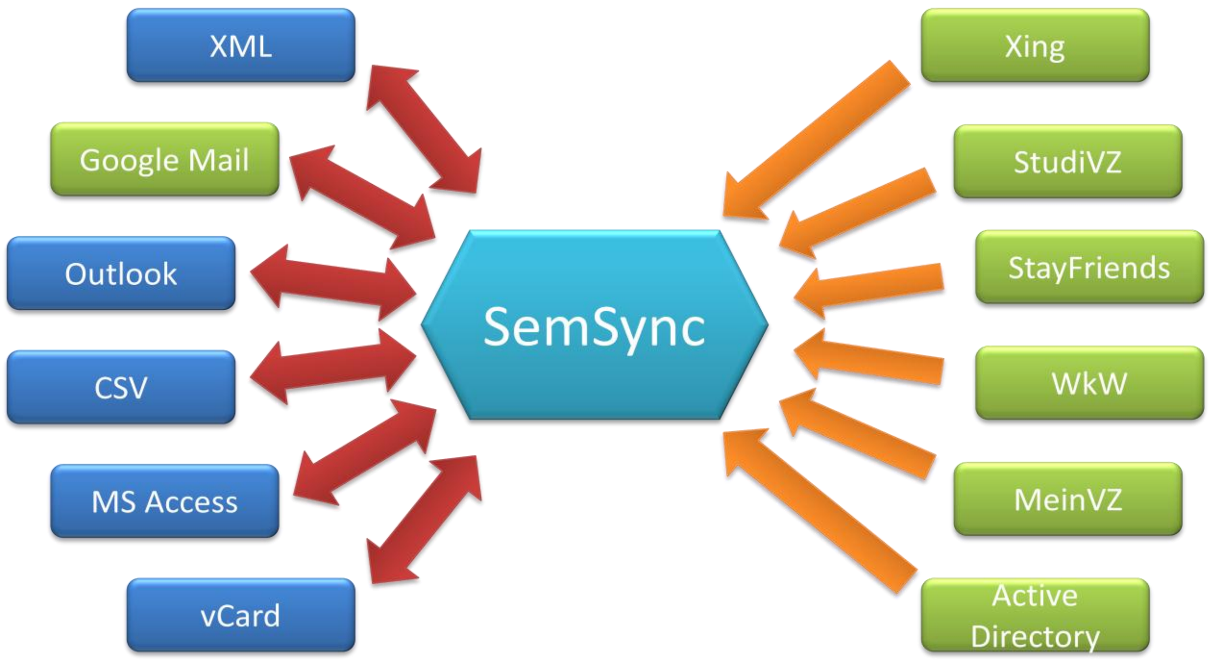
Sem.Sync.

Synchronization Library



Preface

The Sem.Sync Library has been used to implement some “sample” applications. These Applications might be more attractive to you than the base library, because the base library is for programmers (a minor group in the internet) while the applications are useful for users (a major group of the internet users). So the first few chapters on this document are for users, while the later chapters are for programmers.

Please feel free to use the applications and the library as described in the license. You also might mail me to tell about your ideas what else to implement – also hints on how to do things better are welcome.

ATTENTION:
Please be aware that social networks or owners of other information provider do have contracts on how to use the data provided by them. This may include a prohibition on automatic data collection from those sources. I do explicitly NOT research for such policies of the information provider. You are responsible to check the information providers policies whether it is legal to use the functionality of Sem.Sync or not.

Where to start

It depends … as always:

Users

If you have Microsoft Outlook and you have an account for Xing, you might want to try the application “Sync Outlook with Xing” first – then have a look at the application “Sem.Sync.LocalSyncManager”.

If you want to synchronize other sources and targets, “Sem.Sync.LocalSyncManager” is the first choice – later you might have a look at the console client to automate your process.

Developers

If you are a developer and want to implement access to other sources, first have a look at the functionality of “Sem.Sync.LocalSyncManager”, then have a look for the sync engine class `Sem.Sync.SyncBase.SyncEngine`, then have a look at the file system connector class `Sem.Sync.Connector.Filesystem.ContactClientIndividualFiles` (which is really simple to understand). For dealing with web sites as data sources, you might use `Sem.Sync.Connector.Facebook.WebScrapingClient` as a starting point – it does inherit from `WebScrapingBaseClient` which does implement all of the generic web scraping stuff; you will only need some Regular Expression know-how to build a web scraping client.

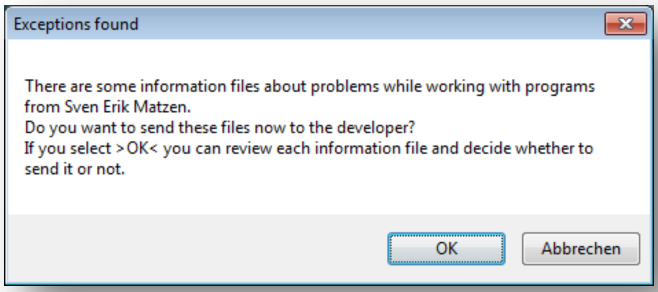
If you have problems

Email me! Yes, you are explicitly encouraged to email your problems to me. I might answer with just a hint where to find the solution, or I might answer with a detailed explanation – I also might simply correct the problem and tell you to download the new version, but each time I will be able to make your life easier with the library.

Exception Reporting

Reporting problems with the software

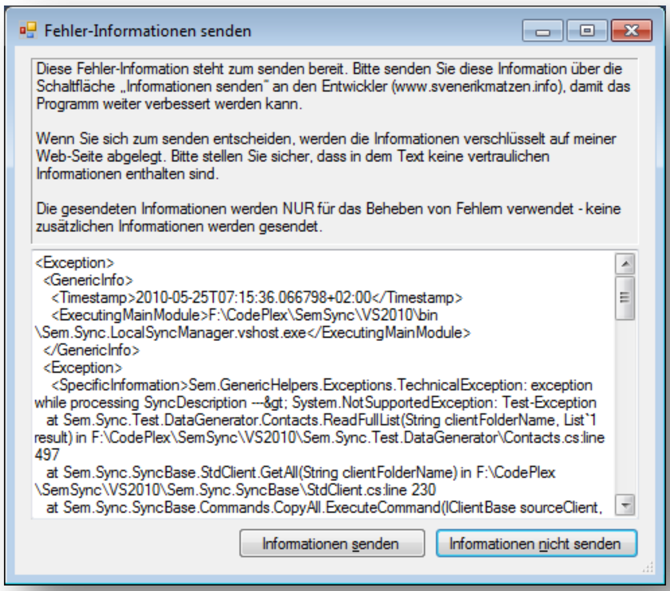
The application described in this document support sending exception information to the developer web site. Each time an exception occurs a file will be written to the exception logging folder. When you start the application, this folder will be scanned for pending exception logs and the user will be asked for sending the information.



There are some information files about problems while working with programs from Sven Erik Matzen.
Do you want to send these files now to the developer?
If you select >OK< you can review each information file and decide whether to send it or not.

Picture 1 - "Send information" question

If the user (you) decides to send the information, the encrypted file will be uploaded via http and will be stored on the developer server (http://www.svenerikmatzen.info). The file content will be encrypted using a random AES key that is protected by public key encryption (RSA). Passwords will **never** be sent in exception files as long as you use the original connectors.



Diese Fehler-Information steht zum senden bereit. Bitte senden Sie diese Information über die Schaltfläche „Informationen senden" an den Entwickler (www.svenerikmatzen.info), damit das Programm weiter verbessert werden kann.

Wenn Sie sich zum senden entscheiden, werden die Informationen verschlüsselt auf meiner Web-Seite abgelegt. Bitte stellen Sie sicher, dass in dem Text keine vertraulichen Informationen enthalten sind.

