

Monday, October 24, 2016

SwissDecTX Transmitter Gateway Service Technical Overview 4.08

PRELIMINARY DOCUMENTATION – SUBJECT TO CHANGE

Starting with version 4.03, the SwissDecTX transmitter supports an optional gateway service, licensed separately, which allow remote access to the transmitter from computers not directly connected to the internet or from computers not running the Microsoft Windows operating system.

System Requirements for the SwissDecTX Transmitter Gateway Service:

- **Windows Vista or later**, any edition except Starter and Home editions.
- **Windows Server 2008 or later**, any edition except IA-64 (Itanium) editions.

The SwissDecTX Gateway service makes use of advanced system-provided facilities such as the Transactional NTFS, and as such the Gateway Service requires at least one of the above operating systems or newer versions. While the SwissDecTX transmitter itself works perfectly well on Windows XP and Windows Server 2003 (theoretically, also on Windows 2000 or even Windows 95 (!) as long as the .NET Framework 2.0 can be installed) the *Gateway* requires a newer operating system. We prefer to run the gateway on Windows 7 SP1 or Windows 8.x 64-bit, or a Server edition such as Windows Server 2008, 2008 R2 or Windows Server 2012 or newer.

The remote computer(s) sending data to the transmitter through the Gateway can run *any* operating system and the only requirement is that those machines are able to connect to a standard SMB network share, with file create/read/write/delete ability. This requirement can be restated as the ability to connect to and use a Windows file server on the LAN or WAN.

The entire system consists of the following:

- One computer running Microsoft Windows (for example Windows 7 or 8, 32-bit or 64-bit, or some Windows Server version). This computer has internet access and the SwissDecTX Transmitter 4.03 or later installed and configured, as well as the SwissDecTX Gateway 4.03 or later software. In the sequel, this machine will be referred to as the “gateway computer” or simply the “gateway”. The gateway computer exposes a local network share that is referred as the “network share”. Local in the sense that a folder local to the gateway computer is shared, in other words this folder must reside on a local volume (and be formatted using the NTFS file system).
- One or more remote computers with the ability to connect to a Windows file-server network share to create, write and read to/from files. In the sequel, those machines will be referred to as the “remote computers”. The remote computers run the salary application that wants to transmit data to SwissDec, later called the “remote application”.
- A local area network or WAN connecting the remote machines to the gateway.

The protocol used by the SwissDecTX gateway is extremely simple and entirely file-based: the remote application writes the XML declaration to the network share, and then writes a small text file, called the “command file”, containing the commands to be executed by the gateway.

The SwissDecTX Gateway monitors the share constantly and detects the arrival of new command files.

When such a new file is detected, the gateway software opens the file and executes the command(s) it contains. Typical commands include sending a SwissDec declaration or retrieving the result of a previously sent declaration.

The command files must follow a simple naming scheme to be detectable by the gateway software: the file extension of the command file must be “.cmd”, the same extension used by Windows batch files.

Please note, however, that the command files are *not* real batch files in the sense that the gateway does not run them using cmd.exe, this convention was chosen to make it easier to test the command files as they can actually be run directly from the command-line if necessary, if you add C:\Program Files\SwissDecTX4 (or C:\Program Files (x86)\SwissDecTX4 on a 64-bit machine) to your PATH.

The command-file syntax is as follow:

Blank lines and lines starting with a semicolon character (;) are ignored.

Lines containing commands to be executed must start with the keyword **SwissDecTX**

The keyword is followed by a space character and the Mode into which the command should run.

Valid modes include:

PING	Check connectivity.
INTEROP	Check interoperability.
INTEROP2	Check interoperability with operands.
TX	Transmit declaration / synchronize contract.
STATUS	Retrieve the status of a pending operation.
DATA	Retrieve the data resulting from an operation.
EIV	Create EIV file for manual transmission.
XSLT	Run XSLT transformation (XML -> HTML)

Most of the modes requires additional parameters, for example the INTEROP mode (SwissDec interoperability test) requires the monitoring ID to be specified. The INTEROP mode thus accepts a mandatory parameter, -i <monitoringid> which must be specified.

Let's start with the simplest example: performing a connectivity check (SwissDec “ping” web service command).

The ping command does not send additional data and does not require additional parameters.

The remote application will create a command file containing the command that directs the SwissDecTX Gateway to call the Ping web service endpoint on the SwissDec server.

The name of the command file can be arbitrary, only the file extension must be “.cmd”. For this example, let's assume the remote application creates the command file under the name **ping.cmd**

The content of the command file will be as follow (highlighted text only)

SwissDecTX PING

The SwissDecTX Gateway software will detect the creation of this file and run the command it contains, then will do two important things:

First, the command file itself will be renamed from **ping.cmd** into **ping.end** by the gateway software: this allows the remote application to detect that the command(s) in this particular command file have been carried over.

Since the actual command is carried-on by running the SwissDecTX.exe command-line utility, the second important thing the gateway software does is write the console output of the command(s) to a file named **ping.log**. Note that the name “ping” is only an example for this documentation, you can name the command file however you like, as long as it is a [valid Windows file name](#).

In summary, the remote application creates files with a .cmd extension, containing commands to be carried on. The gateway software monitors the shared folder and detects the creation of *.cmd files. The commands within *.cmd files are executed, and the command files themselves renamed to *.end, while the console output is written to *.log

Since the gateway software launches the SwissDecTX command-line utility in the background to execute the commands, the command syntax is *exactly the same* as the one of the command-line utility.

You can type SwissDecTX from the installation folder to run the SwissDecTX.exe command-line utility, which as built-in help and extensive parameter validation. You can use the -? command-line option to display some help text about each mode, for example SwissDecTX TX -? Will display the parameters required to send a declaration, with a description of the command and examples, you can thus use the SwissDecTX.exe as a discovery and learning tool to get started with the gateway's command files.

Incidentally, the command syntax (lines starting with SwissDecTX) and the command file's name (ending in .cmd) makes it possible to actually execute the command files locally, to test them manually.

The scheme seems simple enough but make no mistake: the gateway software will very reliably monitor the shared folder (and all its subfolders) in real-time and never, ever miss a command file. If the remote application creates multiple command files (there is no limit other than disk space to the number of commands that can be queued) and the gateway computer is shut-down before all commands are processed, all remaining commands (not yet renamed to *.end) will be processed the next time the gateway is started.

The commands themselves, while requiring the SwissDecTX keyword at the start of each line, do not allow to run arbitrary programs, only the <mode> and command-line parameters are actually carried-over to the SwissDecTX.exe command-line utility. Finally, if the command-line utility (SwissDecTX.exe) is modified or replaced, the gateway software will not run any command and will immediately suspend its execution, so the command files do not represent a cheap way to execute arbitrary code on the gateway machine.

Commands can be chained in the same command file. For example a command file with the following content will direct the gateway to perform the two commands, ping and inter-operability check, in sequence:

```
SwissDecTX PING  
SwissDecTX INTEROP -i tester
```

The remote application should test for the presence of the *.end file and can then read the *.log file with the console output.

Let's consider a slightly more complicated example: sending a salary declaration.

To send a salary declaration, the remote application must first write the XML declarations in XML files. Let's assume that the XML declaration exists and that the file name is "Decl.xml"

The **TX** mode (which maps to SwissDec's **DeclareSalary** web service endpoint) requires a number of parameters, namely:

- dec** Salary declaration XML file (input file name)
- res** Result file (output file name)
- eiv** Message sent file (output file name)
- ans** Message received file (output file name)
- job** Job key XML file (output file name)

You can get the list of required parameter by typing **SwissDecTX TX -?** from a command-prompt in the SwissDecTX transmitter's installation folder (of from any command prompt if you added this folder to your PATH, which we recommend to do on dev machines).

We need to pass all five parameters to the command-line utility. Likewise, all five parameters need to be written in the command file.

Let's assume that we want to write the result of the operation in a file called **Res.xml**, then write the message sent file (a signed but unencrypted SOAP envelope, a SwissDec requirement) to a file named **Eiv.xml**, the server's answer to **Ans.xml** and the job key, a token that is used later in the SwissDec workflow to retrieve the deferred result of the declaration, to a file named **Key.xml**. Our complete command-line thus becomes:

```
SwissDecTX TX -dec Decl.xml -res Res.xml -eiv Eiv.xml -ans Ans.xml -job Key.xml
```

Assuming that the Decl.xml file exists in the current folder, the command-line utility will carry the operation and all output files will be written out and contain the expected results and data upon exit of the utility.

