# 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](#_Toc2691214)

[1. Overordnet løsningsbeskrivelse 2](#_Toc2691215)

[2. Arkitekturbeskrivelse og tegning 2](#_Toc2691216)

[3. Placering af kode, scripts og evt. binære filer 3](#_Toc2691217)

[4. Driftssetup 3](#_Toc2691218)

[5. Udviklingssetup 5](#_Toc2691219)

[6. Klassediagram 6](#_Toc2691220)

## Navn og version

[Beskriv navnet og versionen på løsningen ]

1007+ FileConverter Version 1.0

## Overordnet løsningsbeskrivelse

[Beskriv kort formålet med løsningen og den forretningsmæssige sammenhæng]

Formålet med 1007+ FileConverteren er at give brugere et værktøj for at kunne producere nært fejlfrie afleveringspakker, der kan videregives til Rigsarkivet til bevaringsformål. Der skal kunne oprettes en mappestruktur og laves filer baseret på de statistiske data som brugeren har genereret i sit statistikprogram. Derudover skal der kunne laves en kontrol af data fra brugere, konvertering til bevaringsformatet og slutteligt en mulighed for at kunne konvertere fra bevaringsformatet og til et statistik format (SPSS).

Baggrunden for dette er at mindske den mængde arbejde der skal udføres af Rigsarkivet i forhold til at tilrette pakkerne efter de er blevet indsendt til Rigsarkivet, for eventuelle fejl og mangler.  
Da kan være op til tre måneders behandlingstid på afleveringspakkerne, har det også den fordel at brugeren sjældent skal rette i deres pakker, da de oftere vil blive afleveret korrekt i første forsøg.

## Arkitekturbeskrivelse og tegning

[Beskriv løsningens arkitekturmæssige opbygning

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

Løsningen er bygget på Electron (https://electronjs.org/), da der er ønsket en crossplatform understøttelse. Løsningen er tiltænkt som en selvstændig pakke, som kan bruges både af Rigsarkivet, men også af brugerne. Der vil blive lavet to forskellige versioner af programmet, henholdsvis til Rigsarkivet og til brugerne. Den bagvedliggende kode er ens for begge versioner, men i den udgave der benyttes af Rigsarkivet, vil der være nogle administrative interfaces der er skjult i den version som den almindelige bruger vil benytte.

Løsningen er opbygget af fire program dele der arbejder uafhængigt, men som kan bruges i tandem.

De fire programdele er;

* **Hybris:** Ansvarlig for opbygningen af mappestrukturen. Herunder også placering af filer som genereres igennem det flow brugeren ledes igennem.
* **Nemesis:** Ansvarlig kontrol af det indhold af afleveringspakken som brugeren har lavet ved brugen af Hybris.
* **Athena:** Ansvarlig for at tage indholdet af den generede data og testede afleveringspakken fra Hybris / Nemesis og konvertere det til bevaringsformatet.
* **Styx:** Ansvarlig for at konvertere indholdet af bevaringsformatet til statistik formatet SPSS.



Programmet har intet ansvar i forhold til lagringen af bevaringsformatet. Det eksterne lager er angivet for at give et fuldstændigt billede af programmets flow.

## 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?]

Hele løsningen hostes i GitHub (<https://github.com/the-danish-national-archives/ASTA> ) – herfra kan løsningen klones til egen maskine.

## Driftssetup

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

Løsningen hostes på GitHub og derfor kan den tilgås via almindelige Git kommandoer. Der kræves ikke noget specielt værktøj til dette, så enhver form for terminal kan anvendes.

I pakcage.json filen er der beskrevet alle script kommandoer som programmet anvender, så som ’npm start’, der anvendes til at køre løsningen lokalt. Det skal bemærkes at der er nogle script kommandoer der er reserveret i npm, så som ’npm start’ eller ’npm help’ – den sidste kommando viser hjælpe filen til npm, som indeholder information om den generelle anvendelse af npm.

For at køre en af de efterfølgende scrips brug ’npm run <script>’ – f.eks:  
npm run package-win  
Overstående vil lave en pakke af bruger versionen af programmet.

I package.json filen er anført hvorledes der udgives forskellige udgaver af programmet, dette findes under scripts delen:



Når pakken skal laves, bruges der først ”electron-packager”. Dette laver en pakke af filerne som man efterfølgende kan lave til en .exe / .dmg fil. Hertil er der lavet en række scripts der håndtere pakke oprettelsen, disse starter alle med ”package-”.

”electron-packager” er en NPM pakke, som kan findes på <https://www.npmjs.com/package/electron-packager> - her kan findes relevant information omkring brugen af modulet. Det skal bemærkes at ’–extra-resources’ flaget ikke er beskrevet her. Dette flag tillader at inkludere ekstra filer i bygget (så som script filerne). De øvrige flag omhandler output mappe, ikon til applikationen mv.