Die gesendeten Informationen werden NUR für das Beheben von Fehlern verwendet - keine zusätzlichen Informationen werden gesendet.

```
<Exception>
  <GenericInfo>
    <Timestamp>2010-05-25T07:15:36.066798+02:00</Timestamp>
    <ExecutingMainModule>F:\CodePlex\SemSync\VS2010\bin
\Sem.Sync.LocalSyncManager.vshost.exe</ExecutingMainModule>
  </GenericInfo>
  <Exception>
    <SpecificInformation>Sem.GenericHelpers.Exceptions.TechnicalException: exception
while processing SyncDescription ---&gt; System.NotSupportedException: Test-Exception
   at Sem.Sync.Test.DataGenerator.Contacts.ReadFullList(String clientFolderName, List`1
result) in F:\CodePlex\SemSync\VS2010\Sem.Sync.Test.DataGenerator\Contacts.cs:line
497
   at Sem.Sync.SyncBase.StdClient.GetAll(String clientFolderName) in F:\CodePlex
\SemSync\VS2010\Sem.Sync.SyncBase\StdClient.cs:line 230
   at Sem.Sync.SyncBase.Commands.CopyAll.ExecuteCommand(IClientBase sourceClient,
```

Picture 2 - Information detail review

As you can see in Picture 2 - Information detail review the full information is presented for a user review. The dialog defaults for “Don’t send information” to prevent the user from accidentally send information – by simply pressing “Enter”, the information will **not** be sent.

The process to send exception information can be switched off in the configuration xml:

```
<appSettings>
  <add key="SendExceptionDetails" value="true" />
</appSettings>
```

If the configuration does contain the value `"false"`, the user will not be asked for uploading the information will and no information will be uploaded.

Only the XML shown in the upload screen will be uploaded and the information will only be used for fixing the problem. The uploading user cannot be identified, so she/he cannot be informed about the progress of fixing the bug. Currently the server that will process the information is hosted in USA – this may change in the future.

You also can host the Exception Service in-house. Inside the app.config file the endpoint for the WCF client can be changed:

```
<endpoint
   address="http://www.svenerikmatzen.info/Beta/ExceptionService.svc"
   binding="basicHttpBinding" bindingConfiguration="BasicExceptionService"
   contract="ExceptionService.IExceptionService"
   name="BasicExceptionService" />
```

As shown here, you can simply change the URL to your own exception service instance.

After this upload process the information is deleted from the hard disk. If you decide not to upload the information – it will be deleted, too. The information resides on the hard disk unless you start one of the programs that do support the upload.

The Exception Service

The project Sem.GenericHelpers.ExceptionService does include a small WCF service that can accept uploads of exception information from the library Sem.GenericHelpers. As you might see in code, it will accept one message per second – so even if you wait a minute between two submissions, the message may be rejected, because someone else just uploaded one. The reason to reject messages is simply to not fill up all my hosting space just because someone thinks it’s funny to upload the same content by a script multiple times. Also content that is more than 10 Kilo bytes will be rejected (I don’t do that restriction in the WCF configuration, because I want to be able to write a second method that accepts bigger content).

The service interface does implement two methods:

- `GetEncryptionKey` does provide access to the public key portion of the RSA key pair used for encryption of the message content.
- `WriteExceptionData` does implement the upload and storage of the message.

The message will not be decrypted on the server, so you need a tool to do that after retrieving the content from the server. A command line tool for this purpose is the project Sem.GenericHelpers.Decrypter. Because of using the `SimpleCrypto` class the tool is very simple (there are only 9 lined of code needed for this). The most complex task for the encryption/decryption is the generation of the RSA-key. The program does assume the key in a file named `PrivateKey.xml` inside the folder where the program is executed – together with the exception files that need to follow the file naming "`????-??-??-??-??-??-*.xml`".

Executing this program will load the key and scan the current directory for matching files to be decrypted. In case of already decrypted files the crypto class will throw a `FormatException` (which is suppressed) without altering the file.

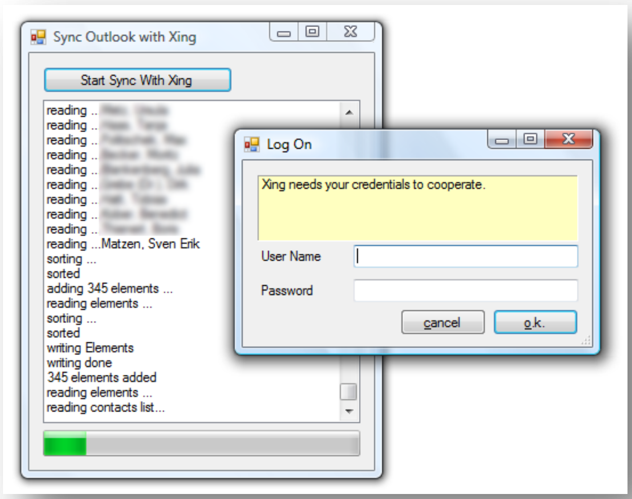
Sync Outlook with Xing

This is a “simple” project using the synchronization library to sync contacts from the social network Xing to the contact store of Microsoft Outlook. I originally did write the library to build this little tool. The contact information will be downloaded using the contact list of Xing, not the profile pages of the users. This has the advantage that your download will not show up in the “visits list” of your contacts, but has the disadvantage that you cannot import information like the contacts of your contacts.

I don’t want to get angry emails of people who download software for free that I have written in hours of work and who 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.

Dialogs in the User Interface

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



```
reading ..[redacted]
reading ..[redacted]
reading ..[redacted]
reading ..[redacted]
reading ..[redacted]
reading ..[redacted]
reading ..[redacted]
reading ..[redacted]
reading ..[redacted]
reading ...Matzen, Sven Erik
sorting ...
sorted
adding 345 elements ...
reading elements ...
sorting ...
sorted
writing Elements
writing done
345 elements added
reading elements ...
reading contacts list...
```

Xing needs your credentials to cooperate.

Picture 3 - "Sync Outlook with Xing" in action

You will be presented a very spartanic user interface with exactly one button. Downloading the contacts, you may need to provide proxy credentials and you will need to enter your Xing credentials.

After downloading the Xing contacts and exporting the Microsoft Outlook contacts, the program will use its own UI to let you first match the contacts (see *How I* do work with it

Address Synchronization

I’m synchronizing contacts from the various networks to my Outlook and the corporate Exchange server by first exporting them into the file system using the connector “[StdContact] Filesystem one Xml per contact” with the template “Template Syncronize”. While this export I’m matching the contacts to the already exported data, so that I will have no duplicates. Then I’m importing from this file system connector to “[StdContact] Microsoft Outlook 2010” and “[StdContact] Microsoft Exchange Web Services via Managed API” to update my address books.

Address normalization

By exporting the contacts into the target "[StdContact] Microsoft Excel OpenXml" I have a very handy tool to process a list of addresses (sorting, copying and correcting data). Then I'm simply syncing back the data into Outlook and Exchange as described in "Address Synchronization".

Reading connections between contacts

The template "Template Get Contact Relations" uses a connector interface to read some more information about the contacts "friends". The implementations of my connectors do only read the IDs of the "friends" and perform a lookup in the already existing contacts for matching entries – so you will not get all "friends", but only those you are connected to, too (otherwise we would blow up the contact information with each additional run and end up exporting really all data from the sources).

Analyzing the data

The connector "[StdContact] DGML Graph" allows having a look for the connection between the contacts (after getting the connection by using the template "Template Get Contact Relations"). By copying data to this connector you will create a dgml-file that can directly opened in Visual Studio 2010 ("Professional" and better).

Matching Contacts on page 12) and then merge conflicts (see *Merging* 14). Solving conflicts does not present you a GUI for adding new contacts because a new contact is not a conflict – it will just be added. New contacts are added to the list without any user interaction.

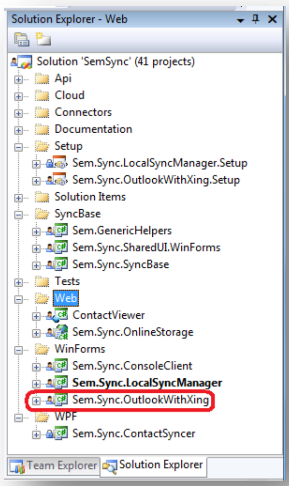
Conflicting pictures will be overwritten in Microsoft Outlook when the binary size of the Xing image is bigger than the binary size of the Microsoft Outlook-exported image – this is usually the case, because Microsoft Outlook does downscale the images and compress them using JPEG image format. So in most cases all contacts with an image will be updated.

