TystofteFonden

System to System communication

Seed Certification System Tystoftefonden

Table of Contents

Authors, revisions, warnings, etc.

Created by: Alexander Balsam, nedeco, 2017-04-06
Rewritten by: Henrik Bønnerup, Tystoftefonden, 2017-04-19
Revised by: Henrik Bønnerup, Tystoftefonden, 2017-04-20
Revised by: Henrik Bønnerup, Tystoftefonden, 2017-04-27
Revised by: Henrik Bønnerup, Tystoftefonden, 2017-05-19
Revised by: Henrik Bønnerup, Tystoftefonden, 2017-05-26
Revised by: Henrik Bønnerup, Tystoftefonden, 2017-06-14
Revised by: Henrik Bønnerup, Tystoftefonden, 2017-06-26
Revised by: Henrik Bønnerup, Tystoftefonden, 2018-01-10
Revised by: Henrik Bønnerup, Tystoftefonden, 2018-03-26
Revised by: Henrik Bønnerup, Tystoftefonden, 2018-04-16
Last revision: 2018-04-16, rev. 1.4

Scope

The seed certification system that was operated by the ministry of agriculture and fishery until now will be transferred to the newly founded Tystoftefonden.

As the seed certification system (called FCS) is a quite complex system that does contain over 15 single services all services have to be reviewed while being migrated to Tystoftefonden. These review process does not only include to migrate the services but also has to make sure that components and dependencies that are either no longer of any use have to be removed or that cannot be used as they are have to be replaced. One of these components is the NAER SIKKER authentication used for the system-to-system transfers.

NAER SIKKER was introduced into FCS „system to system“ 01. January 2015 and is based on the Governmental Authentication System. As only Governmental Institutions are able to validate signed request against the NAER system there will be no possibility for Tystoftefonden to use the NAER system.

As shown in the picture on next page, in the old system customers generated XML-files for four different kind of objects: *AutoriseredeProeveResultater*, *AvlsParti*, *Form300* and *Markbesigtigelse*. These files were then transferred to the FCS system either via a small java application that was provided by the ministry (FCS-Secure-Client) or via a custom implementation on the customer side.

In the background the transfer from the customer to the ministry used NAER SIKKER to authenticate the customer towards the FCS and afterwards transferred the content of the XML-files via the SOAP protocol. The FCS-Cert service on the ministry side received the content of the XML-files and did a very basic verification. Then the FCS-Cert service stored those XML-files in a special “transfer” directory from where the FCS-Basis would fetch the files for further processing.

As the FCS-Cert service only made a basic validation, based on existence of keys in the XML-files, and did not validate any values, the feedback back to the customer was very limited.

The migration of the FCS System to Tystoftefonden introduced the need to remove the NAER SIKKER authentication as Tystoftefonden is not a Governmental Institution.

As some the customers of Tystoftefonden spent quite some effort and money to implement the NAER SIKKER authentication just two years ago, the planned new transfer has focused on limiting the effort on the customers side.

The new FCS-Client will hence, be a “compatible” replacement of the existing NAER-sikker Client and, as an addition to this, the possibility of using the standard protocol sFTP as an alternative for uploading files has been added.

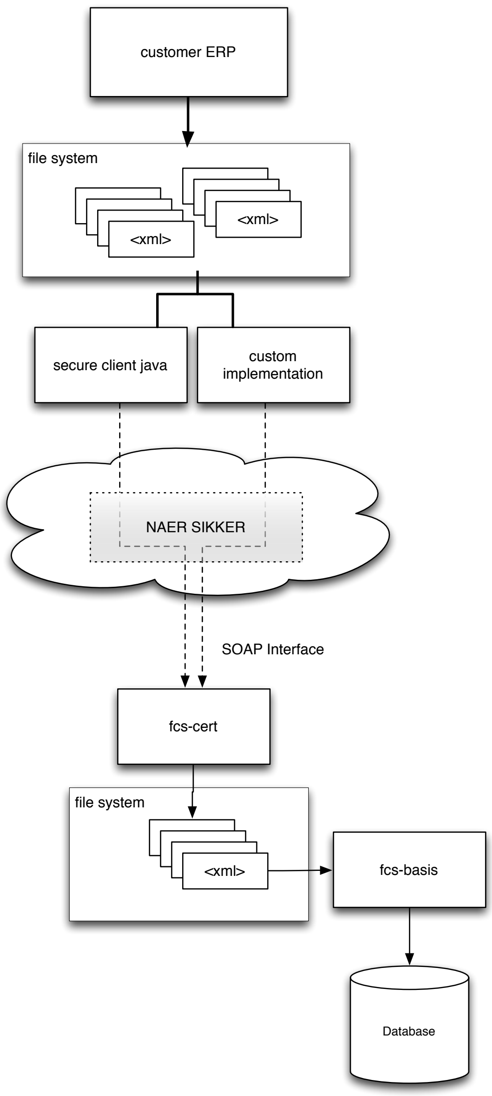
Please Note: For the FCS-Client you’ll need installation (if not already existing) of the Java Cryptography Extension (JCE) or authentication will likely fail.