In the gateway case, all you have to do is to write the same command into a file with a .cmd extension, say **transmit_test.cmd**, and copy both **Decl.xml**, then the **transmit_test.cmd** file to the network share monitored by the gateway. As in the command-line case, the file names passed on the command-line are relative to the "current folder" i.e. the one where the .cmd file was written to.

It is important to write the command file *last*, as the gateway will begin processing the commands immediately and the data files referred to by the commands, in this particular case the **Decl.xml** file, must already be present and ready to be accessed immediately, or an error will occur.

After the command(s) executed, the gateway software will rename **transmit_test.cmd** into **transmit_test.end**, and the application can then read **transmit_test.log**, **Res.xml**, **Eiv.xml**, **Ans.xml** and **Key.xml**.

The remote application is responsible for the file cleanup: the gateway software cannot know for sure when the application has retrieved the results, as such, all files must be erased by the remote application once the transaction is complete.

Please note that the SwissDecTX transmitter, the SwissDecTX.exe command-line utility and the SwissDecTX Gateway maps the SwissDec 4 workflow exactly. If you are not familiar yet with how SwissDec operates regarding transmission and data retrieval, please read the relevant SwissDec documentation and experiment with the SwissDecTX Test & Configuration tool as well as with the SwissDecTX.exe command-line utility.

Transmitting data over the internet is not instantaneous, as such the remote application does not need to poll for the *.end files too quickly. Ping and interop tests should run relatively fast but for most other operations, one try every few seconds should be enough in most cases.

The only hard conventions regarding the SwissDecTX Gateway are:

- The file extension of the command files must be **.cmd**
- The file extension of a processed command file will be **.end**
- The console output of the command-line tool goes into a file with extension **.log**
- The gateway software sometimes creates temporary files with the extension **.jnl**

The .jnl (“journal”) files are used by the gateway to keep track of the progress in multi-commands command files. This file enable the continuation of the command batch in case it was interrupted, say because the gateway computer lost power or was restarted to install updates etc. Upon resuming, the gateway will restart and only run unprocessed commands within a given multi-commands command file that was already partially processed. The .jnl files are automatically removed by the gateway software when a particular command batch (a .cmd file) has completed execution.

Other than the above, you are free to organize and name the files as you see fit. In particular, you can create subfolders within the share if you wish, or any naming scheme that suits your needs.

Some suggestions:

Name the command-file after the declaration ID if you generate those yourself, and prefix all the input and output files with the declaration ID. For example, say your declaration ID is ABCDEF0123456789.

You *could* name your files as follow:

```
ABCDEF0123456789_Declaration.xml
ABCDEF0123456789_Result.xml
ABCDEF0123456789_EIV.xml
ABCDEF0123456789_Answer.xml
ABCDEF0123456789_JobKey.xml
ABCDEF0123456789.cmd
```

This way, since your application obviously knows the declaration ID it generated itself, you don’t have anything else to remember (besides the fact that a transaction is in progress). The application can easily know what *.end file to look for, and the names of all the various files can easily be inferred. Since the declaration IDs are supposedly unique, you won’t run into name conflicts and you can queue as many declarations as needed if for some reason you need to perform a mass transfer.

Another suggestion:

Create a unique subfolder within the share, say named after the declaration ID as above:

Assuming the network share’s name is SwissDecTX and the Gateway computer’s name is GATEWAY, create a folder whose path is [\\GATEWAY\SwissDecTX\ABCDEF0123456789\](#)

Since the folder name will be unique, use fixed names for all files within the folder, such as *for example*:

```
...\ABCDEF0123456789\Declaration.xml
...\ABCDEF0123456789\Result.xml
...\ABCDEF0123456789\EIV.xml
...\ABCDEF0123456789\Answer.xml
...\ABCDEF0123456789\JobKey.xml
...\ABCDEF0123456789\Commands.cmd
```

If you choose to create a subfolder to write both the data and the associated command file, you don't need to pass the subfolder name in the command-line parameters: the current directory, for a given command set in a given .cmd file, is always assumed to be the one where the .cmd file was written into.

One good (and very scalable) technique to name subfolders uniquely is to create a GUID/UUID and use it as folder name. GUID/UUIDs can be created easily, for example on Windows you can use the [CoCreateGuid\(\)](#) or [UuidCreate\(\)](#) functions, followed by a call to [UuidToString\(\)](#) to create a really unique string which can be used to name the folder. From .NET, `Guid.NewGuid().ToString("N")` will do the trick. The main advantage is you don't need to worry about uniqueness: any number of GUIDs/UUIDs can be created independently without any risk of collision and without the need to centralize anything such as when using an auto-increment column or some similar scheme. UUIDs can be created from any environment, Java, Ruby, PHP, .NET, SAP and many others. See the [Wikipedia page describing UUIDs](#)¹ for more information.

Of course, you can ignore the above suggestions and come up with an entirely different scheme if you wish; the only invariants you need to live with are the .cmd, .end and .log extensions.

Remember to always write the .cmd file *last*. Preferably, create it somewhere else then just copy it, or create it in the network share but under a different name, for example *.tmp, if for some reason the command file will take more than a couple of second to get written. When the file is ready, close it then rename it from *.tmp to *.cmd, so when the gateway software detects it (which is instantaneous) and the file itself and all its dependencies – like the declaration to be sent – are already available and readable. If you create the .cmd file directly on the network share, be quick! We advise you to just copy it to the share, after the related data files, as this is the safer option.

As long as you come up with a unique naming scheme, either unique individual file naming or unique subfolder naming, you can queue as many concurrent operations as you want, even thousands of them. The only practical limit is the disk space on the network share. The gateway software does not have any hard limit on the number of command files it can detect and queue – i.e. thousands is perfectly OK – and all the files will be processed as fast as possible.

You can also create command files that send multiple declarations. Say your application creates N declarations, Decl1.xml, Decl2.xml, ..., DeclN.xml and wants to send them all sequentially.

You could just as well create N command files, like send1.cmd, send2.cmd etc, each with a single TX command like `SwissDecTX TX -dec Decl1.xml -res Res1.xml ...` etc inside, or create a single command file that contains one line per declaration:

```
SwissDecTX TX -dec Decl1.xml -res Res1.xml ...
SwissDecTX TX -dec Decl2.xml -res Res2.xml ...
```

¹ http://en.wikipedia.org/wiki/Universally_unique_identifier

.
.
.

SwissDecTX TX -dec DecN.xml -res ResN1.xml ...

In both cases all declarations will be sent of course, but in the second case you have only one .cmd/end file to worry about, and only one .log file to consider. Performance, however, is another matter as the gateway can process multiple command files concurrently.

Multithreading / Multiprocessing

While the commands within a given command-file are guaranteed to be processed sequentially, **separate command files may be processed in parallel by the gateway**. This has implications on the workflow: if your application queues multiple command files *that depend on each other's output* (for example command file N+1 uses the output of command file N) then your application must send command file N first, wait for its completion, *then* send command file N+1 if it uses the output of the previous command, etc. If the application queue command files N and N+1 at the same time, with command file N+1 depending on command file N's output, and the gateway processes them simultaneously, the commands in command file N+1 are almost guaranteed to fail as the processing of command file N+1 will begin long before command file N has completed execution.

For maximum throughput of the gateway, use separate command files, a multi-core CPU with Hyper Threading enabled and enough memory (RAM) to accommodate the handling of several transmissions concurrently (and of course the fastest possible outbound connection speed!)

Checkpoints

After processing each line in a given command file, provided there is more than one, the gateway will store checkpoint data in a small .jnl file to remember what command was just executed. If the gateway is stopped for whatever reason in the middle of a command batch, it will be able to resume and continue with the next unprocessed line. Good reasons for the gateway to be stopped is a computer reboot due to installing updates, a loss of power, some hardware failure etc.

Note that loss of network connectivity is *not* an error that the gateway software itself will notice: all commands will be processed but you will of course get error messages in the -res XML file produced by the SwissDecTX transmitter. Your application must always read all output files to detect errors: the gateway just runs commands, it has no knowledge of salary declarations or even SwissDec and cannot make the difference between a declaration that cannot be sent because the network is down, or because someone is born on February 31, or because its salary is negative. Only the application can interpret the results and decide the course of action, for example retry a few times in case of a network error, or ask the user to correct the data in case the SwissDec server rejected the transmission.

