Zürcher Hochschule
für Angewandte Wissenschaften

zh
aw

Finanzen & Services

Password Letter PDF

Using itextsharp .Net library via PowerShell

Project name: PW Letter AGPL
Project leader: Philipp Hungerbühler

Document date: 25. October 2013

Inhalt

1 Introduction

At the ZHAW accounts are created through an automated process. The corresponding encrypted one-time-password and some metadata are stored in an SQL database. Later on HR staffer can create corresponding PDF password letters on request and provide them to end users.

The ZHAW uses the Adaxes Active Directory management and automation tool (www.adaxes.net) to provide certain AD management capabilities via a web interface to authorized users. This tool allows creation of different AD management web interfaces for specific user groups.

The task was, to extend the default Adaxes functionality with a non-standard functionality, allowing authorized users (HR) to request password letters in order for them to provide the password letters to end users.

The needed functionality was implemented through a PowerShell module using the AGPL version of the itextsharp library (www.itextpdf.com). As requested by the license the Powershell module is released under the AGPL license. This document describes the PowerShell module and illustrates its usage through a small test PowerShell script.

While implementing the needed functionality in PowerShell it became evident, that there is no good example or description, how to use itextsharp in PowerShell. Most of the found examples focus on using the itextsharp library in a C# application or ASP.Net web. We hope that the provided library and the example code can be used by others, when trying to create PDFs in PowerShell using itextsharp.

2 AGPL License

This program is free software; you can redistribute it and/or modify it under the terms of the GNU Affero General Public License version 3 as published by the Free Software Foundation with the addition of the following permission added to Section 15 as permitted in Section 7(a): FOR ANY PART OF THE COVERED WORK IN WHICH THE COPYRIGHT IS OWNED BY 1T3XT, 1T3XT DISCLAIMS THE WARRANTY OF NON INFRINGEMENT OF THIRD PARTY RIGHTS.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Affero General Public License for more details. You should have received a copy of the GNU Affero General Public License along with this program; if not, see http://www.gnu.org/licenses or write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA, 02110-1301 USA, or download the license from the following URL: http://itextpdf.com/terms-of-use/

The interactive user interfaces in modified source and object code versions of this program must display Appropriate Legal Notices, as required under Section 5 of the GNU Affero General Public License.

In accordance with Section 7(b) of the GNU Affero General Public License, you must retain the producer line in every PDF that is created or manipulated using iText.

You can be released from the requirements of the license by purchasing a commercial license. Buying such a license is mandatory as soon as you develop commercial activities involving the iText software without disclosing the source code of your own applications. These activities include: offering paid services to customers as an ASP, serving PDFs on the fly in a web application, shipping iText with a closed source product.

3 Restriction

The provided PowerShell module is an extension implemented for using with the Adaxes Active Directory management and automation tool. The PowerShell module itself does not use or rely on Adaxes functionality and can also be used independently. However the PowerShell module is tailored to the need of the ZHAW and may need adjustment when used in other environments. The PowerShell module is provided as it is, as free code and can be used or adapted as needed. The ZHAW takes no responsibilities and does not provide any support for this PowerShell module.

4 Basic requirements

The following requirements had to be fulfilled be the PowerShell module:

- Provide the possibility to filter the available password entries by start date, end date and a filter term.
- Limit the returned password letters depending on the requesting user's business unit (department).
- For certain cases the requested list of password letters may be quite big (> 3000 entries, resulting in >6000 PDF pages). The way non-standard functionality is implemented in Adaxes it was not possible to let the user wait until the needed PDF output file has been created (timeout). Additionally there is no way to provide the user with a download link. Therefore the PDF's need to be created in an asynchrony task and the resulting PDF is then sent to the requesting user via email.
- The size of the resulting output file should allow sending via email (size restriction).
- Logging should be added, to allow tracing of the different requests.

