Data One – PowerActivity 2010

Installation Guide / Manual

Data One

Saarbrücken, 2015-06-02

Description

This document provides instructions on how to install Data One – PowerActivity 2010 and its features. Apart from the manual itself, a list of frequently asked questions (F.A.Q.) is included. The last section of this guide will then mention limitations that have been observed in the past.

Versions

Version	Date	Author	Changes
1.0	2014-10-14	Data One	Added F.A.Q. / New structure
1.1	2014-11-06	Data One	Changed F.A.Q.
1.2	2015-01-29	Data One	Changed F.A.Q., Windows UAC
1.3	2015-02-24	Data One	Update PowerModules
1.4	2015-05-08	Data One	Registry access not allowed
1.5	2015-05-11	Data One	Predefined variables
1.6	2015-06-02	Data One	ConsoleHost

Table of Contents

1. Installation and Configuration

1.1. PowerActivity

1.1.1. Requirements

- **.NET 3.5**
- **Microsoft SharePoint 2010 Server**
- **Nintex Workflow 2010**
- Microsoft PowerShell 2.0
- Data One Licensing Feature 2010

1.1.2. Data One Licensing

This step is only required if Data One Licensing has not been installed yet.

- Copy the "DataOne.SharePoint.Licensing.wsp" to a SharePoint Server belonging to your farm e.g. "C:\SPSolutions\".
- Start SharePoint Management Shell.
- Execute the following PowerShell commands. Note that there will be a restart of application pools which will cause a downtime during restart.
- `Add-SPSolution "C:\SPSolutions\DataOne.SharePoint.Licensing.wsp"`

```
Install-SPSolution -Identity "DataOne.SharePoint.Licensing.wsp" -GACDeployment
```

1.1.3. PowerActivity for Nintex Workflow 2010

- Copy "DataOne.Nintex.PowerActivity.wsp" to a SharePoint Server belonging to your farm e.g. "C:\SPSolutions\".
- Start the SharePoint Management Shell.
- Execute the following PowerShell commands. Note that there will be a restart of application pools which will cause a downtime during restart.
- `Add-SPSolution "C:\SPSolutions\DataOne.Nintex.PowerActivity.wsp"`

```
Install-SPSolution -Identity "DataOne.Nintex.PowerActivity.wsp" –GACDeployment
```

1.1.4. Activating Web Application Feature

Before you can use PowerActivity on a SharePoint site, you first have to activate a web application feature called PowerActivity for Nintex Workflow 2010. This needs to be done once for each web application that you want to use in future.

It is very important to choose the *SharePoint Central Administration* as the structural level on which to activate this feature. Otherwise you will not able to configure PA2010.

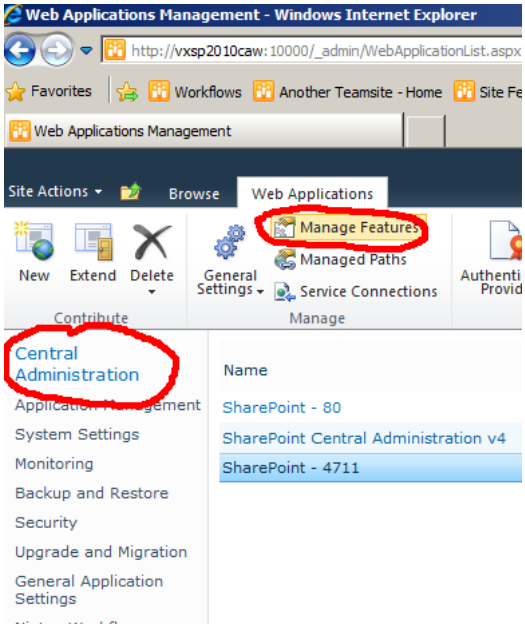


Figure 5: PowerActivity – Activating the Web App feature

1.2. Uninstalling PowerActivity for Nintex Workflow 2010

- Start the SharePoint Management Shell.
- Execute the following PowerShell commands. Note that there will be a restart of application pools which will cause a downtime during restart.

```
Uninstall-SPSolution -Identity "DataOne.Nintex.PowerActivity.wsp"
Remove-SPSolution -Identity "DataOne.Nintex.PowerActivity.wsp"
```

1.3. Configuring PowerActivity for Nintex Workflow 2010

1.3.1. License

- Open the Central Administration of SharePoint 2010.

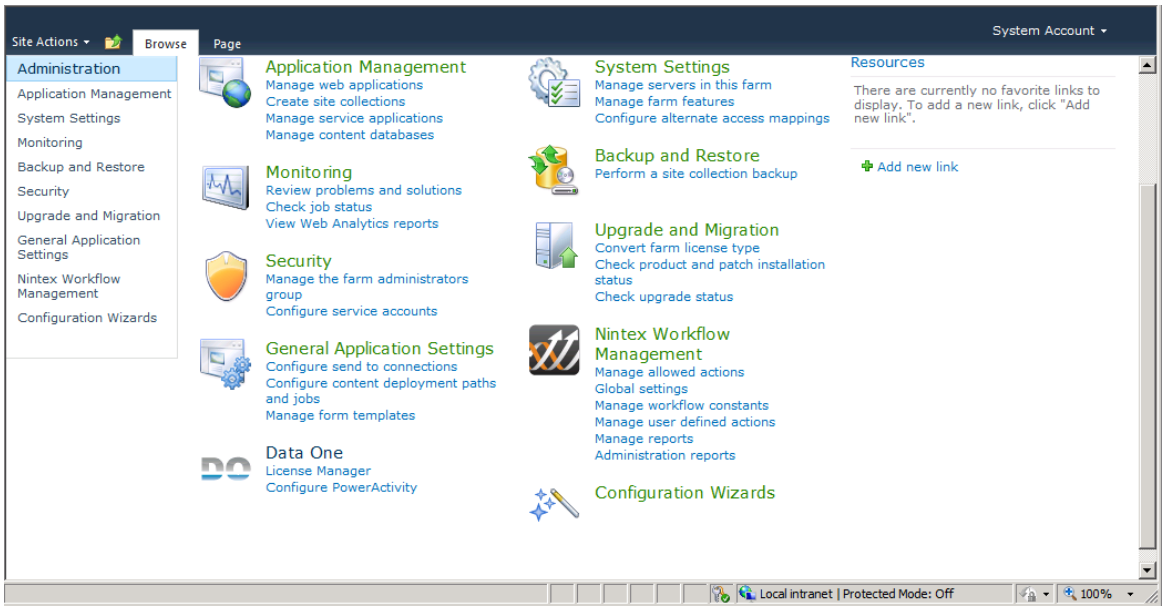


Figure 1: Central Administration with Data One entries

- In the section Data One, click on License Manager.