Customers, with the wish for doing system-to-system transfers will get all needed information on User credentials and IP-address for the remote server, acting as an end-point for upload of XML-files. In the same manner Tystoftefonden will provide the FCS-Client upon request. For sFTP-transfers the Customer can choose from a large variety of “standard” clients. An example on using the OpenSSH sFTP version is given later in this document.

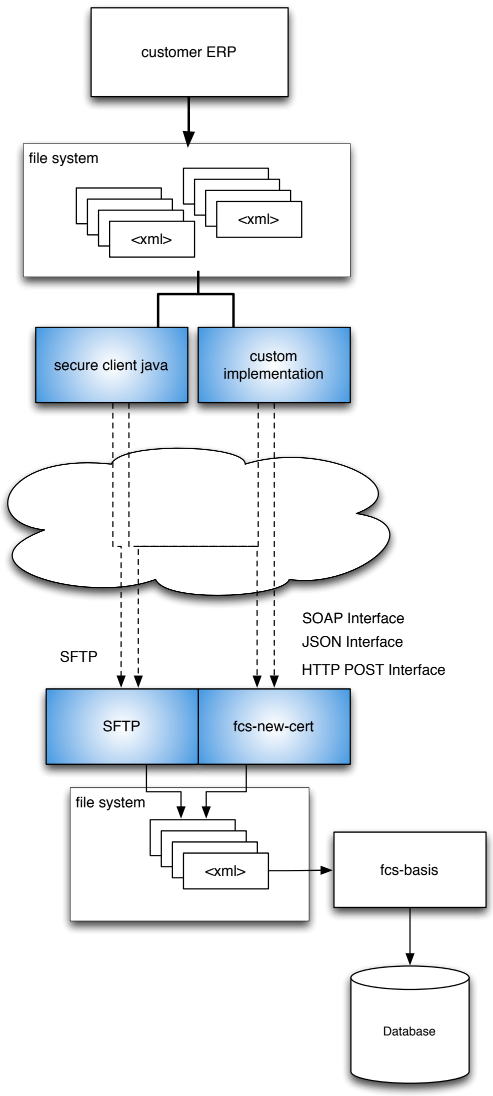
Please Note: Only upload of XML-files adhering to the XML-templates will be accepted by the FCS-system.

The schematic drawing, shown below, shows the planned changes to the system and introduces various new possibilities in the transferral of the XML-files to Tystoftefonden.

old system



new system



General Provisions

IP-address for the Upload-Server

Currently the DNS-entry for the upload-server at Tystofte is: api.tystofte.dk

Please Note: The DNS-entry may change in the future, but *only* with prior notification sent by e-mail to the users of the upload-server.

User Credentials for the Upload-Server

All users of system-to-system transfers will receive necessary User Credentials (login & password) allowing them to use the service.

Directory for uploading of files

With the recent implementation of the upload structure, you'll need to use the upload directory of: **/ transfer** in your default configuration.

Please Note: The actual files will be "rotated" (i.e. moved for external storage purposed) in a 14 day cycle. This means that only files from the last days will be available.

WEB page for viewing status of the uploaded files

There is a possibility of getting af view on the current status of the uploaded files by visiting: https:/ api.tystofte.dk

You'll need the same user credentials for logging in as you've received for the transfer of files.

XML-file Descriptions

The allowed and possible XML-file descriptions and types are contained in another Document (*XML-XSD-Descriptions System-to-System.pdf*), available at our WEB-site.

Flow of information

Handling of files and errors, receiving XML-files in FCS

Whenever the application starts processing of the individual *.xml* file it'll be renamed to *.running*. The **result** of the processing (of the individual file) is then saved to a new file with an identical filename, but now with the extension *.out* and the **original** file (*.xml*) will be renamed to *.done*
Please Note: The .out file can contain a "negative" answer upon processing.