5 Basic concept

The basic concept used in the PowerShell module can be described as follows:

- Initially a temple of the password letter was created as a Word file.
- This word file was converted to PDF and opened in Adobe Acrobat.
- For each dynamic data part a PDF input field was created in Adobe Acrobat, moved to the needed location and named as required. The resulting PDF file is stored in the same location as the PowerShell module.
- The PowerShell module exposes a DB function allowing retrieval of the password and meta-information from the database, limiting the search by date/time range and a filter value.
- Based on the filter values and the access rights of the requesting user, the corresponding password records and additional data are requested from the database.
- The resulting list of accounts is provided to the functionality implementing the PDF production. In our example the production of the PDF file is done as an asynchrony task.
- The function producing the PDF loops through the account data and creates a PW letter for each entry by opening a template file, filling in the form fields, compressing/flattening the PDF and adding the resulting PDF to the output file.
- In order to keep the resulting PDF small several itextsharp options must be set else the resulting PDF gets very big and cannot be sent via email.

6 Module functions

The modules PowerShell code is commented in details. Therefore the different functions are described very short.

6.1 AGPLGeneratePWLetters

This module contains all functions needed to access the database, request the list of passwords and metadata, read the PDF templates, fill in the PDF form fields, flatten the PDF and add it to the output file. The resulting output file is then sent to the requesting user. This module depends on the iTextSharp dll and the also provided module AGPLogging.

6.1.1 Get-AGPLPasswordLettersDataFromDB

This function is called with the different filter values and requests the needed data from the SQL database. It returns an object containing detailed results about the operation.

6.1.2 Generate-AGPLCreatePWLettersAndSend

This function can be called in an asynchrony task. As input it gets the list of accounts found (returned by Get-AGPLPasswordLettersDataFromDB), creates the needed PDF PW letters and sends a mail to the recipient of the PW letter list.

6.1.3 Generate-AGPLPasswordLettersFromList

This is actually the function producing the PDF file based on the passed list of accounts and the available PDF templates. It returned a PDF containing all generated password letters. The contained PDF pages are compressed (means the PDF form fields contained in the template are stamped).

6.1.4 Get-AGPLPasswordLetterExpandFileName

This is a helper function giving access to the mail templates which are also stored in the same location as the PowerShell module.

6.1.5 Write-AGPLPasswordToStore

This function allows writing a password to the database store. It is used, when the password of a given account is reset and the new password needs to be added to the PW store.

6.2 AGPLogging

This is an additional small PowerShell modules used for writing logs or producing debug output.

6.2.1 Write-AGPLLog

This logging functionality is called with an error type, and up to three message strings, which are then written to the log file.

6.2.2 Write-AGPLDebug

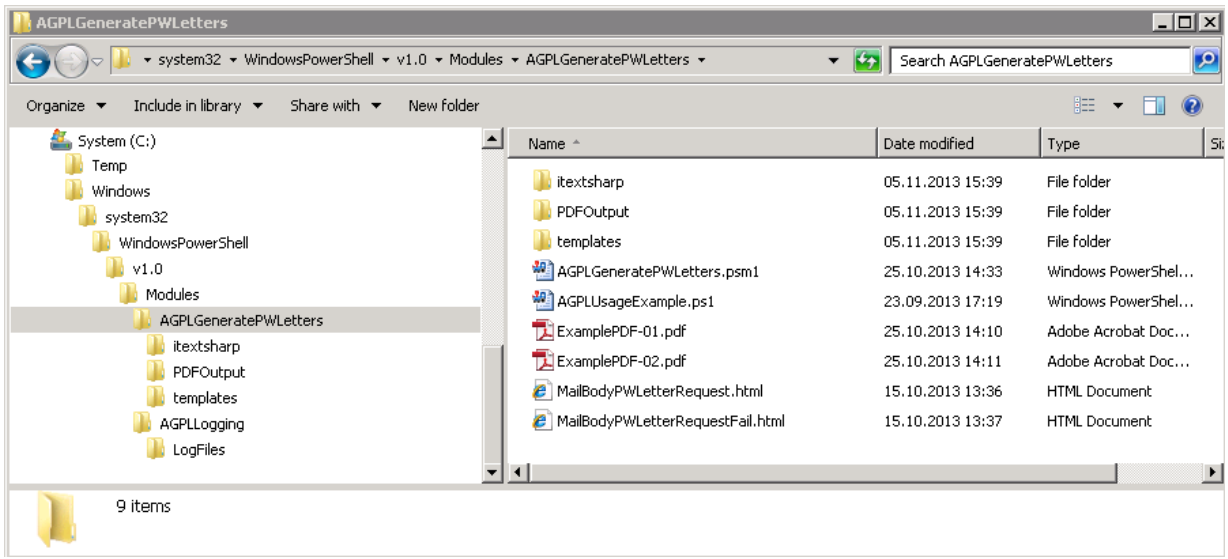
A similar function as the logging one, but it requires just one message string, which is written to the debug file.

