Propel

Reach your servers

**Design**

Contents

[Goals 4](#_Toc83333599)

[Legacy Tool 4](#_Toc83333600)

[Development Phases 4](#_Toc83333601)

[Functional Design 5](#_Toc83333602)

[Menu 6](#_Toc83333603)

[Target 6](#_Toc83333604)

[Script 7](#_Toc83333605)

[Introduction to Tasks & Workflows 9](#_Toc83333606)

[Task 10](#_Toc83333607)

[Run 12](#_Toc83333608)

[Execution Results 13](#_Toc83333609)

[Workflow 14](#_Toc83333610)

[Dashboard 14](#_Toc83333611)

[Architecture 16](#_Toc83333612)

[Technical details 16](#_Toc83333613)

[Technical Design 18](#_Toc83333614)

[Domain Model 18](#_Toc83333615)

[Base Classes 18](#_Toc83333616)

[Script 19](#_Toc83333617)

[User scripts 21](#_Toc83333618)

[Propel special script parameters 21](#_Toc83333619)

[$PropelCredentials 21](#_Toc83333620)

[Execution History and Results 22](#_Toc83333621)

Figures

[Figure 1 - FD - Site Map 5](file:///C:\DATA\Dev\Propel\doc\Propel%20-%20Design.docx#_Toc37163043)

[Figure 2 - FD - Expanded Menu 6](file:///C:\DATA\Dev\Propel\doc\Propel%20-%20Design.docx#_Toc37163044)

[Figure 3 - FD, Script page Code tab 8](file:///C:\DATA\Dev\Propel\doc\Propel%20-%20Design.docx#_Toc37163045)

[Figure 4 - FD - Script page, Details Tab 8](file:///C:\DATA\Dev\Propel\doc\Propel%20-%20Design.docx#_Toc37163046)

[Figure 5 - FD, Script page Finish tab 9](file:///C:\DATA\Dev\Propel\doc\Propel%20-%20Design.docx#_Toc37163047)

[Figure 6 - FD, Task - Parameters tab 10](file:///C:\DATA\Dev\Propel\doc\Propel%20-%20Design.docx#_Toc37163048)

[Figure 7 - FD, Task - Script to run tab 10](file:///C:\DATA\Dev\Propel\doc\Propel%20-%20Design.docx#_Toc37163049)

[Figure 8 - FD,Task - Targets tab 11](file:///C:\DATA\Dev\Propel\doc\Propel%20-%20Design.docx#_Toc37163050)

[Figure 9 - FD Task - Finish Tab 12](file:///C:\DATA\Dev\Propel\doc\Propel%20-%20Design.docx#_Toc37163051)

[Figure 10 - FD Run 12](file:///C:\DATA\Dev\Propel\doc\Propel%20-%20Design.docx#_Toc37163052)

[Figure 11 - FD Execution Results 13](file:///C:\DATA\Dev\Propel\doc\Propel%20-%20Design.docx#_Toc37163053)

[Figure 12 - FD Workflow 14](file:///C:\DATA\Dev\Propel\doc\Propel%20-%20Design.docx#_Toc37163054)

[Figure 13 - FD Dashboard 15](file:///C:\DATA\Dev\Propel\doc\Propel%20-%20Design.docx#_Toc37163055)

[Figure 14 - Architecture 16](file:///C:\DATA\Dev\Propel\doc\Propel%20-%20Design.docx#_Toc37163056)

[Figure 15 - Domain Model 18](file:///C:\DATA\Dev\Propel\doc\Propel%20-%20Design.docx#_Toc37163057)

[Figure 16 - Domain model, Base Classes 19](file:///C:\DATA\Dev\Propel\doc\Propel%20-%20Design.docx#_Toc37163058)

[Figure 17 - Domain model, Script and parameters 19](file:///C:\DATA\Dev\Propel\doc\Propel%20-%20Design.docx#_Toc37163059)

[Figure 18 - Data type conversion between PowerShell and Javascript 20](file:///C:\DATA\Dev\Propel\doc\Propel%20-%20Design.docx#_Toc37163060)

[Figure 19 - Validation attributes support 21](file:///C:\DATA\Dev\Propel\doc\Propel%20-%20Design.docx#_Toc37163061)

[Figure 20 - Execution History and Results 22](file:///C:\DATA\Dev\Propel\doc\Propel%20-%20Design.docx#_Toc37163062)

# Goals

allows easier remote server administration by:

* Allowing you to get a centralized and categorized collection of PowerShell ™ scripts
* Parameterize and run those scripts against know targets.
* Keep track of each execution both in progress and historical.
* In Phase II will allow web access.

# Legacy Tool

Right now, we are using another tool created in 2016 built with PowerShell itself and WPF™ forms.

This tool has some limitations:

* Do not handle script parameters.
* Run as a desktop app in a server.
* Uses WPF as GUI.
* Run the scripts in sequential order.

# Development Phases

Two phases planned so far:

**Phase I**: The tool will be accessible only from a specific server via Remote Desktop technology, reducing any security risk to the minimum.

**Phase II**: The tool will be accessible from the internet adding security to both APP and API.

**This document will focus on the design aspects of Phase I only**.

# Functional Design

This topic will cover the functional design of Propel Phase 1. The Site map in Figure 1 shows main parts of the app and the main interactions. Following sections will explain each one of these parts in more detail.

But here I would like to summarize main topics and provide a brief explanation of the main user flows and design aspects that we are going to tackle in deep later.

There will be two main entities in our domain that are the enablers for the rest of the operations. They are Targets and Scripts.

A **Target** is a remote server on which will be able to run remote commands. So, adding a target is one of the most critical aspects of the application security, therefore will be reserved to Administrator users only.

A **Script** is an entity that represents a PowerShell script. Any user will be able to add his own scripts and keep them private or share them with the other users. Also, the users will be able to specify some target characteristics in order to avoid the script to fail in the case of missing dependencies for example.

In his everyday work, users will be facing two different cases when it comes to perform any operation on remote servers.

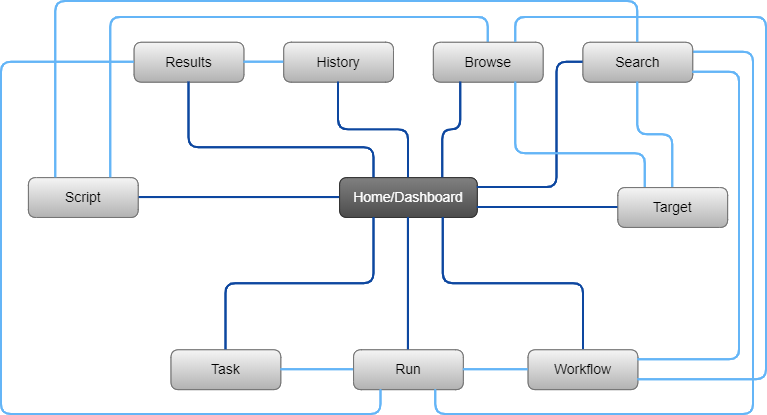


Figure - FD - Site Map

They will need sometimes to execute specific operations on single or multiple servers for troubleshooting issues or extracting some information that is not part of his routine. We can detail here a case for a server that stop responding, get event logs for troubleshooting, get disk free space in some specific server in response to an alert from a monitoring system, etc.

