it easy to understand).

Currently I use project references between the UI projects and the connectors. There is no technical reason for that, but I want Visual Studio to copy the build artifacts of the connectors to the output paths of the UI projects, and a reference is a simple way to do that – this will be removed for the first release version.

The project is split in many assemblies just for keeping the code responsible for one thing away from code that’s responsible for another thing. We have:

Sem.Sync.SyncBase This is the library that contains the basis of the engine together with all the utilities. This assembly does not include anything that might interact directly with the user interface.

Sem.Sync.SharedUI.WinForms This is the user interface for the basis functionality. This includes log on dialogs as well as dialogs for merging and a generic disclaimer.

Sem.Sync.*something*Connector These assemblies do implement storage dependent logic. E.g. here you can find the code that interacts with outlook, Active Directory, Xing, Facebook or other storage. The connectors do implement a specific interface to plug into the project. There’s no need to implement bi-directional communication – e.g. the Xing connector can only read while the CSV connector can only write. There’s also a connector that writes some statistics (aggregated data) to an XML file.

ContactViewer This is a simple Silverlight application to display contacts provided by the WCF service that is also part of this Solution. This may be the reason for a “**Project type not supported**” message while opening the solution. You can simply remove the project from the solution if you don’t want to deal with Silverlight.

Sem.Sync.OutlookWithXing.Setup This is the setup project for the sample application that synchronizes Xing contacts to Microsoft Outlook. This may be the reason for a “**Project type not supported**” message while opening the solution. You can simply remove the project from the solution if you don’t want to install the WiX Toolkit – if you want to have the project working, get the version 3.5 from the WiX homepage: !!!!!!!!

Sem.Sync.Documentation Another candidate for a “**Project type not supported**” message is this project. It’s a DocProject project to perform sandcastle operations from a Visual Studio solution as part of the release build process.

Sem.Sync.OnlineStorage This is a web project to host the WCF sample service. Just remove it from the solution, if you don’t want to deal with WCF. Without this project, you should remove the Silverlight project, too.

Sem.Sync.OutlookWithXing This is a sample application that I do frequently use for synchronizing the contacts from my Xing account into my Microsoft Outlook address book – this was the main reason to develop this library.

## The engine

The sync engine does provide the ability to execute instances of the class SyncDescription or SyncCollection which does inherit from BindingList<SyncDescription> to support two-way data binding.

### Interacting with the user

The engine does provide a property called UiProvider of type IUiInteraction. You can set this property to an instance of an object implementing this interface to “catch” the UI requests from the base library and process them using “some” UI technology.

You can however process the request without any user interaction if your process does already have all information requested by the base library and implements the IUiInteraction interface.

## The connectors

Connectors do read/write from/to data sources like file system, online storage and processes like Microsoft Outlook. If a connector is useful for you depends strongly on your expectations: some connectors do only read from a source, because that source is does not accept writing (like most social networking sites) – some sources do provide only very few information. The sources “Facebook” and “Wer-Kennt-Wen” do not support reading much data, but you might be able to extract useful pictures from there.

### The File System connectors

The file system connectors ContactClient, ContactClientIndividualFiles and GenericClient<T> and do serialize the internal objects or a list of these objects to the file system.

#### GenericClient

The GenericClient is intended to be able to work with any type inheriting from StdElement. In the configuration of a command containing this connector you need to specify the type to handle:

<SyncDescription Name="copy vCard to file system">

<Command>CopyAll</Command>

<CommandParameter></CommandParameter>

<SourceConnector>Sem.Sync.FilesystemConnector.ContactClientVCards</SourceConnector>

<SourceStorePath>{FS:WorkingFolder}\vCards</SourceStorePath>

<TargetConnector>Sem.Sync.FilesystemConnector.GenericClient **of StdElement**</TargetConnector>

<TargetStorePath>{FS:WorkingFolder}\vCards.xmlcontact</TargetStorePath>

</SyncDescription>

#### ContactClientVCards

The vCard implementation of the file system connector does currently not support all properties of the internal Contact class. The vCard implementation does have a configuration value in the Config file to save the pictures externally:

<appSettings>

<add key="FileSystem-Contact-Connector-vCard-Save-Pictures-External"

value="true"/>

</appSettings>

In addition to this configuration value a file name needs to be present in the StdContact.PictureName property. This external picture is “write only” and not processed while reading if there is an internal representation of a photo inside the vCard.

#### GenericClientCsv

The CSV implementation is able to write into a format that can be opened in Excel. Unlike the name suggests, this connector does not use a comma as a separator, but a tab because Excel does not register for the file type “TSV” (tab separated values) by default, but has problems opening “real” CSV files.

