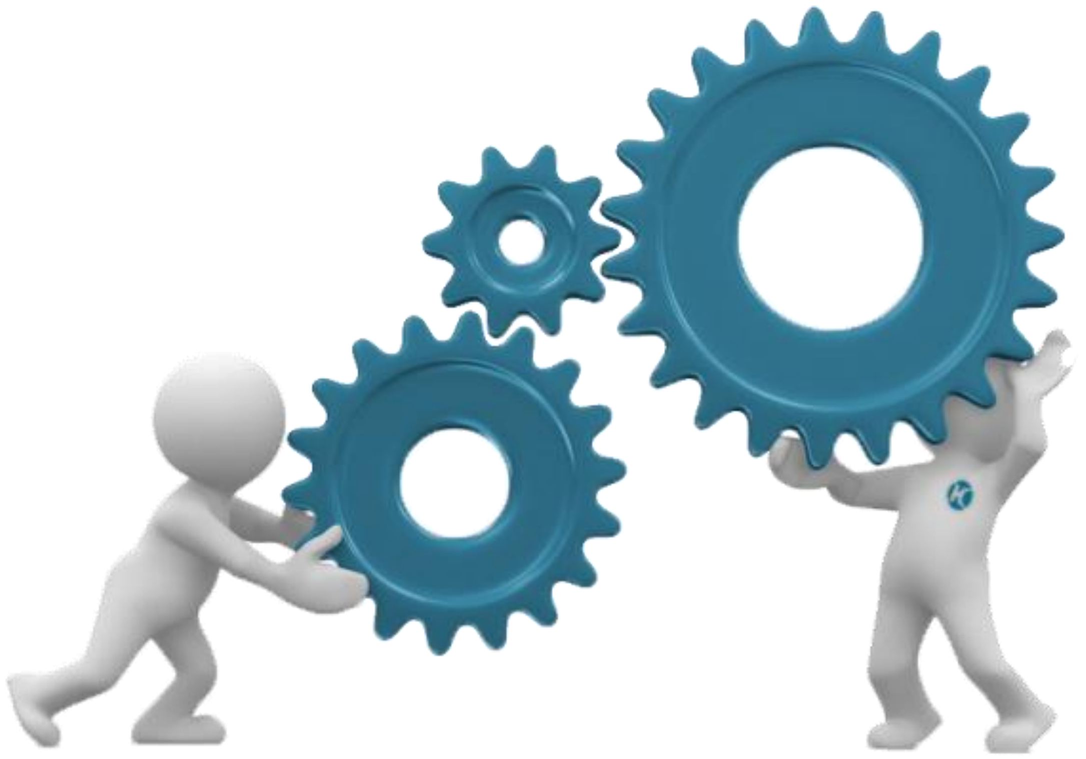
User Manual

for uConnector



1. INTRODUCTION 3
1.1. WHAT IS UCONNECTOR? 3
1.2. THE ARCHITECTURE OF UCONNECTOR 3
1.2.1. Tasks and Operations 4
1.2.2. Receivers, Transformers and Senders 4
2. GETTING STARTED 6
2.1. UCONNECTOR SAMPLES SOURCE 6
2.2. LOCAL DIRECTORIES 6
2.3. UMBRACO SETUP WITH UCOMMERCE 6
2.4. TEST UCONNECTOR 7
3. UCONNECTOR AS A SERVICE 8
3.1. INSTALL 8
3.2. UNINSTALL 9
4. EXAMINING MYSAMPLEOPERATION 10
4.1. CREATING AN OPERATION WITH THE FLUENT API 10
5. TEST RUN A TASK 12
6. ENABLE AND RUNNING AN OPERATION USING THE SERVER 13

1. Introduction

This is not a typical user manual, but more of a guidance of how certain problems can be solved using uConnector.

Furthermore it will contain some tips of some of the built-in senders, receivers, adapters and cogs.

1.1. What is uConnector?

uConnector is a pluggable, extendable framework for integration code, moving data from one system to another.

It is an attempt at bringing structure and reusability to the many console applications created for

1. Receive data from system A.
2. Transforming data from system A to system B format.
3. Sending transformed data to system B.

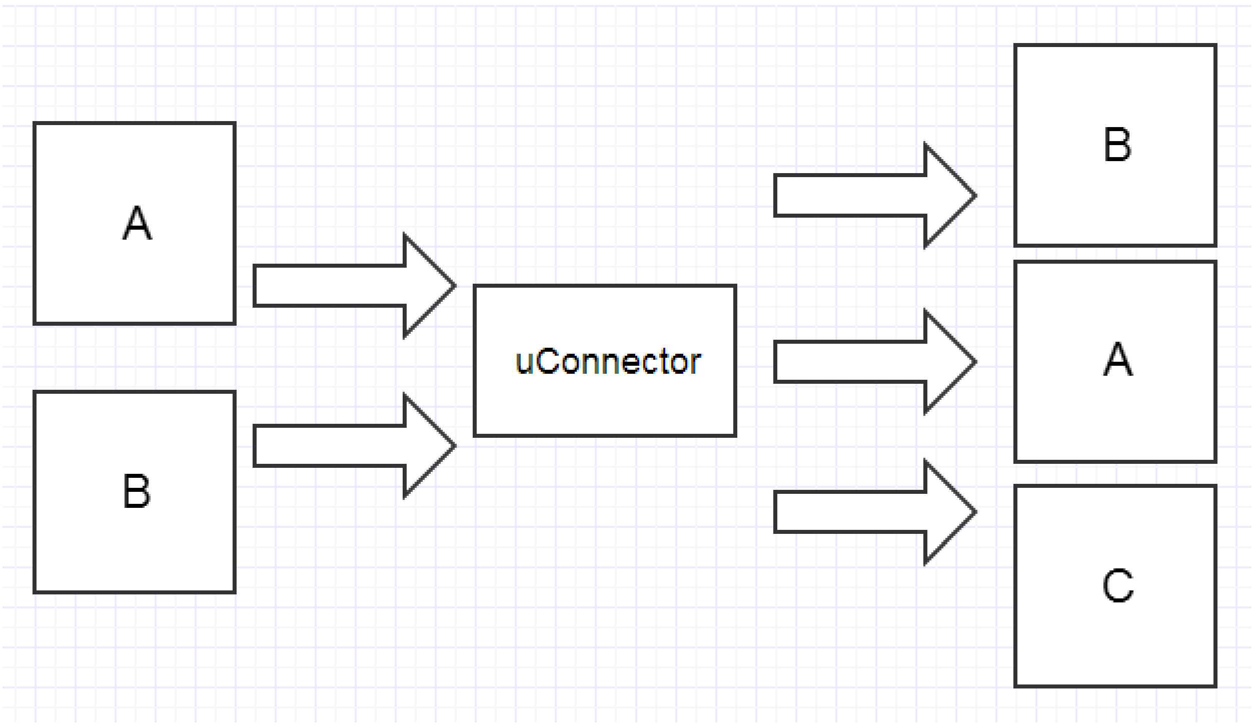
The framework makes it:

- Easy to set up a scheduled execution of code.
- Easy to install and run as a windows service.
- Easy to reuse component for reading data from and sending data to systems.
- Easy to overrule the framework behavior, by adding your own components.

The extendable structure makes it:

- Easy to distribute components created for specific systems, such as “uCommerce”.

1.2. The architecture of uConnector



uConnector is the active part in the transferring of data between systems. This means that uConnector contacts system A for data. System A does not contact uConnector. As a consequence, the transferring of data is intended to be in larger batches at regular intervals.

uConnector is specifically not intended as an event listener, waiting for events to fire in system A.

Frameworks based upon listening for events, and then transferring a small amount of data each time, tends to become very “chatty”. The error and recovery scenarios can easily become very complex. What if the listener is offline? What if the data is lost after reception?

The architecture for uConnector is aiming for simplicity, robustness and reusability.

Users of uConnector can spend their time writing business specific code! The framework handles all the tedious bits.

Please note that uConnector is not tied to any two specific systems. The same server instance can have tasks configured to move data between many different combinations of source and target systems.

1.2.1. Tasks and Operations

An Operation is a description one flow of “Receive, Transform and Send” put together by basic components. So an operation is a sequence of steps to perform. An operation can typically be configured with information like, where to look for data files, usernames and passwords needed to perform the operation, etc.

A Task, is simply a schedule and an operation with a specific configuration. So a Task is “When to perform which operation with what configuration”.

1.2.2. Receivers, Transformers and Senders

Receivers, Transformers and Senders are the basic building blocks of operations.

Receivers are responsible for receiving data from an external system. For example:

- Reading files from a local directory.
- Reading files from an Ftp server.
- Reading data from a table in a database.
- Reading data from an external system: uCommerce, ERP system, etc.
- Reading data from a web service.
- Etc.

Data can be anything and come from anywhere!

Transformers are responsible for transforming the data received by a receiver into a different format. For example:

- XML data into CSV data.
- CSV data into data tables.
- Data tables into a proprietary format.

- Etc.

Data can be transformed from any format to any other format!

Senders are responsible for sending data to an external system. For example:

- Writing data to a file.
- Sending a file to an Ftp server.
- Writing data to a database.
- Posting data to a web site.
- Etc.

Data can be send anywhere!

2. Getting started

The uConnector samples source contains everything you need to get started using uConnector. It contains the latest instance of the uConnector executable “UConnector.Server.exe” and all the dependent assemblies.

2.1. uConnector samples source

Clone the source of the samples project from www.bitbucket.org with your favorite Hg tool.

The url is: https://bitbucket.org/uCommerce/uconnector-samples

Since this is still public there are no password protection on this, this could however change is the future without warning.

This will be cloned to the following directory:

- C:\Hg\uConnector Samples\

For the rest of this “User manual” that will be referenced to as the “source” directory.

2.2. Local directories

For the best experience and so you don’t have to change too many configuration files, we suggest that you use the following structure.

On the C-Drive create this directory structure.

- uConnector
    - ConfigFiles
    - FromFtp
    - In
    - Out

2.3. Umbraco setup with uCommerce

- Installation guide for Umbraco can be found at:
    - http://our.umbraco.org/wiki/how-tos/a-complete-newbie's-guide-to-umbraco
- Installation guide for uCommerce can be found at:
    - http://www.ucommerce.dk/screencasts/uCommerceTV-01-Installing-uCommerce.mov

2.4. Test uConnector

In the source directory to go: “lib\uConnector”.

This should contain a file called:

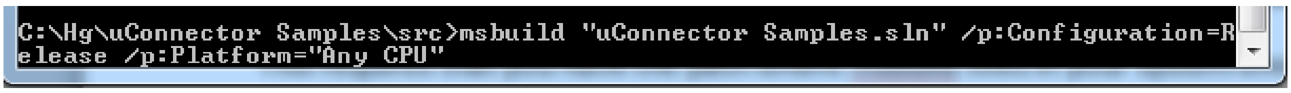
- UConnector.Server.exe.config.default

Make a copy of it and name it:

- UConnector.Server.exe.config

This will be the main configuration file for the uConnector.

Afterwards compile the uConnector samples project.



```
C:\Hg\uConnector Samples\src>msbuild "uConnector Samples.sln" /p:Configuration=R
elease /p:Platform="Any CPU"
```

This requires that you have the path where msbuild lives in your system environment PATH variable. If you don’t, you need to supply the complete path to msbuild.