Decoupled operations

The gateway entirely decouples the transmitter from the application: the remote application can create a declaration and its associated command file and just copy them to the share and go on perform some other tasks while the gateway software and the transmitter operates entirely independently. The application only needs to remember what .cmd files (or which subfolders) it created, as it needs to come back to the network share later to check if the command file(s) was or were renamed to .end

Whenever the application wishes to, it comes back and look for the *.end file that indicates that the transaction has completed on the gateway side, and that the results are ready for consumption. It then read the output and acts according to the results found.

From the end-user point of view, the fact that the communication between the application and the transmitter if done by the mean of disk files over a network makes no difference at all.

XML -> Formatted HTML

It should be noted that the SwissDecTX Transmitter has a built-in facility that transforms the XML replies from the server, for example the plausibility, into ready-to-display HTML, with clickable completion links etc, in a format that is fully admissible for SwissDec certification.

The SwissDecTX.exe command-line utility exposes this functionality through the XSLT mode so you could very well create a .cmd file that runs the command-line tool in XSLT mode after you have received the data from a previous call, to transform the XML replies to formatted HTML ready for consumption.

A possible workflow could be as follow:

- 1) Run a TX command to send a declaration to the SwissDec distributor. If this step is successful (you got a job key in return)
- 2) Run a STATUS command with the job key you got in step 1. If you get a ValidityPlausibilityChecking status in the server reply, wait and retry step 2 until you get a valid answer (plausibility)
- 3) Run an XSLT command to transform the XML plausibility to HTML
- 4) Save all output files and erase all files pertaining to that transaction from the network share, including the .end file, and remove the subfolder (if you opted for that solution).

This is exactly the workflow you'd have to do with the command-line tool or with direct API calls to the transmitter using the File2FileXXX family of functions. The only difference is, in the above, "Run a command" means copy the relevant XML input data and .cmd file to the network share, then loop until you see the .end file, instead of "run the command-line tool" or "call the transmitter API".

The general SwissDec workflow is exactly the same, the only real difference is how the transmitter is invoked.

Tip: get familiar with the command-line utility. Several customers have used this tool to implement their SwissDec-certified solution without a single direct API call. Once the transmitter is configured, all the necessary operations can be performed from the command line tool, which uses disk-based input and output files. The command-line tool is essentially a thin wrapper around the File2FileXXX family of function in the transmitter's API. The gateway software essentially extends this functionality by enabling remote access.

Reliability and dependability

The SwissDecTX transmitter itself and its command-line utility were both extensively tested and have been used in production for a long time (nearly five years at the time of this writing). Tens of thousands of production salary declarations have been successfully sent using this transmitter from hundreds of different sites. The gateway software uses the transmitter and command-line utility virtually unmodified, thus it leverages the stability and robustness of those components.

The gateway service itself is entirely new, but relies on proven components to perform its services so the risk of seeing new transmission issues or different behavior though the gateway is virtually non-existent.

Once the basics are mastered, the other operations not covered on this guide (STATUS, DATA, EIV, XSLT) are all straightforward but please feel free to write if you have any question!

DRAFT

Installation of the gateway software (preliminary)

The SwissDecTX Transmitter Gateway is implemented as a Windows Service (aka NT Service) that runs in the background without any user-interface. You can easily configure the gateway service to start automatically when Windows starts.

Since the gateway is still in preliminary form, there is no monitoring facility at this time, and the installation and configuration must be performed manually.

At the moment and for convenience, the gateway service executable (**SwissDecTX.Gateway.exe**) is deployed by the SwissDecTX Transmitter 4.03 installer, and you will find the service .exe in the installation folder, namely C:\Program Files\SwissDecTX4 or C:\Program Files (x86)\SwissDecTX4 on a 64-bit machine.

You need administrative rights on the computer to install the SwissDecTX transmitter and its gateway. Assuming you ran the SwissDecTX.Setup4.exe silent installer already...

...the installation consists in:

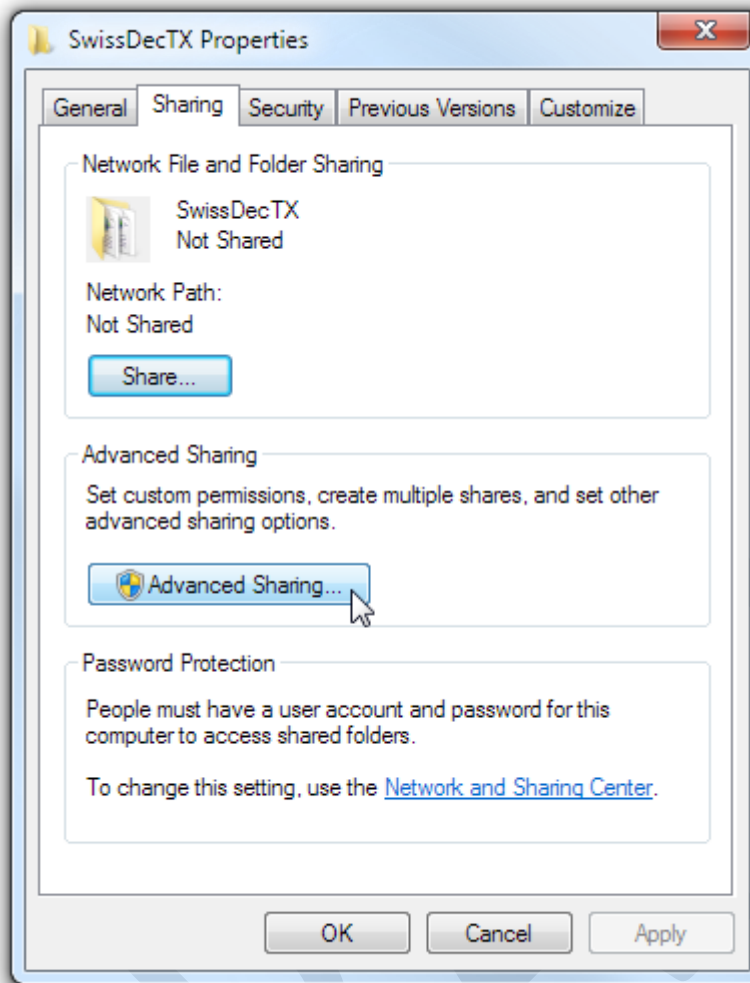
- 1) Create the folder which will be shared on the network
- 2) Actually share the folder
- 3) Create a registry key to tell where this folder is
- 4) Configure the service

Please follow the below steps carefully:

On the computer with internet access where you install the transmitter and its gateway:

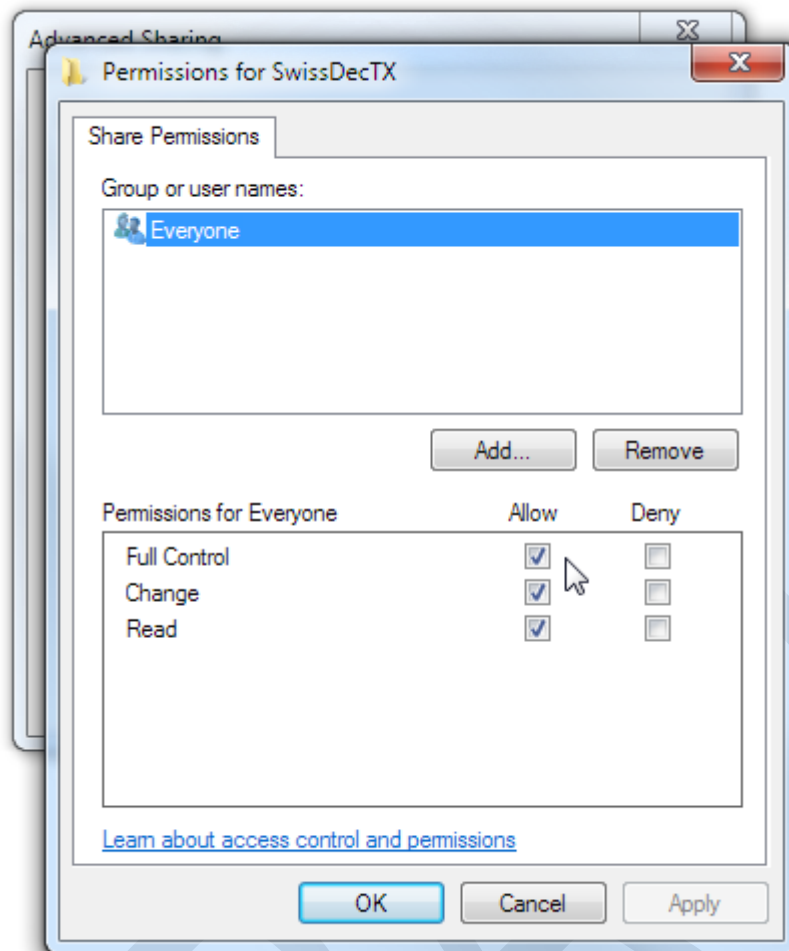
- 1) Login using an account with administrative rights
- 2) Create a folder, for example (and recommended location) **C:\SwissDecTX**
- 3) In Windows Explorer, right-click the **C:\SwissDecTX** folder just created and click Properties in the context menu

