

Paper 4857-2020

## *Efficient Release Management with SAS®*

Lars Tynelius, Infotrek

### ABSTRACT

Release management is a process for managing different software versions between environments. This paper examines how you can improve the quality, speed, and efficiency of building or updating software and have a reliable release management process.

By creating a web application with SAS® Stored Processes as the back end, we can easily create, export, import, and promote releases through the different environments in an automated and highly configurable dynamic way. Different content can be included in the release such as SAS® metadata, configuration files and tables, SAS macros, migration jobs, and so on.

### INTRODUCTION

Release management is a software engineering process intended to oversee the development, testing, deployment, and support of software releases.

Release management usually begins in the development cycle with requests for changes or new features. If the request is approved, the new release is planned and designed. The new design enters the testing or quality assurance phase, in which the release is built, reviewed, tested and tweaked until it is ultimately accepted as a release candidate.

The release then enters the deployment phase, where it is implemented and made available. Once deployed, the release enters a support phase, where bug reports and other issues are collected; this leads to new requests for changes, and the cycle starts all over again.

### BACKGROUND

When you do development in SAS there are a few tools that you could use to promote a new release of the system to a target environment. These tools are usually good for managing one isolated deployment process, but there is no SAS tool or application that combines the different parts to build a complete release that could be deployed to another environment.

There is a need to have an application that can combine the different parts and tools to build a release in automatic way instead of doing it manually.

### DEPLOYMENT PROCESS

#### PROBLEMS

Most often, development teams do not consist of just one resource, but several developers are part of the team. Secondly, other parts of the system can be supplied by another development group and the combination of the new versions should be deployed as a new release. This is usually hard to manage and many sub-processes are handled manually.

Here are some problems listed when not having an automated release management process.

- Manual deployments are error-prone. Steps in the release process can accidentally be missed, faults or errors in the release process may not be spotted, which in turn leads to that the software shipped could be broken and working incorrectly.
- Resource dependence. Performing manual or partially manual deployments are often the responsibility of a small subset of people or even more common, one person. If that person is unavailable, you might have a problem if not every step in the release deployment process is documented and followed in detail.
- Time-consuming. If the process is not automated, it is usually required to follow detailed documentation on every step that should be executed in the release process. Often it is also required to validate that the step was executed successfully. You also need to repeat the manual steps and their validation in all your environments.
- Expensive to deploy frequently. If you want to deploy frequently it is very expensive and will have high overhead in resources and time. If you have a manual process it is more common to deploy less frequently compared to automated deployments.
- Revision and audit. It is hard to track which release version exists in the different environments when it was exported or imported and by who.

Automated deployments are not only repeatable, but they are configurable too. Although the underlying release process is permanent, the target environments and machines can be changed easily.

The automated process will repeat the same steps independently on the environment and the current version of the release. This will significantly reduce risks when promoting a new version of a system or application.

## GOALS WITH THE APPLICATION

The main purpose of the application is to support a process that builds and manages releases in an efficient and productive way. The process should be easy to use and maintain, highly configurable and dynamic. The application should be used as a framework or shell when managing the release management deployments.

The main capabilities of the application should be:

- Create and export a release
- Import a release
- Move a release to a target environment
- Highly configurable
- Restart the release process if an exception occurred
- Configurable steps
- Exception handling and return codes
- Easy overview when a release was exported or imported

## IDENTIFY OBJECTS

### WHAT SHOULD BE MOVED?

The first step is to identify all the objects or parts that are required to be moved to the target environment. The identification process will differ for different systems or applications. All the identified objects will together build the complete release.

**Usually, it's quite easy to identify** the objects, if we have an approach to identify all different objects that are required by a rollback of a release version or that could potentially change during a development phase.

Examples of objects:

- Metadata objects
- SAS functions
- SAS macros
- SAS formats
- Models
- Configuration or control tables
- autoexec\_usermods.sas
- Etc.

The other part that needs to be identified is environment attributes. Which attributes will change depending on in which environment the code is executing on. These are usually static attributes that I suggest that you define in one place instead of spreading them out in different configurations or control tables. I think it most practical to gather all the attributes in the Autoexec file for the application or system, easy to maintain and to locate when everything is in one file.

