# 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](#_Toc23410958)

[1. Name and version 2](#_Toc23410959)

[2. Overall solution description 2](#_Toc23410960)

[3. Architecture description and diagrams 2](#_Toc23410961)

[4. Location of code, scripts and possibly binary files 3](#_Toc23410962)

[5. Operation Setup 3](#_Toc23410963)

[5.1. Creation of packages and releases 3](#_Toc23410964)

[5.2. General setup 5](#_Toc23410965)

[6. Development Setup 6](#_Toc23410966)

[7. Class diagram 7](#_Toc23410967)

[8. Browser version supported 10](#_Toc23410968)

[9. Languages 11](#_Toc23410969)

[10. Profile 13](#_Toc23410970)

## Name and version

ASTA Version 2.0.0

## Overall solution description

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.

## Architecture description and diagrams

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;

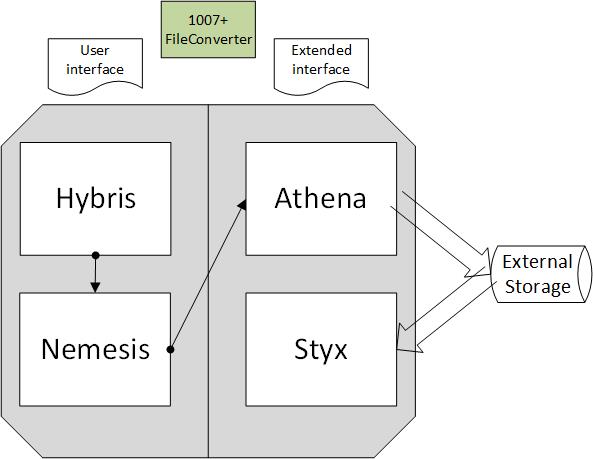
* Nemesis: The creation of the delivery package (program 1)
* Hybris: Test of the delivery package. (program 1)
* Athena: The creation of the storage format. (program 2)
* Styx: The creation of a statistical file format (SPSS) from the data in the storage format. (program 2)

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.



## Location of code, scripts and possibly binary files

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.

## Operation Setup

## Creation of packages and releases

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:

Et billede, der indeholder indendørs

Automatisk genereret beskrivelse

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 several 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 have not been described here. This flag allows for the inclusion of extra files in the build (such as the scripts, languages and PDF 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 “-extended” is for Rigsarkivet only. In this package the administrative tools are included in the interface (they are hidden, but inaccessible in the user version).

Firstly the ‘version’ field in package.json needs to be updated. This field will be reflected in the interface and program list (on Windows). The release number consists of three digits e.g. 1.0.4, these digits are defined as;

1 – Major change (e.g. new language pack, addition of R scripts)  
0 – Minor change (new functionality in existing features in ASTA)  
4 – Bug fixes

After this a package can be made using the package- scripts.

Once the package has been made, an 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 attempted.

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.

After the executable files have been created, a release needs to be created on GitHub. Releasing on GitHub follows the GitHub guide on releases, which can be found at <https://help.github.com/en/articles/creating-releases>

This does in short include;

* Draft a new release
* Create version number and tag for the release
* Include the binary files for the release (non-code related files)
* Create the (pre-)release.

## General setup

[%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/asta/ASTA\_systemlog.log
* on Windows: %USERPROFILE%\AppData\Roaming\asta\ 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\asta\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\asta\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\asta\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\asta\styx\_systemlog.log

## Development Setup

The solution for program 1 is made in [Elctron](https://electronjs.org/), 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

## Class diagram

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)



## Browser version supported

The program is developed in HTML and JavaScript, and was tested and worked on the following versions of the following browsers (as of September 2019);

* Chrome: 77.0.3865 (7.7.299)
* Firefox: 69.0
* Internet Explorer: 11.0.145
* Safari (tested on Mac): 12.1.2

Note that the Internet Explorer is blocking (as a setting) embedded JavaScript from executing when opening a file, therefore there is need to click a checkbox in order to make the log files work in this browser.

On Mac there is an issue with the HTML5 element, data list, which makes dropdown menus does not work correctly. There are 3rd party modules that can help solving this issue.