For other side they will need to face also repetitive task, that they would like to automate on a set of specific and known targets and running one script already developed or, (why not!), a set of script each one with a specific set of parameters and values for them. Some cases of this kind could be: Execute backups, install service patches, start or stop servers, etc.

provides the best fit tool for each case, and they are **Tasks & Workflows**.

Tasks provide a way to quickly configure a script to run in one or multiple targets in order to approach non-repetitive tasks likes the ones we mentioned.

For other side we have Workflows, workflows are no more than tasks on steroids! A workflow is composed by several steps each one is a Task. Also, workflows will be saved so you can share them and repeat the execution as many times you want.

For both, Task & Workflows, a complete log will be recorded enabling the users to access at any time the detailed activity including errors as also the steps returned results.

Please continue reading, below you’ll find a detailed explanation of each one of these topics.

## Menu

A fixed top page menu will be present in all the pages, you can see below all option with the fully expanded version of that menu.

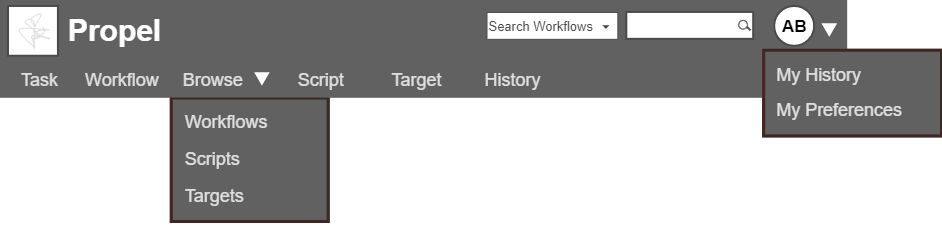


Figure - FD - Expanded Menu

Regarding the user menu: For Phase I, the user will be the interactive one, so user menu options are really constrained. So far, we are going to add the ability to access his history, (the jobs he executed), and a preference menu, (not designed yet), that could allow to change the user name, or customize certain aspects of the app.

## Target

A “Target” is a server definition that is going to be a possible target invocation for a script run.

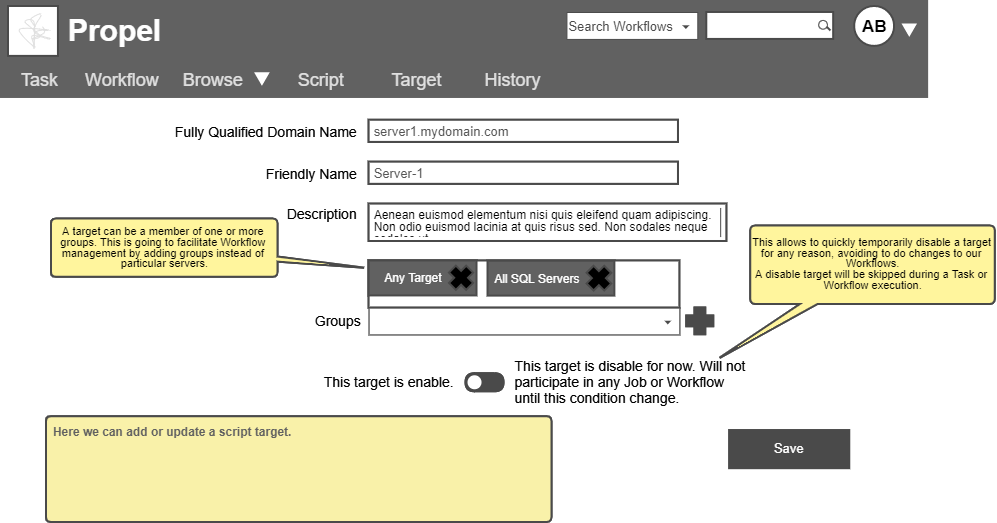
You can define target groups, that could facilitate the creation of tasks or workflows and each target can have one or more tags that allows to filter them for some specific scripts in order to avoid unintended errors. e.g.: You can create a tag named “Web Server” and use that tag for all the IIS Servers. Then, if you create a script to stop all the sites in a target server, you can tag that script with the same tag, so every time a new Task or Workflow step is using that script, only those targets will be available.

## Script

In , one of the goals is to count with a centralized repository of PowerShell scripts that, (in Phase 1), could be accessed and owned by any user.

Scripts have the following characteristics:

* They could be of **2 types**:
  + **USER Scripts**: Are the ones created by regular users.
  + **SYSTEM Scripts**: Are internal scripts from , that are used for internal operations, but they will be available to the users also.
* A script must have also a **Category** that facilitate to search and pick the right ones when creating a Workflow.
* Scripts can be developed to **target physical servers**, (like a remote server reboot script), or to **target a service**, (like an AD query or a request to a web service)





After finishing the basic script details, you will be able to go to the “Code” tab where you can paste or drop the script.