If you will have different values for each environment, you could easily create a format that will be grouped on each level or environment. So for example, if we need configure different ports for a Postgres database in each environments:

```
/* create postgres format */
proc format;
  value $postgresport
    Lev1 = "5432"
    Lev2 = "5433"
    Lev3 = "5434"
    other = 'ERROR'
  ;
quit;

/* extract the level from the autoexec path */
%let level = %cmpres(%substr(&autoexec,%index(%lowcase(&autoexec),lev),4));

/* create the global macro variable from the format postgresport */
%let postgresport = %sysfunc(putc(&level, $postgresport));
```

## HOW CAN WE AUTOMATE THE IMPORT/EXPORT FOR EACH IDENTIFIED OBJECT

When the different objects have been identified, we need to investigate how it could be possible to export and import different types of objects in an automated way. There are few tools that are shipped with SAS as out-of-box for extracting and importing metadata.

The SAS Intelligence Platform Batch Tools provides different batch tools that could be used in the release management process:

Tool	Description
SASPlatformObjectFramework/9.4/ExportPackage	Export SAS metadata
SASPlatformObjectFramework/9.4/ImportPackage	Import SAS metadata
SASPlatformObjectFramework/9.4/tools/sas-delete-objects	Delete specific metadata objects
SASDataIntegrationStudioServerJARs/4.8/DeployJobs	Deploy SAS® Data Integration Studio jobs.

For some objects, that are identified to be included in the release, there might not have a predefined tool or command-line interface to extract or import information. In most cases, **it's possible to create custom code to achieve that functionality through a third API.**

Many of the SAS solution and their products could today be consumed through a REST API both for doing export and import of content. Here are some examples of code that are required to have custom code implementations.

- Scheduled LSF jobs, need to extract which LSF jobs that are scheduled in the source environment with SAS® 9.4 Open Metadata Interface to be able to set which LSF job that should be scheduled in the target environment.
- Promote changes created in the source environment when using check-in/checkout in SAS Data Integration Studio to the target environment.
- Adjust the subprop file for each target environment. The subprop file is created when **we're using the ExportPackage tool from SAS and it's possible to substitute values.** We can substitute the value specified from the source configuration to the target **configuration in the file. When you import the metadata it's possible to specify the** subprop file as the template for substitution.
- Some of the objects moved might need to be adjusted after they are imported in the target environment.

Undependably if you will use an existing command or create custom code for exporting/importing content you should create command instructions into a temporary script file and then execute that file.

If you try to execute the complete command in the terminal you might have problems that the command was too long and could additionally break the command logic. Another benefit with a temporary script file is that it will be easier doing testing and debugging of the actual script.

Here is an example for creating a temporary file with the export of SAS metadata:

```
%let dt = %sysfunc(datetime(),dtpic);

filename exp2spk temp;

/* create temporary file so we can execute, make sure to delegate the
return code */
data _null_;
  file exp2spk;
  put "/usr/local/SASHome/SASPlatformObjectFramework/9.4/ExportPackage -
subprop \" /
      @2 "-user &username. -password &password. -domain DefaultAuth -host
&metaserver. -port &metaport. \" /
      @2 "-package &releasePath./Metadata/application1.spk \" /
      @2 "-objects ""/Application1"" \" /
      @2 "-includeEmptyFolders -disableX11 -log
&releasePath./Logs/MetadataExport_&dt..log \" /
      @2 "echo $?";
run;

/* echo file content in the log */
%echofile(fileref=exp2spk);

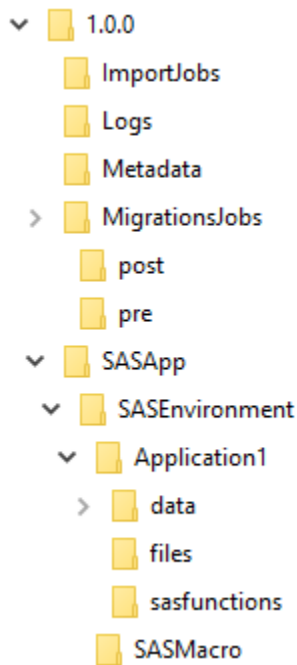
/* change authorization of the file and execute it */
%let exp2spk = %sysfunc(pathname(exp2spk));
%sysexec chmod 775 &exp2spk;
%sysexec &exp2spk;

/* save away the return code */
%let _sysrc = &sysrc;
```

## HOW TO STRUCTURE A RELEASE

When we export or import a new release from the application a folder structure on disc will be created with the name of release number, the release folder will have the following naming convention major.minor.bug.