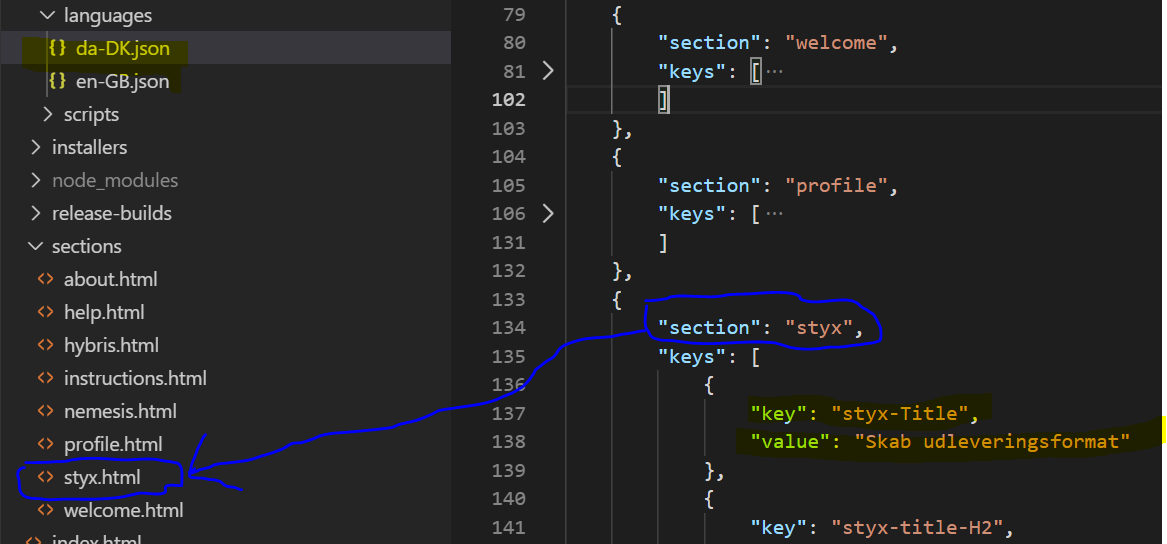
## Languages

Hybris & Nemesis (program 1) are supporting multiple languages. The default language is Danish. JSON ([https://www.json.org](https://www.json.org/)) files are used to save different languages texts. You can use online validation site ([https://jsonlint.com](https://jsonlint.com/)).

File’s name is using LCID standard (<https://docs.microsoft.com/en-us/openspecs/office_standards/ms-oe376/6c085406-a698-4e12-9d4d-c3b0ee3dbc4a>). Files are placed at electron/assets/languages and structured as array of sections. Each section has a name related to HTML page and keys array. Each key element has key/value pair. All Keys are unique and starting with related section name separating by “-“ with extra text. A new language file required add file to release package ([Creation of packages and releases](#_Creation_of_packages))

Followings are examples using Visual Studio Code.

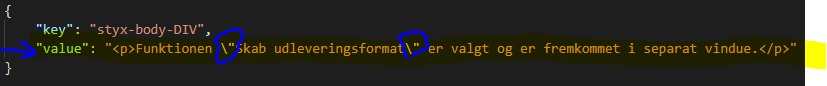
Remember keep each key name uniquely a cross language file. See da-DK.json example:



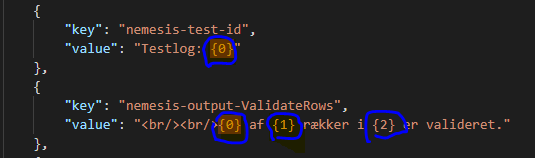
In most cases above key is representing HTML element id and value will be placed as element value on program starting event. This will be done by using reserved “languages” CSS class. See key’s related HTML element example:



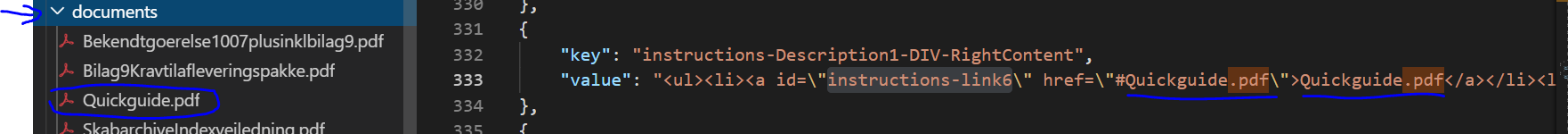
Key’s value has variant possible contents from simple text to long HTML contents with menu and PDF links. Because of value is string typed then remember to use backslash as escape character See example:



Key’s value has also Composite Formatting notation { index } (<https://docs.microsoft.com/en-us/dotnet/standard/base-types/composite-formatting>). These will be replace by self application do not change them. See:



Key’s value has also PDF links references do not change id attribute since it used by application. In order to update link reference to new language just change self filename then added new file to electron/assets/documents folder. Of course you need also to add new PDF file to release package ([Creation of packages and releases](#_Creation_of_packages)) See:

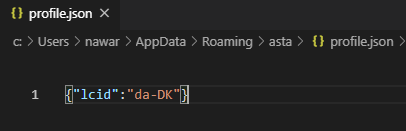


## Profile

Hybris & Nemesis (program 1) is generating for first run JSON profile file:

* on macOS: ~/Library/Logs/asta/profile.json
* on Windows: %USERPROFILE%\AppData\Roaming\asta\ profile.json

File is saving user language id. This can be changed by user at profile section. Save action will restart application. See:



Notice : you need to delete this file manually on ASTA uninstallation.