This compiles the uConnector Samples project with the configuration “Release” and target platform “Any CPU”. This can also be done from visual studio if you are more confident with that way, but is out of scope of this guide.

Changes to above configuration file:

In the section “uConnector” make the following changes.

The “taskPaths” should point to directories containing operation configurations. If you compiled the Release is should be some like this:

- <add path="C:\Hg\uConnector Samples\src\uConnector.Samples\bin\Release\Tasks" />

If you have more directories that contain Tasks just append them.

The “assemblyPaths” should contain paths where the assemblies for the tasks is placed. If you compiled the “Release” this should be placed in the following directory.

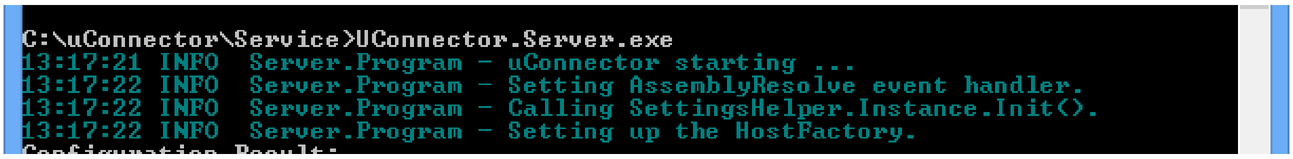
- <add path="C:\Hg\uConnector Samples\src\uConnector.Samples\bin\Release" />

Paths can also be relative from the “UConnector.Server.exe”.

Now try and run the application:

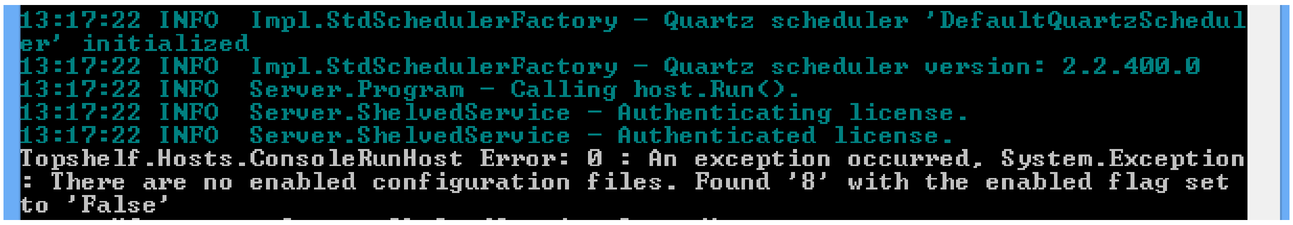
Figur 1 When starting uConnector for the first time

You should see some like the above in the beginning.



```
C:\uConnector\Service>UConnector.Server.exe
13:17:21 INFO  Server.Program - uConnector starting ...
13:17:22 INFO  Server.Program - Setting AssemblyResolve event handler.
13:17:22 INFO  Server.Program - Calling SettingsHelper.Instance.Init().
13:17:22 INFO  Server.Program - Setting up the HostFactory.
```

After some time it should display some exception like the following



```
13:17:22 INFO  Impl.StdSchedulerFactory - Quartz scheduler 'DefaultQuartzSchedul
er' initialized
13:17:22 INFO  Impl.StdSchedulerFactory - Quartz scheduler version: 2.2.400.0
13:17:22 INFO  Server.Program - Calling host.Run().
13:17:22 INFO  Server.ShelvedService - Authenticating license.
13:17:22 INFO  Server.ShelvedService - Authenticated license.
Topshelf.Hosts.ConsoleRunHost Error: 0 : An exception occurred, System.Exception
: There are no enabled configuration files. Found '8' with the enabled flag set
to 'False'
```

Figur 2 Exception when no operations are enabled

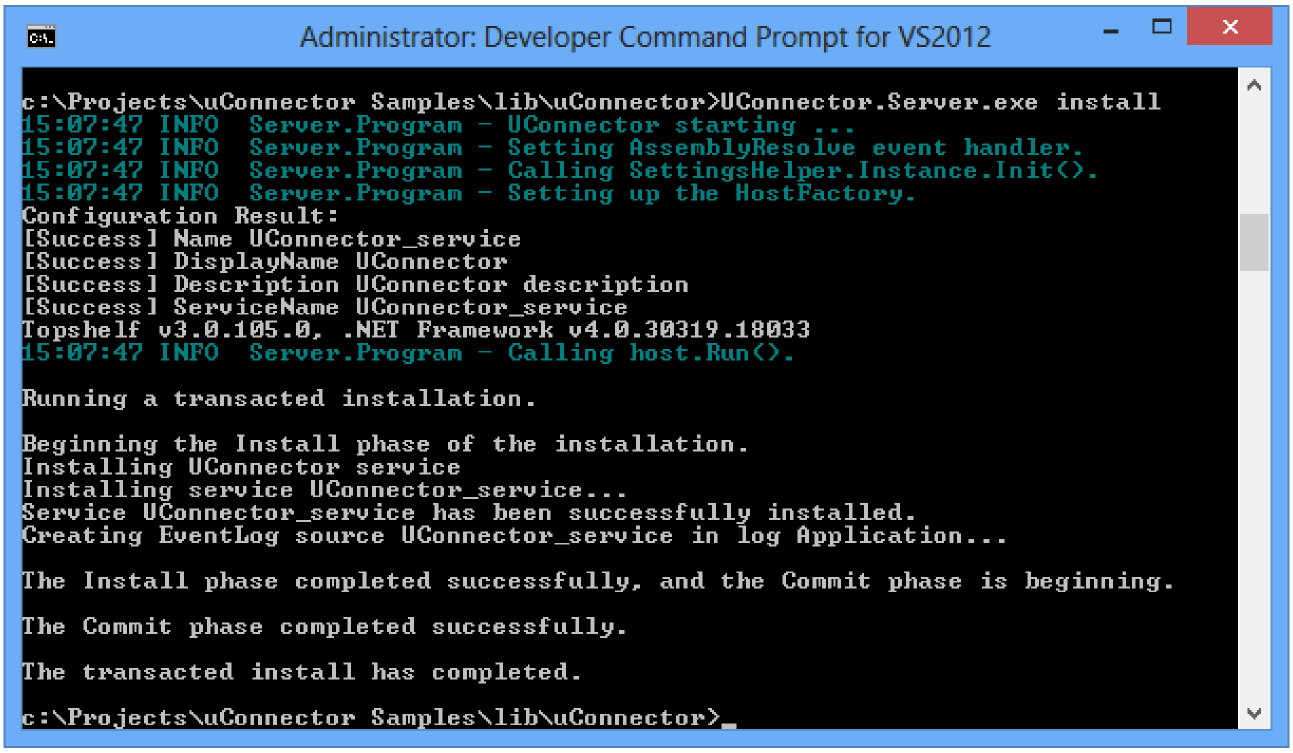
That just tells you that it found 8 operations, but none of them were enabled. This is exactly what it should do, since none of them are enabled by default.