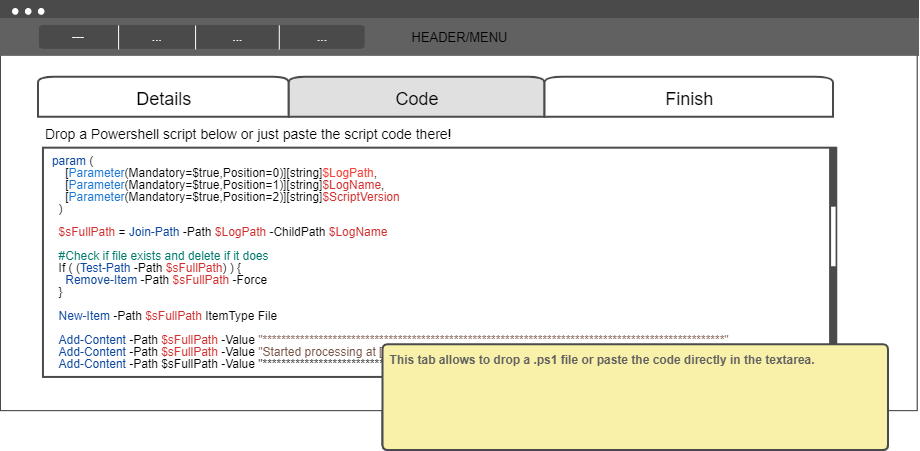


Figure - FD, Script page Code tab

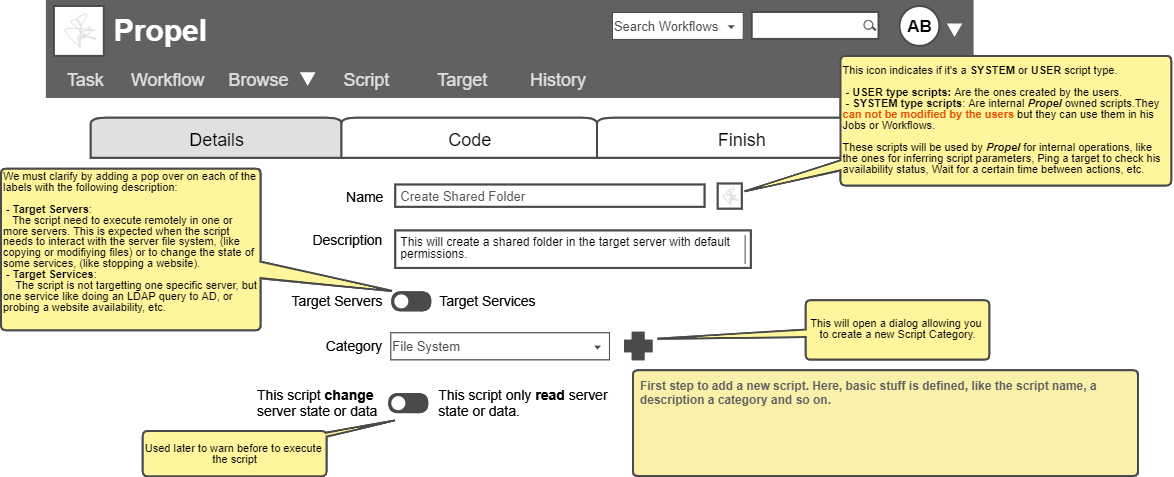


Figure - FD - Script page, Details Tab

When pasting the script and moving to the final tab, the script will be analyzed, and his parameters will be inferred. You will see in this tab all the script details and if is all ok, you can save the script.

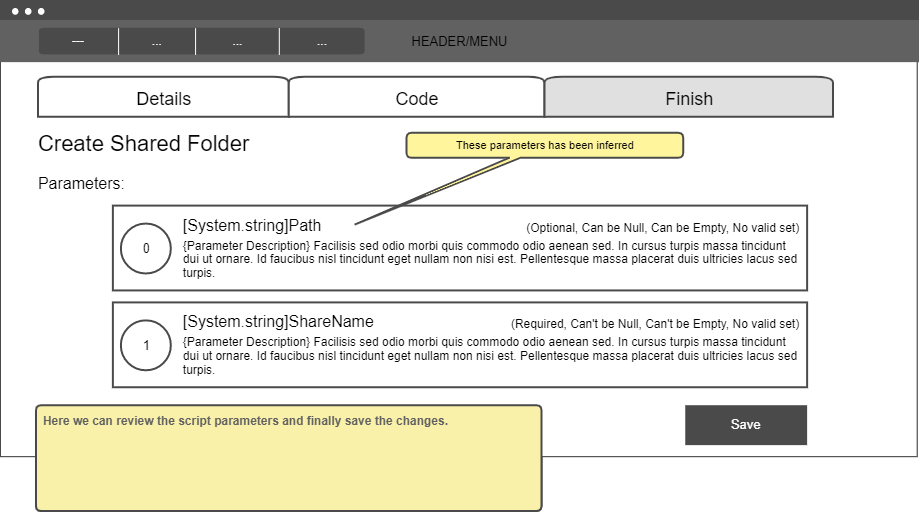


Figure - FD, Script page Finish tab

## Introduction to Tasks & Workflows

In , users will have two important tools for automation. The most basic tool is the “Task”.

**A Task is a run unit composed from a single script to run on one or more targets**. This is useful for non-repetitive tasks like to quickly check if a specific server is up and running, get the PowerShell version installed in some servers, etc.

Tasks can be executed more than once if required from both the Execution Results page and the History execution logs. But tasks are not intended to endure. The task definition will be stored for a limited time. Also, every time you need to run this kind of quick executions, you will need to recreate the task again.

When you have repetitive automation tasks, the best option is to create a Workflow. **A Workflow is a defined set of steps that are going to execute sequentially in a defined order, and where each one of these steps is a Task**.

So, if you have some repetitive tasks that could be composed by one or more steps, you can define a Workflow that could run every time you want.

Examples of workflows could be the setup of a new server where you need to create websites, assign permissions to defined user groups, and configure multiple other things in order to have all set and ready.

### Task

The first step to define a Task, is to choose the script to run.