Der findes pt to forskellige pakketerings scripts til de gængse operativsystemer. Dem der er kaldt ”-admin” er tiltænkt Rigsarkivet selv. I denne pakke, kommer de administrative værktøjer med på grænsefladen, så som Athena og batchkørsel af test mapper i Nemesis.

Når pakken er lavet, skal der efterfølgende laves en eksekverbar fil. Dette gøres ved at man kører et de scripts der starter med ’create-’. Her er der igen scripts til at lave admin og almindelig bruger filer.

OBS: .dmg filerne skal laves på en Mac. Dette kan ikke køres på Windows. Scriptet vil kaste en fejl og informere om dette såfremt man forsøger.

På GitHub siden kan man desuden finde forklaringer på hvordan GitHub anvendes, skulle der være tvivl.

Man skal være opmærksom på at nå man laver exe filen, kan der opstå en fej hvis man har en anden exe fil liggende i output mappen fra scriptet. Man bør derfor altid sørge for at slette en exe fil når man er færdig med den (f.eks. når man har lagt exe filen ud i live eller når man skal til at lave en admin pakke efter man har lavet en normal bruger fil.

Fejlen består i at programmet kan komme til at pakke de ekstra filer fra den eksisterende exe fil med i den nye exe fil og programmet så vil starte op to gange når man starter det og derfor kan man have to vinduer med programmet kørende, hvor den ene kan være bruger udgaven og den anden kan være administrator udgaven.

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)

## Udviklingssetup

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

Løsningen er lavet i Electron, så ethvert IDE der kan understøtte dette vil være at foretrække. Da filerne aldrig kompileres, kan et simpelt tekstværktøj også anvendes.

Når løsningen er klonet lokalt, skal man tilsikre at de NPM moduler løsningen anvender (anført i package.json filen i projektet) er installeret og man skal derfor køre kommandoen ’npm install’. Det installerer de korrekte pakker på maskinen.

Herefter kan man køre kommandoen ’npm start’ og så startes løningen op i udviklings mode lokalt på maskinen.

Der er brugt for skellige moduler i udviklingen af løsningen, disse listes her:

* Electron (<https://www.npmjs.com/package/electron>). Driver hele løsningen.
* Electron-packager (<https://www.npmjs.com/package/electron-packager>). Klarer pakketeringen af hele løsningen til operativ specifikke pakker.
* Electron-installer-dmg (<https://www.npmjs.com/package/electron-installer-dmg>). Laver en Mac dmg fil til installation af programmet på Mac.
* Fortawesome/fontawesome-free (<https://www.npmjs.com/package/@fortawesome/fontawesome-free>). Bruges til at lave ikoner I log viewet I Nemesis delen af programmer (specifikt rettetegn, udråbstegn og fejltegn).
* Chardet (<https://www.npmjs.com/package/chardet>). Finder fil formatet (kontrollerer om filerne som testes af Nemesis er gemt som UFT-8).
* Junk (<https://www.npmjs.com/package/junk>). Sorterer filer fra i søgninger.
* Nodejs-base64 (<https://www.npmjs.com/package/nodejs-base64>). Håndterer base64 encoding /decoding.
* XmlDom (<https://www.npmjs.com/package/xmldom> ). Xml fil håndtering.
* electron-log (<https://www.npmjs.com/package/electron-log> ). By default it writes logs to the following locations:
  + on macOS: ~/Library/Logs/file-converter/log.log
  + on Windows: %USERPROFILE%\AppData\Roaming\file-converter\log.log
* fast-csv (<https://www.npmjs.com/package/fast-csv> ). Parsing CSV data files

Alle moduler er angivet med en minimums version i koden, men når koden hentes ned på ny og installeres, vil den seneste version hentes.

Listen af dependencies kan der ud over også ses på:  
<https://github.com/the-danish-national-archives/1007plus/network/dependencies>

## Klassediagram

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

Der er for alle .js filer lavet kodebeskrivelse der angiver filens funktionalitet og anvendelse i filen. Da der ikke er tale om OO-programmering er et klassediagram redundant. Men føglende er Athena C# .Net 4.5 klassediagram



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.

Tables Property updates through above converter’s run methods ends with list of Table objects. Each object contains related conversion output data like Columns objects. Data will be used by generating 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. This objects needs to be based through all converter’s objects in order to collecting all logs in one object.
* SIP json-file path
* AIP output path
* AIP folder name

By subscribing to LogManager event handler “LogAdded” possible different output displays for each LogEntity object can be implemented. AthenaConsole & AthenaForm projects has different implementation for it see [Driftssetup](#_Driftssetup).