Press "CTRL + C" to exit the console runner.

3. uConnector as a Service

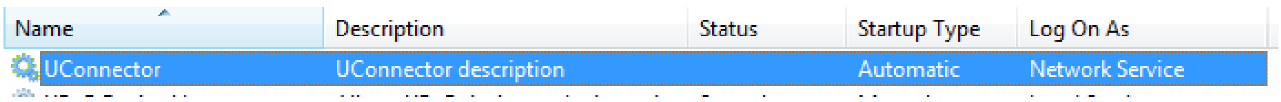
3.1. Install

If you install uConnector on a server, it's more likely that you want to run uConnector as a service, since it will start automatically when the server boots.



```
c:\Projects\uConnector Samples\lib\uConnector>UConnector.Server.exe install
15:07:47 INFO  Server.Program - UConnector starting ...
15:07:47 INFO  Server.Program - Setting AssemblyResolve event handler.
15:07:47 INFO  Server.Program - Calling SettingsHelper.Instance.Init().
15:07:47 INFO  Server.Program - Setting up the HostFactory.
Configuration Result:
[Success] Name UConnector_service
[Success] DisplayName UConnector
[Success] Description UConnector description
[Success] ServiceName UConnector_service
Topshelf v3.0.105.0, .NET Framework v4.0.30319.18033
15:07:47 INFO  Server.Program - Calling host.Run().

Running a transacted installation.

Beginning the Install phase of the installation.
Installing UConnector service
Installing service UConnector_service...
Service UConnector_service has been successfully installed.
Creating EventLog source UConnector_service in log Application...

The Install phase completed successfully, and the Commit phase is beginning.

The Commit phase completed successfully.

The transacted install has completed.

c:\Projects\uConnector Samples\lib\uConnector>_
```

This will create the following entry in the services snap-in.



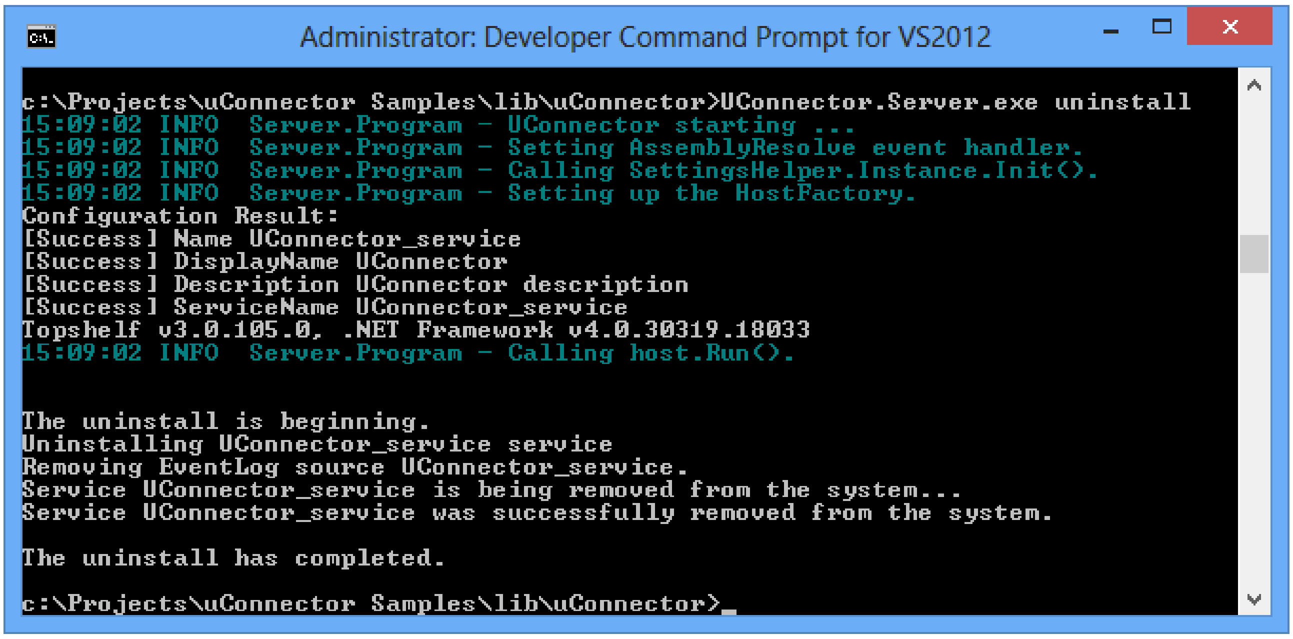
Name	Description	Status	Startup Type	Log On As
UConnector	UConnector description		Automatic	Network Service

Please note after installation of the service it will not start. You have to do that manually afterwards.