To write all data the configuration of this connector is straight forward:

<SyncDescription Name="Export contacts from FS-XML to FS-CSV">

<Command>CopyAll</Command>

<SourceConnector>Sem.Sync.FilesystemConnector.ContactClient</SourceConnector>

<SourceStorePath>{FS:WorkingFolder}\Outlook.xmlcontact</SourceStorePath>

<TargetConnector>Sem.Sync.FilesystemConnector.GenericClientCsv

of StdContact</TargetConnector>

<TargetStorePath>{FS:WorkingFolder}\test.csv</TargetStorePath>

</SyncDescription>

As shown above the connector is a generic class that needs a description for what type it should be created. In this case the configuration will create a connector for StdContact elements.

To read the CSV, some more information (mapping of columns to properties) is needed. To specify the mapping file, you need to add one more path to the …StorePath property in the configuration:

<SourceStorePath>

{FS:WorkingFolder}\test.csv

{FS:WorkingFolder}\test.csv.config

</SourceStorePath>

In this case the source file does specify a config-file in the second line of the property with the mapping. Such a mapping file does have the following structure:

<ArrayOfColumnDefinition

xmlns:xsi=<http://www.w3.org/2001/XMLSchema-instance> xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<ColumnDefinition Title="GetFullName()" Selector="GetFullName()" />

<ColumnDefinition Title="PersonGender" Selector="PersonGender" />

<ColumnDefinition Title="DateOfBirth" Selector="DateOfBirth" />

<ColumnDefinition Title="Name.AcademicTitle" Selector="Name.AcademicTitle" />

<ColumnDefinition Title="Name.FirstName" Selector="Name.FirstName" />

<ColumnDefinition Title="Name.LastName" Selector="Name.LastName" />

</ArrayOfColumnDefinition>

You can generate such a file with all supported entries by adding .{write} to the 2nd path parameter:

<SourceStorePath>

{FS:WorkingFolder}\test.csv

{FS:WorkingFolder}\test.csv.config.{write}

</SourceStorePath>

For each column you want to export / import you can specify the Title and the Selector. The Title is the column title inside the file and only used while exporting. Imports do match the column definition to the column by the position. The Selector is the full path of the property. E.g. Name.FirstName does mean “match the FirstName property of the Name property of the contact element to the value of this column”. The properties specified in the selector should have a meaningful overwrite for the ToString() method. Specifying a culture for culture dependent formatting of the values is not supported yet, so you need to format the values for reading in the current culture format.

Parenthesis at the end of the selector like in Selector="GetFullName()" match to a method (in this case the GetFullName() method of the StdContact element). When matching to a method, this method must return a value and must not accept any parameter. Also the methods will only be called while writing the elements to a file – they will be skipped while reading from the CSV.

### The Online Storage connector

This connector communicates with a WCF service that is also part of the package. You will have to modify the service and the connector to match your needs (security, physical storage etc.).

### The Xing connector

Xing is an online business community to handle contacts.

You can configure the login credentials for this portal inside the app.config. If there is no password configured, a login dialog will appear to ask for login credentials. The user id can be preconfigured inside the app.config, even if you don’t want to store the password locally. The user id does always have the focus, even if there is one configured.

<appSettings>

<!-- you might add your credentials here to not being asked -->

<add key="Xing-Contact-Connector-LoginUserId" value=""/>

<add key="Xing-Contact-Connector-LoginPassword" value=""/>

</appSettings>

The connector logs in, detects the contacts from the own contact list and downloads the vCards from the Xing portal. It then converts the vCards into the internal contact representation. You can combine the Xing connector and the file system connector to export the Xing contacts into the file system.

There is a caching mode for Xing that will download the information and use that downloaded information in subsequent executions. This mode is for debugging purpose only and should not be activated by a normal user or to synchronize real data.

<appSettings>

<!-- using the cache will use already downloaded data -->

<add key="Xing-Contact-Connector-UseCache" value="false" />

<!-- skipping non-cached entries will not download any data

not skipping means that we will download deleted files -->

<add key="Xing-Contact-Connector-SkipNotCached" value="false" />

<!-- using the ie cookies you may be able to skip the login

screen, because you are already authenticated by the cookie -->

<add key="Xing-Contact-Connector-UseIeCookies" value="false" />

</appSettings>

### The Active Directory Connector

The AD connector works via LDAP with an active directory. The connector does lookup the credentials from the registry (key current user – missing entries will be created). If there’s nothing inside the registry, it will by default use the current user.

If you want to authenticate to another domain, you need to modify the password value inside the registry to {ask}. If the user id contains a backslash, the first part of the user id will be treated as the LDAP server to query. If you don’t want to see that in the UI, you can specify this server (full qualified DNS server name) in the registry, too.