- 4) Click the Sharing tab and the Advanced Sharing button

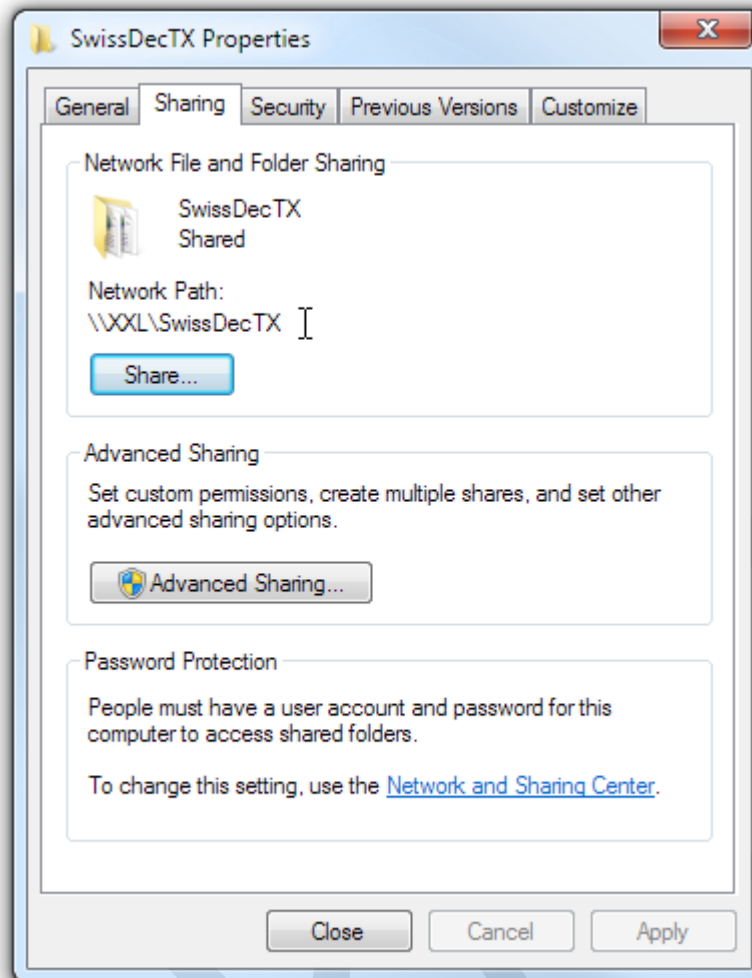


- 5) In the Advanced Sharing dialog, check "Share this folder" at the top, add a meaningful comment, like "SwissDecTX Transmitter Gateway Network Share" then click the Permissions button underneath the comments field.

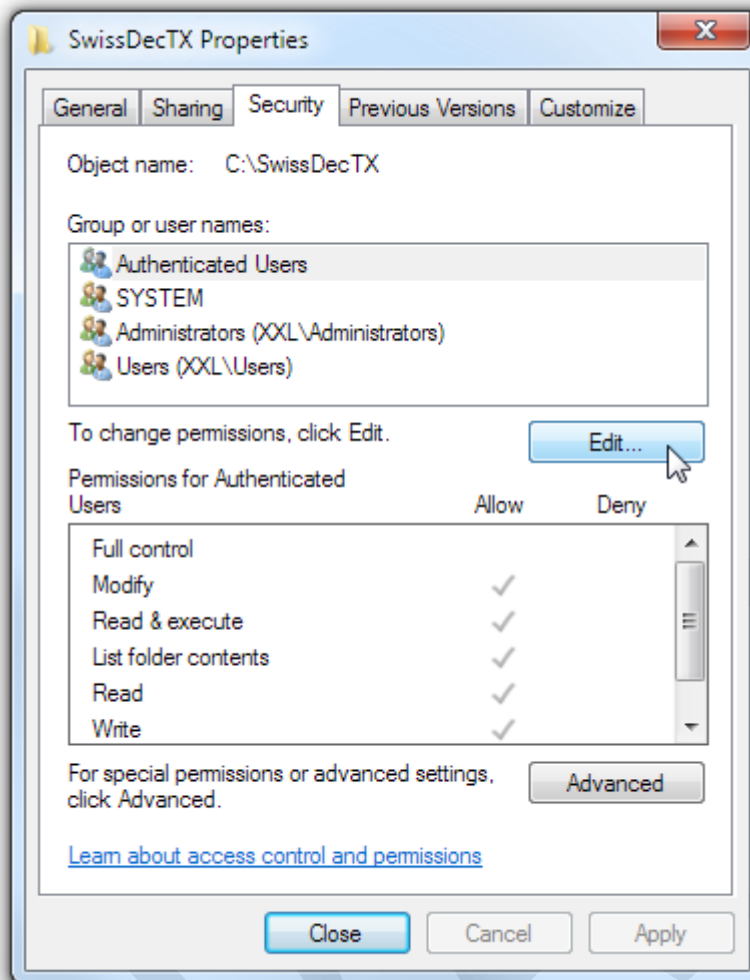
6) In the Permissions dialog, give Full Control to Everyone on the folder:



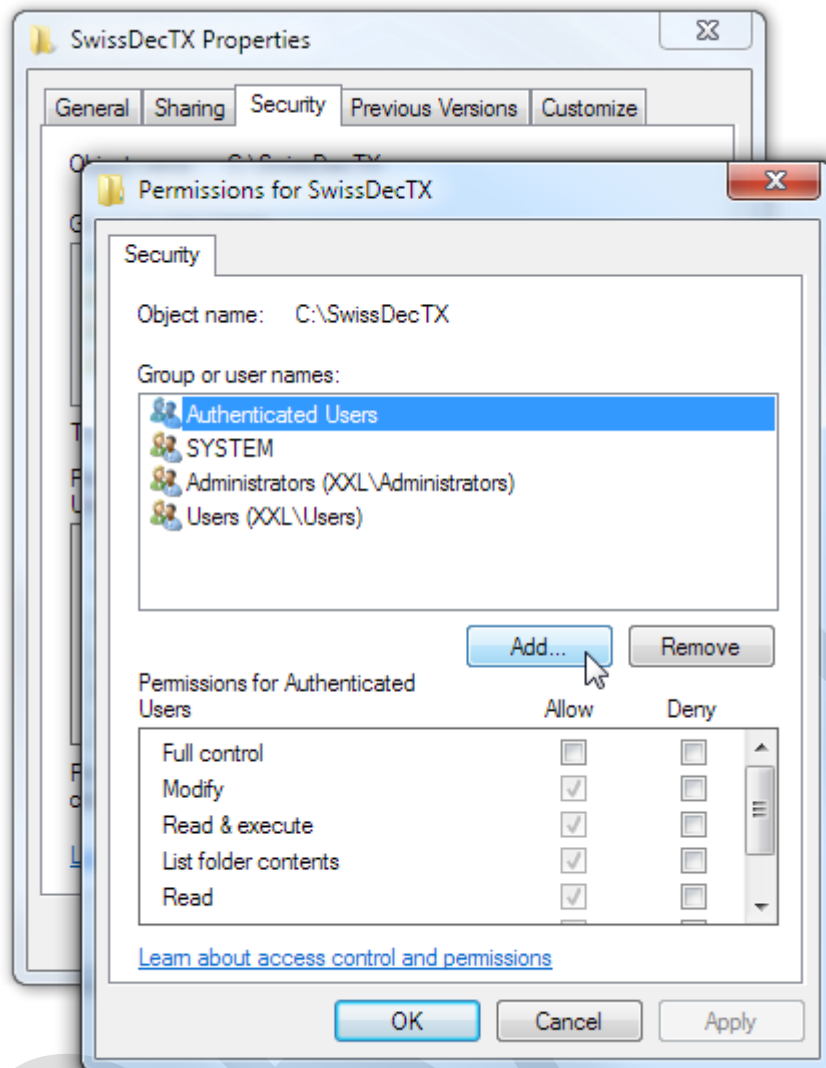
- 7) Click OK to close the permissions dialog, then click OK to close the Advanced Sharing dialog. You should see the Network Path in the Properties dialog. Do **not** close the dialog yet.