If you need to see what's going on, the system uses log4net. By default uConnector uses a RollingFileAppender and logs to: log.txt . You can change the logging settings in the uConnector configuration file created in "Test uConnector" section of this document.

3.2. Uninstall

If you need to uninstall it, just issue the following command and you are done.



```
c:\Projects\uConnector Samples\lib\uConnector>UConnector.Server.exe uninstall
15:09:02 INFO  Server.Program - UConnector starting ...
15:09:02 INFO  Server.Program - Setting AssemblyResolve event handler.
15:09:02 INFO  Server.Program - Calling SettingsHelper.Instance.Init().
15:09:02 INFO  Server.Program - Setting up the HostFactory.
Configuration Result:
[Success] Name UConnector_service
[Success] DisplayName UConnector
[Success] Description UConnector description
[Success] ServiceName UConnector_service
Topshelf v3.0.105.0, .NET Framework v4.0.30319.18033
15:09:02 INFO  Server.Program - Calling host.Run().


The uninstall is beginning.
Uninstalling UConnector_service service
Removing EventLog source UConnector_service.
Service UConnector_service is being removed from the system...
Service UConnector_service was successfully removed from the system.

The uninstall has completed.

c:\Projects\uConnector Samples\lib\uConnector>_
```

4. Examining MySampleOperation

It is time to take a closer look at the first sample operation!

4.1. Creating an operation with the fluent API

MySampleOperation is a very simple operation, which reads an XML file containing book data and writes the names of the authors to a text file.

This is done using only already available components!

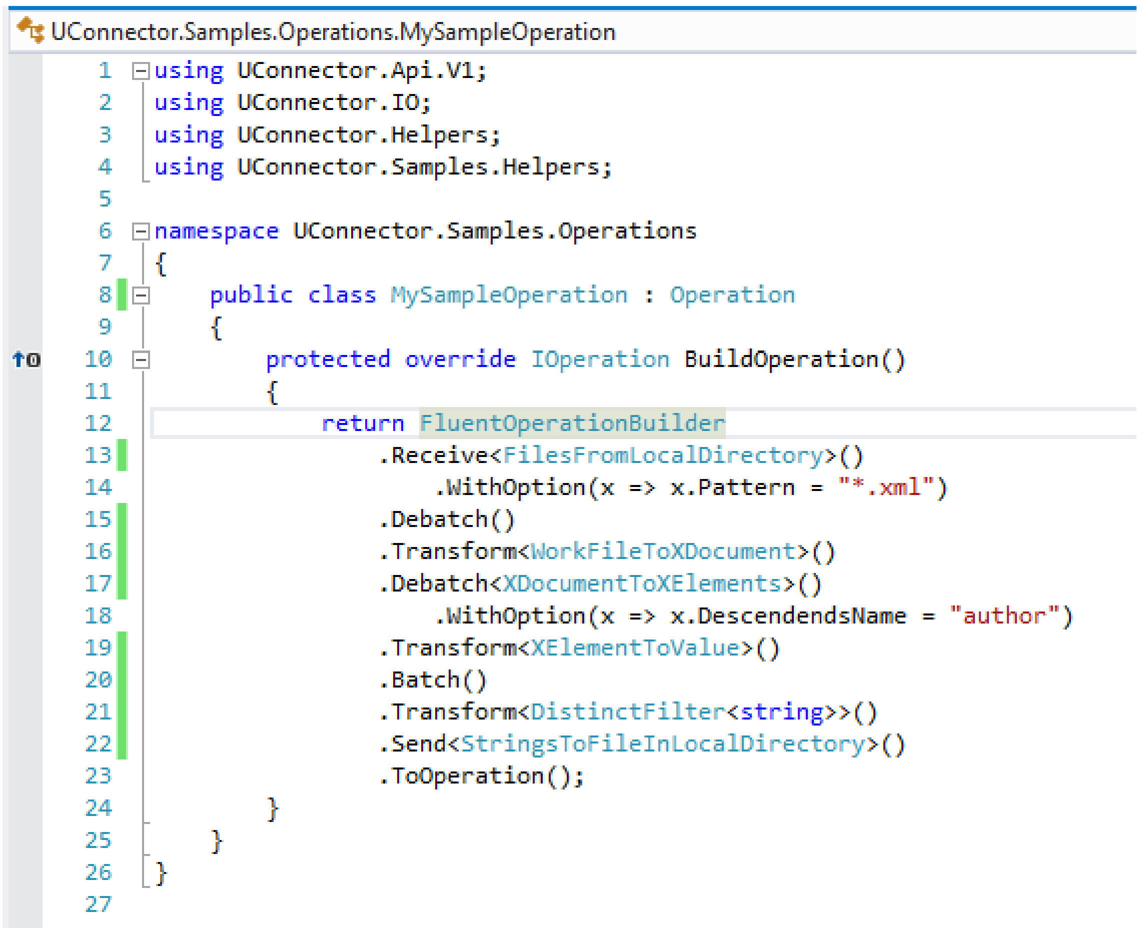
Find the file called MySampleOperation.cs in the Samples project. It is in the folder called "Operations".

Make the class inherit from the uConnector framework class "UConnector.Operation".

UConnector.Operation is an abstract class with one virtual method called BuildOperation() that returns a UConnector.IOperation instance.

