# Dokumentationsstandardskabelon – RA ITU

Denne skabelon er tiltænkt som minimum dokumentation af en applikation løsning. Dokumentationen bør lægges i readme.md i github under det pågældende projekt.

Tekst i [eksempel] er vejledninger og kan slettes fra skabelonen når den er udfyldt.

Indhold

[Dokumentationsstandardskabelon – RA ITU 1](#_Toc16790186)

[1. Navn og version 2](#_Toc16790187)

[2. Overordnet løsningsbeskrivelse 2](#_Toc16790188)

[3. Arkitekturbeskrivelse og tegning 2](#_Toc16790189)

[4. Placering af kode, scripts og evt. binære filer 3](#_Toc16790190)

[5. Driftssetup 3](#_Toc16790191)

[6. Udviklingssetup 6](#_Toc16790192)

[7. Klassediagram 7](#_Toc16790193)

## Navn og version

[Beskriv navnet og versionen på løsningen ]

ASTA(1007+ FileConverter) Version 1.0

## Overordnet løsningsbeskrivelse

[Beskriv kort formålet med løsningen og den forretningsmæssige sammenhæng]

ASTA is a tool for converting statistical data files into a delivery package, that can be stored by Rigsarkivet. The program is to create a folder structure and create files based on the statistical data that have been created by the user in his local statistical software. After the program have created the folder structure a control will be made of the data the user has created. After that creation another program takes over and creates the package that Rigsarkivet will use for storage. That program will also be able to convert from the storage format to a statistical file type again (SPSS).

The reason behind this is minimize the workload performed by Rigsarkivet in regard to correct the delivery package to the correct format after they have been submitted to Rigsarkivet and to have removed most errors before the data are even submitted. As the wait for having your package accepted by Rigsarkivet can be as much as three weeks, this have the bonus of making the users almost 100 % sure they have delivered what all required material and correct data in the first try.

## Arkitekturbeskrivelse og tegning

[Beskriv løsningens arkitekturmæssige opbygning

* Tegning af løsningsarkitektur
* Sammenhæng med andre løsninger
* Særlige valg/beslutninger i forhold til arkitektur begrundes]

The full solution consists of two different pieces of software (program 1 and 2 for ease of identification). These two consists of the following four subprograms;

* Nemesis: The creation of the delivery package (program 1)
* Hybris: Test of the delivery package. (program 1)
* Athena: The creation of the storage format. (program 2)
* Styx: The creation of a statistical file format (SPSS) from the data in the storage format. (program 2)

There will be two different program releases. One for the regular users and one for Rigsarkivet, the latter containing extra features and program parts (only this version will have access to Athena and Styx). The one for Rigsarkivet will also contain extra options in the program running Nemesis and Hybris. So, the overall setup of the two releases will be as follows;

* User version: Program 1, which has the basic functionality the user needs to create and test packages.
* Rigsarkivet version: Program 1 + 2, which has extra functionality in program 1.

Program 1 is a crossplatform solution made with Electron (<https://electronjs.org/>).

Program 2 is made with .NET technologies (WinForms). The applications have no responsibilities in regard to the storing of the transformed data.



## Placering af kode, scripts og evt. binære filer

[Beskriv:

* Hvor løsningens kildekode er placeret?
* Hvilke scripts findes der og hvor?
* Hvor evt. binære filer er placeret?]

The solution is hosted in GitHub (<https://github.com/the-danish-national-archives/ASTA> ) -from here the solution can be cloned to a local machine.

## Driftssetup

[Beskriv løsningens driftssetup, inkl. Guide til deployment mm.]

The solution is hosted in GitHub as described above and can be access via Git commands. There are no requirements for tools to do this, only a terminal with Git installed. GitHub does have a desktop application that can be used instead of a terminal.

In the package.json file, all script commands used by the program are detailed, like ‘npm start’, which is used to start the program 1 locally.

To run one of the other custom scripts, use ‘npm run <script>’ – e.g. npm run package-win. That command will make a user version of the solution. In the package.json file it is detailed how to make different versions of the program, this is listed in the ‘scripts’ part, see below:



When creating a release package, you need to use the ‘electron-packager’ component. This will make a package of the solution, which then can be made into an .exe / .dmg file. There is made a number of scripts to handle this, these all start with “package- “.