7 Installation

In order for both modules (AGPLGeneratePWLetters and AGPLogging) to be easily used they are copied to the default PowerShell modules folder. On Windows systems this is:

- C:\Windows\System32\WindowsPowerShell\v1.0\Modules

The folder structure should look as follows:



The folders are shortly described in the following list.

Folder	Content	Description
AGPLGenerate PWLetters	Itextsharp (folder)	Contains the itextsharp.dll library file.
	PDFOutput (folder)	This is the output folder for the created PDF files. In order for the function to be able to write to this folder the user executing the function needs write permissions on this folder.
	Templates (folder)	This folder contains some template files used to create the PDF template files and the different mail bodies.
	AGPLGenerate PWLetters.psm1	The actual PowerShell module.
AGPLLogging	LogFile (folder)	This is the folder where the actual log and debug files are created.
	AGPLLogging.psm1	The actual PowerShell module.

There are no more prerequisites in order to use the two PowerShell modules. However as stated above the module AGPLGeneratePWLetters has some dependencies on the infrastructure. For easier testing an example script using the module and a PW database is provided. See the next chapter.

8 Example

The PowerShell module AGPLGeneratePWLetters was tailored to the needs and the infrastructure at the ZHAW. In order to be able to see the module working and test the itextsharp library some additional elements are needed.

8.1 SQL Password database

As described above the module relies on the passwords and related information to be stored in a Microsoft SQL database. The current implementation uses Microsoft SQL 2008, however the example should also work on version 2003 or 2005.

An empty template database can be created by using the script **CreateDB.sql**. An adjustment of the storage location of the database file (line 5 and 7 of the script) may be needed. Else the database files are created in the following location:

- Database file: F:\MSSQL\DB\AUMPWLetterStoreAGPL.mdf
- Transaction log file: F:\MSSQL\DB\AUMPWLetterStoreAGPL.mdf

After adjustment of the storage location the script can be executed and should create an empty database named: AUMPWLetterStoreV2AGPL.

Once the database is created it can be populates with some test data by using the script: **CreateTestData.sql**.

The SQL database should now be ready and contain test data for a test request.

8.2 Example script

Beside the SQL database, the provided example contains the following files:

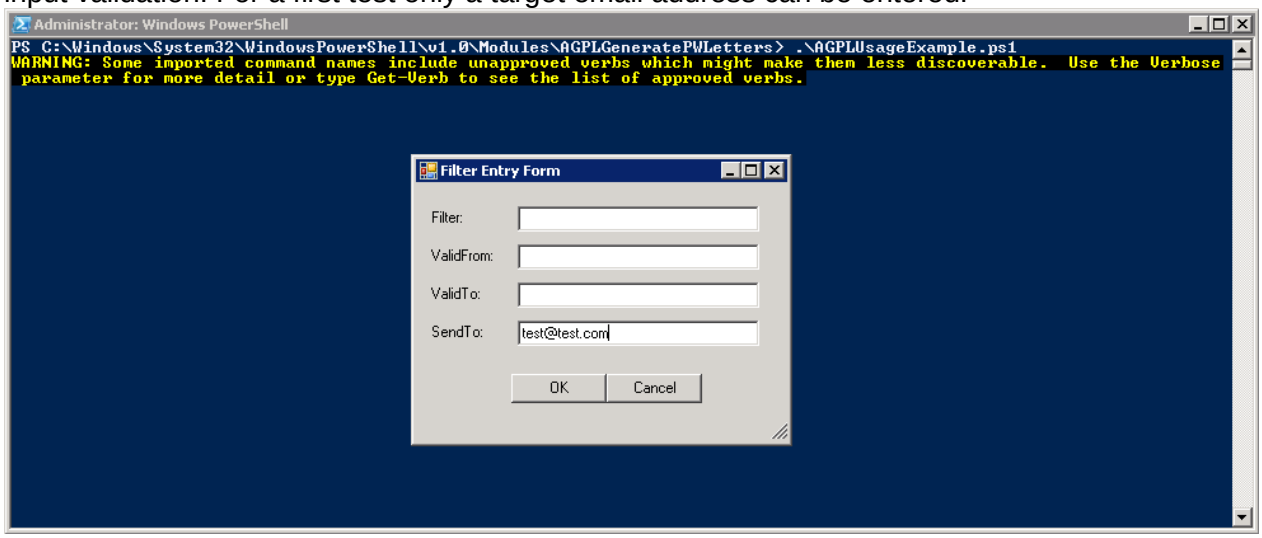
AGPLUsageExample.ps1	The actual example script.
ExamplePDF-01.pdf	A two page PDF file with form fields to be filled out with data from the database.
ExamplePDF-02.pdf	A second tow page PDF file with form fields to be filled out with data from the database.
MailBodyPWLetterRequest.html	The mail body used, when the requested PDF could be generated.
MailBodyPWLetterRequestFail.html	The mail body used, in case there was an error when executing the request.