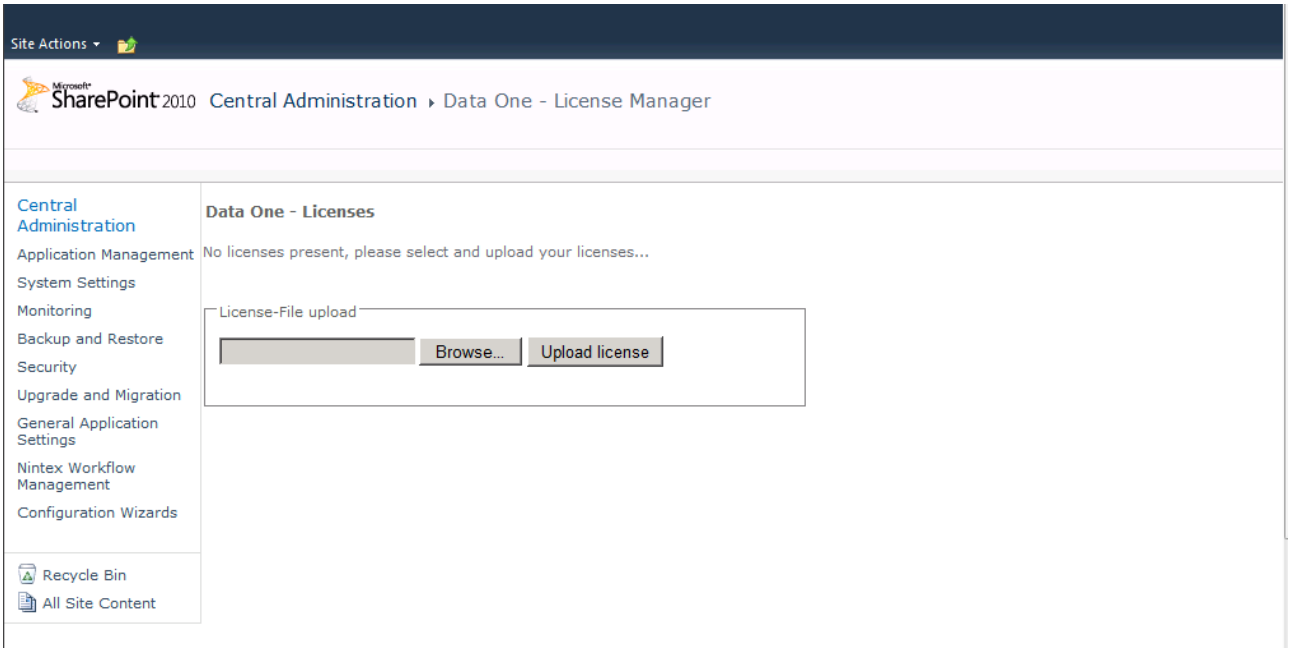


Figure 2: License Upload Dialog

- Click the Browse-Button and choose your license file.
  - Click the Upload button.
  - Once the license has been successfully imported, a page is displayed that shows the relevant product details.

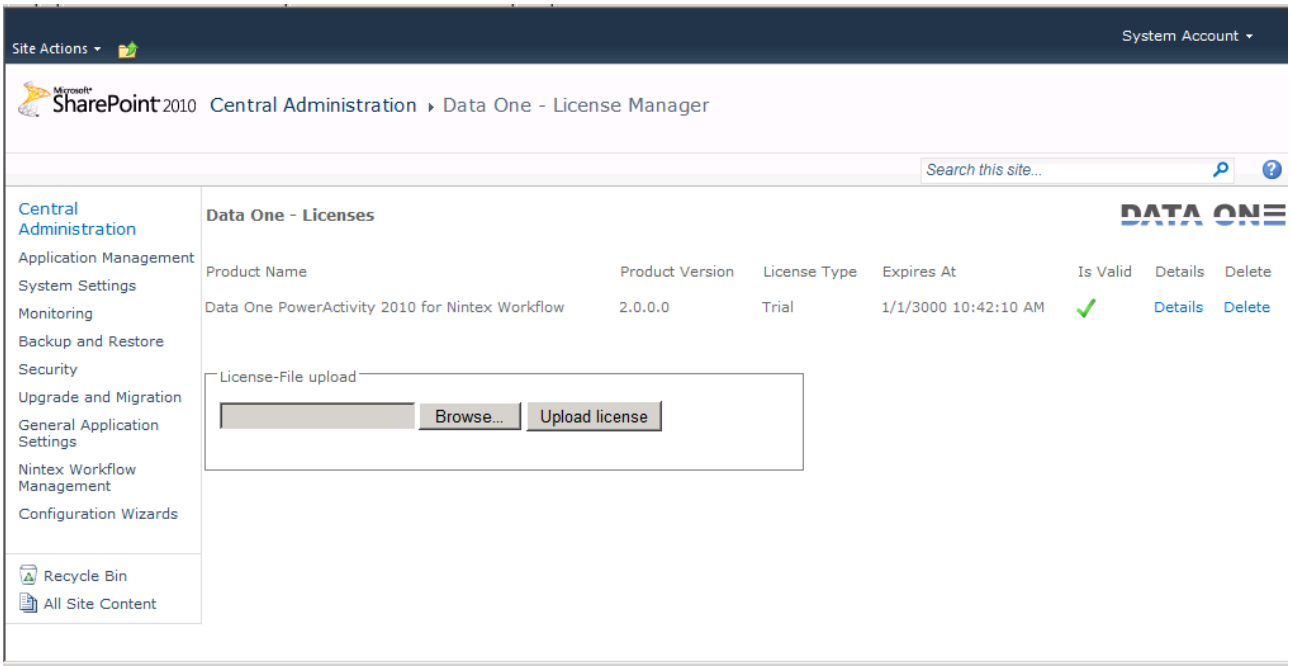


Figure 3: License Dialog with a valid key file

1.3.2. Settings

- In order to configure your PowerActivity for Nintex Workflow 2010, go to *Central Administration* -> *Data One* -> *Configure PowerActivity*.

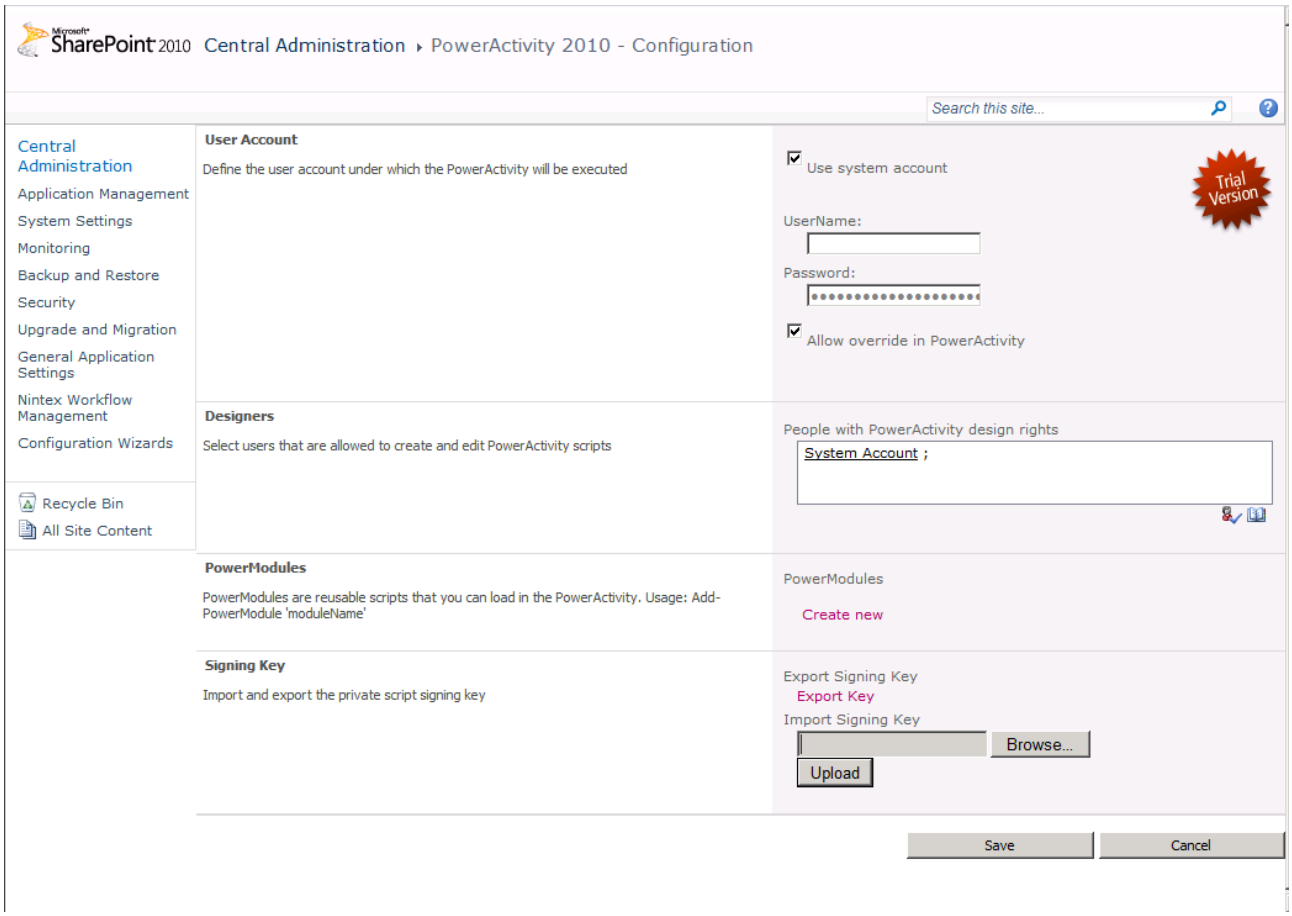


Figure 4: PowerActivity settings

- In the section User Account, you can define the windows credentials (“Domain\user”) under which the PowerActivity will be executed. To execute SharePoint CMDLETS, the user must have all the privileges that are relevant to the SharePoint object model and to the SharePoint databases. Use Add-SPShellAdmin to change user privileges if you decide to choose a different account.