The command parameter SourceStorePath is handled as a filter for the directory query. An example of a query filter is:

<SourceStorePath>(memberOf=CN=MyGroupName,OU=Security-Gruppen, DC=company,DC=de)</SourceStorePath>

You might include more advanced filters – the string is not processed, so you have full power of LDAP queries here.

Inside the app.config you can specify a logging path for the LDAP properties of the objects:

<add key="Active-Directory-Connector-DumpPath"

value="C:\AD-Path"/>

The connector is currently “Read only” – writing is planned for one of the next releases.

### The Facebook connector

The Facebook API does not provide much information that can be handled in a contact-application. So this connector is more for completion and photo-extraction.

### The Wer-Kennt-Wen connector

The social network site “Wer-Kennt-Wen.de” does not provide a client API, so I do use web-scraping again (like with Xing). The amount of information at this site is very limited, so this connector is more for completion and photo-extraction.

## Authoring connectors

Writing a new connector is very simple: just inherit from Sem.Sync.SyncBase.StdClient and override the two abstract methods. The complexity might come with implementing these two methods, because you will need to provide a conversion from your “native” data to the StdContact class. Depending on the type of data you read/write this might be easy or complex.

The connectors are instantiated by a class factory. In the scripts just use the full qualified class name (include the assembly name if that is different to the namespace) and the engine will do the rest. You can even use generic classes as connectors (see GenericClientCsv).

## Authoring commands

The “commands” you can execute are classes like the connectors. You can write a command by implementing the ISyncCommand interface (which only consists of one property and a single method). Have a look at the AskForContinue command as a sample of a very basic implementation of a synchronization command.

## Auto-Update-Check

The library does contain an update check and will inform you by a log entry if there’s a new version of the library available. This check does download a file from my server (http://svenerikmatzen.info/Content/Portals/0/sem.sync.version.xml). If you don’t want the app to “phone home”, you can simply add a Config value to the app.config:

<appSettings>

<add key="Sem.Sync.SyncBase-VersionCheck" value="false"/>

</appSettings>

The only information sent to my server is the request itself – it’s just a file download.

## Working with contacts

The internal contact class does have an Id to identify the contact. This Id is a Guid and will be generated when creating an instance of the class. The engine is capable to match the contacts using this Id. If you do export contacts from different sources, you can also match the contacts using the name and replace the Id in the target (the target connector will read the contacts, the Ids will be overwritten and the target connector will then save the contacts).

Some connectors cannot write contacts (like Xing, because Xing does not allow altering the contact information) – other connectors **may change the external data even when only reading**: the Outlook connector writes back the Id into Outlook as a *user defined property* to have always the same Id for an exported contact.

## Importing data

I’m currently using the following sequence to import and synchronize contacts from Xing to Microsoft Outlook:

1. Delete previous export/work files
   * Otherwise the contacts exported last time will stay in place while adding the current contacts from Xing
2. Export from Outlook to file system
   * This will create a file system representation of the Outlook contacts. Using the file system representation is much better to handle for the connectors.
3. Export from Xing to file system
   * This will download the contacts from Xing. You might configure the credentials in the app.config file to bypass the login screen. You also might use the IE cookies if you did save the login while using the IE. The download will be done using the normal Html UI of Xing – this may be a problem when Xing does change the GUI.
4. Normalize Xing (file system)
   * Some people do use other spellings of the company name than others (while working for the same company). Normalizing does allow you to specify replacements for strings, so that all contacts do use the same “wording”.
5. Normalize Outlook (file system)
   * Same as the one before – just for the outlook contacts.
6. Match Outlook to Xing by name
   * This will do lookups for the Xing contacts inside the Outlook contacts. We need something like this, because Xing is not capable to write back the Sync-Id.
7. Add missing from Outlook to Xing (file system)
   * For comparison in third party tools it’s much easier if we have no missing contacts.
8. Add missing from Xing to Outlook (file system)
   * For comparison in third party tools it’s much easier if we have no missing contacts.
9. Merge high evidence from Xing to Outlook (file system)
   * Some of the information can be overwritten without user intervention. E.g. if the information is only available on one side or the other side is out of scope (date of birth > 01.01.2020)
10. Merge high evidence from Outlook to Xing (file system)
    * Same as above but other direction.
11. Detect Conflicts
    * This will include the user by presenting a dialog to “merge” the conflicting information. The user should select the correct information (color it green).
12. Merge Xing to Outlook via BeyondCompare
    * This will open a third party tool (BeyondCompare[[1]](#footnote-1)) to show still not matching information.
13. Ask if we want to overwrite Outlook
    * Before writing to Outlook we will ask the user if she is really sure … ;-)
14. Import all from file system to Outlook
    * Now we write all the data from the manipulated Outlook file system representation to the Outlook store.

After this sequence Outlook “should” contain all new and updated information. If that’s not the case you might file a bug and provide enough information to let me reproduce the problem in detail and on my person PC while debugging the program.