The underlying folder structure can be configured to your **specific needs but it's wise to have** the folder structure that is reflecting the current folders that you have under your SAS configuration directory. Here is an example of folder structure:



**Figure 1. Example of release folder structure.**

The top folder 1.0.0 is the release folder; the underlying folders under SASApp will be filled with content and files that should be moved with the release.

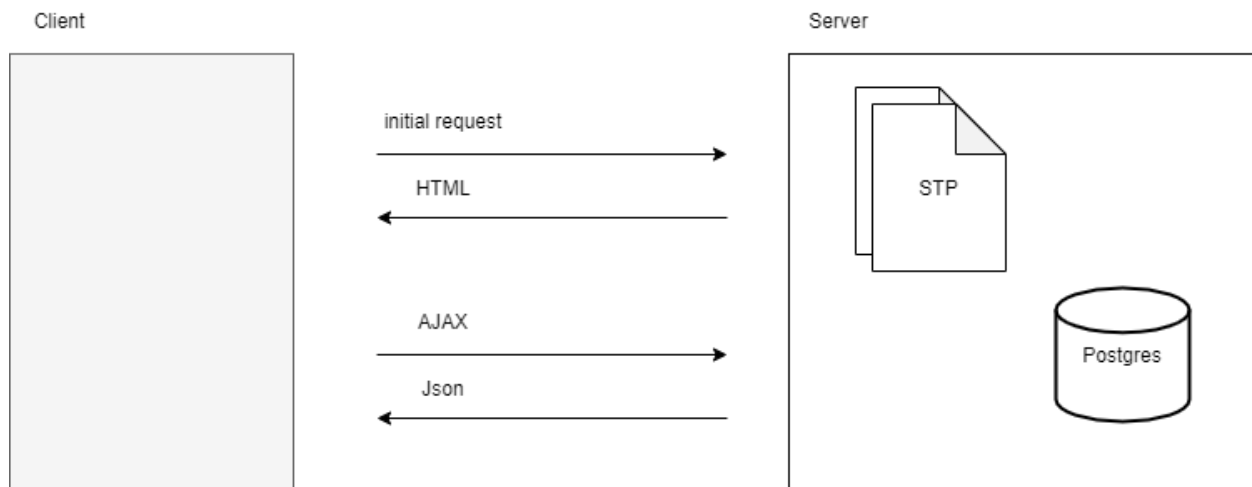
The folders ImportJobs, Logs, Metadata and MigrationsJobs are utility folders and will have content that will be used when the current release is either exported or imported. The folder structure will be replicated for all the different releases, they will have the same structure but with different content.

The release folder will be added to a zip file when the export of the release is finished. The zip file will have the same naming convention as the release folder but with the extension zip. To store the information in a zip file will have benefits when transferring to other environments.

## APPLICATION

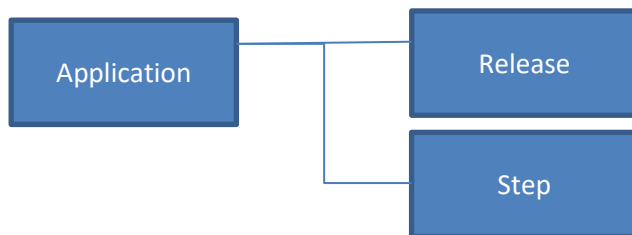
### OVERVIEW

The application consists of three different parts or tiers. The web app is built as Single-Page application (SPA), which is dynamically rewriting the current page rather than loading entire new pages from a server, which means that small parts of the page can be updated. The service tier consists of multiple SAS Stored Processes that will return JSON response. This will have less footprint compared to returning the complete page through each request. As the backend data source, we use the Postgres database, a fast scalable database that handles concurrent updates well.