The last interaction is the question if you want to import/overwrite the data into Microsoft Outlook. Until this step all data manipulation has been done in memory and your Microsoft Outlook data has only got one user defined field for a GUID that uniquely identifies each contact.

Configuration

The configuration does only include information for the Xing connector. See chapter *Xing* on page 27 for a description of the configuration.

Implementation Details



The project can be found inside the solution folder "WinForms". According to the code metric the project is really simple: 58 Lines of code in the project.

I've extracted all from the project what may be useful in other projects, so that inside this project you will only see the really project specific code.

The code of the form does create and initialize an instance of `SyncEngine` to execute an XML script that's part of the project (commands.xml). Before executing the script the code does attach to two of the events of the engine to render the progress. Then it loads the list of commands, executes it and detaches from the events – that's all.

You might find something interesting like the lambda that handles the `ProgressEvent` of the engine. The code for handling that is so simplistic, that writing an own method for it would be over engineering. I had to put that into a variable to be able to detach it from the delegate after the work has been done. In some other projects of this solution I was able to attach the lambda directly, because there was no need to detach it before the program ends.

There's another funny thing I did use inside this project:

```
this.listLog.Items.Add(
    e.Message + ((StdContact)e.Item).NewIfNull().GetFullName());
```

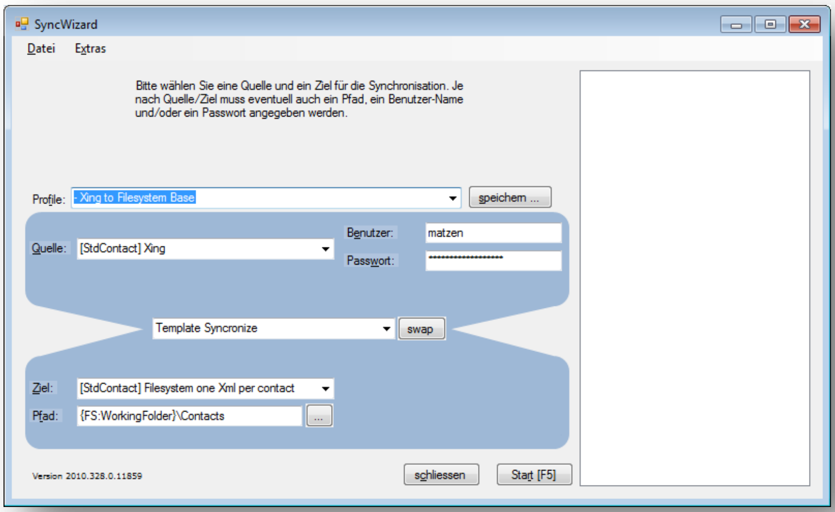
As you can see I do call a function `NewIfNull` of the casted `Item` property. The `Item` property might be `null`, but the generic extension method `NewIfNull` does check the value of the object and creates a new instance of that object it the current instance is `null`. This does reduce the code and prevents me from massive `if` statement usage just to skip null reference exceptions. The code for such a method is really simple, too:

```
public static T NewIfNull<T>(this T testObject) where T : class, new()
{
    return testObject ?? new T();
}
```

Sem.Sync.LocalSyncManager

The synchronization manager does provide access to all connectors that have been implemented in a "productive" state – that is not a "bug free", but an "it's performing some useful action" state. There might be connectors that show up only in debug mode. These connectors are not in a stable state and should only be used for debugging them.

Currently there are two implemented forms inside the application. One for defining some data for execution templates (I don't want to call them "workflows", because there is no relation to the workflow foundation of the .net framework). The following picture does show the UI of this "Wizard-View".



Bitte wählen Sie eine Quelle und ein Ziel für die Synchronisation. Je nach Quelle/Ziel muss eventuell auch ein Pfad, ein Benutzer-Name und/oder ein Passwort angegeben werden.

Profile: - Xing to Filesystem Base

Quelle: [StdContact] Xing

Benutzer: matzen

Passwort: ********************

Template Syncronize

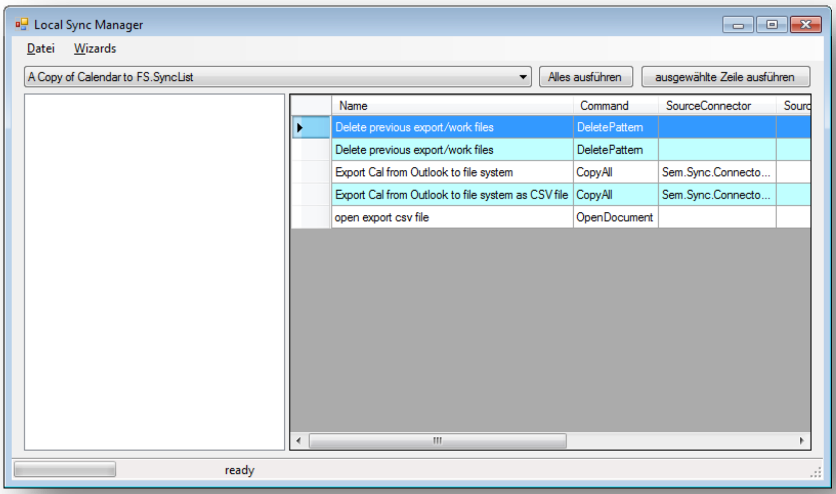
Ziel: [StdContact] Filesystem one Xml per contact

Pfad: {FS:WorkingFolder}\Contacts

Version 2010.328.0.11859

Picture 4 - Wizard-View of the LocalSyncManager

The other provides a list of predefined execution command lists that can be customized in the file system and executed with a log:


Local Sync Manager
Datei
Wizards
A Copy of Calendar to FS.SyncList
Alles ausführen
ausgewählte Zeile ausführen
ready

Name	Command	SourceConnector	Sourc
Delete previous export/work files	DeletePattern		
Delete previous export/work files	DeletePattern		
Export Cal from Outlook to file system	CopyAll	Sem.Sync.Connecto...	
Export Cal from Outlook to file system as CSV file	CopyAll	Sem.Sync.Connecto...	
open export csv file	OpenDocument		

Picture 5 - "Command-View" of the LocalSyncManager

This screen 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.

How I do work with it

Address Synchronization

I'm synchronizing contacts from the various networks to my Outlook and the corporate Exchange server by first exporting them into the file system using the connector "[StdContact] Filesystem one Xml per contact" with the template "Template Syncronize". While this export I'm matching the contacts to the already exported data, so that I will have no duplicates. Then I'm importing from this file system connector to "[StdContact] Microsoft Outlook 2010" and "[StdContact] Microsoft Exchange Web Services via Managed API" to update my address books.

Address normalization

By exporting the contacts into the target "[StdContact] Microsoft Excel OpenXml" I have a very handy tool to process a list of addresses (sorting, copying and correcting data). Then I'm simply syncing back the data into Outlook and Exchange as described in "Address Synchronization".

Reading connections between contacts

The template "Template Get Contact Relations" uses a connector interface to read some more information about the contacts "friends". The implementations of my connectors do only read the IDs of the "friends" and perform a lookup in the already existing contacts for matching entries – so you will not get all "friends", but only those you are connected to, too (otherwise we would blow up the contact information with each additional run and end up exporting really all data from the sources).

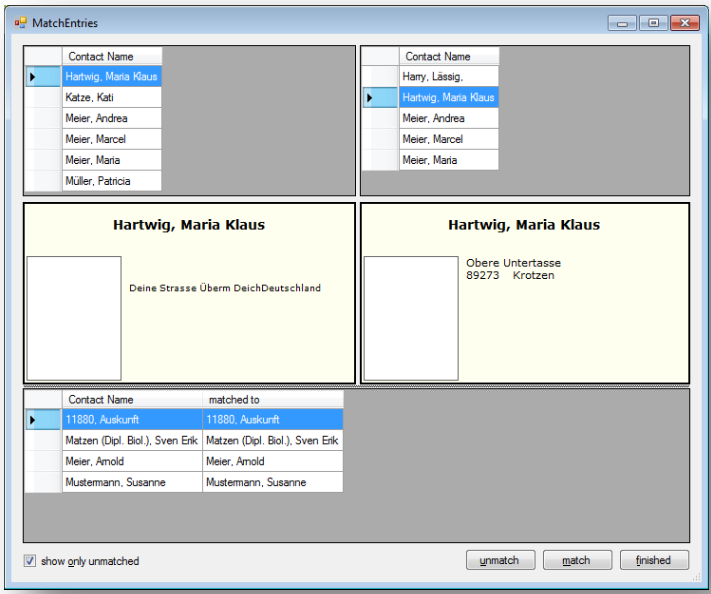
Analyzing the data

The connector “[StdContact] DGML Graph” allows having a look for the connection between the contacts (after getting the connection by using the template “Template Get Contact Relations”). By copying data to this connector you will create a dgml-file that can directly opened in Visual Studio 2010 (“Professional” and better).