“electron-packager” is an npm package that can be found at: <https://www.npmjs.com/package/electron-packager>  
here you can find relevant information about the use of the module. Please not that the ‘—extra-resources’ flag is not described her. This flag allows for the inclusion of extra files in the build (such as the script files). The other used flags are handling out folders, icon for the application and so on.

There is two different packaging script for the various operating systems. Those that are called “-admin” is for Rigsarkivet only. In this package the administrative tools are included in the interface (they are hidden, but inaccessible in the user version).

Once the package has been made, a executable file then needs to be made. This is done by running the scripts starting with “create-“. Here are also scripts for making admin and user versions of the files.

NOTE: the .dmg files must be made on a Mac machine. These cannot be created on Windows. The script will throw and error and inform about this if you try.

On the GitHub page, you can find more information about how to use GitHub.

When creating the .exe file, be aware that an error can occur if you have an old file in the same output folder as you’re packaging to. Make sure to delete old executable files before creating new ones. The error is that the packaging module will include the old .exe file in the build and the program will then start twice when you run it.

[%USERPROFILE%](http://environmentvariables.org/UserProfile): UserProfile environment variable represents the path to the user's profile folder (C:\Users\{username})

By default, it writes system logs to the following locations:

* on macOS: ~/Library/Logs/file-converter/ASTA\_systemlog.log
* on Windows: %USERPROFILE%\AppData\Roaming\file-converter\ ASTA\_systemlog.log

asta.log file format: [timestamp] [log type] text [code path]

* log type: info, warning or error
* code path: Rigsarkiv.[Model].[Class].[Function]

example as followings:

[2019-07-03 10:24:04.604] [info] selected path: C:\VSO\Rigsarkivet\SPSS\spss23765\_short.sav Rigsarkiv.Hybris.DataExtraction.AddEvents

[2019-07-03 10:24:24.909] [error] ENOENT: no such file or directory, scandir 'C:\VSO\Rigsarkivet\test\FD.12345\Data' Rigsarkiv.Hybris.DataExtraction.EnsureData

Athena C# .Net has 2 packaged applications within administrator version of electron:

* AthenaConsole.exe: used to batch converting by starting & passing parameters explicitly as following:

AthenaConsole.exe “SIP json-file path” “AIP output path” “AIP folder name”

* AthenaForm.exe: Invoked from electron as child process with required parameters. Implements by using .Net 4.5 Windows Forms

Both above applications depending on Athena.dll component see [Klassediagram](#_Klassediagram). Both above applications System logs settings using log4net configuration in (AthenaConsole.exe.config & AthenaForm.exe.config) files:

<log4net>

<appender name="RollingFile" type="log4net.Appender.RollingFileAppender">

<file value="${USERPROFILE}\AppData\Roaming\file-converter\athena\_systemlog.log" />

<appendToFile value="true" />

<maximumFileSize value="100KB" />

<maxSizeRollBackups value="2" />

<layout type="log4net.Layout.PatternLayout">

<conversionPattern value="%date %level %logger - %message%newline" />

</layout>

</appender>

<root>

<level value="INFO" />

<appender-ref ref="RollingFile" />

</root>

</log4net>

Log path on Windows: ${USERPROFILE}\AppData\Roaming\file-converter\athena\_systemlog.log

Styx C# .Net has 2 packaged applications within administrator version of electron:

* StyxConsole.exe: used to batch converting by starting & passing parameters explicitly as following:

StyxConsole.exe “AIP path” “DIP output path” “DIP folder name” “script type”

* StyxForm.exe: Invoked from electron as child process. Implements by using .Net 4.5 Windows Forms

Both above applications depending on Styx.dll component see [Klassediagram](#_Klassediagram). Both above applications System logs settings using log4net configuration in (StyxConsole.exe.config & StyxForm.exe.config) files:

<log4net>

<appender name="RollingFile" type="log4net.Appender.RollingFileAppender">

<file value="${USERPROFILE}\AppData\Roaming\file-converter\styx\_systemlog.log" />

<appendToFile value="true" />

<maximumFileSize value="100KB" />

<maxSizeRollBackups value="2" />

<layout type="log4net.Layout.PatternLayout">

<conversionPattern value="%date %level %logger - %message%newline" />

</layout>

</appender>

<root>

<level value="INFO" />

<appender-ref ref="RollingFile" />

</root>

</log4net>

Log path on Windows: ${USERPROFILE}\AppData\Roaming\file-converter\styx\_systemlog.log

## Udviklingssetup

[Beskriv løsningens udviklingssetup, inkl. Guide til opsætning mm.]