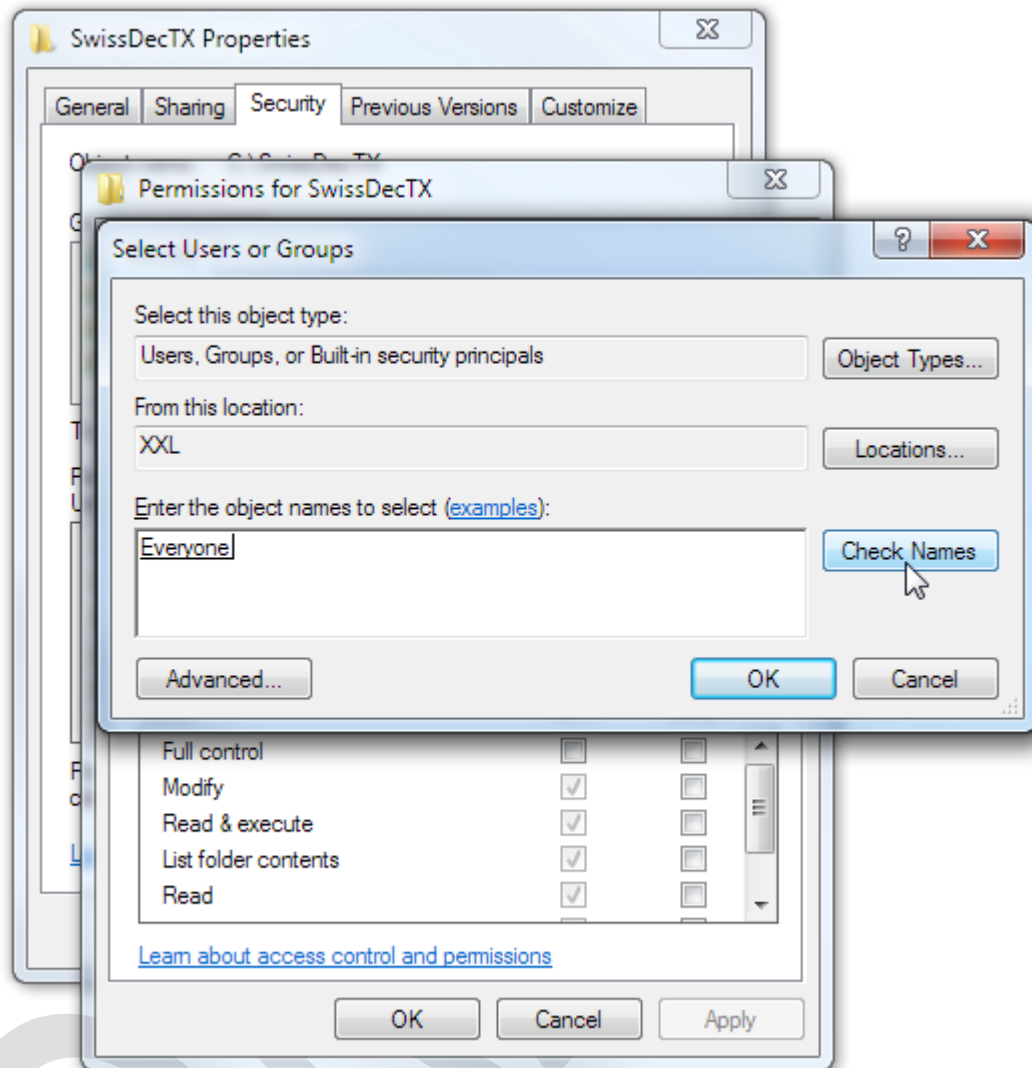
8) Click the Security tab, then click the Edit button:



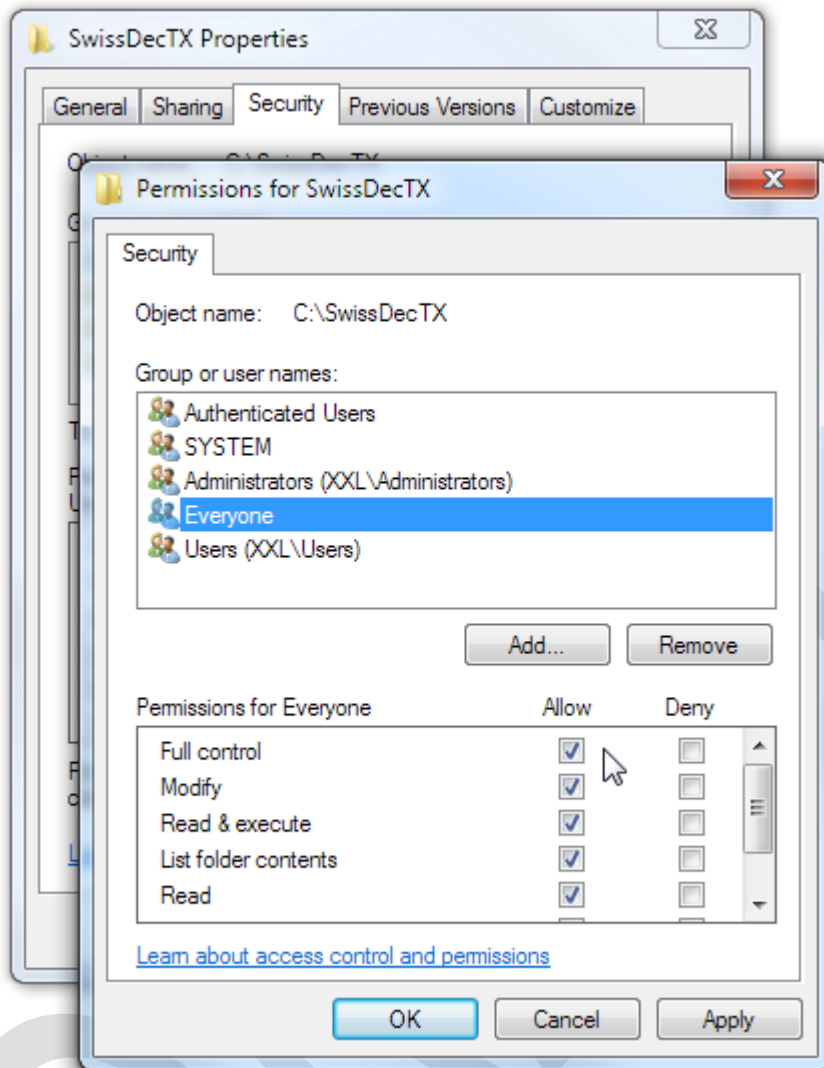
9) In the Permissions for SwissDecTX dialog, click Add...



10) In the Select Users Or Groups dialog, type Everyone in the “Enter the object names to select and click the Check Names button, then click OK to close this dialog:



- 11) Grant Full control to the folder the Everyone group just added, click OK to close the Permissions dialog, then click Close to close the SwissDecTX Properties dialog:



Congratulations: you just created a network share, and gave access to it to everyone, including unauthenticated users. Keep in mind that only the C:\SwissDecTX folder just created is affected by the sharing and security changes just made. Depending on the nature of the remote computers, you may be able to tighten the security on this folder, but for now let's just make things work.

Selecting the input folder

(this section will soon be superseded by GUI controls in the Test & Configuration utility)

The next thing that we need to do is to create a registry key to tell the gateway service where to look for command files. There is no user-interface at the moment to perform this operation as the gateway is still at an early stage, you thus need to enter the value "manually".

There are two options: the first is to create a .reg file with the following content:

For Windows 64-bit:

Windows Registry Editor Version 5.00

[HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\ARSD\SwissDecTX4\Gateway]

"MonitoredFolder"="C:\\SwissDecTX"

For Windows 32-bit

Windows Registry Editor Version 5.00

[HKEY_LOCAL_MACHINE\SOFTWARE\ARSD\SwissDecTX4\Gateway]

"MonitoredFolder"="C:\\SwissDecTX"

Then double-click the .reg file to enter the data.