# Sem.Sync.LocalSyncManager

This application takes all “.SyncList”-files from its working folder and provides a list of them in a combo box. You can choose one of them and execute the whole script or even a single command while watching the progress in a list of log entries.

## Configuration

### Working folder

The working folder is configured inside the Config-file of this application:

<configuration>

<userSettings>

<LocalSyncManager.Properties.Settings>

<setting name="WorkingFolder" serializeAs="String">

<value />

</setting>

</LocalSyncManager.Properties.Settings>

</userSettings>

</configuration>

If no value is defined in the Config file, the application will use a subfolder named \SemSyncManager\Work in the folder Environment.SpecialFolder.ApplicationData as a default (that’s C:\Users\*[user name]*\AppData\Roaming\SemSyncManager\Work on a Windows 7 machine).

## C:\Users\Sven Erik\Documents\Visual Studio 2008\Projects\Sem.Sync\Misc files\Sem.Sync.LocalSyncManager.Menu.png

You can open the working folder from the file menu of the application (the screen shot is from the localized German version of the application).

## Sample scripts

### Sync with Xing

This script does include commands to sync contacts from Xing to Outlook. This script does use an external tool (BeyondCompare) to solve merge conflicts. You will need to download the tool if you want to use this script. Actions:

1. Cleanup working folder
2. Export Outlook contacts
3. Export Xing contacts
4. Merge automatically missing entries in both files
5. Open external tool to perform manual merge fixing
6. Import into Outlook
7. Remove duplicate calendar entries from Outlook (excluded in sample script)

You might include the last entry of the script. I’ve tested that process many times with my own appointments, but I cannot guarantee that it will not remove non-duplicate entries, so I’ve commented it out. The reason to include this is that Outlook automatically adds calendar entries for contact birthdays – so when importing from many sources, you might blow up your calendar.

### Sync with WCF (Simple)

This is very similar to “Sync with Xing” with the exception that it will also upload the synchronized list to the Online Storage WCF service (in contrast to Xing, that one is capable to update data).

### Sync TESTS

**This is my experimental script and will include changing commands. Review it carefully before executing.**

# Sync Outlook with Xing

This installer includes a very simple user interface to download contacts from Xing and synchronize with Microsoft Outlook.

When starting the executable, you will be presented a disclaimer that will tell you that I will not be responsible for any damage this software will do to your PC and that you understand that you are responsible for performing a backup of your data, not me. I don’t want to get angry emails of people who download software for free that I have written in hours of work and that lost a space in a street name (or even the whole address book), so backup your data before executing software that is designed to alter your data.

If you tell the software that you do understand the terms, you will be presented a very spartanic user interface with exactly one button. After downloading the Xing-contacts (you will need to enter your credentials) and exporting the Microsoft Outlook contacts, the program will use its own UI to let you merge conflicts and then ask if you want to import the result. Solving conflicts does not include adding new contacts (because a new contact is not a conflict – it can just be added).

## Planned things

The items listed here are planned, but not promised. Also I don’t have a fixed order in which I will do the planned items.

1. Authentication and authorization for the web service … I hope to be able to host the web service on my site – some day.
2. Better code … there are some things in the current code that have been implemented to quickly go forward – I will clean up that code and also plan to document all code.
3. More connectors … I will write connectors for Web.de and other social network sites. You have the code, you might be able to implement some by yourself – please send me the code if you write some implementations so that I can use that code in this project and publish it. If you send code to me and I publish it, I will add comments that will clearly state that you did submit the code. But if you send code to me that I already have (implemented by myself or sent by another person) you will not be mentioned. If you don’t like this rule: don’t send me code.

## FAQ

Question: Why does “sync” update so many contacts in Outlook every time?

Answer 1: I currently do not have a good picture comparison, so if there is a picture in the source, it will always override the one in the target.

Answer 2: Outlook does not accept the picture binary, but converts the specified picture before saving it to its internal storage, so extracted pictures from Outlook are different to the source images that have been imported.

Question: If you want me/us to use your library, you need to support…

Answer: I’m sorry, but I don’t want anyone to use this project – I work on it to get my personal synchronization between Outlook and Xing … and to have fun. If you need to support feature X, feel free to implement it ;-)

Question: Why do I have to deal with so many assemblies?

Answer: See “Architecture thoughts”

Question: Why is building a release-build so extraordinary slow?

Answer: “Release” build does include compiling the help file and because of the way sandcastle works it’s a very slow process – exclude the project Sem.Sync.Documentation from the solution and building will run a lot faster.

1. You can get this commercial tool from <http://www.scootersoftware.com/>. I’m not related in any way to this company and I will also not get any money from them. [↑](#footnote-ref-1)