**Figure 2. Communication process between client and server.**

The most important subjects in the application are Application, Release, and Step. An application is just a way to group related content that should be bundled in a release, could be a system or a subset of a system that you want to release independently.



**Figure 3. Subjects in the application.**

The release is only is just a version of that specific application; the release will have the same structure but different content.

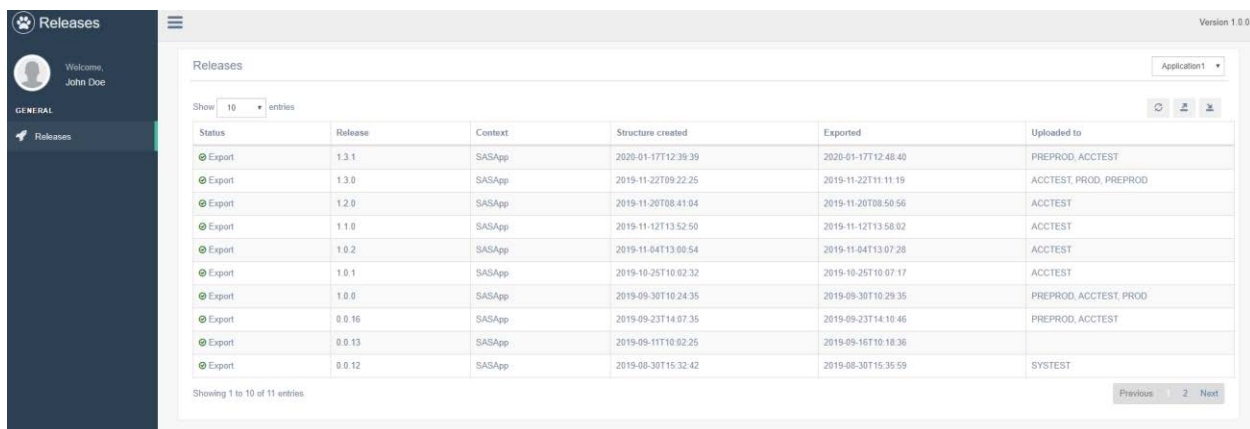
A step is one isolated task that will execute a command to do a specific task; all steps are executed in a **specified sequence**. **For each application, it's required that steps are defined** for doing an export/import of a release. For each step, a type must be defined which could have one of the following values CreateStructure, Export, and Import. The different types are used for grouping steps depending when they should be executed.

Type	Description
CreateStructure	Steps that you want to be executed to create the release folder structure. Steps with this definition will be executed before steps defined with type Export. These steps are executed when you want to build and export a release.
Export	Execute steps that will extract information and fill the release structure with content. Steps with type of Export will be executed after steps defined with type

	CreateStructure. These steps are executed when you want to build and export a release.
Import	Only the steps that should be executed when doing an import of a release.

## APPLICATION VIEWS

The application consists of two main screens; the first screen will list every exported/imported release for the current application. For each release the following information will be displayed: status, release number, context, structure created, timestamp for last step executed and if the release have been uploaded to a target environment. On the same screen it is possible to create a new export or import of a release and switch to another application.



The screenshot shows a web application interface for 'Releases'. On the left is a dark sidebar with a 'Releases' header, a user profile 'John Doe', and a 'GENERAL' section with a 'Releases' link. The main area is titled 'Releases' and features a table with columns: Status, Release, Context, Structure created, Exported, and Uploaded to. The table contains 11 rows of data, all with a status of 'Export'. At the bottom, it says 'Showing 1 to 10 of 11 entries' and has 'Previous', '2', and 'Next' navigation buttons.