In order for the script to be able to connect to the database the corresponding connection string needs to be configured in the PowerShell module (this is currently is hardcoded).

In the PowerShell module line 73 needs to be adjusted to match the configuration of the environment:

```
$script:sqlConnectionString = $("Data Source=localhost;
Database=AUMPWLetterStoreV2AGPL2; User ID=AGPLTestUser;
Password=Test.12345")
```

Once this is done the example script can be called by executing ".\AGPLUsageExample.ps1" in a PowerShell command line. The following small dialog will be shown and allows input of a filter, a start and end date, and a target email address. The example script does not do any input validation. For a first test only a target email address can be entered.



This will result in a four page PW PDF file to be created and sent to the target email address using the mail template found in the modules home folder.

8.3 Example PDF and filling in Adobe form fields

In order for the PowerShell module to be able to fill in the PDF template files the PDF form fields need to be defined with names, which are mapped correctly in the PowerShells module code.

```
# iterate through all available fields and define value
$value = ''
foreach($key in $keys){
  switch($Key){
    'ID'                    {$value = $AccountInfo.UniqueID}
    'Name'                  {$value = $AccountInfo.Nachname + ' ' + $AccountInfo.Vorname}
    'CurrentDate'           {$value = $(Get-Date -Format dd.MM.yyyy)}
    'LastName'              {$value = $AccountInfo.Nachname}
    'FirstName'             {$value = $AccountInfo.Vorname}
    'SAMAccountName'        {$value = $AccountInfo.KurzZeichen}
    'Class'                 {$value = $AccountInfo.AnlassEvento}
    'Password' {
                                    if ($AccountInfo.PWIsValid){
                                        $value = $AccountInfo.PasswordDecrypted
                                    } else {
                                        $value = 'The password was already changed by the user!'
                                    }
                                }
    default {
      $value = 'UNKNOWN'
      Write-AGPLLog "W" "psPWLetter" "Found unknown AcroField in template PDF" $key
    }
  }
  $result = $form.SetField($key,$value)
  Write-AGPLDebug "Looking for key $($key) in PDF, assigning value: $($value)"
}
```

This code maps to the following form fields in the PDF template.

ID	ID
Name	Name
CurrentDate	CurrentDate
LastName	LastName
FirstName	FirstName
Class	Class
SAMAccountName	SAMAccountName
Password	Password

8.4 PDF file size

On requirement was to create small PDF output files in order to be able to send the files via email. Finding the correct configuration settings for the iTextSharp library proved to be difficult. Once the correct settings were found, a reduction from 300MB to 2MB could be observers.

- It is important to set the corresponding settings before starting production of the output PDF. Else the settings will not apply.
- Use an iTextSharp.text.pdf.pdfstamper to write (stamp) the content of the PDF form fields. Use the following settings:
  - $stamper.SetFullCompression()
  - $stamper.FormFlattening = $true
- For the PDFReader used to access the template PDF use
  - $pdfReader.RemoveUnusedObjects

  Before after filling the form fields, but before adding the new page to the output template.
- If the flush operation is not used the output PDF will be fully put together in the servers RAM. In case of big PDF files and multiple requests this may be a problem. Therefore changes should be flushed to disk after a certain amount of pages have been added to the output PDF.
  - $stream.Flush()