- In the section Designers**,** the users or groups are defined who are allowed to design and publish workflows that contain PowerActivity components. **The system account should NOT be chosen in this context.**

- In the next section, called PowerModules, some additional modules can be selected to be used within PowerActivity. For example, you may want to choose a module that sets variables and values that are frequently used. If so, just enter *Add-PowerModule '#MODULENAME#'* into the PowerShell Script Mask. Please replace #MODULENAME# and insert the name of the module in question.

- The Signing Key functions as the signature of the workflows on the SharePoint server. This ensures that the imported workflows and custom actions run on the correct machine. You can import and export keys from or to

another system to copy workflows or custom actions to a different target system such as from a test system to a productive system after the completion of a test period.

1.4. Checking the Installation and Configuration while Nintex Workflow Designer is Open

- Create a new Nintex Workflow and take a look at the workflow action category Data One. Ideally, there should be an action called PowerActivity.

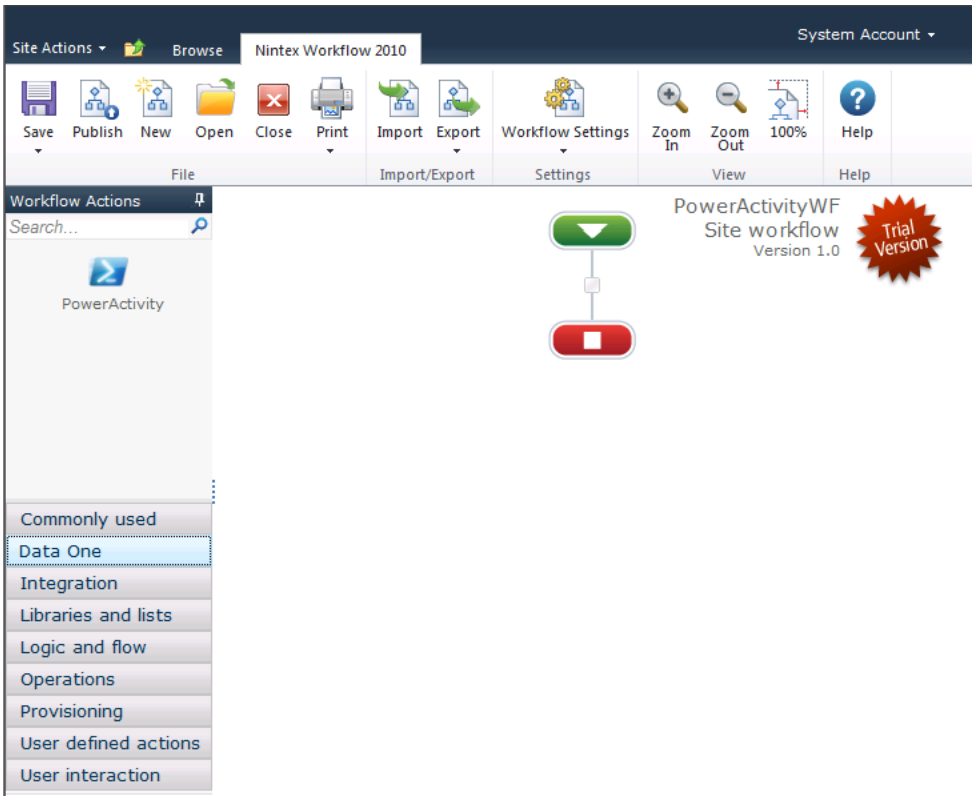


Figure 6: Nintex Workflow Designer

- Insert the PowerActivity action and start configuration.

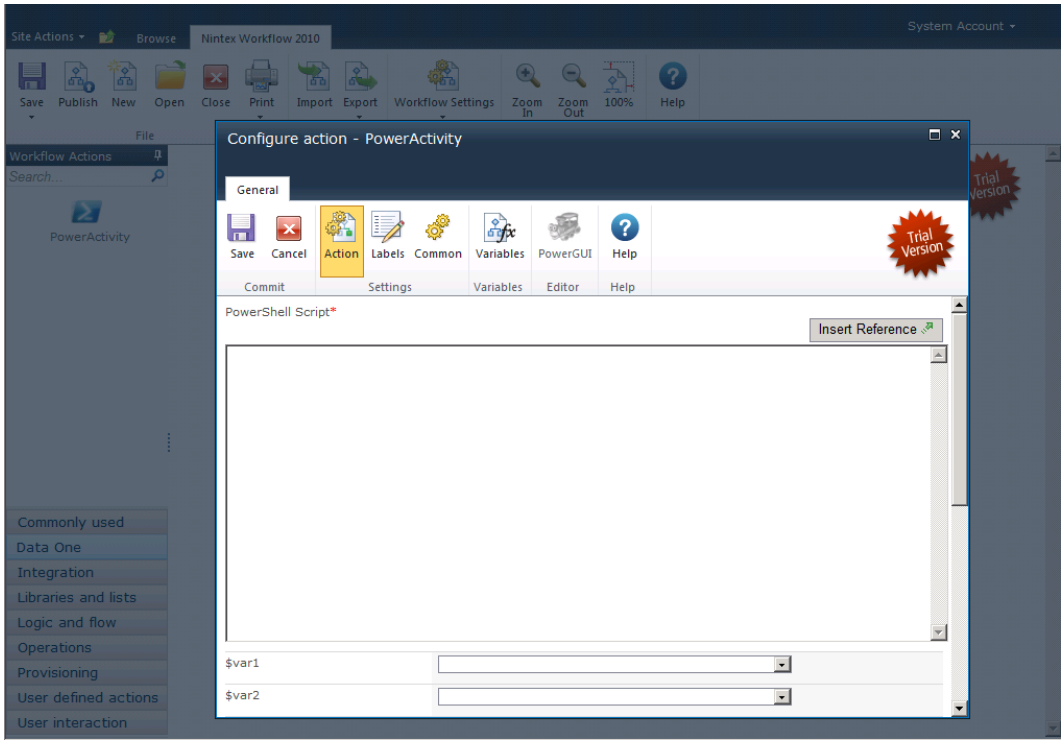


Figure 7: Power Activity - Workflow action configuration

- If the PowerActivity configuration dialog above appears, the installation of Data One PowerActivity has been successful.

1.5. Optional: Installing PowerGUI Integration for PowerActivity

Within Nintex Workflow Designer, is it possible to use PowerGUI as a script editor inside PowerActivity. In so doing, you can easily edit PowerModules.

The following installation steps are then necessary for every individual **client** who needs PowerGUI to be available:

The first step is to download the necessary files:

- Quest PowerGUI (http://www.powergui.org)
- iLoveSharePoint PowerGUI Launcher (http://ilovesharepoint.codeplex.com/releases/view/20019)

1.5.1. Installing PowerGUI on the Client

- Download the current version from http://www.powergui.org
- Start the Setup Wizard and click on Next.

Figure PGUI 1: Installing Wizard PowerGUI

- Read the license agreement. Click Yes to accept.

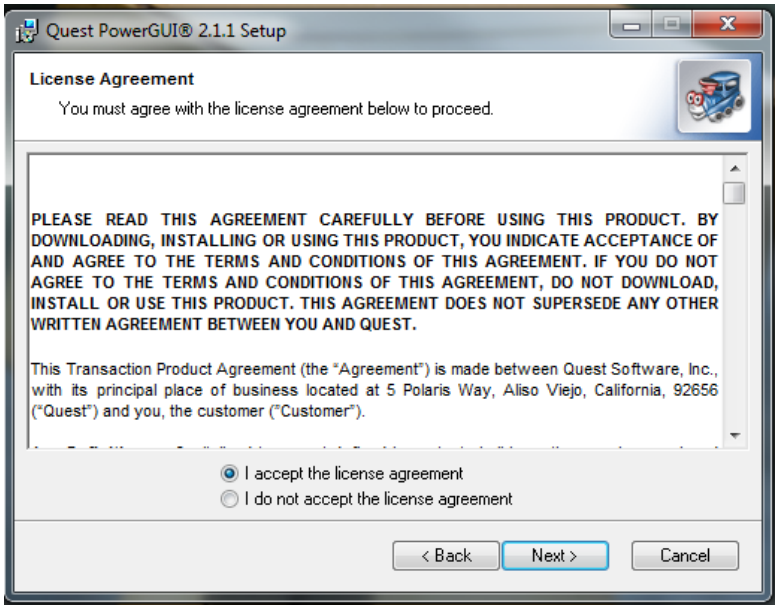


Figure PGUI 2: License Agreement

- Next, select the features you wish to install. Exchange and Active Directory Features can only be installed if they are available on the local system. Click Next.