The solution for program 1 is made in Elctron, so any IDE that supports this, is optimal. As the files never compiles, a simple text editor can do as well. Program 2 is made in .NET and requires Visual Studio to compile.

Once the solution is cloned locally, you need to ensure that the required npm modules for program1 are installed (these are listed in package.json), therefore you need to run ‘npm install’ once the initial clone is done. This will install the packages on the machine.

When this is complete you can run ‘npm start’ which runs program 1 locally.

For program 2 you just run the application from Visual Studio.

There have been used various modules (npm) in the solution for program 1, these are listed here;

* Electron (<https://www.npmjs.com/package/electron>). Core component for the solution.
* Electron-packager (<https://www.npmjs.com/package/electron-packager>). Handles the packaging for the various operating systems in the program.
* Electron-installer-dmg (<https://www.npmjs.com/package/electron-installer-dmg>). Creates a mac dmg file (this needs to be run on a Mac machine).
* Fortawesome/fontawesome-free (<https://www.npmjs.com/package/@fortawesome/fontawesome-free>). Used to make icons in the log view in the Nemesis part of the program.
* Chardet (<https://www.npmjs.com/package/chardet>). Checks the format of the data files from the user (if they are UTF-8 formatted).
* Junk (<https://www.npmjs.com/package/junk>). Filters files in searches.
* Nodejs-base64 (<https://www.npmjs.com/package/nodejs-base64>). Does base 64 encoding / decoding.
* XmlDom (<https://www.npmjs.com/package/xmldom> ). Xml file handling.
* electron-log (<https://www.npmjs.com/package/electron-log> ). Creates a log file.
* fast-csv (<https://www.npmjs.com/package/fast-csv> ). Parsing CSV data files.

All modules are listed with a minimum version in the code, but once the code is cloned and installed, the latest modules will be downloaded as well.

The list of dependencies can be found at:  
[https://github.com/the-danish-national-archives/ASTA/network/dependencies](https://github.com/the-danish-national-archives/1007plus/network/dependencies)

Athena C# .Net has followings modules:

log4net (<http://logging.apache.org/log4net/> ) .NET logging

## Klassediagram

[Indsæt klassediagrammer eller anden relevant teknisk dokumentation for sammenhænge i løsningens opbygning.]

For all .js files there have been made code description which describes a files functionality and usage. As this is not OO-programming a class diagram is redundant.  
The following illustrates program 2’s components diagram.



Asta.dll is responsible for custom logs. By subscribing to LogManager event handler “LogAdded” possible different output displays for each LogEntity object can be implemented



Athena.dll component has 4 inherited converter classes (Structure, MetaData, Data & Index) that’s implement Run method. Each of these classes is responsible for partially convert action as followings:

* Structure: create AIP folder structure and copy embedded resource XSD files.
* MetaData: build table index & research index XML files with related code lists data tables
* Data: use stream writer to convert CSV to XML data table files.
* Index: Ensure files indices XML file and create report.

Tables Property in Report class updates through above converter’s run methods ends with list of Table objects. Each object contains related conversion output data like Columns objects. Index Flush method generates report. Data GetRow function take Table object and row index return specific detailed Row object with before, after values & errors. The Converter structure takes the followings parameters:

* LogManager: responsible for add & flush custom logs.
* SIP json-file path
* AIP output path
* AIP folder name



Styx.dll component has 3 inherited converter classes (Structure, MetaData & Data) that’s implement Run method. Each of these classes is responsible for partially convert action as followings:

* Structure: create DIP folder structure and copy embedded script files based on script type.
* MetaData: build related texts files for (VARIABEL, VARIABELBESKRIVELSE, KODELISTE & BRUGERKODE)
* Data: use stream writer to convert XML data to CSV files.

Tables Property in Report class updates through above converter’s run methods ends with list of Table objects. Each object contains related conversion output data like Columns objects.

The Converter structure takes the followings parameters:

* LogManager: responsible for add & flush custom logs.
* AIP path
* DIP output path
* DIP folder name
* Script type (SPSS, SAS, Stata)