Alternatively, you can use the registry editor (regedit.exe) and navigate to

HKEY_LOCAL_MACHINE\SOFTWARE (on a 32-bit machine) or

HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node (on a 64-bit machine)

Create a Key named "ARSD"

Within ARSD create a Key named "SwissDecTX4"

Within SwissDecTX4 create a Key named "Gateway"

Within Gateway create a Value (REG_SZ) named "MonitoredFolder" equal to C:\SwissDecTX

At this point you need to install the SwissDecTX 4.03 transmitter if not done already, by running the SwissDecTX.Setup4.exe silent installer program (this is a *nearly*-silent installer designed to be embedded within your own setup, so it only displays a minimal user-interface consisting in a progress dialog and nothing else). **If no error message gets displayed then the transmitter was successfully installed.**

You now need to configure the transmitter and make sure it works. In particular, you need to configure the server's URL, install and select the three SwissDec certificates, select the Demo license, pass the Quick Test and the Inter-operation test on the next tab, as well as successfully send the ICHAGCompany.xml file that comes with the transmitter. Once the transmitter works as expected, proceed with the next steps. All necessary files can be found in the SwissDecTX **3.02** retail package, the current production version that you can find on www.swissdectx.ch

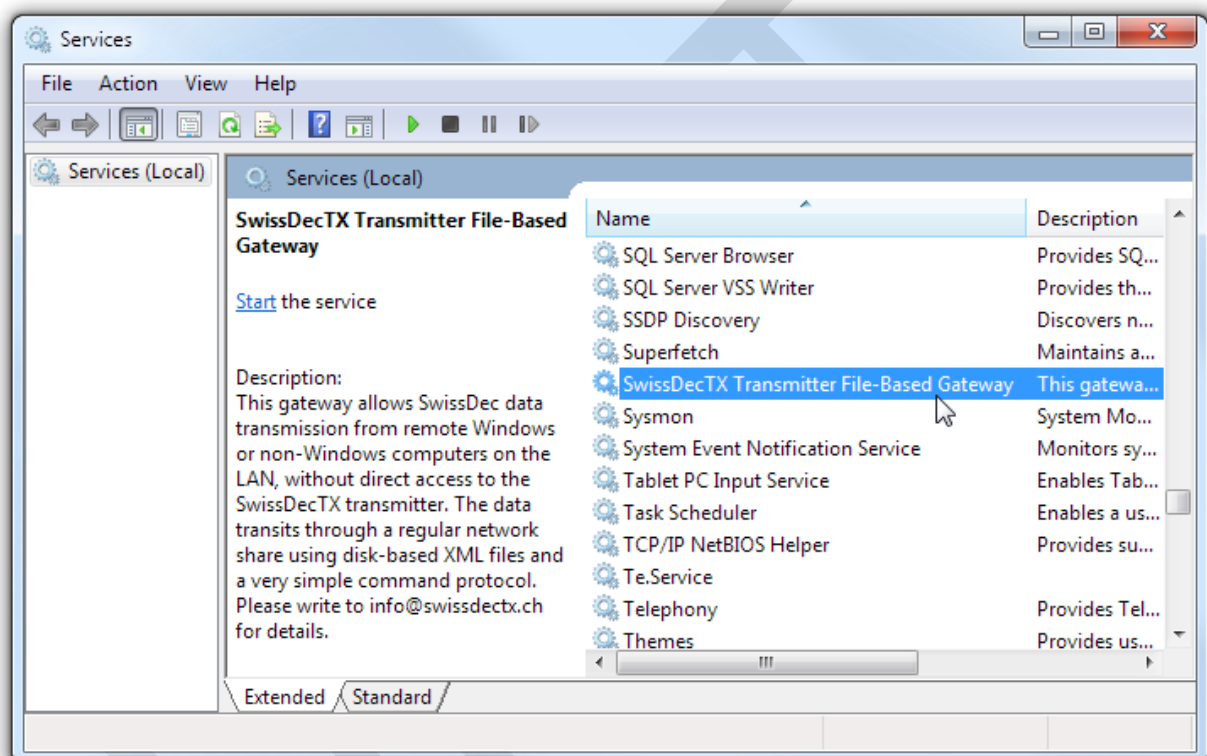
The last step is to configure the SwissDecTX Gateway service.

NT Services normally run in special identities like LocalSystem, LocalService or NetworkServices to be able to perform their duties. In our case, we want the service to have access to the folder we created and also to the SwissDecTX Transmitter 4.03 configuration, which is stored per-user.

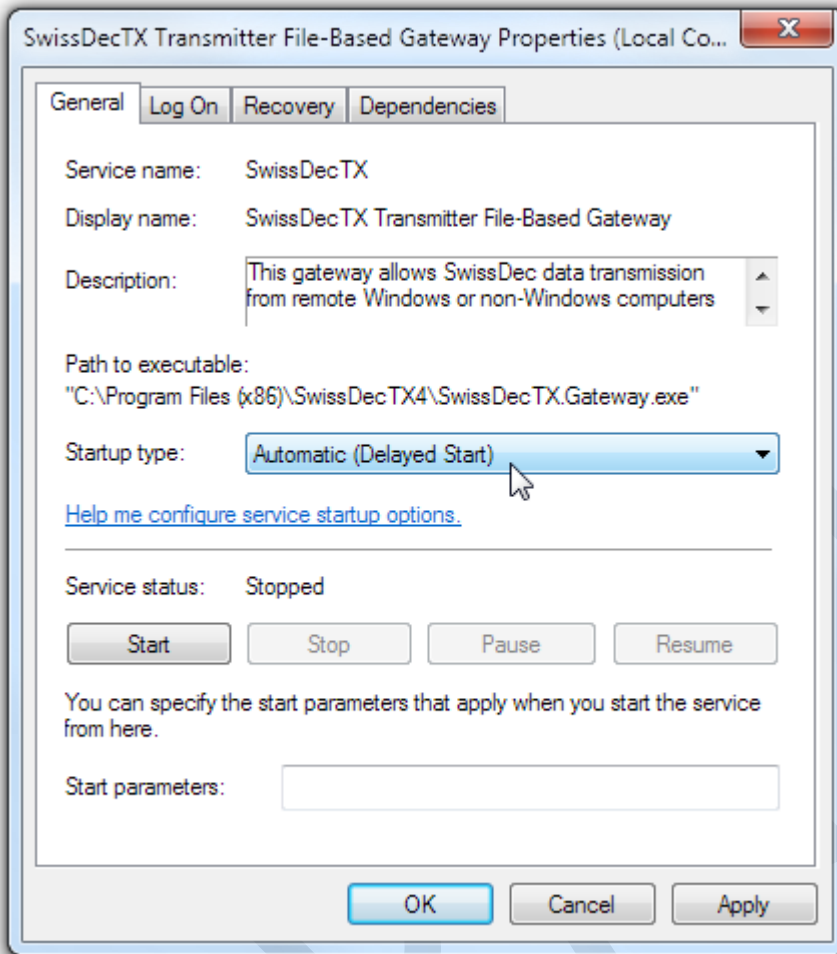
As such, we **must** run the service in the installing user's identity.