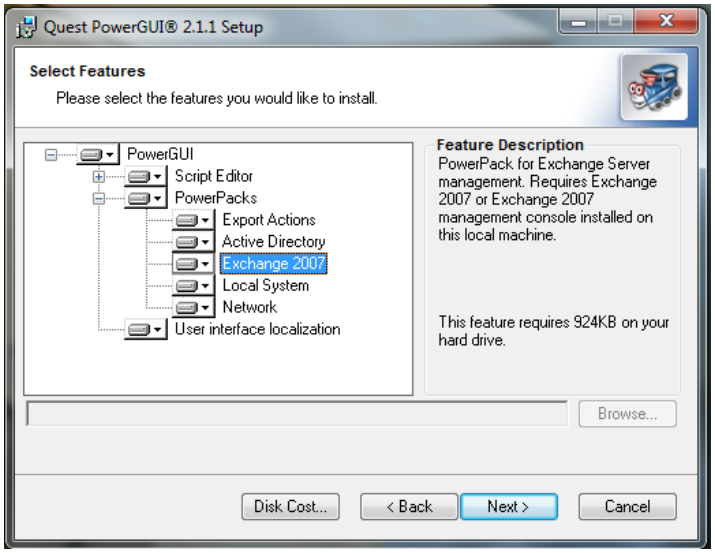


Figure PGUI 3: Feature list

- Now, it will be checked whether all relevant requirements are met. If so, you press the next button and start the installation process.

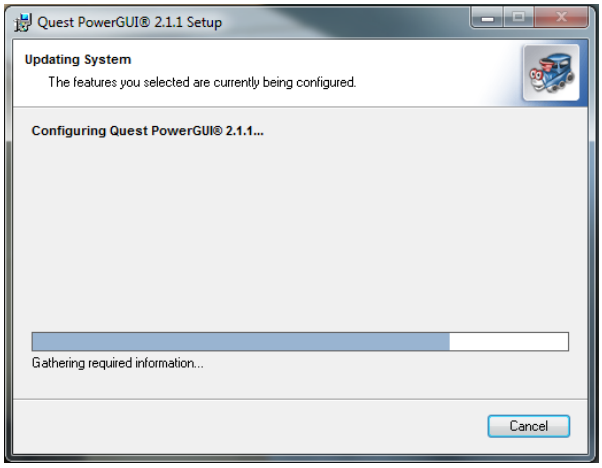


Figure PGUI 3: Power GUI installation process

- After completion of the installation, click Finish. PowerGUI is now installed.

Figure PGUI 4: Power GUI - Installation completed

1.5.2. Installing iLoveSharePoint PowerGUI Launcher on the Client

- Download the current version at http://ilovesharepoint.codeplex.com/releases/view/20019
- Extract the archive.
- Start the Setup file. Afterwards, click Next.

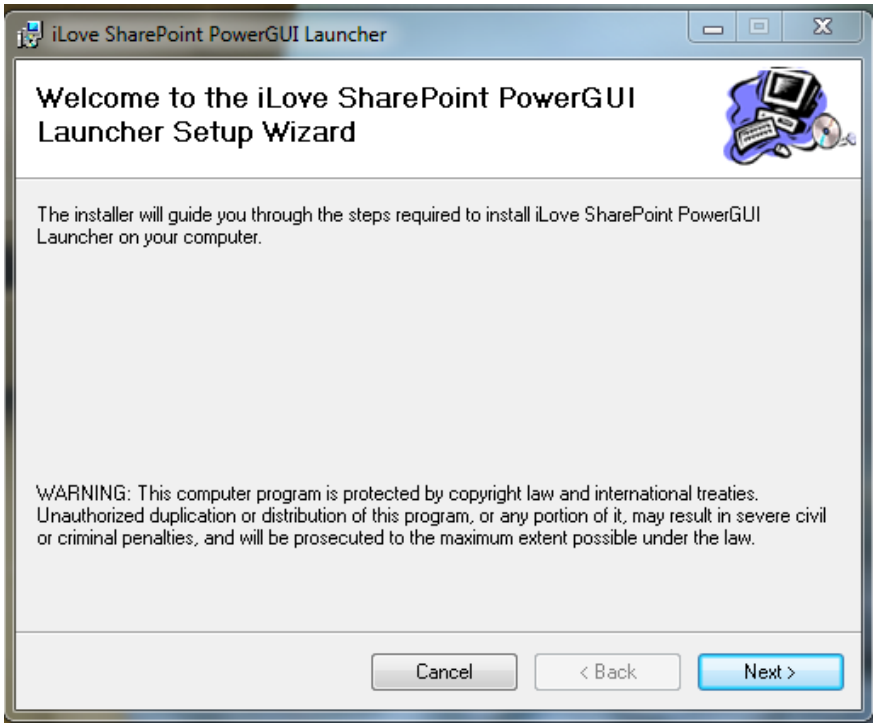


Figure PGUI 5: Power GUI – Launcher setup wizard

- Select an installation folder and define a user scope. Click next.

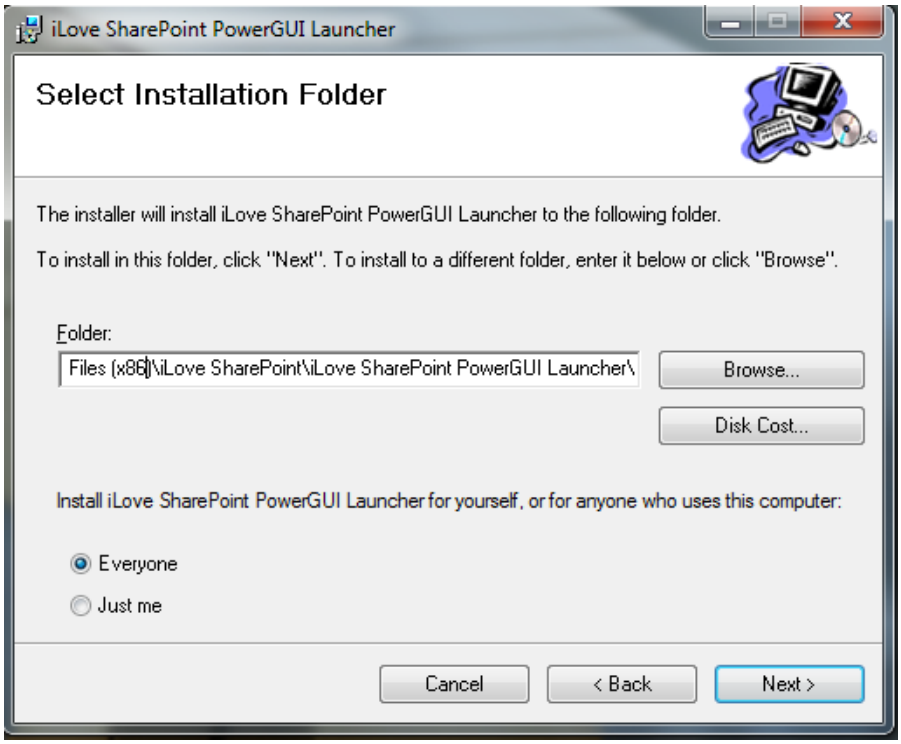


Figure PGUI 6: Power GUI – Selecting an installation folder

- Now, it will be checked whether all relevant requirements are met. If so, you press the next button and start the installation process.