Status	Release	Context	Structure created	Exported	Uploaded to
Export	1.3.1	SASApp	2020-01-17T12:39:39	2020-01-17T12:48:40	PREPROD, ACCTEST
Export	1.3.0	SASApp	2019-11-22T09:22:25	2019-11-22T11:11:19	ACCTEST, PROD, PREPROD
Export	1.2.0	SASApp	2019-11-20T08:41:04	2019-11-20T08:58:56	ACCTEST
Export	1.1.0	SASApp	2019-11-12T13:52:50	2019-11-12T13:58:02	ACCTEST
Export	1.0.2	SASApp	2019-11-04T13:09:54	2019-11-04T13:07:28	ACCTEST
Export	1.0.1	SASApp	2019-10-25T10:02:32	2019-10-25T10:07:17	ACCTEST
Export	1.0.0	SASApp	2019-09-30T10:24:35	2019-09-30T10:29:35	PREPROD, ACCTEST, PROD
Export	0.0.16	SASApp	2019-09-23T14:07:35	2019-09-23T14:10:46	PREPROD, ACCTEST
Export	0.0.13	SASApp	2019-09-11T10:02:25	2019-09-16T10:18:36	
Export	0.0.12	SASApp	2019-08-30T15:32:42	2019-08-30T15:35:59	SYSTEST

**Figure 4. Screen in the application that list all exported/imported releases**

If you select one of the imported/exported releases or if you start to export or import a new release another screen will be displayed with all steps related to that application. Information about each step will be displayed, type of step, if it was executed or not, return code, start time and end time of the step.

If you created a new release you can start to build the release on the current screen. All steps defined will be executed in sequence, if one of the steps was not successfully **executed, i.e. return code not equal to 0, the execution will stop. It's possible to restart the execution process of the release from where it was stopped.**



Status	Step	Name	Type	Rc	Message	Start	End
Completed	1	Create structure for release folder	CreateStructure	0		2020-01-17T12:39:37	2020-01-17T12:39:37
Completed	2	Create folder Metadata for release folder	CreateStructure	0		2020-01-17T12:39:37	2020-01-17T12:39:37
Completed	3	Create pre folder for the release	CreateStructure	0		2020-01-17T12:39:37	2020-01-17T12:39:37
Completed	4	Create post folder for the release	CreateStructure	0		2020-01-17T12:39:37	2020-01-17T12:39:37
Completed	5	Create Logs folder for the release	CreateStructure	0		2020-01-17T12:39:37	2020-01-17T12:39:37
Completed	6	Create SASMacro folder for the release	CreateStructure	0		2020-01-17T12:39:37	2020-01-17T12:39:37
Completed	7	Create Modelingsystem folder for the release	CreateStructure	0		2020-01-17T12:39:37	2020-01-17T12:39:37
Completed	8	Create ImportJobs folder under the release	CreateStructure	0		2020-01-17T12:39:37	2020-01-17T12:39:37
Completed	9	Create /SASApp/Application1/sascode/programs/ folder for the release	CreateStructure	0		2020-01-17T12:39:37	2020-01-17T12:39:37
Completed	10	Create /SASApp/Application1/sascode/storedprocesses/ folder for the release	CreateStructure	0		2020-01-17T12:39:37	2020-01-17T12:39:37
Completed	11	Create /SASApp/Application1/sascode/programs/UserWritten/transformsCode/ folder for the release	CreateStructure	0		2020-01-17T12:39:37	2020-01-17T12:39:37
Completed	12	Create /SASApp/Application1/scriptstatic folder for the release	CreateStructure	0		2020-01-17T12:39:38	2020-01-17T12:39:38
Completed	13	Create /SASApp/Application1/sascode/functions folder for the release	CreateStructure	0		2020-01-17T12:39:38	2020-01-17T12:39:38
Completed	14	Create CustomData folder contains parameter tables for the release	CreateStructure	0		2020-01-17T12:39:38	2020-01-17T12:39:38
Completed	15	Create control folder for the release	CreateStructure	0		2020-01-17T12:39:38	2020-01-17T12:39:38
Completed	16	Create in folder for the release	CreateStructure	0		2020-01-17T12:39:38	2020-01-17T12:39:38
Completed	17	Create trigger folder and file to avoid warning when doing import	CreateStructure	0		2020-01-17T12:39:38	2020-01-17T12:39:38
Completed	18	Create source1 folder and file to avoid warning when doing import	CreateStructure	0		2020-01-17T12:39:38	2020-01-17T12:39:38
Completed	19	Create source2 folder and file to avoid warning when doing import	CreateStructure	0		2020-01-17T12:39:38	2020-01-17T12:39:38