Matching Contacts

When working with contacts of different sources it’s important to match the contact data, so that no duplicates are written into the destination system. The dialog for matching contacts of different sources does use an XML file in the working folder to keep track of matched entries. As you can see in the following picture, the contacts of the two sources are shown like a business card. This way you can quickly decide whether to contacts do match, or don’t match. If a picture is available, it’s shown, too. Each time a “match” is performed, a new entry is generated in the lower grid of the dialog (this match can be “unmatched”) and the next possible matching entries are selected automatically.

With pressing the “finished” button you do write the matching list to the local file system and continue.



Contact Name
Hartwig, Maria Klaus
Katze, Kati
Meier, Andrea
Meier, Marcel
Meier, Maria
Müller, Patricia

Contact Name
Harry, Lässig,
Hartwig, Maria Klaus
Meier, Andrea
Meier, Marcel
Meier, Maria

Hartwig, Maria Klaus

Deine Strasse Überm DeichDeutschland

Hartwig, Maria Klaus

Obere Untertasse
89273 Krotzen

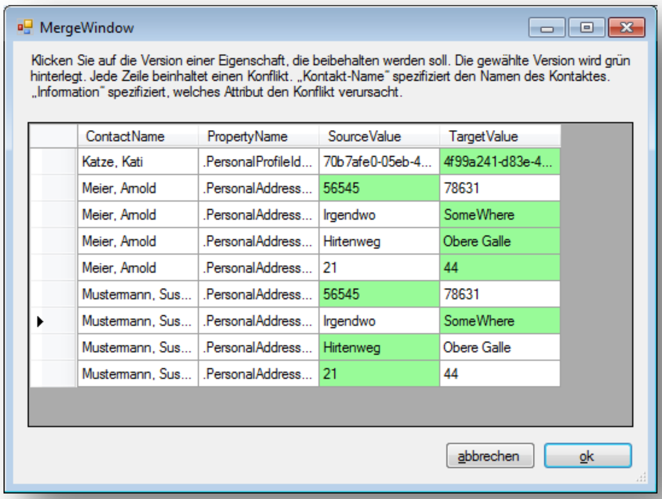
Contact Name	matched to
11880, Auskunft	11880, Auskunft
Matzen (Dipl. Biol.), Sven Erik	Matzen (Dipl. Biol.), Sven Erik
Meier, Arnold	Meier, Arnold
Mustermann, Susanne	Mustermann, Susanne

[✓] show only unmatched

Picture 6 - LocalSyncManager Matching Dialog

Merging Conflicting Data

In the next dialog, the contacts conflicting attribute values are shown, so that you can select which version should be selected for the destination:



Klicken Sie auf die Version einer Eigenschaft, die beibehalten werden soll. Die gewählte Version wird grün hinterlegt. Jede Zeile beinhaltet einen Konflikt. „Kontakt-Name" spezifiziert den Namen des Kontaktes. „Information" spezifiziert, welches Attribut den Konflikt verursacht.

	ContactName	PropertyName	SourceValue	TargetValue
	Katze, Kati	.PersonalProfileId...	70b7afe0-05eb-4...	4f99a241-d83e-4...
	Meier, Arnold	.PersonalAddress...	56545	78631
	Meier, Arnold	.PersonalAddress...	Irgendwo	SomeWhere
	Meier, Arnold	.PersonalAddress...	Hirtenweg	Obere Galle
	Meier, Arnold	.PersonalAddress...	21	44
	Mustermann, Sus...	.PersonalAddress...	56545	78631
▶	Mustermann, Sus...	.PersonalAddress...	Irgendwo	SomeWhere
	Mustermann, Sus...	.PersonalAddress...	Hirtenweg	Obere Galle
	Mustermann, Sus...	.PersonalAddress...	21	44

Picture 7 - Property value selection of LocalSyncManager

In case of non-conflicting properties, the change is applied without explicit permission from the user. Non-conflicting situations are:

- One side does contain information, but the other does not
- One side does contain wrong data (e.g. Birthday before 1900) and the other does contain "more correct" data (e.g. Birthday 1985).

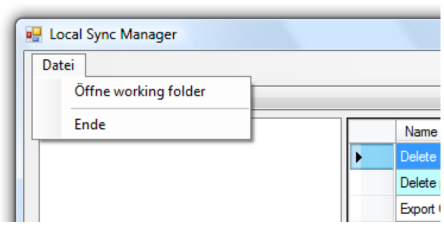
After a last question if the data should now be written to the destination, the data will be persisted.

Configuration

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

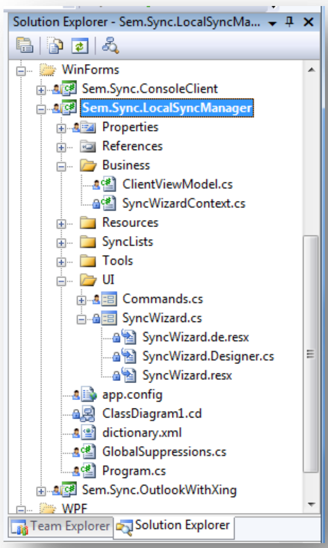
```
<configuration>
    <userSettings>
        <LocalSyncManager.Properties.Settings>
            <setting name="WorkingFolder" serializeAs="String">
                <value />
            </setting>
        </LocalSyncManager.Properties.Settings>
    </userSettings>
</configuration>
```

If no value is defined in the configuration 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).



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).

Implementation Details



This implementation does come with some more complex things:

- Data binding is implementing using a little method called `SetupBind`. This method encapsulates some setup of the `BindingSource` and some event handler. I was trying to get rid of the addition control for data binding which I do not like.
- All events are set up using Lambdas. Events from the “`DataContext`” which might come from another thread do use a `MethodInvoker` to handle cross thread UI updates.
- The class names might let you think of the MVVM pattern, but I did not follow any strict pattern in this project – I only was searching for a name that describes the class.
- The `SyncWizardContext` does implement `INotifyPropertyChanged`, but does not handle all property changes using this interface, because that would lead to many field backed properties … but I want “auto properties” with as less code as possible.
- I do make some use of Linq there to setup the list of sources/targets, enumerate folders and add them to a list using an extension method `ForEach`.
- I heavily make use of the `var` keyword and object initializer. At the moment I think they do make the code more readable.

Sem.Sync Library

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. You also might give the "Synchronization Manager" a try – this does provide access to all currently implemented connectors.

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 (read and write including images)
    - CSV files (configurable mapping for columns)
2. FTP servers
3. Microsoft Outlook 2010
4. Microsoft Outlook 2007
5. Microsoft Outlook 2003 (also compatible with Outlook 2007)
6. Microsoft Exchange via "Managed Web Service API"
7. Microsoft Excel Xml Spreadsheet 2003
8. Microsoft Excel Xml Spreadsheet OpenXml (the preferred one)
9. Xing contacts using web scraping technology (read only)
10. Active Directory via LDAP (read only)
11. Facebook (read only)
12. Wer-Kennt-Wen.de (read only)
13. StudiVZ (read only / the social network for students)
14. MeinVZ (read only / the social network for StudiVZ members that are not students any more)
15. StayFriends (read only / a social network site for finding schoolmates)
16. Google Mail ("Gmail") contacts via Google API
17. A cloud storage (not yet fully functional, but started and in progress)
18. A simple WCF online storage
19. A simple write-only statistic module
    (the XML generated by the module can be read by Microsoft Excel)
20. A generator for DGML-files to be loaded into the Visual Studio DGML-Editor/Viewer
    (available in Visual Studio 2010 Professional and better)
21. An FTP client to support internet storage.