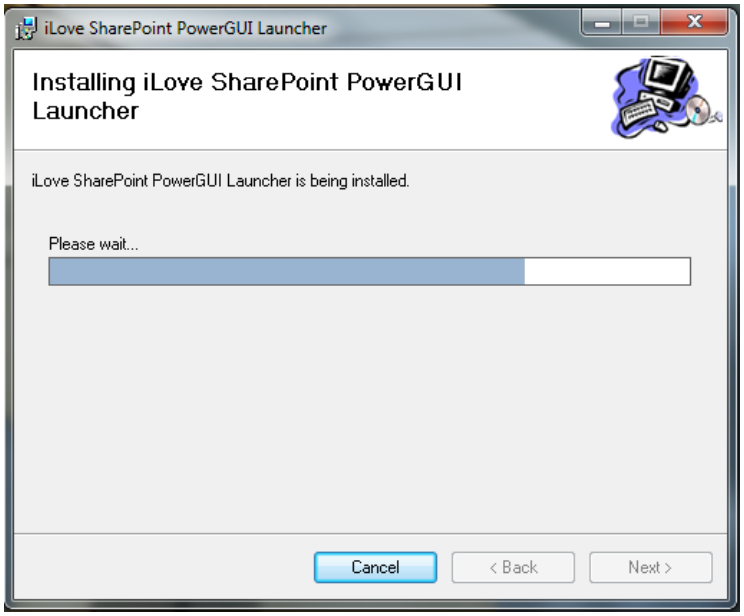


Figure PGUI 7: Power GUI – Installation progress

- After completion of the installation, click Close. iLoveSharePoint-PowerGUILauncher is now installed.

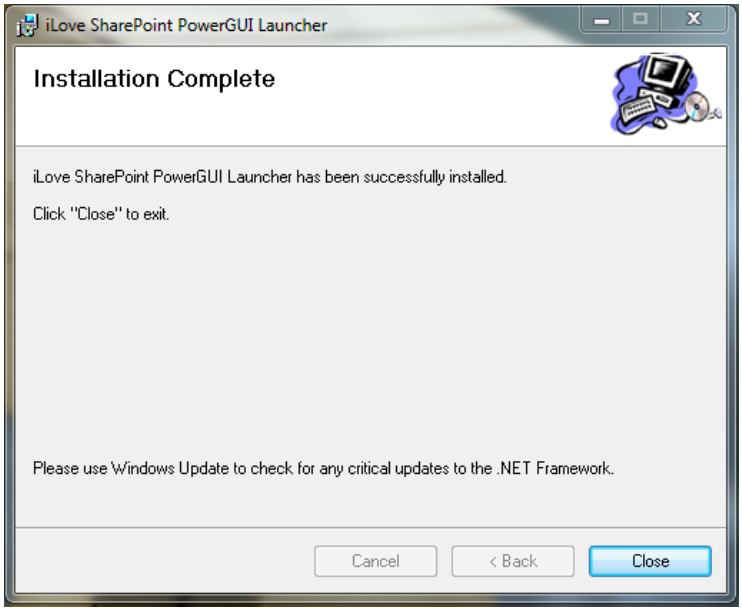


Figure PGUI 8: Power GUI – Installation completed

1.5.3. Checking PowerGUI PowerActivity Integration

- Connect to SharePoint where PowerActivity is installed.
- Start Nintex Workflow Designer.
- Create a new workflow.
- Add the PowerActivity to your workflow and start configuration.
- Click the PowerGUI button in the ribbon menu.

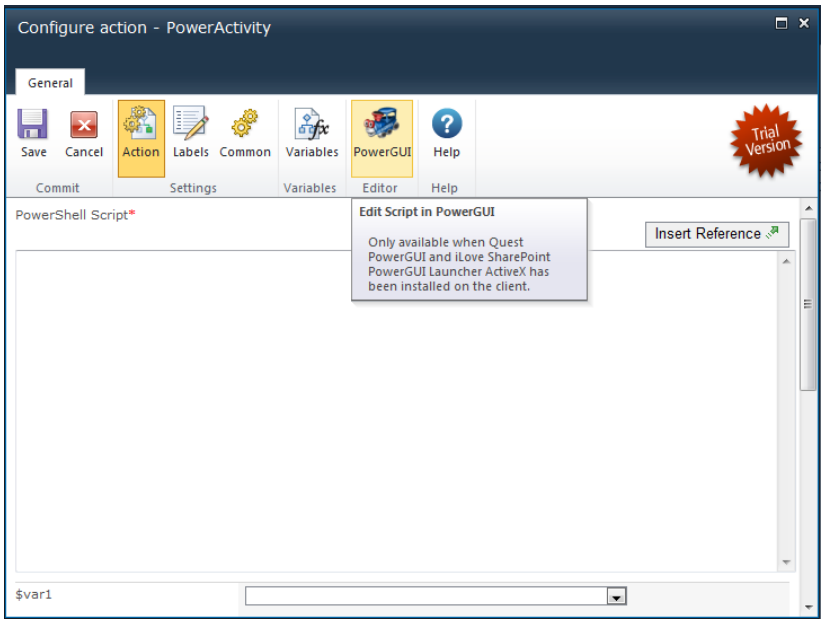


Figure PGUI 8: PowerActivity menu with PowerGUI integration

- PowerGUI opens. Insert a script, press save and close PowerGUI. The script you have just inserted should be shown in the PowerActivity now.

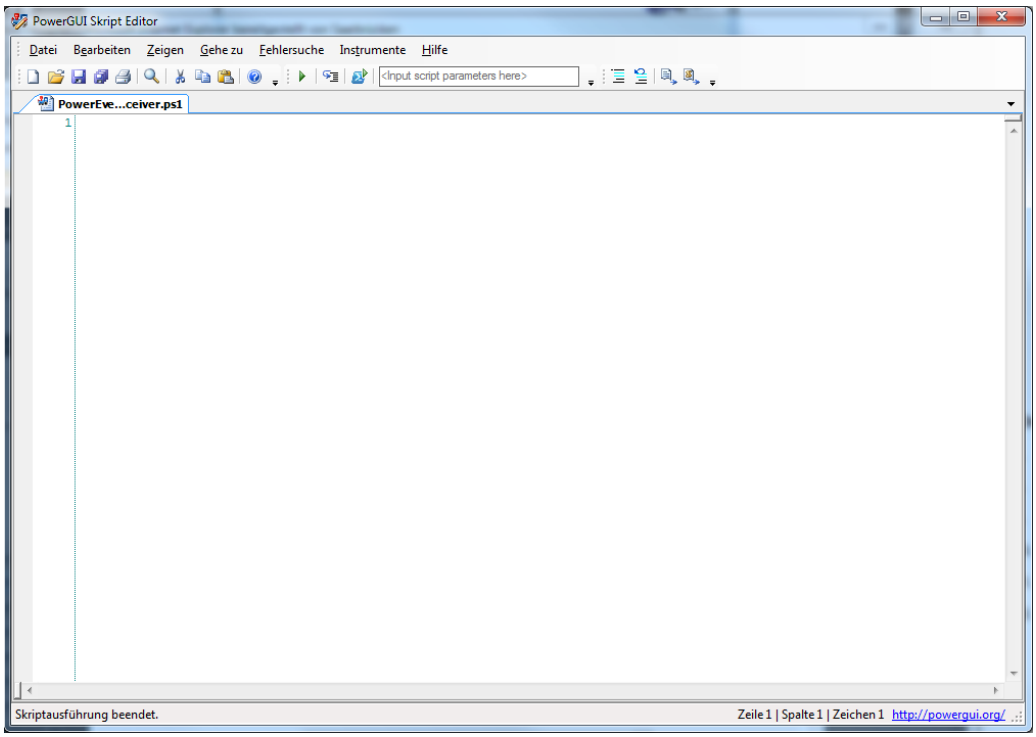


Figure PGUI 9: PowerGUI script editor

2. Manual: PowerActivity for Nintex Workflow 2010

Data One PowerActivity has been developed to enable users to easily exploit the opportunities inherent to PowerShell within Nintex Workflow. Once PowerActivity has been successfully installed, it can be used like any other Nintex Workflow action. To do so, just drag the PowerActivity action to your workflow and configure it. The following manual will cover key aspects of using PowerActivity and its features.

2.1. How to Use Data One PowerActivity

PowerActivity is quick and easy to apply: Open the Nintex Workflow Designer and open or, if needed, create a workflow of your choice. Choose the category Data One and drag and drop the PowerActivity action to the workflow.