To get an IOperation instance we use the UConnector fluent configuration api. Add the using statement "`using UConnector.Api.V1;`"

This gives us access to the class FluentOperationBuilder. This is what is used to build uConnector operations. The code looks like this:



UConnector.Samples.Operations.MySampleOperation

```
using UConnector.Api.V1;
using UConnector.IO;
using UConnector.Helpers;
using UConnector.Samples.Helpers;

namespace UConnector.Samples.Operations
{
    public class MySampleOperation : Operation
    {
        protected override IOperation BuildOperation()
        {
            return FluentOperationBuilder
                .Receive<FilesFromLocalDirectory>()
                    .WithOption(x => x.Pattern = "*.xml")
                .Debatch()
                .Transform<WorkFileToXDocument>()
                .Debatch<XDocumentToXElements>()
                    .WithOption(x => x.DescendendsName = "author")
                .Transform<XElementToValue>()
                .Batch()
                .Transform<DistinctFilter<string>>()
                .Send<StringsToFileInLocalDirectory>()
                .ToOperation();
        }
    }
}
```

What does all this mean? Well, remember that an operation is basically something that receives data, transforms data and finally sends data.

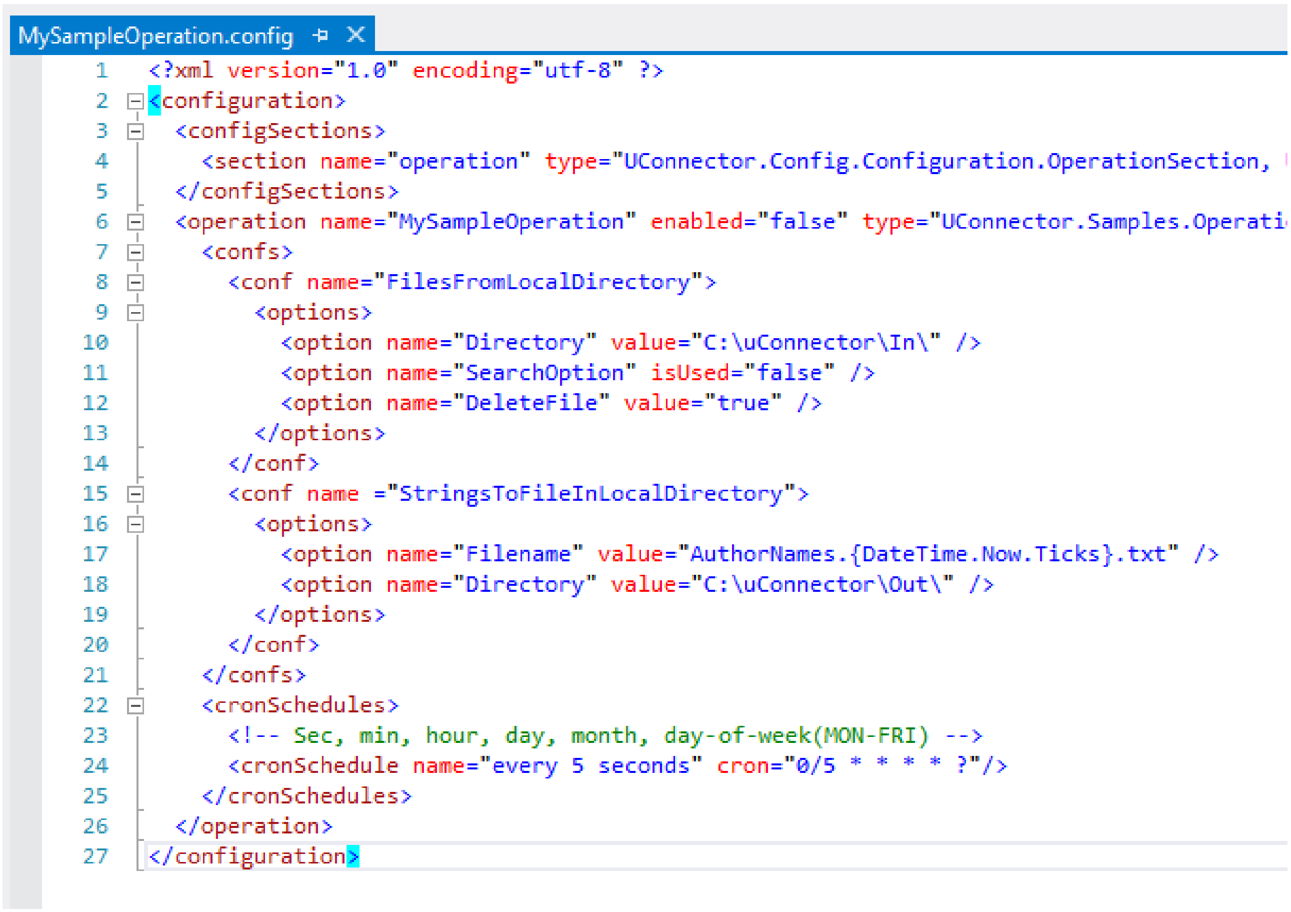
Let us go through the code, line by line.