Click the Start button and type View local services. In the installed services list, locate

“SwissDecTX Transmitter Gateway”:

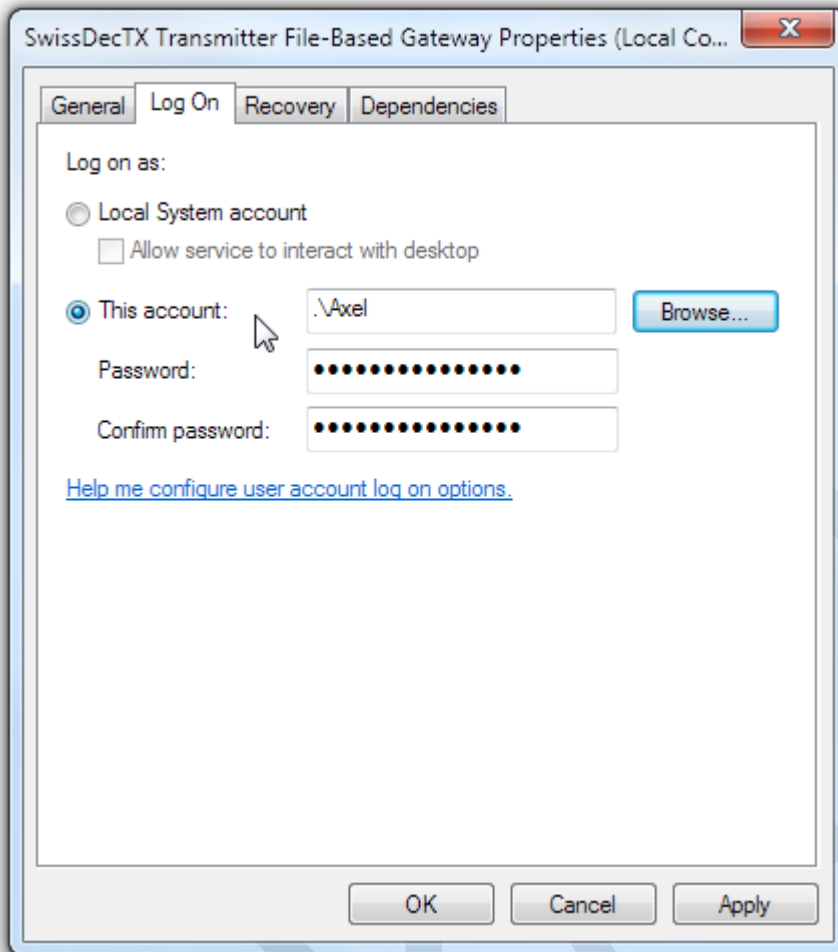


Right-click and click Properties. In the properties dialog, set the Startup Mode to “Automatic (delayed start):



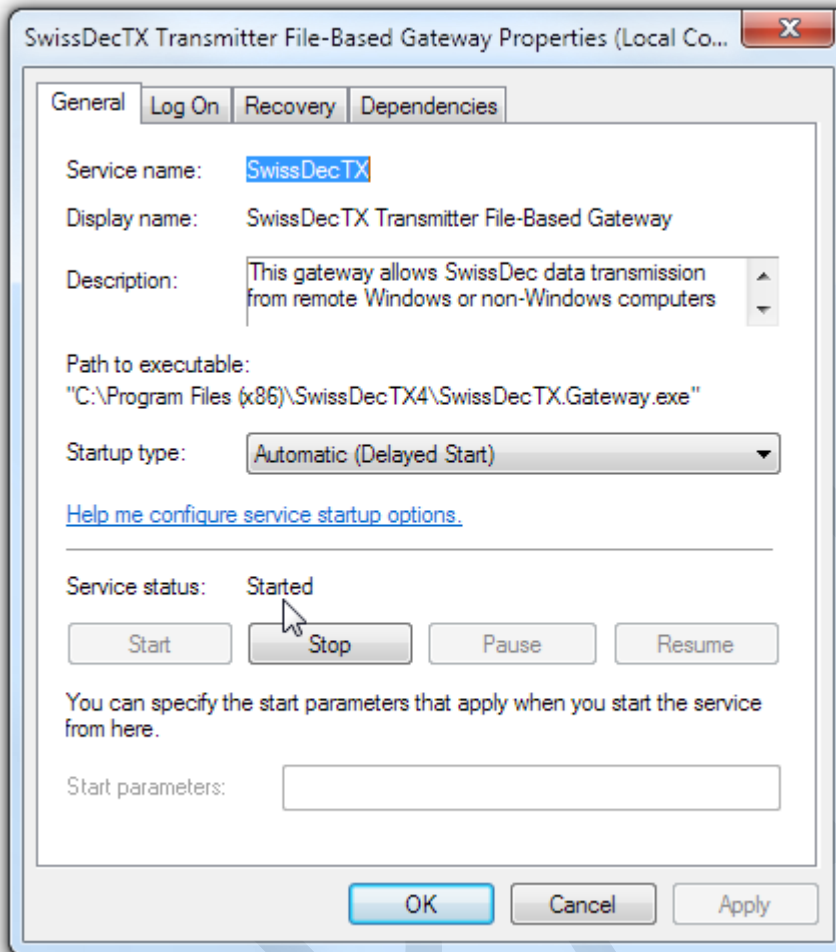
Do **not** click the Start button yet, but press the “Log On” tab instead, then click “This Account” and enter the username/password of the installing user (your’s, that’s it).

Note that this same user must configure the transmitter.



You can optionally set the Recovery options on the Recovery tab, then go back to the General tab, click the **Apply** button to save your changes, *then* click the Start button.

If everything goes well, the service control manager should soon report that the service has started:



You can now test the gateway locally from a command prompt: Go to the C:\SwissDecTX folder which is our “inbox” and run the **dir** command to convince yourself it is empty.

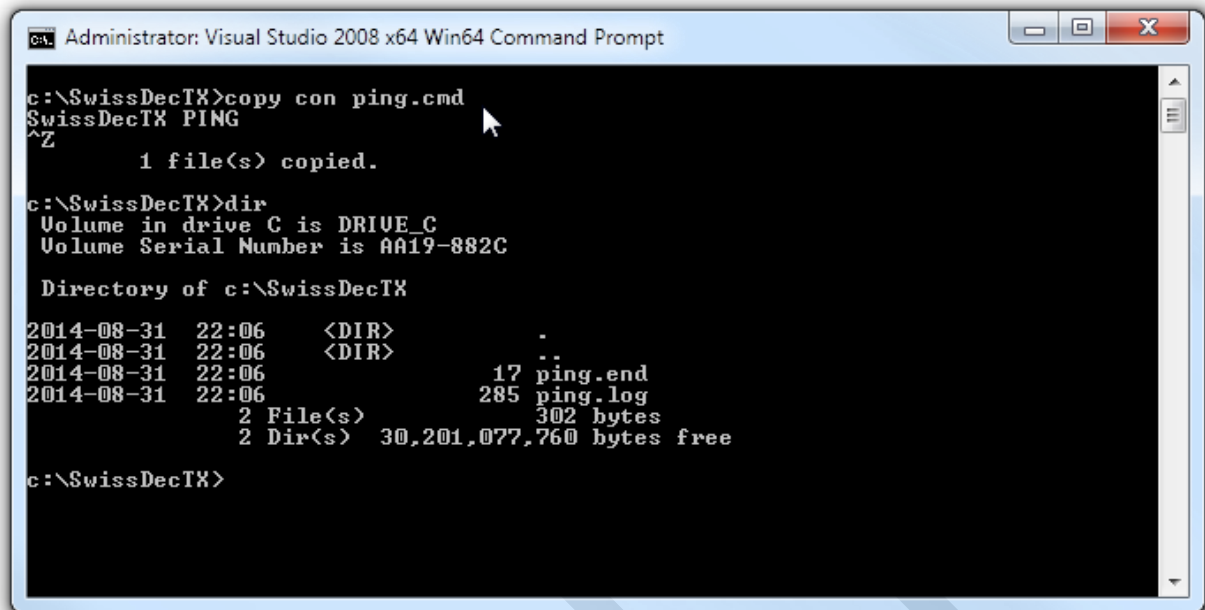
Type the following commands exactly, where <return> means the Enter key and <F6> means the F6 function key:

```
copy con ping.cmd<return>
SwissDecTX PING<return>
<F6><return>
```

You have about 10 seconds, after the first <return>, to type the **SwissDecTX PING** command, hit return again and hit F6 then return once more to close the file, so be quick!

Wait a couple of seconds and type dir again: Your **ping.cmd** file should have been renamed to **ping.end** and a new file **ping.log** should have been created, with the console output of the command-line tool. During the call to the SwissDec web service, the .log file will exist but the command file will still have its .cmd extension. The gateway renames the file only after the command has executed, independently of the outcome of the command, and flushes and closes the

.log file so you can access it the see the console output of the command-line utility that was launched in the background.



```
Administrator: Visual Studio 2008 x64 Win64 Command Prompt
c:\SwissDecTX>copy con ping.cmd
SwissDecTX PING
^Z
        1 file(s) copied.
c:\SwissDecTX>dir
Volume in drive C is DRIVE_C
Volume Serial Number is AA19-882C

Directory of c:\SwissDecTX

2014-08-31  22:06    <DIR>          .
2014-08-31  22:06    <DIR>          ..
2014-08-31  22:06                17 ping.end
2014-08-31  22:06                285 ping.log
                2 File(s)          302 bytes
                2 Dir(s)  30,201,077,760 bytes free

c:\SwissDecTX>
```

Congratulations !