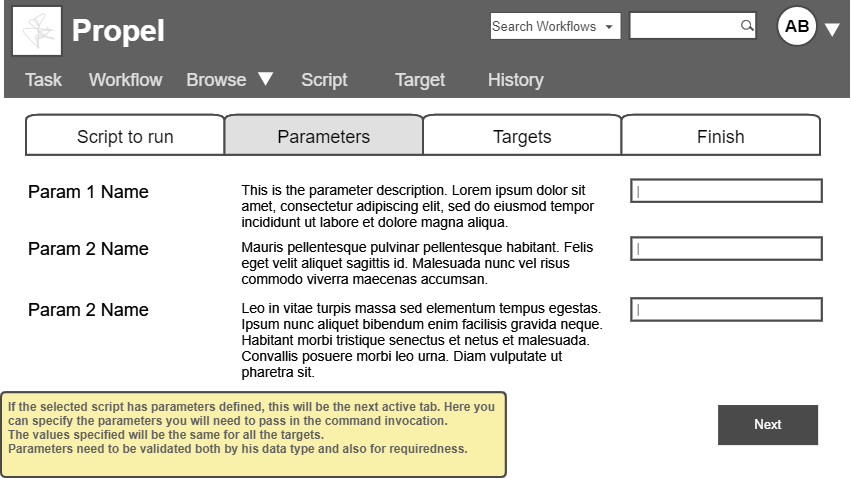


Figure - FD, Task - Parameters tab

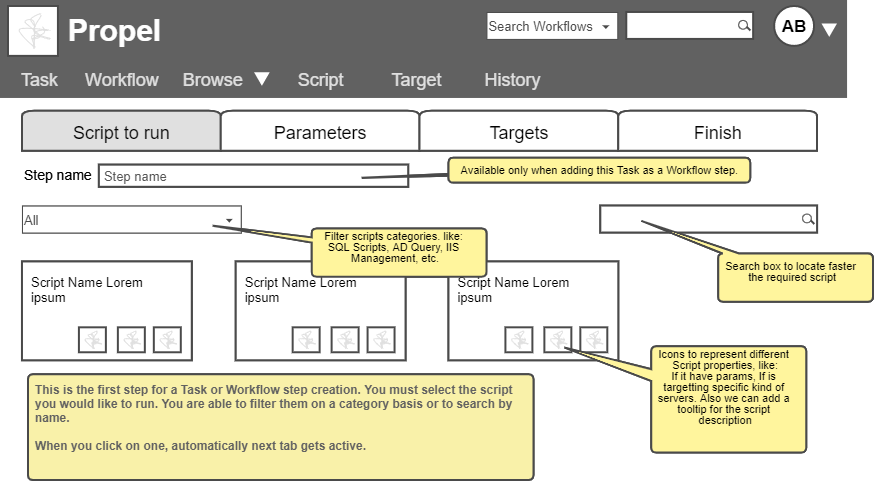


Figure - FD, Task - Script to run tab

You can filter the scripts by his categories as also search by the script name. The “Step name” field you can see in Figure 7, will be there only when our Task is created as part of a Workflow step. We will come back to that topic later when we describe the Workflow page.

If the script we choose, had any params defined, our next step will be to fill the values we are going to use for them, as you can see in Figure 6.

As you recall from script page, **parameters are inferred each time the script is created or updated**, so there will be certain validations made in the data you enter on those fields that are dependent on the parameter definition in the script.

For example: If some parameter is defined as a number, or if some other is defined as required or if it has a set of valid values. All of this will be considering in order to minimize as much as possible any runtime error.

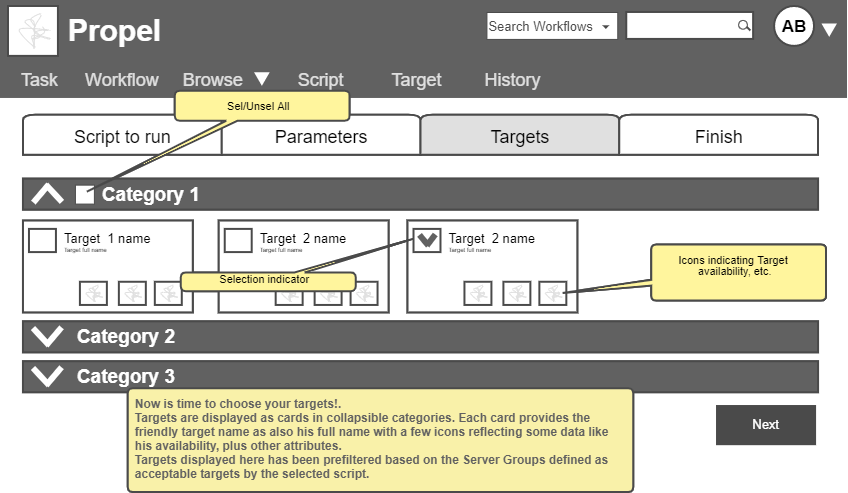


Figure - FD,Task - Targets tab

As soon we have all the required parameters, we are going to be redirected to the final tab that you can see in Figure 9. Here e can review all the task definition we just set and then click in one of the action buttons you will find there.

Only one will be visible at each time and depending if you are creating this as a standalone task or this is intended to be one step of a workflow.

If It’s the first case, as soon as you click the ”Run” button you will be redirected to the “Run” page.

### Run

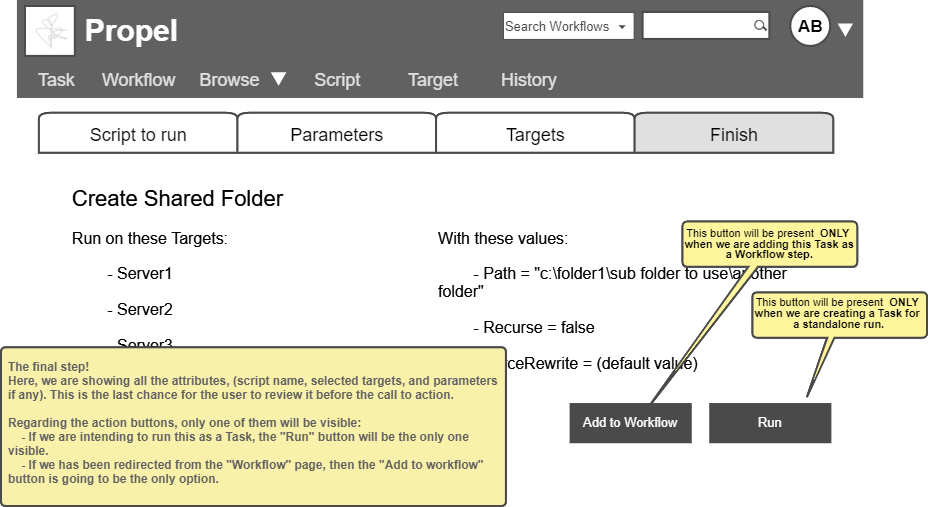


Figure - FD Task - Finish Tab

Run page takes care of start a Task or Workflow execution and monitoring in real time.



Figure - FD Run

The run page shows any messages and runtime errors during the execution in real time. And as soon the execution ends, it will automatically redirect to the “Execution Results” page.

### Execution Results

This page can be navigated too from the “History” log page, but the normal flow for one user is to arrive here from the “Run” page right after a Task or Workflow ends his execution.