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 ;-).

Can I use the library in unattended environments (windows services, web sites)? Yes, 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). Also it does simplify some things that are more complex in other factories, by "guessing" the assembly name from the namespace if no one has been provided.

All projects do build into a central BIN folder (all build configurations make use of the same build folder). This removes the need of project references from the executable to the connectors.

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.GenericHelpers	This assembly provides a library of functionality developed in the context of this solution, but not tightly related to the business case of synchronizing objects. E.g. the class factory and the http helper classes are provided by this library.
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 – that's write only.
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. This Silverlight implementation does display the contacts in a list of pictures with some additional text information.
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. As a consequence of this being the functionality that is used most often, this is also the functionality that is tested in the best way.
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 (see below: *The Tools*).
Sem.Sync.LocalSyncManager	The synchronization application Sem.Sync.LocalSyncManager provides access to all currently implemented connectors. The GUI is much more complex than Sem.Sync.OutlookWithXing, but allows defining and storing profiles to synchronize from one (readable) connector to another (writable) connector.
Sem.Sync.LocalSyncManager.Setup	This is the setup project for the synchronization application Sem.Sync.LocalSyncManager that provides access to all currently implemented connectors.
Sem.Sync.ConsoleClient	This console application does provide the ability to execute commands from a serialized `SyncCollection` which is a list of

	commands and connectors that do describe a kind of workflow to perform a synchronization operation.
Sem.Sync.ContactSyncer	This WPF project might become active someday to provide a nice and user friendly WPF interface to the features of the synchronization project. Currently it is not in a functional state.
Sem.Sync.Documentation	It's a DocProject project to perform sandcastle operations from a Visual Studio solution as part of the release build process. Another candidate for a "**`Project type not supported`**" message is this project (see below: *The Tools*).
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.Cloud	This project is an Azure Cloud Service providing access to the `IStorage` interface to get or put a list of contacts.
Sem.Sync.Cloud.Storage	The project provides Azure Cloud Service definition and configuration for Sem.Sync.Cloud.
StorageClient	This is a library of abstracting the REST interface of the Azure storage engine implemented by Microsoft. This library has been used for convenience and has not been evaluated for performance, security or any other aspect. Reviewing this library is one point of the list of To-dos.

The Tools

The projects of the solution do imply installing some free tools to integrate new project types into Visual Studio. The solution has been written with "Visual Studio Team Developer" and not tested under any other development environment. If you don't want to install the tools in the list below, you might need to exclude some of the projects from the solution.

WiX Setup	Homepage: http://sourceforge.net/projects/wix The Windows Installer XML (WiX) is a toolset that builds Windows installation packages from XML source code. This project does use the version 3.5 which you can download from http://wix.sourceforge.net/releases/
DocProject	Homepage: http://www.codeplex.com/DocProject DocProject facilitates the administration and development of project documentation with Sandcastle, allowing you to use the integrated tools of Visual Studio to customize Sandcastle's output.
Microsoft Silverlight	Homepage: http://silverlight.net Silverlight is a RIA framework from Microsoft enabling .Net developers to write client side web application components in a well-known language line C# or VB.net. You will need to install the

	component and the Visual Studio extensions to work with the project.
Microsoft Azure	Homepage: http://www.microsoft.com/azure/sdk.mspx Azure is the cloud computing framework of Microsoft and enables developers to write web applications that can be deployed into a Microsoft computing center without any considerations about the physical infrastructure or the OS.
Microsoft Pex	Homepage: http://research.microsoft.com/en-us/projects/pex/ Pex is a tool for generating parameterized unit tests right from the Visual Studio code editor, Pex finds interesting input-output values of your methods, which you can save as a small test suite with high code coverage. Pex performs a systematic analysis, hunting for boundary conditions, exceptions and assertion failures.
Google Data API	Homepage: http://code.google.com/p/google-gdata/ this project provides access to the Google address book through a .Net API.
Exchange Web Service Managed API	What a name! This will be used to implement the connector to Microsoft Exchange via web services. Currently this is in BETA, but I hope this will become stable some time.

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. A `SyncDescription` contains all information that is needed to perform a transformation of data from a source to a target connector with the help of a baseline connector.

The engine is not designed for transforming high volumes of object. I've tested the engine now with more than 300 contacts synchronizing from Xing to Outlook and from Outlook to the file system. All data (including the binary image data) is loaded into the objects before executing the commands – currently streaming is not implemented.

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.

Path Token

File system paths can include tokens to specify the path of a file independent from the installation path and working folder location:

`{FS:WorkingFolder}`	The working folder – usually `C:\Users\`*`[user name]`*`\AppData\Roaming\SemSyncManager\Work` on a Windows 7 machine

`{FS:ApplicationFolder}`	The folder where the executable is running.

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.

Active Directory

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. An application might decide to prepare the connector with credentials (like Sem.Sync.LocalSyncManager does) – in this case you don’t have to fill out any dialog while the connector does access the Active Directory server.

Be aware that you should use the full qualified name of the LDAP server to query – otherwise you may experience delays or you might not be able to connect to the directory.

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=Unit,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. If one of the returning entities is a group of users, the members of this group (recursively) are included in the result – so when having groups that contain other groups of members, the result will contain all users in all sub-groups.

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"/>
```

This path is the destination for downloaded data from the Active Directory and will dump any information accessible from the `SearchResult` objects by the `ResultPropertyCollection Properties` property. Using this path you can look up the information to better filter your query.

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

Microsoft Exchange Web Services via Managed API

This connector does implement accessing (read and write) Microsoft Exchange via the Managed API (EWS MA) assembly provided by Microsoft. It does support the auto discovery feature of EWS MA.

The command parameter `SourceStorePath` is a combination of the target system and the contacts folder inside that system. The target system can be specified with the URL to the service (e.g.:

https://serverName.subDom.myCompany.com/ews/exchange.asmx) or an email address that would be processed using the auto discovery of Exchange WS (e.g. klaus.mustermann@myCompany.com). The contacts folder name is separated from the server by a pipe character “|”. A valid path specification might look like this:

```
https://serverName.subDom.myCompany.com/ews/exchange.asmx|BusinessContacts
```

The contacts folder specification can be omitted, but if the server is specified (by an URL or an email address) the pipe character must be added.

The server specification can be omitted, if it’s configured by the app.config file:

```
<appSettings>
  <add key="ExchangeWSMA-Connector-ServerUrl"
       value="https://srvffm15a.frankfurt.sdx-ag.de/ews/exchange.asmx" />
</appSettings>
```

In this case the contacts folder can still be specified. Many organizations do deploy exchange servers without an “official” certificate. Normally in order to correctly install a certificate from “private” CA, the root certificate of that CA must be imported into the *machine* store of all computers of the organization – this is often not done correctly. In order to access such “non-trusted” SSL servers, we need to skip the trust check for the server certificate (which makes the certificate useless, because without that check a man-in-the-middle is really easy). You can configure the connector using the app.config to bypass the trust check:

```
<appSettings>
  <add key="ExchangeWSMA-Connector-IgnoreCertificateErrors" value="true" />
</appSettings>
```

WARNING: This setting will open a security hole, because it’s bypassing SSL security features. This setting disables the check for the complete process, not only for this connector.

Facebook

The Facebook client has been re-implemented without the Facebook API, but using web-scraping technology. This enables all users to use this client without requesting Facebook for a key. Currently the connector does not download the contact image from Facebook – this will be changed soon.

File System

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 configuration 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.

Property to Column Mapping

To read the CSV or to write a CSV with a specific structure, 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. You can specify the additional configuration file using a line break or the pipe character (“`|`”):