In case the transfer and processing of the uploaded xml-file be successful (*from a technical point of view*) the .out file will have a content similar to whats shown below:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?><ns2:IndberetningResponse
xmlns:ns2="http://pdir.dk/fcs/ws/indberetning"><status>Complete</status></
ns2:IndberetningResponse>
```

Note the status of "Complete", which is identical to what would be shown on the WEB-page for your account on api.tystofte.dk

The status "Complete" is (currently) *only* an indication of whether the XML-fil could be parsed and only incorporates a *very limited* set of checks for the actual content.
Hence, it's (unfortunately) *merely* a test for syntax of the XML-file and care should be taken when interpreting "Complete" as a "valid content" of the file.
This will be changed in the future.

Should the .xml files are processed and there is a communication-, application- or xml-error it can be identified by one of the following options:
A *.err* file exist
A *.running* file exist (after the application has stopped running)
A *.out* file contains a status with an error message.

An example of this would be (again look for the <status>xxx</status> tag):

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?><ns2:IndberetningResponse
xmlns:ns2="http://pdir.dk/fcs/ws/indberetning"><status>Element
'{urn:dk:naturerhverv:fcs:indberetning}AutoriseredeProeveResultater': No matching global
declaration available for the validation root.</status></ns2:IndberetningResponse>
```

Using the FCS-Client

Installation

The Customer must ensure the Java Runtime Environment (JRE version 7) and Java Cryptography Extension (JCE) has been properly installed on the server doing the uploads to FCS.

The FCS-Client is provided, as a *jar-file*, by Tystoftefonden and can be installed based on user requirements e.g. in /FCS/java-client/build/libs (on Windows <Drive:>\FCS\java-client\build\libs)

Make sure you have proper permissions for running the FCS-Client in your environment.

Running

Enter a terminal-window (shell) and run the command:
java -jar build/libs/fcs-upload-client-x.y.z.jar <file|dir>

Apart from the jar file a properties file must exist in the same directory as the jar file:

```
### fcs-client.properties
user=fcsapi
password=fcssecretpassword
host=api.tystofte.dk
```

If a file is specified <*file*>, only the named file will be uploaded. The file directive may (should) include any valid path specification e.g. C:\FCS\files-for-upload\<filename>

If a directory is specified <dir>, all files in that directory are considered.

The FCS-Client will exit with the following exit codes:
- Exit code \`0\` is used if all files were uploaded successfully.
- Exit code \`1\` is used if an error occurred.
- Exit code \`2\` is used if no \`.xml\` files were specified or found in the given directory.

Note: Only files ending with extension of \`.xml\` will be uploaded and renamed to \`.done\` on success.

Scheduling of upload(s)

As with the NAER Sikker Client, all wanted scheduling of uploads is based on implementation of proper calls to the FCS-Client for the given operating system.

Example

```
2 directories have been defined:
/Users/grumpy/Desktop/FCS/JAVA-Client          <—— For java-client AND properties-file
/Users/grumpy/Desktop/FCS/upload-directory     <—— Where the xml-files are placed

In the directory: /Users/grumpy/Desktop/FCS/JAVA-Client, run the jar-file
Grumpys-Mac:JAVA-Client grumpy$ java -jar fcs-upload-client-0.0.1.jar /Users/grumpy/Desktop/FCS/upload-directory/
Launching Tystofte FCS-Client version 0.0.1
Connecting to api.tystofte.dk ... connected!
Uploading file /Users/grumpy/Desktop/FCS/upload-directory/<xml-file(s)>.xml ... done!
Exit code should be 0 (zero)
version 1.0
Now check if the file(s) have been renamed to “.done”
Grumpys-Mac:JAVA-Client grumpy$ ls -lart ../upload-directory/
-rw-r—r-- 1 grumpy staff 0 Jun 26 17:49 <xml-file(s)>.xml.done
```

Using “native” sFTP for uploads (not implementing the FCS-Client)

sFTP is an interactive file transfer program, similar to ftp, which performs all operations over an encrypted ssh transport. It may also use many features of ssh, such as public key authentication and compression. sFTP connects and logs into the specified host, then enters an interactive command mode -or- by correct configuration, enters a batch processing mode.

This example is based on the OpenSSH version of sFTP (another possibility could be PuTTY).

For Microsoft Windows it can be installed by following instructions from the following link:
https://github.com/PowerShell/Win32-OpenSSH/wiki/Install-Win32-OpenSSH

For Linux, please use the following link: *https://www.openssh.com*

Note: Ensure your firewall(s) allow sFTP (port 22) to the remote host.