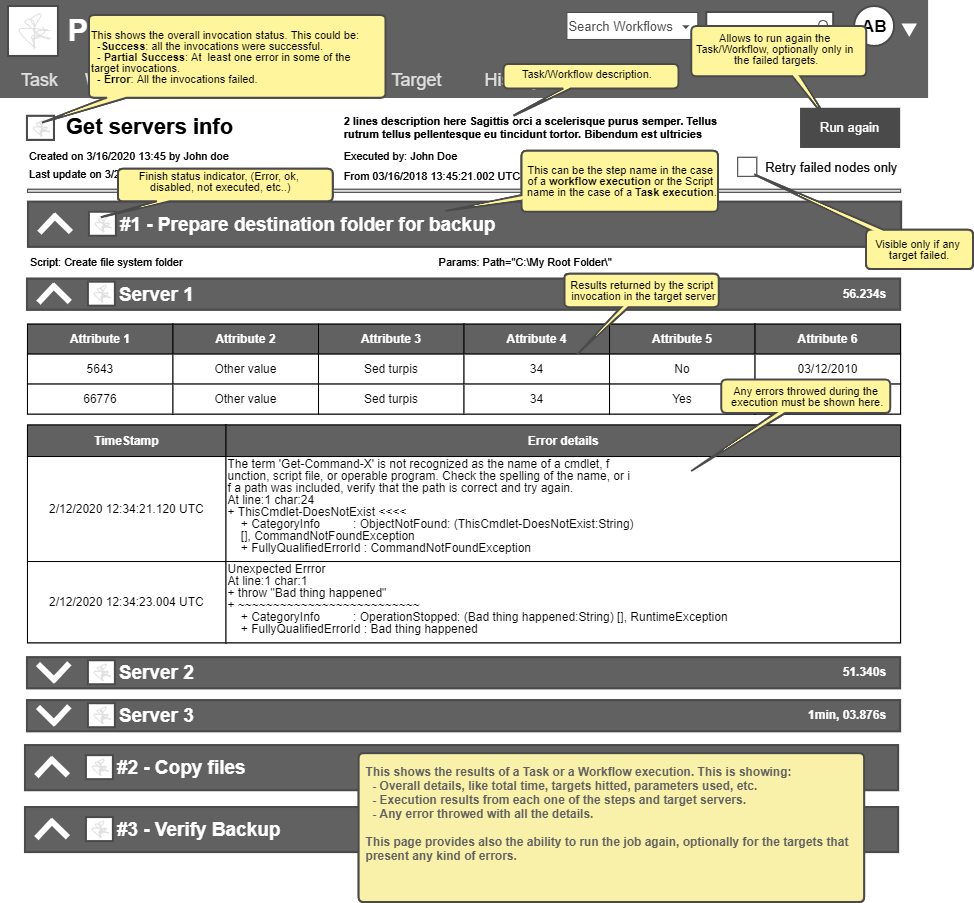


Figure - FD Execution Results

In Figure 11 you can see sample results from a workflow execution. Any possible detail is there, and you will have also the chance to re-run the Task or Workflow, and even do this just for the targets that present any errors.

### Workflow

Workflows are, like we said, a set of steps, (each one a Task), that are intended to be executed sequentially as a unique and big Task.

Also, Workflows are intended to be run several times because they represent daily-basis maintenance task or operations, one important part of workflows is that they are intended to be shared by any team member. Anyway, in certain specific cases like when you are trying one configuration that could be error-prone or can lead to unexpected results if misused, you can keep a Workflow as private as much as you want.

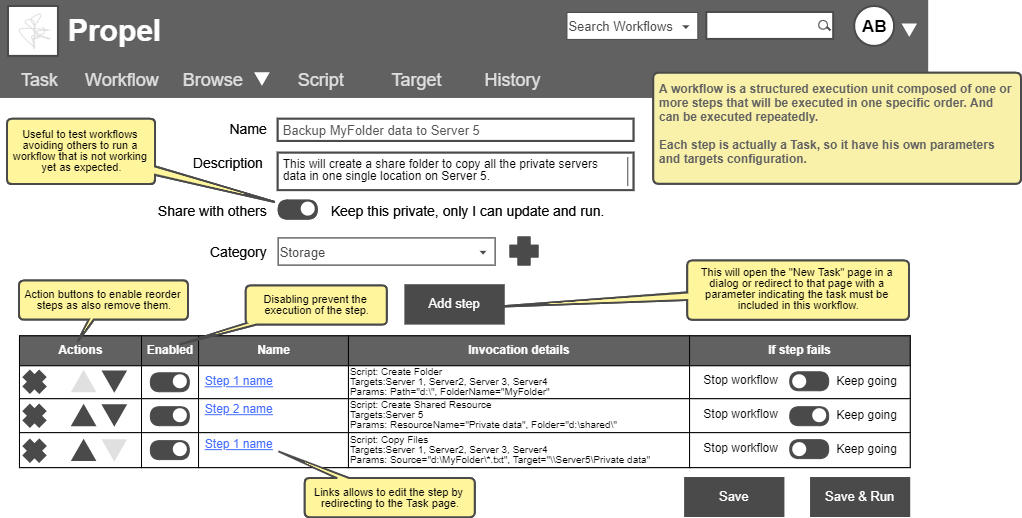
**

Figure - FD Workflow

The “Add step” button will drive you to the “Task” page, where you will be able to configure a new task as a step for this workflow.

All the added steps will be displayed in the below grid and you will be able to edit, order as also remove or disable them for the workflow.

At the step level also, you will be able to configure a defined action for each step in the case there is some errors.

## Dashboard

Dashboard is intended to be Home page.

Dashboard is presenting some basic stats like the amount of scripts, targets, executions made in last month, etc. Allowing also a quick access to the most used workflows and to see what errors occurred and access the results of the last executions. You can see the wireframe of this page in Figure 12.

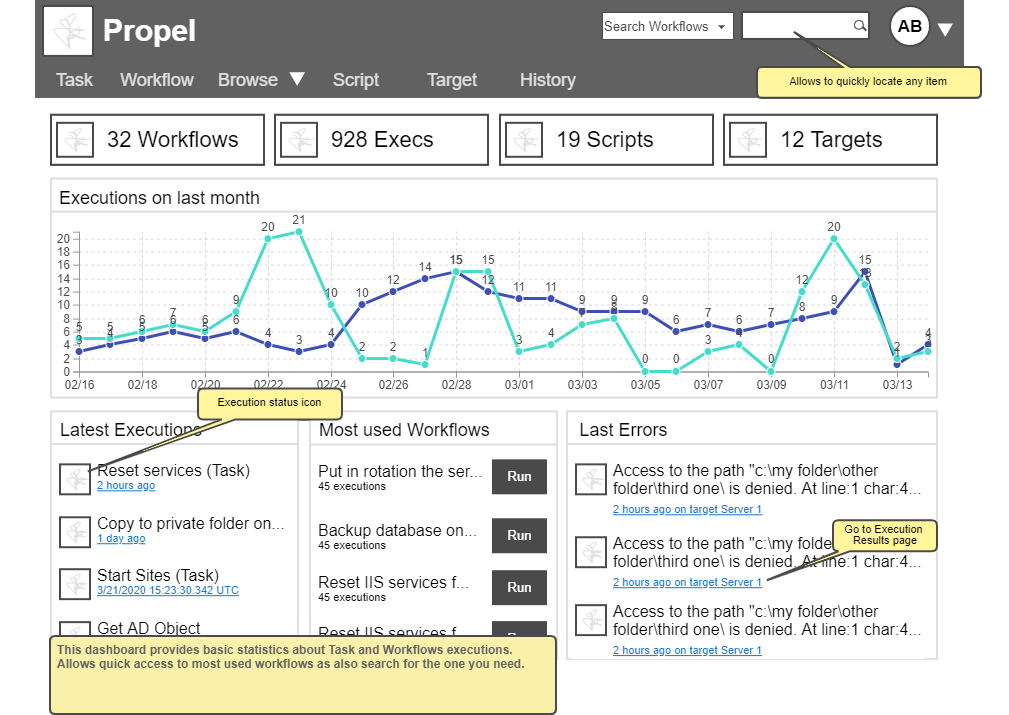


Figure - FD Dashboard

# Architecture

For Phase I, architecture will be the one depicted in Figure 14.