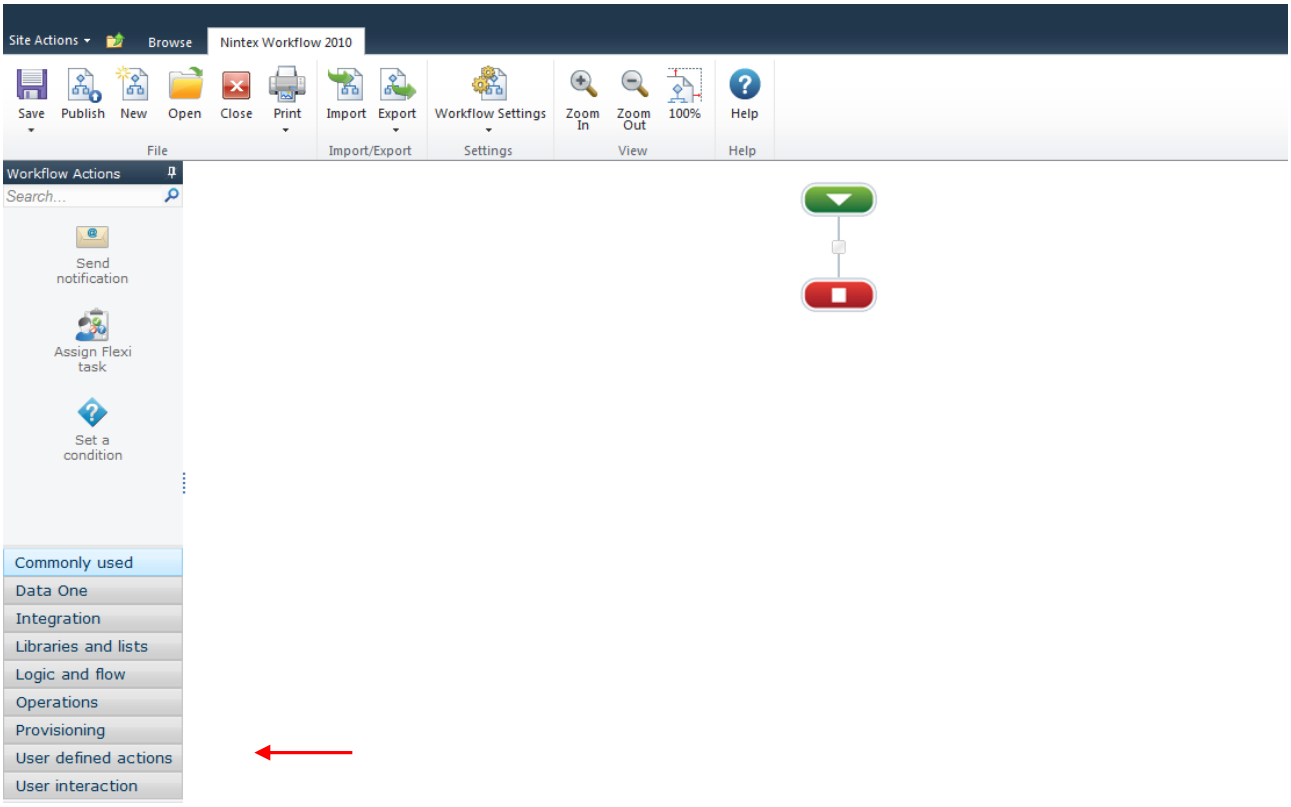


Figure 8: Nintex Workflow Designer with Data One PowerActivity

Now, you will have to configure the workflow action. To do so, double click on the action or, alternatively, select Configure from the menu in the right upper corner.

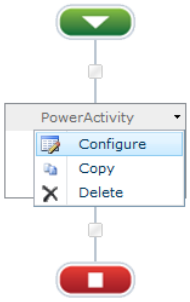


Figure 9: Open configuration

The configuration mask below will open and you can start configuring your PowerActivity action.

2.2. The Configuration Mask

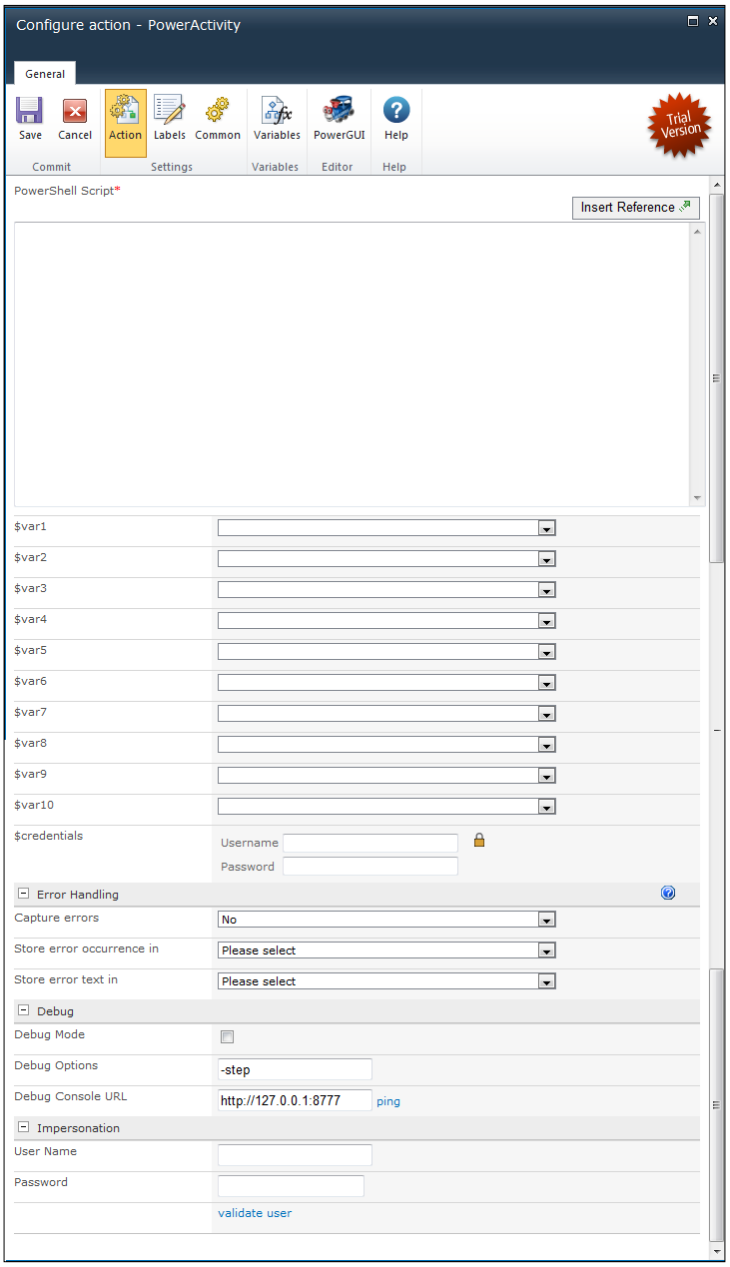


Figure 10: PowerActivity – Configuration mask

2.2.1. The Ribbon Menu

The Ribbon Menu consists of eight buttons that correspond to main functions.

Button Save	The Save button as the first button is for saving the current configuration.
Button Cancel	Pressing the next button, the Cancel button, the configuration mask is closed without saving the changes that have been made so far.
Button Action	The Action button contains the main configuration of the activity. Chapter 2.3 to 2.3.9 will cover various aspects in more detail.

Button Labels	Fourth, Labels allows users to define labels which will be displayed in specified places in the action. These labels are visible in the workflow itself and facilitate communication as things are more transparent.
Button Common	Under Common, a log message can be left. It will be listed in the workflow history if the workflow is over. In the section Expected duration, you can enter the time you expect this action to take until completion. This information is used for statistics and reports.
Button Variables	The button Variables enables users to define workflow variables.
Button Power GUI	If PowerGUI has been installed as written in Chapter , this button opens the Quest Power GUI.
Button Help	Clicking on the Help button, general information regarding SharePoint 2010 can be found.

2.2.2. PowerShell Script

In the script dialog, you can define the PowerShell script you want to run at this point of your workflow.

If you frequently use distinct scripts, you have the possibility to choose predefined modules. You can use these PowerModules by typing *Add-PowerModule '#ModuleName#'*. #ModuleName# is a place holder for the name you specified in the configuration of the PowerActivity (Location: *Central Administration* -> *Data One* -> *Configure PowerActivity*).

For more information, check the example in Chapter 3.8.

2.2.3. Insert Reference

Clicking on the Insert Reference button, you can choose some common variables, inline functions and workflow specific variables. To add one of these to the PowerActivity, just double-click on it and press OK.

Insert Reference is not supported in User Defined Actions (UDAs).

Security note: The usage of Insert Reference may potentially lead to an increased vulnerability towards script injection attacks. However, the usage of variable binding is secure.