Full description of the sFTP (client) can be found on: *http://man.openbsd.org/sftp.1*

Setting up a connection for the first time (DEPRECATED, use sFTP in the same way if your client supports it)

The first time connecting to a remote host, the key itself should be verified. Usually this is done by comparing the fingerprint or the ASCII art visual host key, metadata about the key, rather than trying to compare the whole key itself.

Note: If installed using instructions including ssh-add for adding keys, this step has already been prepared.

```
$ ssh -l <login> <FCS-server>
The authenticity of host ‘FCS-server (<some-IP>)’ can't be established.
ECDSA key fingerprint is SHA256:LPFiMYrrCYQVsVUPzjOHv+ZjyxCHlVYJMBVFerVCP7k.
Are you sure you want to continue connecting (yes/no)?
```

Answering *yes* to the question above, should store the server key in your key store for future use, hence later connections shouldn’t ask for the key exchange.

Once key-based authentication is working, a batch file can be used to carry out activities via sFTP by use of the batchfile option (-b).

Using batch file processing

Batch mode reads a series of commands from an input batchfile instead of using the terminal/ console. Since it lacks user interaction it should be used in conjunction with non-interactive authentication.

Note: sFTP - in batch-mode - will abort if any of the following commands fail: get, put, reget, reput, rename, ln, rm, mkdir, chdir, ls, lchdir, chmod, chown, chgrp, lpwd, df, symlink, and lmkdir.

Termination on error can be suppressed on a command by command basis by prefixing the command, with a '-' character (for example, -reput /tmp/somefile).

You can recursively copy entire directories when uploading and downloading by using the *-r* option. Note that sftp does not follow symbolic links encountered in the tree traversal.

Example on transferring files from a Windows platform to a UNIX/linux remote server.

```
## upload file(s)from a Windows system to UNIX/linux system.
## First change the local Directory to where the files for upload resides.
lcd <C:\SomeDir\FCS-files>
## Now copy the file(s)
## Since we want to copy all files, note the use of "-r" for recursive copy (doesn't work for
symbolic links)
put -r *.xml
## Quit the sFTP session
quit
```

Example on transferring files from a UNIX/linux platform to a UNIX/linux remote server.

```
## upload file(s)from a Unix/linux system to UNIX/linux system.
## First change the local Directory to where the files for upload resides.
lcd </SomeDir/FCS-files>
## Now copy the file(s)
## Since we want to copy all files, note the use of "-r" for recursive copy (doesn't work for
symbolic links)
put -r *.xml
## Quit the sFTP session
quit
```

Use the SFTP Command to Authenticate and Run the File

In both examples, the batch-file is named Upload-to-FCS.cfg, the Username is FCSuser and the FCS server used for upload is FCS-server (Note: UNIX/linux is case-sensitive).

On a Windows Client the syntax is: *sftp –b <path>\<batch_file> <username>@<hostname>*
Example: sftp –b C:\MyFiles\Upload-to-FCS.cfg FCSuser@FCS-server:/transfer

On a UNIX/linux Client the syntax is: *sftp –b /<path>/<batch_file> <username>@<host name>*
Example: sftp –b /home/FCS/Upload-to-FCS.cfg FCSuser@FCS-server:/transfer

Logging of upload(s)

Depending on your choice of Operating System, logging of uploads done by sFTP is possible.
As the sFTP client does **not** rename your original files after uploading them to the FCS-server, you should take care of this based on the logging facilities for your particular OS.
The option *-v* is used for logging and for UNIX/linux the logging should appear in your syslog
Depending on your choice of sFTP client, advanced logging can be possible.

If you rely on the **default** feature of sFTP, that a batch file will terminate upon an error (with an exit code of "1") you can be pretty sure that an exit code of "0" will mean "successful transfer".

In order to be (absolutely) sure a transfer is o.k. you'd probably depend on doing the individual transfers based on a more advanced script.
A **very basic** example of this is listed on the next page.

The (very crude) code example, relies on a number of variables which easily could be parameters to the script. As it is, it'll only deal with XML-files with a lowercase extension of *.xml* this could also easily be changed.

The core (and purpose) of the code is based on single calls to the sftp client, fetching the return value (exit code) for the sftp client. For OpenSSH sftp client the exit code for success is 0 (zero) and for any failure, the exit code is 1 (one).
Based on the exit code either succes or failure of the individual transfer is written to a log. In the example the log does not "rotate" but is kept as a single file (this can be changed by adding a few lines of code).
The log is located in the directory and named as stated in the variable *TransferLog*.