Because we are executing the PowerShell script in a remote target server by using WinRM[[1]](#footnote-1), we need to connect with a server in the same network zone. If the target servers are in a protected zone behind a gateway, we must ensure the API is deployed in one server in the same network zone in order to ensure connectivity.

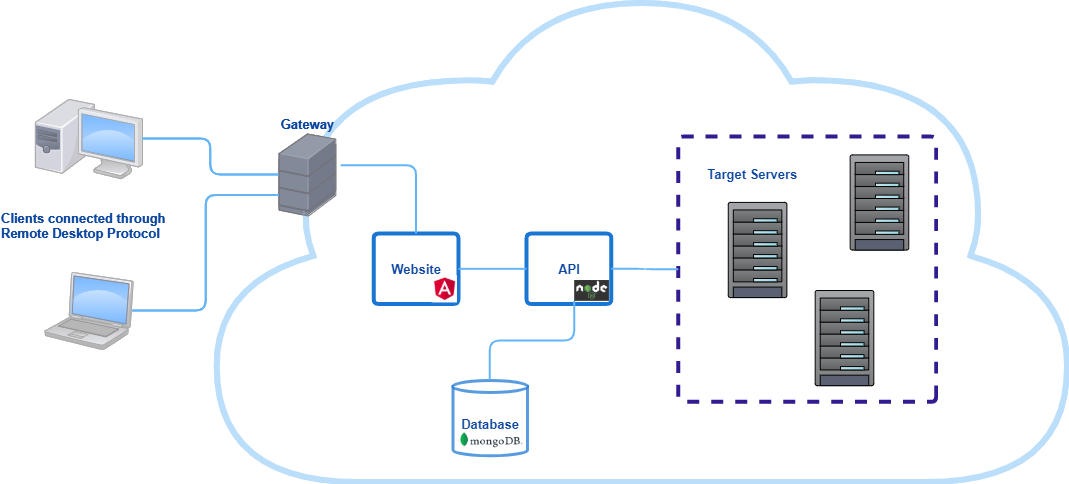


Figure - Architecture

For Phase I, the API will run as a Windows Service and it will be critical that service to run with an Administrator account, also that account need to be added to the Administrators built-in account in all the target servers too.

## Technical details

At architectural level, in Phase 1 will be composed by 3 main components, (that you can see in Figure 14).

* **Frontend**: Will be an Angular app implementing Electron to run as a desktop app.
* **API**: Will be a Node.JS API
* **Storage**: A MongoDB database running in the same server as the API.

These are the key topics; we already show in the architecture graph. The key part of is of course the task execution.

We already worked in a proof of concept Node.JS API that implementing the [node-powershell](https://www.npmjs.com/package/node-powershell) package to facilitate running PowerShell scripts, we can execute a remote script invocation. Even using Server-Sent events to monitor in real time the script execution on each remote server by streaming the STDERR & STDOUT.

Also, we found very useful for Phase I development the [node-windows](https://www.npmjs.com/package/node-windows) package that internally uses [winsw](https://github.com/kohsuke/winsw) to wrap the entire API as a windows service.

The approach will be to deploy this API as a windows service in a server located in the same network zone as the target servers and run the service with an administrator account that must be also granted on the target servers too.

# Technical Design

## Domain Model

In Figure 15, you can see the full domain model.

Is mostly self-explanatory, but I would like to dissect some parts in order to note some important points and the design reasons behind them.

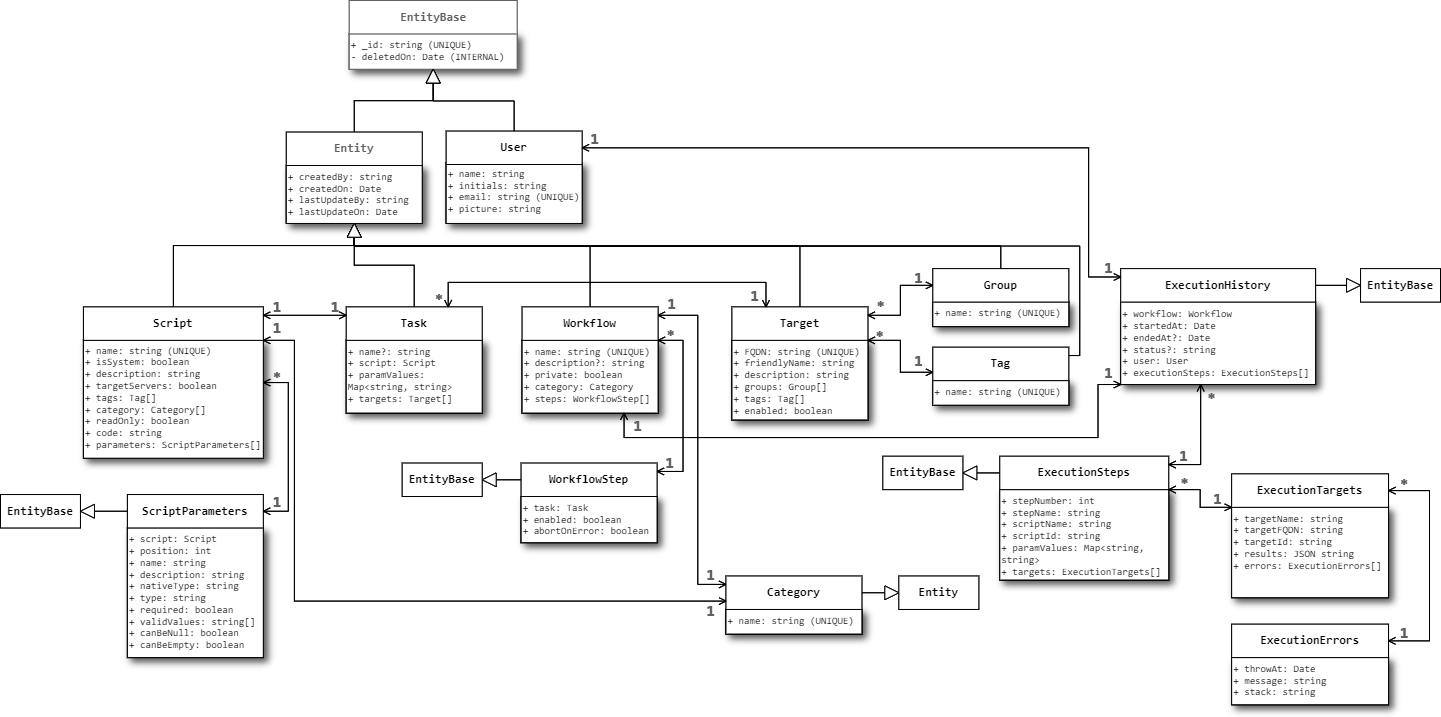


Figure - Domain Model

### Base Classes

All of the classes in the Domain Model will inherit from one of the main classes. These are:

* EntityBase: This is the base class that implements the ID attribute that allows to identify uniquely any entity document.
* Entity: This one inherits from “EntityBase” and adds audit information that will be in most of the entities in this model.