**Figure 5. Screen that list all steps that have been executed for the current release.**

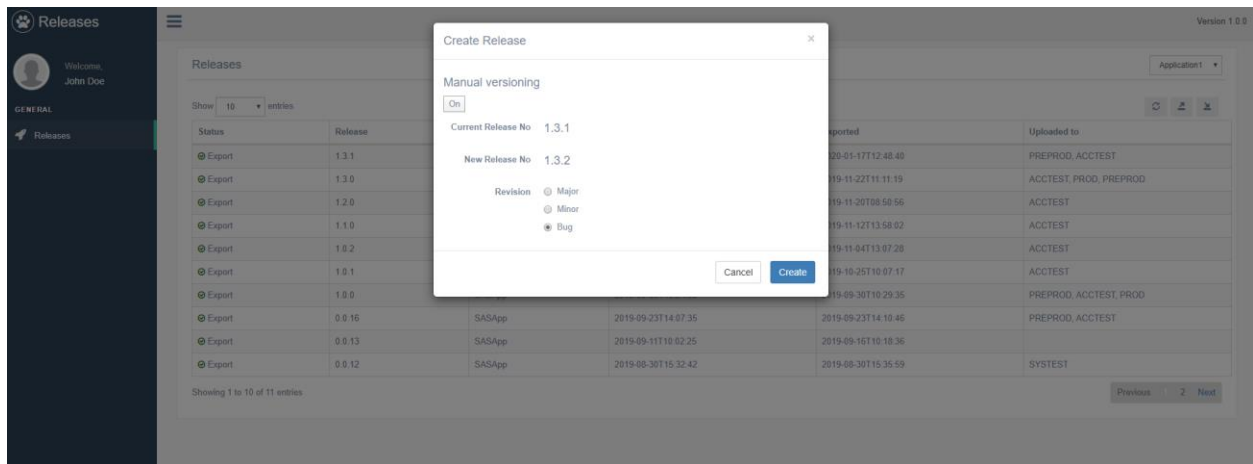
On the other tab named Upload the application, a list will be displayed with predefined target environments for that specific application. If you click on the action for the specific environment and an upload is started for the current release, it's the zipped release file that will be uploaded (1.0.0.zip).

Action	Status	Description	Rc	Host	Port	Start	End
Execute	Not executed	DEV		centosserver1.local	7551		
Execute	Not executed	SYSTEST		centosserver2.local	7552		
Execute	Completed	ACCTEST	0	centosserver3.local	7553	2020-01-17T13:14:52	2020-01-17T13:14:58
Execute	Completed	PREPROD	0	centosserver5.local	7555	2020-01-27T14:59:27	2020-01-27T14:59:35
Execute	Not executed	PROD		centosserver4.local	7554		

Showing 1 to 5 of 5 entries.

**Figure 6. Predefined target environments**

Before you build and create the release you need to specify the release number. There are two options on how to set the new release number either through the automated screen showed below or manually set the release number. If the automated screen is used then the release number will be increased by one depending on the revision selected.



**Figure 7. Screen for specifying a new release number**

## FRONTEND

To speed up the development time of the user interface (UI) you always want to use a JavaScript framework or library. This application uses jQuery, which is an open-source JavaScript library that simplifies the interactions between an HTML/CSS document, or more precisely the Document Object Model (DOM), and JavaScript.

jQuery simplifies HTML document traversing and manipulation, browser event handling, DOM animations, Ajax interactions, and cross-browser JavaScript development.

To display the data in tabular form we used the DataTables plugin for jQuery, the plugin comes with a lot built-in features such as pagination, search, data source configuration, them-able and many more functions.

To get even more functionality out of the box the application uses Bootstrap, which is one of the popular HTML, CSS and JS frameworks. The framework comes with a lot of predefined styles that are ready to use and even some pre-build component that could be useful when creating an UI.