```
<SourceStorePath>
  {FS:WorkingFolder}\test.csv
  {FS:WorkingFolder}\test.csv.config
</SourceStorePath>
```

In this case the source file does specify a configuration-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="Full Name"     Selector="GetFullName()" />
  <ColumnDefinition Title="Sex"           Selector="PersonGender" />
  <ColumnDefinition Title="Date of Birth" Selector="DateOfBirth" />
  <ColumnDefinition Title="AcademicTitle" Selector="Name.AcademicTitle" />
  <ColumnDefinition Title="FirstName"     Selector="Name.FirstName" />
  <ColumnDefinition Title="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.

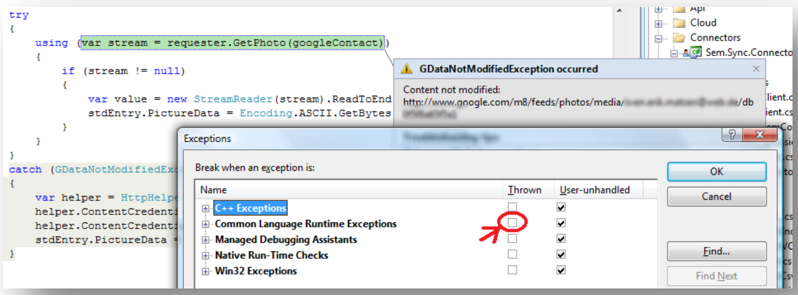
Google

The Google connector connects to the Google Contacts store using the Google Data API via a .Net reference implementation from Google (need to be installed).

You might experience an account lock in case of many updates – in this case the GUI will guide you to the page where you have to solve a captcha in order to reactivate the account. This is normal behavior of the Google Contacts site.

This connector implements read/write access, so you might use it as an external store or backup. To use this connector you need to setup a Google account.

While debugging the code you might get some (or many) `GDataNotModifiedException` that will be caught by a try-catch-block. This is also “normal” behavior of the API, because Google will tell you for each picture that you did already download that image – but currently a distinct Google image cache is not scope of the connector. The connector does handle that exception and download the image directly. Make sure that breaking in handled exceptions is deactivated in Visual Studio to prevent interruptions while debugging the Google connector (un-check the check box shown in Picture 8).



```
try
{
    using (var stream = requester.GetPhoto(googleContact))
    {
        if (stream != null)
        {
            var value = new StreamReader(stream).ReadToEnd
            stdEntry.PictureData = Encoding.ASCII.GetBytes
        }
    }
}
catch (GDataNotModifiedEx
{
    var helper = HttpHelpe
    helper.ContentCredenti
    helper.ContentCredenti
    stdEntry.PictureData =
}
```

GDataNotModifiedException occurred

Content not modified:
http://www.google.com/m8/feeds/photos/media/[redacted]/db [redacted]

Exceptions

Break when an exception is:

Name	Thrown	User-unhandled
C++ Exceptions	☐	✓
Common Language Runtime Exceptions	☐	✓
Managed Debugging Assistants	☐	✓
Native Run-Time Checks	☐	✓
Win32 Exceptions	☐	✓

Picture 8 - deactivating "break in handled exceptions"

Lotus Notes

This connector is not implemented yet, but planned.

MeinVZ / StudiVZ

Both of these social networks are strongly connected to each other, so the connectors do share much functionality and are written in one single assembly. Currently there is limited capability of paging through the contacts, because I do not have enough contacts at these networks in order to test paging. Fortunately a user of this connector did send me come html, so I was able to implement the paging, but I still cannot test/debug it. If you want me to implement more functionality and/or fix bugs in one of these connectors you need to add me as a contact ;-)

Memory

This is an “internal” connector that is not accessible through the GUI. It’s implemented to provide a temporary store for workflow operations like merging and normalizing.

Microsoft Access Database

This connector implements a very simple database mapping from Contact data to one single table inside a database. As a path parameter for the connector you need to specify a configuration file. If that file does not exist, a sample file will be generated that you need to modify in order to sync with a database.

The configuration file for a database connection is also relative simple (as long as there is no value transformation needed). As you can see in the following example the file does contain:

- the full path of the database file
- the name of the table to synchronize
- the database field to contact property mapping

In this example only two fields are mapped:

PersonalProfileIdentifiers.MicrosoftAccessId	Id
Name.FirstName	Vorname

Table 1 - Mapping inside the example XML

The configuration file is an XML file like this:

```
<?xml version="1.0"?>
<SourceDescription     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                       xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <DatabasePath>C:\Users\MyName\Desktop\AccessDatabaseSample.mdb</DatabasePath>
  <MainTable>ContactInformation</MainTable>
  <ColumnDefinitions>
    <ColumnDefinition Title="Id"
                      Selector="PersonalProfileIdentifiers.MicrosoftAccessId"
                      IsPrimaryKey="true"
                      IsAutoValue="true" />
    <ColumnDefinition Title="Vorname"
                      Selector="Name.FirstName" />
  </ColumnDefinitions>
</SourceDescription>
```

You might see that it's a serialized object structure with two strings and a list of mappings (default values for the Boolean values can be omitted). To support more complex mappings I've included a library for serialization/deserialization of expression trees (ExpressionSerialization Copyright © MSIT 2007 – you can reach the projects homepage at http://code.msdn.microsoft.com/exprserialization). If you generate a sample mapping file, such a serialized expression tree is also included, but I think it needs to be reworked a lot in order to enable non-programmers to generate such expressions in xml. The XML serialized expression

```
(ColumnDefinition, value) =>
     ((Gender)value) == Gender.Male
     ? "'m'"
     : ((Gender)value) == Gender.Female ? "'f'" : "NULL",
```

would fill 5 pages of this document.

The syntax to specify property paths using a `Selector` is exactly the same as described in `GenericClientCsv`. The syntax for field names is simply the field name ;-) without braces.

Microsoft Excel Xml

The Excel XML connector does work with the XML version of Excel files – not with the binary format or the OpenXML document format. Since 2003 Excel supports XML as a storage container for worksheets. The parsing and generation of such XML files does not require the installation of Excel or any Microsoft Office components (which is the key to use Microsoft Excel files on a server).

This connector works the same way as the File System CSV connector does (see chapter "Property to Column Mapping" on page 23).

Warning: the connector does manage the XML as one entity in memory – there is no streaming support for the data while reading. This implies that even huge XML files will be loaded completely into memory and parsed to an XML DOM before being processed. This may affect performance of the whole system.

Microsoft Excel OpenXML

This connector does support the new Microsoft Office format for Excel – the xlsx-files. The handling of this connector is identical to the connector "Microsoft Excel Xml".

Microsoft Exchange Web Services

The Microsoft Exchange connector supports Exchange 2007 and Exchange 2010 through the "managed API" of Exchange Web Services.

Microsoft Outlook

There are "some" distinct connector versions for Microsoft Outlook: one for Outlook 2003 (Sem.Sync.Connector.Outlook2003) and one for Outlook 12 (Sem.Sync.Connector.Outlook). There's a hard coded fallback that does use the Outlook 2003 connector if the Outlook 12 connector fails to instantiate (e.g. because Office 12 is not installed). The Outlook connectors do support read/write and there is support for the "Categories" that Outlook does use. Also the Outlook connectors do write the internal contact Id back to the Outlook storage. This way matching is much simpler and you have a stable Id basis. Pictures are also supported (read and write).

Online Storage

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 code implemented is currently only a "prove of concept".

Statistic

The simple report statistic provider does write statistics about the contacts into an XML file. This includes the ratio of male/female contacts and the top 10 cities of private/business addresses. There's nothing to configure but the destination file.

StayFriends

The social network site "StayFriends" is specialized in finding school mates. This connector does read from that site some of the contact data and the pictures. This connector is a web-scraping client, so it will fail some day and then needs to be adapted to the changes in the web sites pages. I'm trying to simplify the web-scraping clients from time to time and consolidate shared methods into a base class. So the code inside the web scraping clients should shrink a lot after some time.

Wer-Kennt-Wen

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

Xing

Xing is an online business community to handle contacts.

You *can* configure the login credentials for this portal inside the app.config, but configuring the password inside the configuration file is explicitly NOT recommended and only implemented for testing purpose.

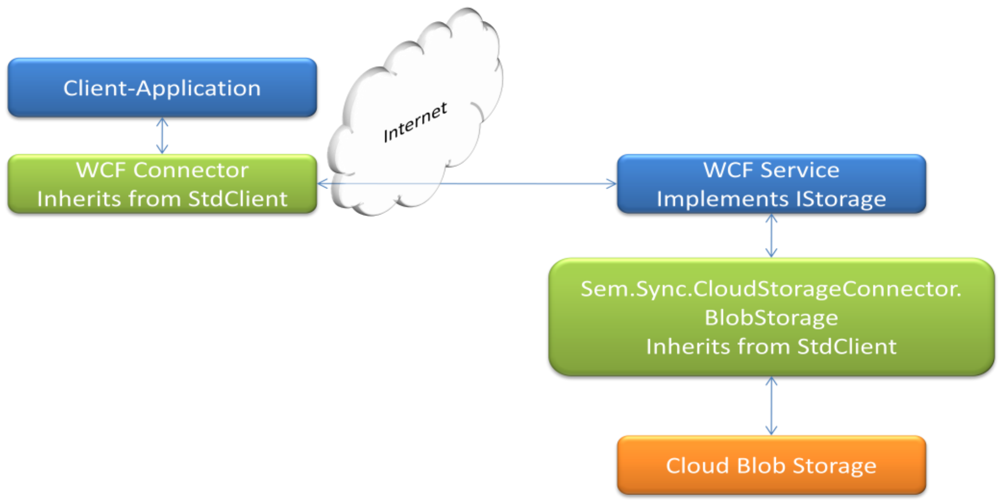
If there is no password configured, a login dialog will appear to ask for login credentials. The user id can be preconfigured inside the app.config, even if you don't want to store the password locally. The user id does always have the focus, even if there is one configured.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>

    <!-- you might add your credentials here to not being asked -->
    <add key="Xing-Contact-Connector-LogonUserId"   value=""/>
    <add key="Xing-Contact-Connector-LogonPassword" value=""/>

    <!-- 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 logon
         screen, because you are already authenticated by the cookie -->
    <add key="Xing-Contact-Connector-UseIeCookies"  value="false" />

  </appSettings>
</configuration>
```