Now you can connect to the [\\YOURCOMPUTER\SwissDecTX](#) share from anywhere on the LAN/WAN and start remotely writing data files and command files to use the SwissDecTX transmitter remotely!

Note: the above tutorial was made on Windows 7. The screens may vary slightly on Windows Vista or Windows 8/8.1 but the idea is the same: **after** having successfully installed, configured and tested the SwissDecTX Transmitter 4.03...

...create a folder somewhere, preferably `C:\SwissDecTX`, give Everyone full access to it at the file system level, share it, give Everyone full access to the network share, create the registry key that tells the gateway where the folder is, configure the gateway to start automatically **and under the identity of the installing user** and, finally, start the gateway and you are ready to test it.

64-bit computing

The SwissDecTX Transmitter itself is a C# .NET component and is architecture agnostic: it will run as a 32-bit component if invoked from a 32-bit process, and as a 64-bit component if instantiated from a 64-bit process (that's the magic of .NET just-in-time compilation).

The SwissDecTX Gateway and the SwissDecTX command-line tool are native components written in C++ and ship in 32-bit and 64-bit flavor within the SwissDecTX.Setup4.exe installer. On 64-bit operating systems, the SwissDecTX installer will deploy native 64-bit components. 32-bit software can still instantiate the transmitter as before and/or run the 64-bit command-line utility without noticing a difference. The gateway service will run natively in 64-bit in the background. On 32-bit computers only 32-bit components are deployed.

Is 64-bit really needed? Well, it depends: if you are using the gateway in a low to medium transmission volume environment then the answer is probably no. In a high volume, high throughput enterprise environment, the answer is a definite yes!

Performance and scalability

The SwissDecTX Gateway is designed for maximum performance. It will take advantage of multiple cores and Hyper Threading automatically to process multiple command files concurrently. Given adequate CPU and RAM resources, the actual throughput of the gateway on a 64-bit computer is most likely to be limited by the computer's network I/O abilities, i.e. the ability of remote machines to write and read from the network share, and the ability of the gateway to upload to, and download from, the SwissDec servers over the internet connection.

Advanced registry settings

The Gateway supports a few additional registry entries as follow. These are for advance use only, none of them is required for typical installations of the SwissDecTX Gateway.

DisableNtfsTransactions	REG_DWORD	Set to 1 to disable the use of NTFS transactions. Use to troubleshoot issues with some networking software, when the monitored folder is a remote share from the point of view of the Gateway software itself, and the server OS on the remote share is not Windows. Note that, while possible, is it not recommended to monitor a remote folder: make sure the monitored folder and the transmitter are on the same computer.
MaxConcurrency and TargetOversubscriptionFactor	REG_DWORD	Those parameters control the level of concurrency inside the Gateway. The default values should do just fine for typical installations, but you could use them for, say, restrict the Gateway to only 1 CPU core, for example. See Concurrency::Scheduler on MSDN for details.
DisableParallelCommandFilesProcessing	REG_DWORD	Set to 1 to process command files sequentially. Use for debugging or if the destination server complains that you send too many concurrent requests.

Troubleshooting:

The gateway does not appear to work:

- Verify that the transmitter is working normally (properly configured, access to the internet...) using the **SwissDecTX Test & Configuration** tool and the **SwissDecTX.exe** command-line utility.
- Verify that the gateway service is running (autostart), and started **under the identity of the user who installed and configured the transmitter**. **This is super-important.**
- Verify that the user account under which the gateway service is running has full read/write/delete (full control) access to the designated folder.
- Verify that the registry key that tells the folder's path is at the right place and points to the correct folder (*soon to be superseded by a GUI interface*).

The gateway works for PING and INTEROP commands, but you get a **"SwissDecTX: A valid license is required to perform this operation!"** message when trying to send a declaration using the TX and the supplied SwissDecTX.Demo.lic:

- The gateway service is not running in the identity of the current user and does not find the configuration parameters. See point 2 above (the super important one!)
- You are not actually using the Demo license (if you are an existing customer using your production license).

The gateway appears to react slowly when playing with command files from Windows Explorer:

- *Explorer* is sometimes slow to pick up the changes, just press F5 to refresh the view ;-)

While playing with multiple simultaneous command files to test the gateway, you got the following error: **"SwissDecTX: Logging exception! (Access to the path 'C:\ProgramData\SwissDecTX4\TransmissionLog\ etc etc .zip' is denied.)"**:

- You are using the transmitter's built-in logging facility and you ran into a name conflict on the log because you sent the exact same declaration multiple time from different command files at once, and the declaration has a fixed ID in the <ct:RequestID> tag. Since the gateway processes command files in parallel up to the number of CPU cores, it is possible that two or more of your commands got started in the same millisecond, resulting in duplicate log name entries as the request ID and transmission time are both used to make the log files unique. Set the request ID to '*' in your test declaration (<ct:RequestID>*</ct:RequestID>) so the transmitter generates a unique request ID on-the-fly for you every time, or use different declarations, each with their own unique ID **as it should be in actual transmissions** (you should never use the same request ID twice for any reason, as per SwissDec specs).

While playing with multiple simultaneous command files to test the gateway, you got the following error: **"The process cannot access the file because it is being used by another process."**

- You used the same output file names in one or more of the parameters of several command-files. Since command files are processed in parallel, the transmitter ran into conflicts because of name collisions in the same folder. Please make sure to use either a *unique* subfolder name for each command file and associated data, or to use *unique* declaration file names and *unique* output file names, if you put multiple data files and command files in the same folder. Please refer to the suggested naming schemes earlier in this document.

You can of course get support at info@swissdectx.ch

Near future

The final version will *probably* be packaged in a separate installer, and the transmitter's Test & Configuration tool will be augmented with a new tab where the installing / managing user will be able to control the gateway (set the account, start/stop it, change the folder, and get some statistics like the number of command files processed, number errors and the like).

Keep in mind that the gateway software is still preliminary at this stage: if you encounter a bug, report it – if possible with clear repro steps - and it will be fixed.

Some remarks on the SwissDec workflow

The SwissDec workflow requires to get data from one call (for example the job key) and use it in subsequent calls to retrieve further data. Sometimes the server does not reply readily with a final answer but instead indicates that the data is still being worked on (for example ValidityPlausibilityChecking). The application must wait and retry and the retry policy is entirely in the hands of the application as the gateway is stateless and (deliberately) ignorant: once a particular command file and the command(s) it contains has completed, the gateway renames the command file and completely forgets about it. In particular the gateway does not retry on your behalf, this is the application responsibility to decide if retrying is needed or appropriate, and what the retry interval should be. In the long run, this is the better choice: the gateway is kept simple (read: bug free!) and your application stays in control of the high-level workflow.

In some cases, the results can take up to 15 *days* to get prepared by the server, so the entire SwissDec workflow is slow-paced: send something, come back the next day to see if it was accepted, wait for the answer that should come back on the expected *date* specified by the server.

Of course the transmitter gives you immediate feedback on the technical aspects: was the XML file correctly formatted? Was the file successfully uploaded? But the server-side *processing* of the data can take minutes, hours or days depending on the cases.

Gateway management and monitoring

The gateway still being in development, there is no management or monitoring mechanism at this point (planned for inclusion into the existing SwissDecTX Test & Configuration tool). The gateway being a Windows Service, it can be managed (started and stopped) using the Windows Service Control Manager (SCM): the display name of the service is "SwissDecTX Transmitter Gateway" and the internal service name is "SwissDecTX", so it can be managed like any other service from a command prompt:

```
net start SwissDecTX
```

and

```
net stop SwissDecTX
```

will start and stop the gateway, respectively. The [Service Control Manager](http://msdn.microsoft.com/en-us/library/windows/desktop/ms685150(v=vs.85).aspx)² can also be invoked programmatically and Windows Services can be managed using [Windows Management Instrumentation](http://msdn.microsoft.com/en-us/library/aa394602(v=vs.85).aspx)³ (WMI) so the SwissDecTX Gateway *is* manageable, just not by its own tooling yet.

² [http://msdn.microsoft.com/en-us/library/windows/desktop/ms685150\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/ms685150(v=vs.85).aspx)

³ [http://msdn.microsoft.com/en-us/library/aa394602\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/aa394602(v=vs.85).aspx)