Here is the code for executing a SAS Stored Process with jQuery:

```
function getApplications() {
    var data = [];
    var applications = null;

    // add new parameter _program ie which STP to call
    data.push({
        name: "_program",
        value: stpPath + "/GetApplications"
    });

    $.ajax({
        url: "/SASStoredProcess/do",
        data: data,
        type: 'GET',
        contentType: 'application/x-www-form-urlencoded; charset=UTF-8',
        async: false,
        success: function(response, status, xhr) {
```

```

        if (xhr.getResponseHeader('X-SAS-STP-ERROR') === null) {
            // check if error exists in the json back
            var error = response[0].error;
            if (typeof error !== 'undefined') {
                // Do something
                console.log(error);
            } else {
                applications = response;
            }
        } else {
            // We might have an error in the stp execution
            alert("Error in the execution of the STP");
        }
    },
    error: function() {
        console.log("An exceptions was thrown");
    }
});

return applications;
}

```

## BACKEND

The backend part consists of different SAS Stored Processes that will handle the requests for specific tasks in the application.

The application tables are stored in a Postgres database and all access will go through the SAS/ACCESS Interface to PostgreSQL. The Postgres database has good support for concurrent read and updates and avoids table locks.

For the backend all the data is fetched with AJAX request and all the data is returned back as JSON (**JavaScript Object Notation**). The application is built using the SAS procedure `proc json` that creates JSON from SAS, here is a simple example:

```

proc json out=_webout pretty;
    export sashelp.class(obs=2) / nosastags;
run;

```

Response from the executed code:

```

[
  {
    "Name": "Alfred",
    "Sex": "M",
    "Age": 14,
    "Height": 69,
    "Weight": 112.5
  },
  {
    "Name": "Alice",
    "Sex": "F",
    "Age": 13,
    "Height": 56.5,
    "Weight": 84
  }
]

```

```

    }
]

```

If you want to create more complex structure you could tweak the code and have more control over how the output JSON should be structured. This is an example of the code and output.

```

proc sql noprint;
    select distinct(ApplicationId) into :ApplicationId_1 -
:ApplicationId_&sysmaxlong
    from postgres.release_application
    order by name;
    %let ApplicationIdObs = &sqlObs;
quit;

%macro ApplicationIdContexts(ApplicationId);
    write open object;
    write value "applicationid";
    write value "&applicationid";

    write value "attributes";
    write open array;
    export postgres.release_application(where=(applicationid eq
"&applicationid") keep=applicationid name path) / nosastags;
    write close;

    write value "contexts";
    write open array;
    export postgres.release_application_context(where=(applicationid eq
"&applicationid")) / nosastags;
    write close;
    write close;

%mend;

%macro ByApplicationId;
    %do i = 1 %to &ApplicationIdObs;
        %ApplicationIdContexts(&&ApplicationId_&i);
    %end;
%mend;

%macro crtjson;
    proc json out=_webout pretty;
        write open array;
        %ByApplicationId;
        write close;
    run;
%mend;
%crtjson;

```

And the output from the executed code:

```

[
  {
    "applicationid": "b6e99677-53566",
    "attributes": [

```

```

    {
      "applicationid": "b6e99677-53566",
      "name": "Application One",
      "path": "/usr/local/config/Levl/SASApp/Releases/Application1"
    }
  ],
  "contexts": [
    {
      "contextid": "1",
      "applicationid": "b6e99677-53566",
      "context": "SASApp1"
    },
    {
      "contextid": "2",
      "applicationid": "b6e99677-53566",
      "context": "SASApp2"
    }
  ]
}
]

```

## CONCLUSION

To have a more reliable and efficient release management process that could be executed repeatedly in different environment an application or tool is required. The application could be built as a web application and the backend with SAS Stored Process.

The application could support many different systems or subsystem that want to promote releases through each environment in the deployment process.

The advantage is that you easily get an overview of which version is deployed in each environments plus what steps in the process were executed successfully and by who.

## REFERENCES

SAS Intelligence Platform Batch Tools documentation. Available at

<https://documentation.sas.com/?docsetId=bisag&docsetTarget=n1syra2xz7myan0zvix72mgx44.htm&docsetVersion=9.4&locale=en>

jQuery library documentation. Available at

<https://jquery.com/>

Datatables jQuery plugin can be found here:

<https://datatables.net/>

Bootstrap css framework can be found here:

<https://getbootstrap.com/>

## CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Lars Tynelius  
Infotrek  
lars.tynelius@infotrek.se  
<http://www.infotrek.se>

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.