Format for the log-file is shown below:

```
test1.xml - Transferred with success at - 04-19-2017 14:53:33 - now renamed to .done
test2.xml - Transferred with success at - 04-19-2017 15:01:37 - now renamed to .done
test3.xml - Transferred with success at - 04-19-2017 15:02:36 - now renamed to .done
test4.xml - Transferred with success at - 04-19-2017 15:10:47 - now renamed to .done
test5.xml - ERROR on transfer at - 04-19-2017 15:11:17 - file is NOT renamed
test6.xml - ERROR on transfer at - 04-19-2017 15:12:30 - file is NOT renamed
```

The variable *UploadDirectory* is the directory in which the XML-files are located.
The variable *targetHostname* is the IP-address or hostname for the FCS-server.
The variable *targetUsername* is the Username for the sFTP credentials for the FCS-server. While the username is kept in the code in clear text and hence, potentially could be a security risk - in reality this is not so. The transfers are based on exchange of keys between the servers, so knowing the username (and not the password) would not suffice to gain access to the FCS-server. If an intruder has access to the server at the customer site, well this is quite another scenario, but of scope for this document.

The script creates a unique sftp-batchfile for every transfer and passes the name to the sftp-client. Afterwards the batchfile is deleted and - in case on a successful transfer - the *.xml* file is renamed to *.done.* If the transfer fails - for any reason - the *.xml* file keeps it's extension.
The script uses a special directory for storing the batch-files, this directory must exist.

Example script for uploading files with a check for failure/success (OSX based)

```
#!/bin/sh
# BASED ON OSX !!!!!!
# Runs through the list of xml-files in the upload directory
# NOTE: They MUST have a lowercase xml extension and NO white spaces

# Transfer logging for copying - should be refined for rotation e.g. adding date
TransferLog=/home/FCS/logs/sftptransfer.log
       # for debug - echo $TransferLog
# Upload Directory i.e. where the xml-files for uploading to FCS i located
UploadDirectory=/home/FCS/upload-directory
       # for debug - echo $UploadDirectory
# Username for FCS sFTP server - should not be unsafe, but could be encrypted on the client side
targetUsername=FCSuser
       # for debug - echo $targetUsername
# IP-address or hostname for the FCS sFTP server
targetHostname=FCSserver
       # for debug - echo $targetHostname

# Change directory to where the xml-files are
cd $UploadDirectory
```

```
# List and process all xml-files (renamed to .done on successful transfer)
for uploadfile in *.xml
do
    if [[ -f $uploadfile ]]; then
    # It's a real file
                # for debug - echo $uploadfile
       # create a temporary file containing sftp commands
       SFTP_BATCH_FILE=/Users/henrik/Desktop/FCS/sftp-batch-files/$(basename $0).$$.txt
                # for debug - echo $SFTP_BATCH_FILE
       sFTPcmd1="lcd $UploadDirectory"
                # for debug - echo $sFTPcmd1
       echo ${sFTPcmd1} > ${SFTP_BATCH_FILE}
                # for debug - echo $uploadfile
       sFTPcmd2=put $uploadfile  /transfer/$uploadfile
                # for debug - echo $sFTPcmd2
       echo ${sFTPcmd2} >> ${SFTP_BATCH_FILE}

       # Now, run the sFTP client in batch mode
       sftp -b ${SFTP_BATCH_FILE} ${targetUsername}@${targetHostname}
       # Get the exit code (0 = succes, 1 = failure)
       sftpExitStatus=$?

       # Get a DateTime stamp for logging - NOTE this is valid for OSX
       EXECUTION_DATE_TIME=$(date +"%m-%d-%Y %T")

       # rm ${SFTP_BATCH_FILE}

       if [ $sftpExitStatus -eq 0 ]; then
                # Write the result of transfer to the log.
                echo "$uploadfile - Transferred with success at - "$EXECUTION_DATE_TIME" - now
renamed to .done" >> ${TransferLog}
                # Rename (move) the file from .xml to .done
                mv "$uploadfile" "${uploadfile/.xml/.done}"
        else
                # Write the failure of transfer to the log.
                echo "$uploadfile - ERROR on transfer at - "$EXECUTION_DATE_TIME" - file is NOT
renamed" >> ${TransferLog}
        fi
fi
done
```

Scheduling of upload(s)

As with the NAER Sikker Client (and new FCS-Client), all wanted scheduling of uploads is based on implementation of proper calls to the chosen sFTP client for the given operating system.
Some sFTP clients allow for setting up "internal" scheduling.
For the script, shown above, it would be natural to put execution into a cron-job.