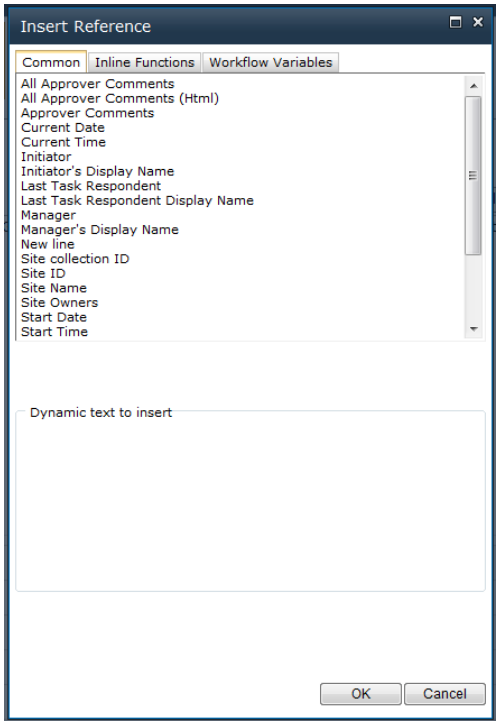


Figure 11: Insert Reference dialogue

2.2.4. Variables

Inside of PowerActivity, you can use a set of ten variables to link your PowerActivity variables to the global workflow variables. The values you assign must be serializable. If they are not, the workflow will fail. The variable binding is bidirectional, which means you can read and modify the values of the variable. In every dropdown field, you can choose the variable you specified before as a workflow variable.

Example 1: Set Workflow Variables within PowerActivity

- $var1 = Get-Date
- $var2 = "Hello Variable Binding"
- $var3 = $web.Title
- ….

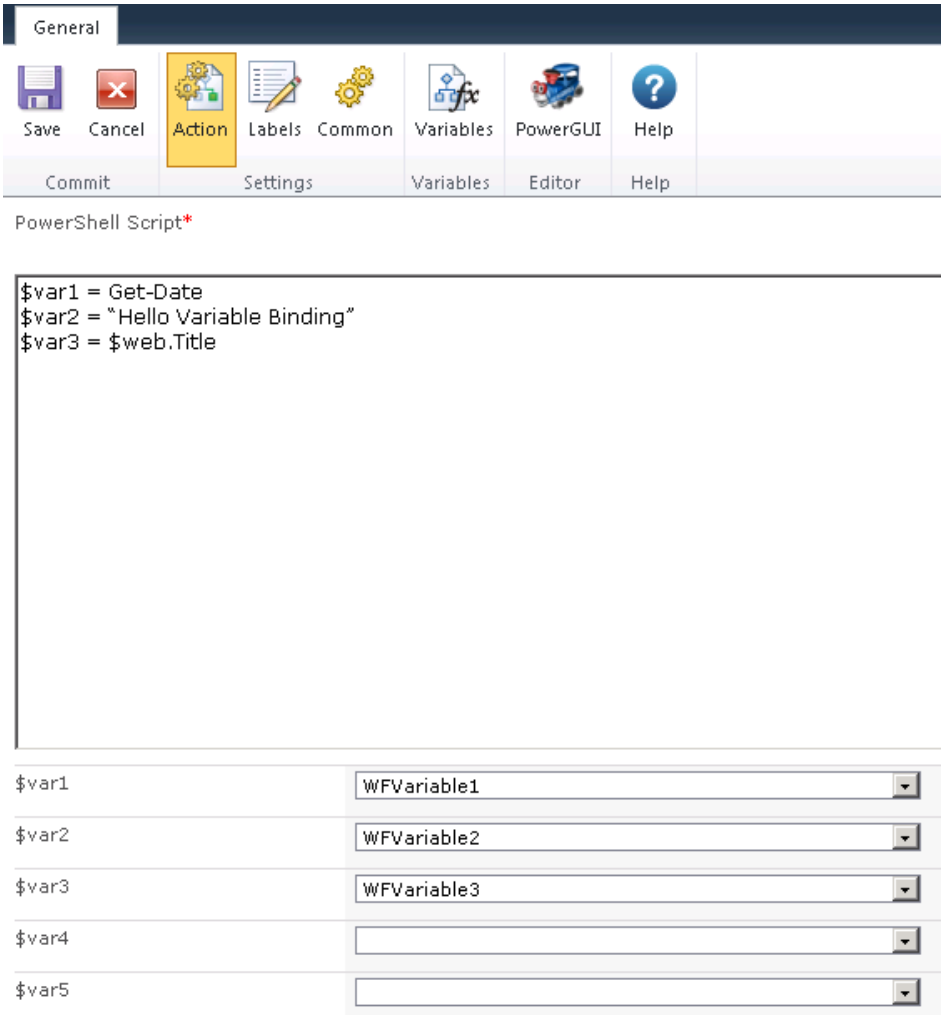


Figure 12: Workflow example 1

Example 2: Read Workflow Variable

- $web.Title = $var1
- $web.Update()
- …

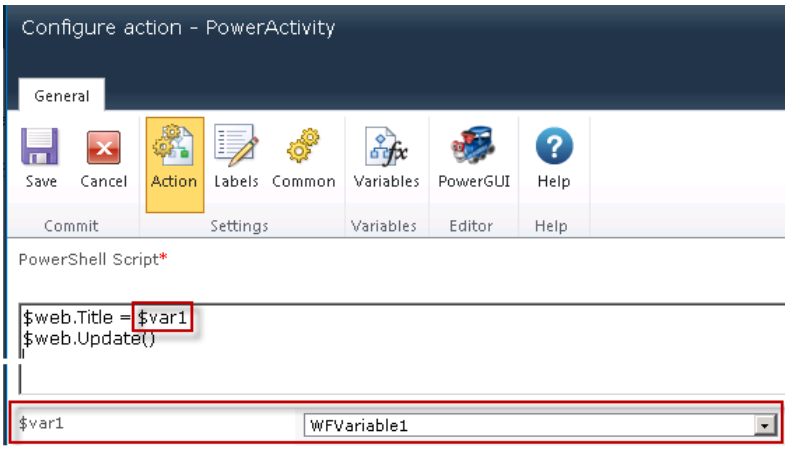


Figure 13: Workflow example 2

2.2.5. Credentials

The credential variable contains a mask for user credential input. If you wish to run a PowerShell script with special credentials, you can specify these there and use them inside of your script with the $credentials variable.

The password will not be shown as plain text.

By choosing the lock symbol, you can choose your stored credentials. Stored credentials are a Nintex feature. You can define them under *Central Administration* → *Nintex Workflow Management* → *Manage Workflow Constants*.

2.2.6. Error Handling

The PowerActivity error handling is based on the Nintex error handling. It enables you to catch exceptions.

To use error handling correctly, define two workflow variables – one Boolean, i.e. *error bool,* and one string variable for the error message, i.e. *error_message*. Now, activate error handling by setting *Capture errors?* to Yes. For *Store error occurrence in*, choose a Boolean variable. For the error message, choose a string variable.

Now you can react to every error for example by using 'set a condition' action. You can check the Boolean value and determine consecutive steps from within your workflow.

2.2.7. Debugging

With the PowerActivity Debug Mode, you are able to debug your PowerActivity action as well as included scripts.

For the integration of the debug functionality, just check the box Debug Mode and enter the IP of the system on which the debug console runs. Start the debug console and push **ping** in order to test whether your debug console is available. If you receive a positive popup message and your debug console shows a message that ping has been received, your debug mode is correctly configured.

To use the debug console, start your workflow and switch to the system on which the debug console runs. For each PowerShell command, an individual prompt will appear. You have five different options:

[Y] Yes	Yes is the default option in cases you press enter without input. The execution of the current command will be accepted.
[A] Yes to all	This accepts all commands and no more question will appear.
[N] No	By choosing the No option, the execution of the present command will stop and the next command starts.
[L] No to all	Similar to Yes to all, this option rejects all upcoming commands.
[S] Suspend	The last option is Suspend. This option enables you to interrupt the workflow. It opens a PowerShell command line prompt within the debug console. You exit the suspend mode through typing exit. In this mode, you have access to all workflow variables and can check their values. You also have access to all PowerShell commands.

2.2.8. Predefined Variables

The following table contains the available variables for the element on which the workflow runs.

Avoid overwriting them as this can give rise to datatype problems.