- “`Receive<FilesFromLocalDirectory>()`”
  Starts the operation by using the standard receiver called FilesFromLocalDirectory. This standard receiver reads files from a local directory, and passes the files along to the next step in the operation.
- “`WithOption(x => x.Pattern = "*.xml")`”
  Instructs the operation to set the Pattern option on the FilesFromLocalDirectory receiver to look for xml files. FilesFromLocalDirectory has many other options, but we leave the setting of those to the task configuration file. In general, the method WithOption always refers to the previously added step. You can have as many WithOptions in a row as needed.
- “`Debatch()`”
  The type returned by FilesFromLocalDirectory is IEnumerable<WorkFile>. So it is a batch of files. So calling Debatch() instructs the framework to process the files one at a time.
  The type WorkFile is a simple wrapper type around a Stream, for reading the file, and information about the filename and location.
- “`Transform<WorkFileToXDocument>()`”
  The output from the previous step was a WorkFile. This standard transformer takes a WorkFile and transforms it into an XDocument.
- “`Debatch<XDocumentToXElements>()`”
  This transforms the XDocument into a number of elements, and instructs the framework to Debatch the resulting list of XElements. The type is XElement after the debatching.
- “`WithOption(x => x.DescendendsName = "author")`”
  Sets the configuration option “DescendantsName” on the previous step to “author”. This instructs the previous step to iterate over all elements called “author” in the XDocument it receives.
- “`Transform<XElementToValue>()`”
  A simple transformer that takes an XElement and returns the value of the element. This is of type string. You can find the code for this transformer included in the Samples project.
- “`Batch()`”
  The previous step produces a whole number of strings. This instructs the framework to gather them into one large batch, the type of which is IEnumerable<string>
- “`Transform<DistinctFilter<string>>()`”
  Yet another simple transformer. This one simply filters the input to only pass on distinct values.
- “`Send<StringsToFileInLocalDirectory>()`”
  The last step in the operation is, of cause, a sender. This particular sender takes a number of strings writes them to a file. The configuration of the sender, such as the file name and the directory to write the file to, we leave for the configuration of the task.
- “`ToOperation()`”
  And finally we instruct the fluent builder that we are done defining the operation, and to please build it for us.

Phew! A lot of words for not so much code. Hopefully the code is easier to intuitively read, than to describe in words.

This was a simple example of an operation. When if we want to run this operation we need to create a Task for it. This is done using a config file placed in the Task directory.

Let's take a look at a Task that uses MySampleOperation. Look at the file called "MySampleOperation.config".



```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <section name="operation" type="UConnector.Config.Configuration.OperationSection, 
  </configSections>
  <operation name="MySampleOperation" enabled="false" type="UConnector.Samples.Operati
    <confs>
      <conf name="FilesFromLocalDirectory">
        <options>
          <option name="Directory" value="C:\uConnector\In\" />
          <option name="SearchOption" isUsed="false" />
          <option name="DeleteFile" value="true" />
        </options>
      </conf>
      <conf name ="StringsToFileInLocalDirectory">
        <options>
          <option name="Filename" value="AuthorNames.{DateTime.Now.Ticks}.txt" />
          <option name="Directory" value="C:\uConnector\Out\" />
        </options>
      </conf>
    </confs>
    <cronSchedules>
      <!-- Sec, min, hour, day, month, day-of-week(MON-FRI) -->
      <cronSchedule name="every 5 seconds" cron="0/5 * * * * ?"/>
    </cronSchedules>
  </operation>
</configuration>
```

Sorry, it can't all fit on one page!

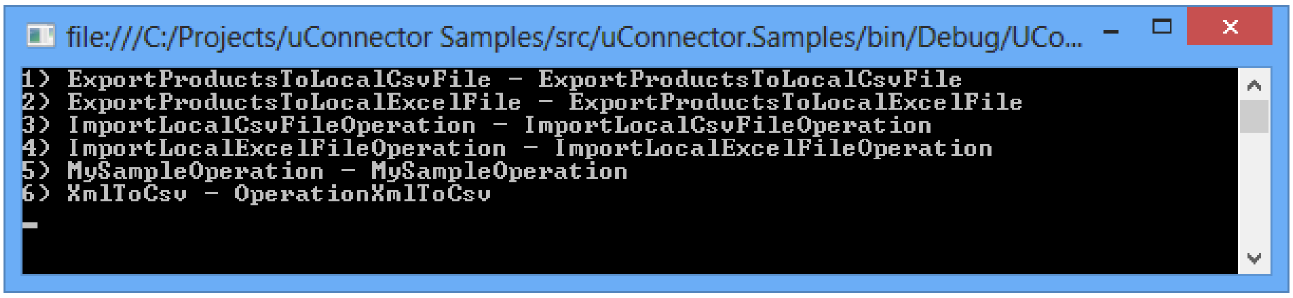
Two of the steps in "MySampleOperation" requires configuration. The "FilesFromLocalDirectory" requires details concerning the place to look for files. And "StringsToFileInLocalDirectory" requires details about where to write the output file.

The last thing to configure is the scheduling of the task. How often should it run? In this example we use the "cron" syntax for running the task once every 5 seconds. I strongly encourage you to take a closer look at the "cron" syntax, to get familiar with it. It is very powerful, and quite easy once you get used to it.

5. Test run a task

Now let's try running the task "MySampleOperation"!

In Visual Studio select the Samples project as the StartUp project. When you run it, you should see something like this on the screen:

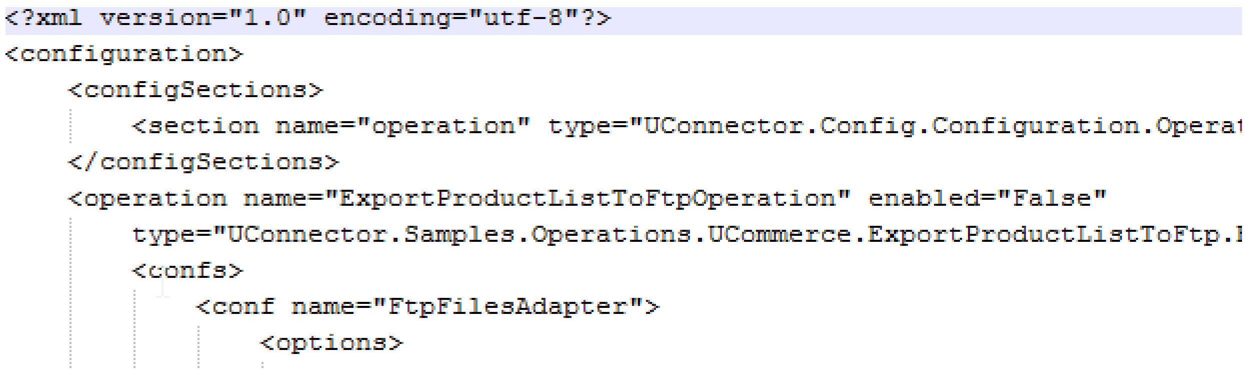

file:///C:/Projects/uConnector Samples/src/uConnector.Samples/bin/Debug/UCo...

```
1) ExportProductsToLocalCsvFile - ExportProductsToLocalCsvFile
2) ExportProductsToLocalExcelFile - ExportProductsToLocalExcelFile
3) ImportLocalCsvFileOperation - ImportLocalCsvFileOperation
4) ImportLocalExcelFileOperation - ImportLocalExcelFileOperation
5) MySampleOperation - MySampleOperation
6) XmlToCsv - OperationXmlToCsv
_
```

Enter 5 to run the task!

6. Enable and running an operation using the server

- To enable an operation, go the directory you set in the uConnector configuration.
- Find the operation you want to enable.
- Open it in your favorite editor

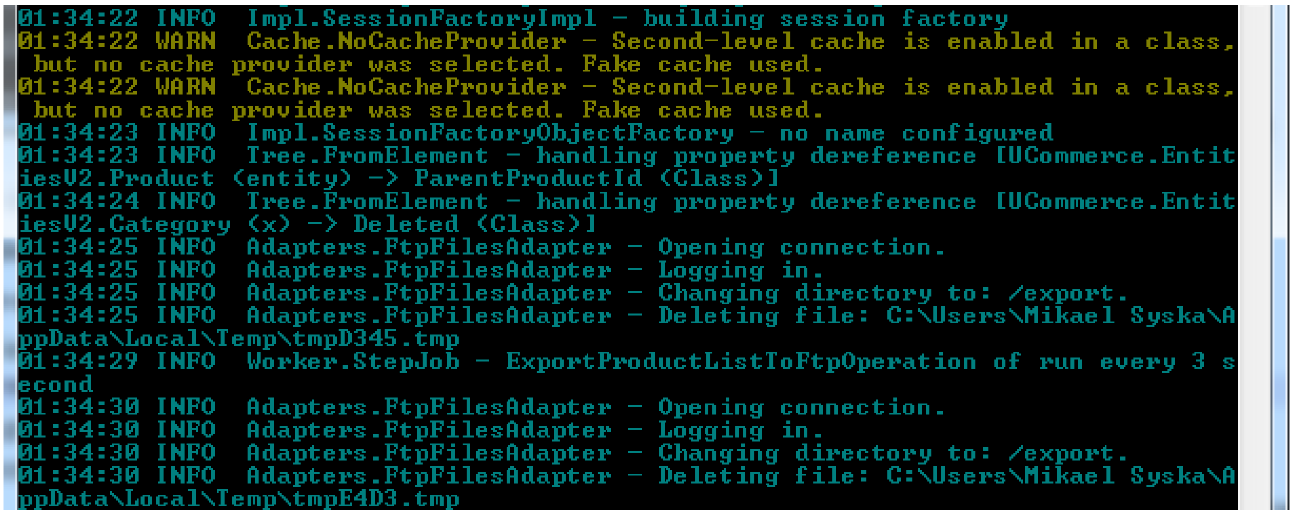


```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <configSections>
        <section name="operation" type="UConnector.Config.Configuration.Operat
    </configSections>
    <operation name="ExportProductListToFtpOperation" enabled="False"
        type="UConnector.Samples.Operations.UCommerce.ExportProductListToFtp.I
        <confs>
            <conf name="FtpFilesAdapter">
                <options>
```

- Find the operation section in the XML file and set the “enabled” attribute to “True”
- The operation is now enabled.

You should now be able to able to run uConnector as a Service or console runner and see the Operation run.

Depending on what operation(s) enabled you should see something like this:



```
01:34:22 INFO  Impl.SessionFactoryImpl - building session factory
01:34:22 WARN  Cache.NoCacheProvider - Second-level cache is enabled in a class,
 but no cache provider was selected. Fake cache used.
01:34:22 WARN  Cache.NoCacheProvider - Second-level cache is enabled in a class,
 but no cache provider was selected. Fake cache used.
01:34:23 INFO  Impl.SessionFactoryObjectFactory - no name configured
01:34:23 INFO  Tree.FromElement - handling property dereference [UCommerce.Entit
iesV2.Product (entity) -> ParentProductId (Class)]
01:34:24 INFO  Tree.FromElement - handling property dereference [UCommerce.Entit
iesV2.Category (x) -> Deleted (Class)]
01:34:25 INFO  Adapters.FtpFilesAdapter - Opening connection.
01:34:25 INFO  Adapters.FtpFilesAdapter - Logging in.
01:34:25 INFO  Adapters.FtpFilesAdapter - Changing directory to: /export.
01:34:25 INFO  Adapters.FtpFilesAdapter - Deleting file: C:\Users\Mikael Syska\A
ppData\Local\Temp\tmpD345.tmp
01:34:29 INFO  Worker.StepJob - ExportProductListToFtpOperation of run every 3 s
econd
01:34:30 INFO  Adapters.FtpFilesAdapter - Opening connection.
01:34:30 INFO  Adapters.FtpFilesAdapter - Logging in.
01:34:30 INFO  Adapters.FtpFilesAdapter - Changing directory to: /export.
01:34:30 INFO  Adapters.FtpFilesAdapter - Deleting file: C:\Users\Mikael Syska\A
ppData\Local\Temp\tmpE4D3.tmp
```