Then we have the “User” class that represents a user.

I would like to comment about one particular attribute called “deletedOn” in the ”EntityBase” class.

Don’t want to start digging here about the great debate on soft vs hard deletes[[2]](#footnote-2) on application data. Just remind that we choose mongoDB as our data store; and mongoDb doesn’t support referential integrity between collections as also cascading deletes or even Transactions!

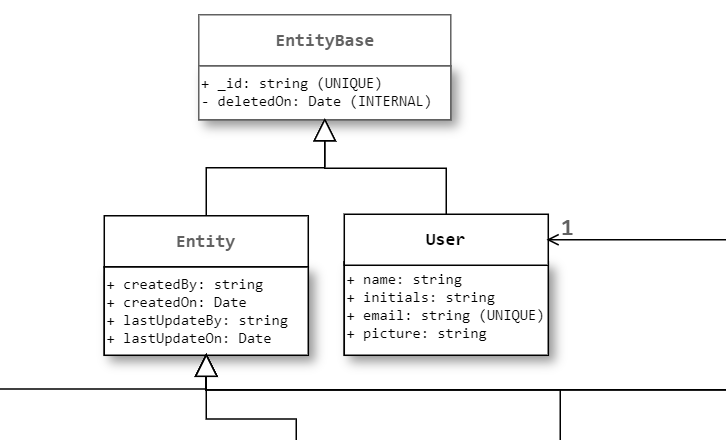


Figure - Domain model, Base Classes

Well, actually [start supporting them in v4.0](https://www.mongodb.com/blog/post/mongodb-multi-document-acid-transactions-general-availability), and I expect to use it for this feature, even when working on soft deletes.

Soft deletes also allow to keep data not required in the app but useful to troubleshoot errors, keep good usage stats and easy and granular data recovering when needed.

### Script

Scripts has been modeled with 2 different classes. First one is Script that will contain all the basic information about the user submitted script including the script code itself.

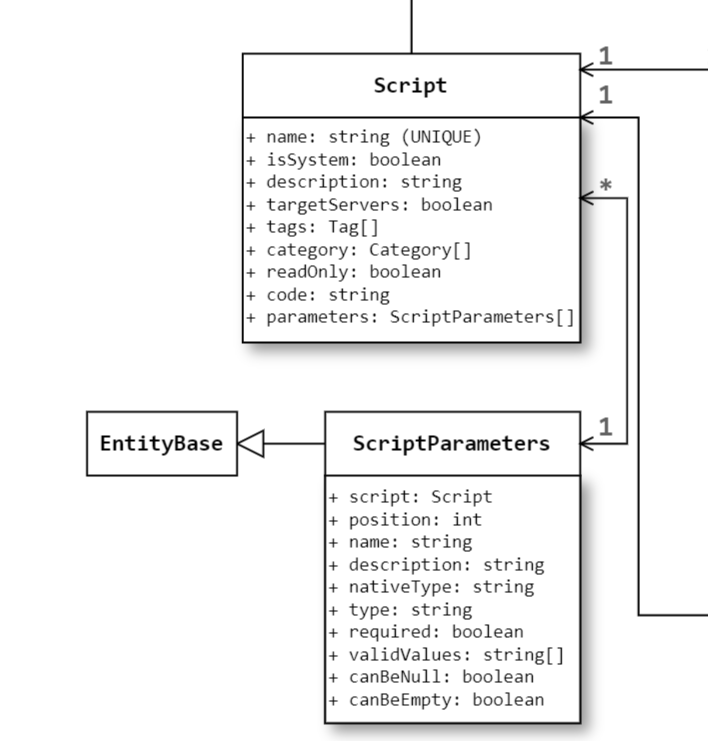


Figure - Domain model, Script and parameters

But I would like to comment some tech details about the script parameters.

As was detailed in the functional design, the **script parameters will be inferred directly by Propel every time a user creates or updates a script**.

Important note

df

**At least for Phase I development, there is no plan to implement script versioning, so this must be responsibility of each script owner.**

T**his means, that if a script change is not backward compatible, it can cause a Workflow to fail if is not properly remediated.**

There is two topics to cover when we talked about PowerShell script parameters. One is data conversion and the other is specific parameter validation attributes. In Figure 18 you can see the intended conversion. Anyway, we must serialize any value to be passed to the script as text during the remote invocation.

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Description** | **Support Status** |
| **ValidateCountAttribute** | Validates that the parameter argument count is in the specified range. | **NOT SUPPORTED**, need to be validated inside the script. |
| **ValidateDriveAttribute** | Validates that the path has an approved root drive. | **NOT SUPPORTED**, no action will be taken if this is present.  Is not possible to evaluate at least you can connect to the remote target. |
| **ValidateLengthAttribute** | Validates that the length of each parameter argument's Length falls in the range specified by MinLength and MaxLength  Note: Cannot be applied to a parameter that is not a string or string[]. | **NOT SUPPORTED**, (for now), it must throw a runtime exception based on the configured values for the execution. |
| **ValidateNotNullAttribute** | Validates that the parameters's argument is not null. | SUPPORTED by the "canBeNull" property. |
| **ValidateNotNullOrEmptyAttribute** | Validates that the parameters's argument is not null, is not an empty string, and is not an empty collection. | SUPPORTED by the "canBeEmpty" property. |
| **ValidateRangeAttribute** | Validates that each parameter argument falls in the range specified by MinRange and MaxRange.  Must be applied to a int or string parameter. | **NOT SUPPORTED**, (for now), it must throw an exception based on the configured values for the execution. |
| **ValidateSetAttribute** | Validates that each parameter argument is present in a specified set | SUPPORTED by the “validValues” property. |

Figure - Validation attributes support

|  |  |
| --- | --- |
| **PowerShell Data Type** | **JavaScript Data Type** |
| **[string]** Fixed-length string of Unicode characters | String |
| **[char]** A Unicode 16-bit character | String |
| **[byte]** An 8-bit unsigned character | Number |
| **[int]** 32-bit signed integer | Number |
| **[long]** 64-bit signed integer | Number |
| **[bool]** Boolean True/False value | Boolean |
| **[decimal]** A 128-bit decimal value | Number |
| **[single]** Single-precision 32-bit floating point number | Number |
| **[double]** Double-precision 64-bit floating point number | Number |
| **[DateTime]** Date and Time | Date Object |
| **[array]** An array of values | Array Object |
| **[hashtable]** Hashtable object | Object |

Figure - Data type conversion between PowerShell and Javascript

The other topic is the specific validation attributes that you can use to decorate PowerShell parameters. There is a lot of them that you can see in Figure 20. Some of them will be supported and others maybe in the future.