There is a caching mode for all web-scraping based connectors like 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. It does cache the whole request response structures, so if you activate this caching, you will not get any new data until you deactivate it again.

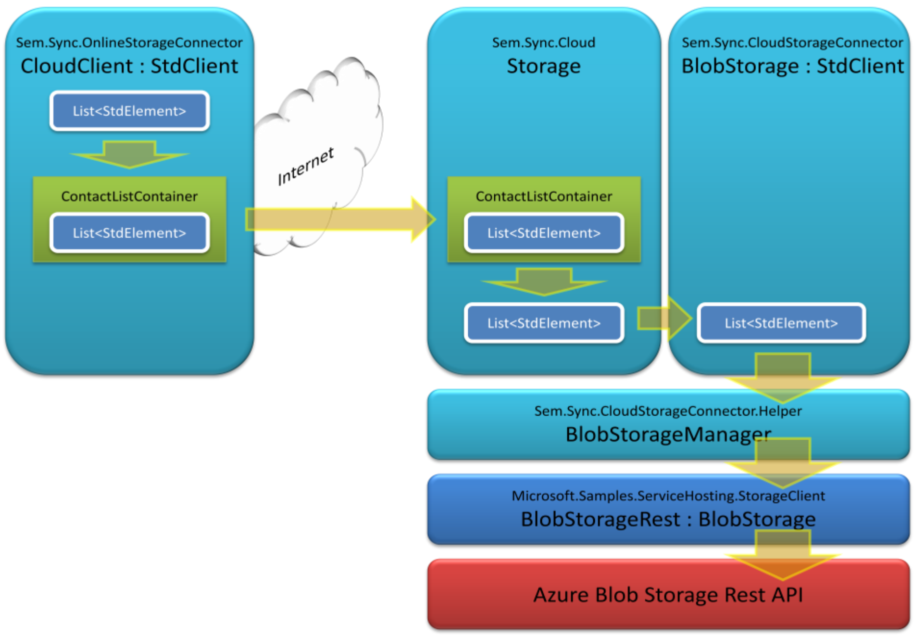
The connector logs in, detects the contacts from the own contact list and downloads the vCards from the Xing portal. It then converts the vCards into the internal contact representation. You can combine the Xing connector and the file system connector to export the Xing contacts into the file system. The profile information is not being downloaded, because such a download would show up as a profile visit in the contact account at Xing. Tags are converted to categories.

Connection to the Cloud


Client-Application
WCF Connector Inherits from StdClient
Internet
WCF Service Implements IStorage
Sem.Sync.CloudStorageConnector. BlobStorage Inherits from StdClient
Cloud Blob Storage

Picture 9 - Azure storage component architecture

The assembly `Sem.Sync.OnlineStorageConnector` does provide a `CloudClient`. This client does provide a connection to the WCF service implemented by the Web Role of the cloud project `Sem.Sync.Cloud`. As you can see in the image above, the WCF is only a proxy that uses the WCF service to provide access to the `Sem.Sync.CloudStorageConnector.BlobStorage` which does inherit from StdClient, again. This BlobStorage does interact with the Azure blob storage. The classes are in detail:



Picture 10 - Azure connector classes – call into the cloud

As you can see the CloudClient is on the (left) client side while the rest of the classes are in the (right) cloud, putting the storage logic into the cloud while keeping the business logic at the consumer. The service consumer may also be hosted inside the cloud.

This implementation does use the Microsoft Azure SDK.

To transport a List<StdElement>, it is being wrapped inside a ContactListContainer which does also contain a List<TechnicalMessage> to provide a way of adding technical information. Write/Delete operations do use a BooleanResultContainer which also contains a List<TechnicalMessage> together with the Boolean result.

The architecture of this access path provides flexibility as the interface at the server side is the same as at the client side and the access to the storage is abstracted by another interface. This abstraction means easier change from *blob storage* to *table storage* as needed. The StorageClient (a sample project from Microsoft to deal with the REST API) does use http to access data at the BlobStorage of Azure. This means the Service is completely isolated from proprietary data access methods.

Currently there's no authentication implemented, but planned to be implemented via Microsoft .NET Access Control Service.

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.

If your "native" system does support access using vCards, this would ease the development substantially, because there is already a vCard connector using the file system. This connector does use the class VCardConverter from the namespace Sem.Sync.SyncBase.Helpers. You should have a look if that converter already matches your needs.

For communication with an XML based system, you might use XSLT to simply transform the XML data from your system to the serialization format of the class StdContact. That way you can easily fill a list of StdContact and return it from the connector to the engine.

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) – this might be helpful when synchronizing different types of entities (like calendar items).

Anatomy of a Simple Connector

```
namespace Sem.Sync.Connector.Console
```

The namespace of the connector class should match Sem.Sync.Connector.*myConnectorName* with the assembly name matching the namespace name in order to ease the configuration. Addressing the connector class inside the configuration is made using the full qualified class name, but you can omit the namespace and the assembly name if they do match this convention.

```
[ClientStoragePathDescriptionAttribute(Irrelevant = true)]
```

This attribute describes the way a UI should help the user to deal with the client storage path parameter while reading/writing data. In our very simple sample this attribute contain the information that this path is irrelevant, so the UI can hide the input elements to not confuse the user.

```
[ConnectorDescription(
        DisplayName = "Console output",
        CanWriteContacts = true,
        CanReadContacts = false)]
```

The ConnectorDescription attribute describes capabilities of the connector. The defaults of the settings are:

```
CanReadContacts = true,
CanWriteContacts = true,
NeedsCredentials = false,
IsGeneric = false,
MatchingIdentifier = ProfileIdentifierType.Default
```

As shown in the defaults, the connector might be a generic class and does have a fixed matching identifier. By default all connectors tell the UI that they are capable for reading and writing, but our example connector (the console connector) does only support writing to the console, not reading.

```
public class ContactClient : StdClient
```

Most connector classes are named "ContactClient" – I will change the term "client" to the term "connector" someday, but that's not a priority 1 task at the moment. As you can see the class does inherit from StdClient – this class does provide things like password handling with the sync engine virtual methods to add provider specific things for read/write, event handling etc.

```
private static readonly XmlSerializer ContactListFormatter
                        = new XmlSerializer(typeof(StdContact));
```

The console connector does print the contact serialized as xml to the console, so we need an instance of the XmlSerializer to build the xml.

```
public override string FriendlyClientName
{ get { return "Console output Connector for individual contacts"; }}
```

Overriding this virtual property is optional, because the base class property does return the name configured inside the ConnectorDescription class attribute.

```
public override void AddRange(List<StdElement> elements, string clientFolderName)
{ this.WriteFullList(elements, clientFolderName, true); }
```

Overriding this method suppresses the read operation from the base class while adding new elements which is not implemented in our case and would lead to an exception. If you don't override this method, the base class will use the GetAll-method to get all elements, remove existing elements and add the elements from the source – this might add significant processing overhead. In most cases you should use this override to speed up write operations.

```
protected override void WriteFullList(List<StdElement> elements,
                                       string clientFolderName,
                                       bool skipIfExisting)
{
    foreach (var element in elements)
    {
        ContactListFormatter.Serialize(Console.Out, element);
    }
}
```

This is the "working horse" method – this is the part where things may become complex (in this case they don't).

That's all. For a simple writing connector that's all sync-specific stuff to be implemented. All other code has to do with dealing with provider specific protocols, data formats, etc.

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:

```
namespace Sem.Sync.SyncBase.Commands
{
    using GenericHelpers.Interfaces;
    using Interfaces;

    public class AskForContinue : ISyncCommand
    {
        public IUiInteraction UiProvider { get; set; }
        public bool ExecuteCommand( IClientBase sourceClient, IClientBase targetClient,
                                    IClientBase baseliClient, string sourceStorePath, string
                                    targetStorePath, string baselineStorePath, string
                                    commandParameter)
        {
            return this.UiProvider == null
                || this.UiProvider.AskForConfirm(commandParameter,
                                                  (targetClient == null)
                                                  ? "Sem.Sync"
                                                  : targetClient.FriendlyClientName);
        }
    }
}
```

As you can see the implementation of the `ISyncCommand` interface is really simple – just provide a public property and implement one single method that returns a Boolean specifying if the execution should continue. The execution of the command can contain as much logic as you need; in this case it's just calling `this.UiProvider.AskForConfirm()` to ask the user if she/he wants to continue. You should return `true` for "continue execution" and `false` for "abort the workflow".

Logging and Exception Handling

The `Sem.Sync.SyncBase` provides inside the base class `SyncComponent` methods for logging capability. This includes logging normal processing events (like reading a specific contact) with the method `LogProcessingEvent` and logging exceptions with the method `LogException`. You should always call `LogException` in case of catching a nonspecific exception. `LogException` will test for some internal exceptions like the `ProcessAbortException` (which needs to be re-thrown in order to really cancel the execution).

Auto-Update-Check

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

The file will not be updated with every release on CodePlex. Also there's no automatic action in case of an "old" version – just a log entry (I may implement a UI action for this case to download the new installation package some day).

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 and an identifier of a social network. 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).

You can currently only define one source per connector – that's enough for social networks, but will become a problem when handling two or more different databases or file system objects that should be merged. This problem will be solved soon, but solving the issue will break compatibility with already saved workflow definitions.

Some connectors cannot write contacts (most social network connectors 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.

Matching Contacts of different Sources

Each contact does have a unique identifier (a GUID). Unfortunately we cannot calculate that value from the source data, so each time we read the data we will get a new GUID. Additionally the contact can have one or more Profile Identifier. Each Profile Identifier does contain a value identifying the source data for this particular contact. The library does manage the mapping of Profile Identifier to Contact ID with a “base line” (the scripts in this solution assume the base line to be an xml file, but because it’s loaded using a connector, you can even write a connector for your proprietary storage system and place it there).

Design was originally made for synchronizing social networks to Microsoft Outlook, so there’s still some limitation for working with other data stores: currently there’s only one identifier possible per connector. This was correct for social networks, but is a problem when working e.g. with two databases. In this case you may have identical IDs for different contacts in the databases, so the library will become confused. I’m currently working on a change in handling Profile Identifiers to enable a per storage path mapping.

Localization

Localization of the applications is done directly inside the projects using satellite assemblies. The neutral language for the assemblies is English (`"en-US"`) but I did already add resources for German localization (`"de-DE"`) – which are far away from being complete.

If you want localization for other languages: send me some resource files with the translation and I’ll include them into the project.

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 – someday.
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.

Interesting things in code

Usage of Attributes

The attributes `ClientStoragePathDescription` and `ConnectorDescription` do determine changes inside the UI. They are interpreted by the UI classes to show/hide elements like credential input boxes or path selection controls. Also they do describe capabilities of the connectors without the need of multiple interfaces – using the attributes you can tag connectors as read only or as internal to hide them from the GUI.

Abstraction of UI

All UI interaction has been "outsourced" from the base libraries through interfaces to different assemblies. This way you can use the base library in any UI environment – WinForms, WebForms, and WPF or even inside a service. Otherwise the base libraries where bound to one presentation technology.

Null Reference Prevention

In a specific area of application (mapping entities) properties that might be null are extremely bad. For this purpose I've created a small generic extension method:

```
public static T NewIfNull<T>(this T testObject) where T : class, new()
{
    return testObject ?? new T();
}
```

This method tests if an object is null and returns a new object of that type in case of the object being null. If the entities do provide meaningful default values, such a method enables code like this:

```
var domain = x.myProp5.NewIfNull().Domain;
```

In this line the property `myProp5` might be null. In that case accessing its property `Domain` would generate a null-reference exception. The correct implementation without `NewIfNull` would first test for `x.myProp5` being null and use an empty string in that case, otherwise the property `Domain` would be read. The use of the `NewIfNull()` method implements exact this behavior by "on the fly generation" of an instance of the missing property – but in only one line.

I've done some additional overloads to deal with non-existing elements in arrays, lists and dictionaries.

Invoking a Method with every Member of a Collection

Linq does not implement a `ForEach` as an extension for `IEnumerable` because of a philosophical reason: "all Linq extensions should be side-effect-free". In this case I think they've lost the goal: to be productive, not to be academically right. Here you have it:

```
public static void ForEach<T>(this IEnumerable<T> enumerable, Action<T> action)
{
    foreach (var element in enumerable)
    {
        action.Invoke(element);
    }
}
```

There's an overload with `Action<T1, T2>`, too – you can build one for `Action<T1, T2, T3>` if you want to.

GetPropertyValue to parse variable property-paths

When mapping entities in an XML file, you need to specify how to access the values. There's a recursive method `GetPropertyValue` and a method `SetPropertyValue` to support the mapping functionality. These methods do search through the object hierarchy to get a value from a property or method. Using these methods you can store mappings in XML or other textural containers.

The DefaultValue attribute

You always have a default for the properties when serializing and deserializing objects (e.g. an empty `string`, `false` in case of `bool` or `0` in case of `int`). You can reduce the size of a serialized object substantially by removing these defaults from the serialized XML – the attribute `[DefaultValue(…)]` will do this for you. It instructs the `XmlSerializer` to not write these values into the XML.

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 override the one in the target if the source picture is larger (in binary size).

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 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.

Question: Can you please add the site XYZ?

Answer: It depends – just drop me a mail if you want some other source / target to be implemented. I’ll reply with a date when I’ll start implementing or with a statement why I’ll not implement that.

Question: Using crawlers to spider site XYZ is illegal!?

Answer: It depends. There has been some trouble in the past for someone who did spider a full site and did try to get some money from the owner of that site in order to delete the data. I’m not interested in the data of site XYZ – neither am I interested in getting trouble or asking some site owner for money to stop this project. I just want to sync my personal contacts between different systems and I will NEVER collect data to publish it. If see a problem in using the connectors for reading contacts from your site, contact me – we may find a way to enable syncing in a way comfortable for you and me.

Question: Why do I get “Project type not supported” errors?

Answer: One of the goals of this project is to get familiar with different tools. If you don’t want to install the tools, just load the alternative solution “SemSync-Standard.sln”, which does only load the project that are based on standard project types.

Question: Each time I write data to the destination storage, I see read operations.

Answer: You should override the `AddRange`-method as described in chapter "Anatomy of a Simple Connector" on page 30. The `GetAll`-method is called by the base class in this method, so you need to override this method in order to suppress a full read operation.