### User scripts

User scripts can return only text or JSON value. If the script is not returning a JSON value, the last text returned by the script will be took as the return value.

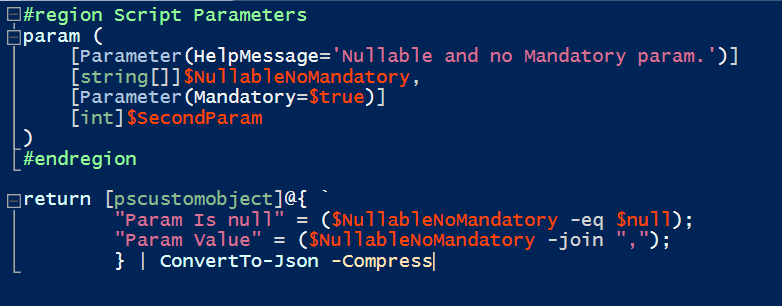


Figure - Converting output as JSON

To return JSON as the script results, you must convert to JSON the returned data, in the way you can see in Figure 19.

## Propel special script parameters

### $PropelCredentials

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Description** | **For credential type** | **PowerShell Type** |
| **Name** | Name of the credential | Any credential type | [string] |
| **Fields** | PSCustomObject with the fields defined for the Credential. | Any credential type | [pscustomobject] |
| **cred** | PSCredential object with the Username, Domain and Password defined. | Only Windows Credentials | [pscredential] |
| **AccessKey** | AWS Access Key. | Only AWS Credentials | [string] |
| **SecretKey** | AWS Secret key | Only AWS Credentials | [string] |

Figure - Attributes of the $PropelCredentials variable.

This is a special parameter that can be declared in a script.

The value for this parameter will be handled by during runtime and allows the script to access some of the credentials saved in Propel. (see Figure 21). You can set one or more credentials for this parameter for any script that have it defined, at the moment you create a Quick Task or a Workflow.

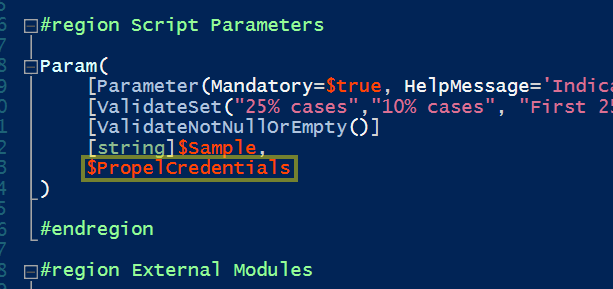


Figure - Sample $PropelCredentials parameter definition in a user script.

To use this parameter in your script you just need to declare the parameter in the script as depicted in Figure 22. is going to identify the parameter on runtime and provide the values automatically.

Also, there is a way you can see in Figure 23 to emulate and test the values of this $PropelCredentials variable in your script.

$PropelCredentials = @([pscustomobject]@{

  Name = "MyCred01WIN";

  Fields = [pscustomobject]@{

    f01 = "First field";

    };

  cred = (New-Object System.Management.Automation.PSCredential "MyDomain\MyUser01",

    (ConvertTo-SecureString "MyPassword01" -AsPlainText -Force));

}, [pscustomobject]@{

  Name = "MyCred02AWS";

  Fields = [pscustomobject]@{

    ASWFLD01 = "1";

    ANotherAWSfield = "2";

    };

  AccessKey = "MyAccessKey02";

  SecretKey = "MYSecretKey02";

})

Figure - Emulating $PropelCredentials variable in your script.

## Execution History and Results

For this set of entities just would like to comment a few things. In order to not impact application performance, the execution steps property in the “ExecutionHistory” entity need to be lazy populated and only if required.

Also “ExecutionTargets” and “ExecutionErrors” must be embedded documents in MongoDB collection for the same reason and also for his immutability.

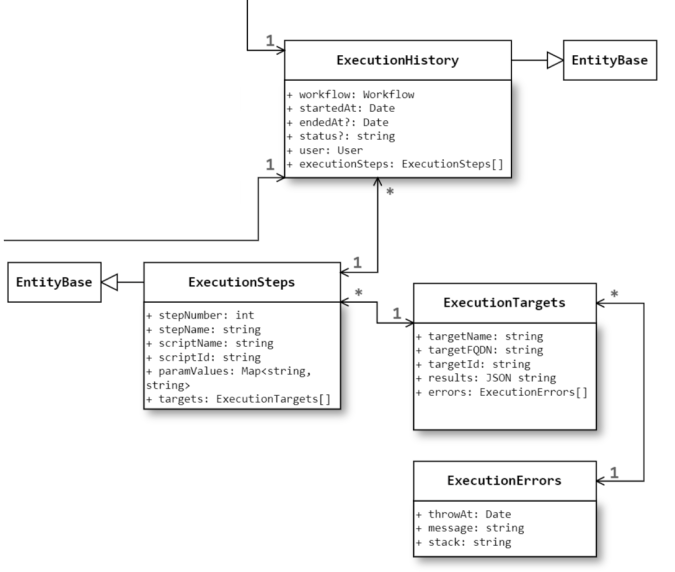


Figure - Execution History and Results

1. WinRM stands for [Windows Remote Management](https://docs.microsoft.com/en-us/windows/win32/winrm/portal). [↑](#footnote-ref-1)
2. Just for a glimpse: [Avoid Soft Deletes](https://ayende.com/blog/4157/avoid-soft-deletes) by Oren Eini, [Don’t Delete – Just Don’t](http://udidahan.com/2009/09/01/dont-delete-just-dont/) by Udi Dahan and the interesting [Deleting Data Is Not a Recommended Practice](https://www.infoq.com/news/2009/09/Do-Not-Delete-Data/) by Abel Avram where both positions are compared. [↑](#footnote-ref-2)