Security note: The variables are initialized within the scope of the user who starts the workflow. If you use impersonation, create a new object of the types SPSite, SPWeb etc. to get an additional instance.

Variable		*List*	*Site*
$site	Microsoft.SharePoint.SPSite	X	X
$web	Microsoft.SharePoint.SPWeb	X	X
$list	Microsoft.SharePoint.SPList	X	
$listItemService	Microsoft.SharePoint.Workflow.IListItemService	X	X
$taskService	Microsoft.SharePoint.Workflow.ITaskService	X	X
$sharePointService	Microsoft.SharePoint.Workflow.ISharePointService	X	X
$contex	Microsoft.SharePoint.WorkflowActions.WorkflowContext	X	X
$item	Microsoft.SharePoint.ListItem	X	

Table 1: List of predefined variables

2.2.9. Impersonation

The last section in the PowerActivity configuration dialogue is impersonation. Here, you can overwrite the user credentials which are used to run PowerActivity on the target system.

Sometimes, it may be necessary to run a script with credentials that are different from the standard ones. In this case, you can use impersonation to integrate these credentials into your workflow.

Note that the username should contain the domain name i.e. DOMAIN\username.

2.3. Features

2.3.1. Impersonation

After PowerActivity is installed, the administrator has to define a default user which runs the embedded scripts. Sometimes, it may be necessary to have more or special rights to run a workflow. The impersonation feature allows you to overwrite the standard user of the workflow. In this case, you can define a user inside the PowerActivity configuration. For more information, see chapter 2.2.9.

2.3.2. User Defined Actions

You can use PowerActivity 2010 in Nintex User Defined Actions (UDA) like every other Nintex action. To create an UDA that contains PowerActivity, you have to be a PowerActivity Designer. However, for using the UDA later on, you do not need to be a PowerActivity designer.

2.3.3. Debugging

PowerActivity allows you to debug your scripts. You can interrupt your workflow. The debug console is helpful in detecting errors. It is possible to check variables and/or to set up any PowerShell command you want. For more information about the usage, see chapter 2.2.7.

2.3.4. PowerModules

Data One PowerActivity provides the opportunity to define a set of modules that can be used inside the PowerActivity script editor. This saves time and, besides, reduces errors. The editor of these PowerModules has to define them in the Central Administration. You can find the PowerModules under the PowerActivity configuration. As shown in Figure 15: PowerActivity configuration you can see a list of present PowerModules there. If you want to use a PowerModule, it is necessary to know the name of the respective module. In the PowerActivity script mask, you can use this module by typing **Add-PowerModule 'moduleName'.** Please replace #ModuleName# with the name of the module in question.

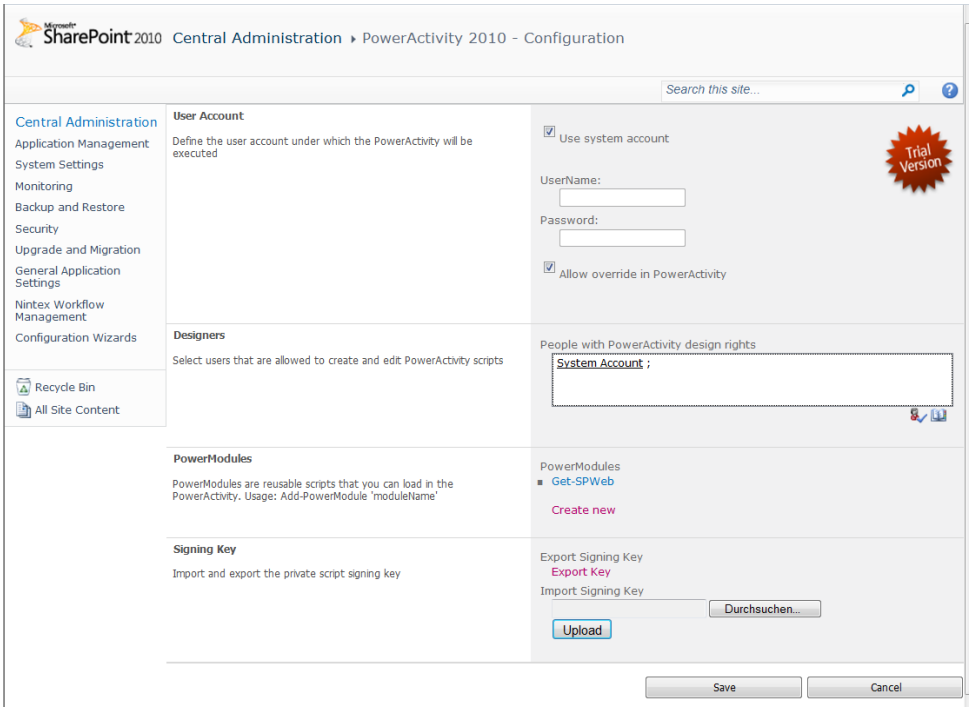


Figure 15: PowerActivity configuration

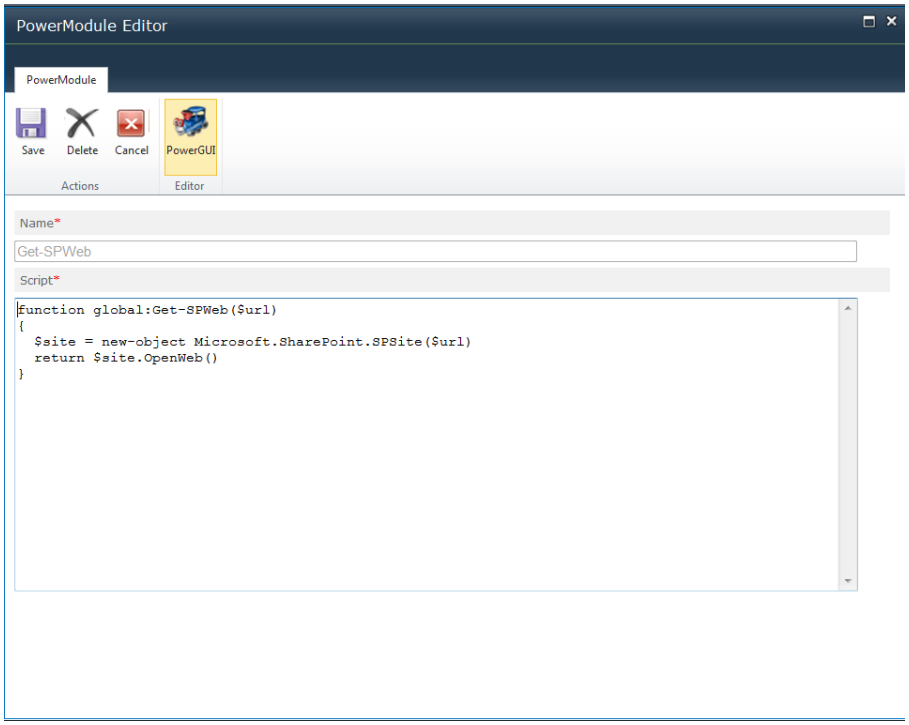
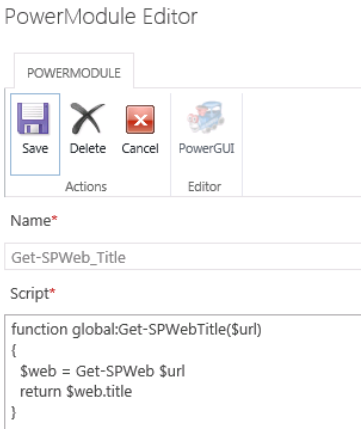


Figure 16: PowerModule editor

PowerModule-Example:

➔ Step 1: Adding PowerModule (Central Administration)



➔ Step 2: Using a PowerModule

```
Add-PSSnapin Microsoft.Sharepoint.Powershell

Add-PowerModule 'Get-SPWeb_Title'

$var1 = Get-SPWebTitle {ItemProperty:URL}
```

2.3.5. Support for Site Workflows

Nintex Workflow 2010 supports site workflows. With this feature, it is possible to start workflows at site level. The PowerActivity for Nintex Workflow 2010 is also available at site level.

2.3.6. Import and Export

With PowerActivity 2010, you can import and export workflows that contain a PowerActivity action. In order to export a Nintex workflow that contains a PowerActivity, open the Nintex Workflow Designer. Open or create the workflow you want to export. In the ribbon, press the Export button.

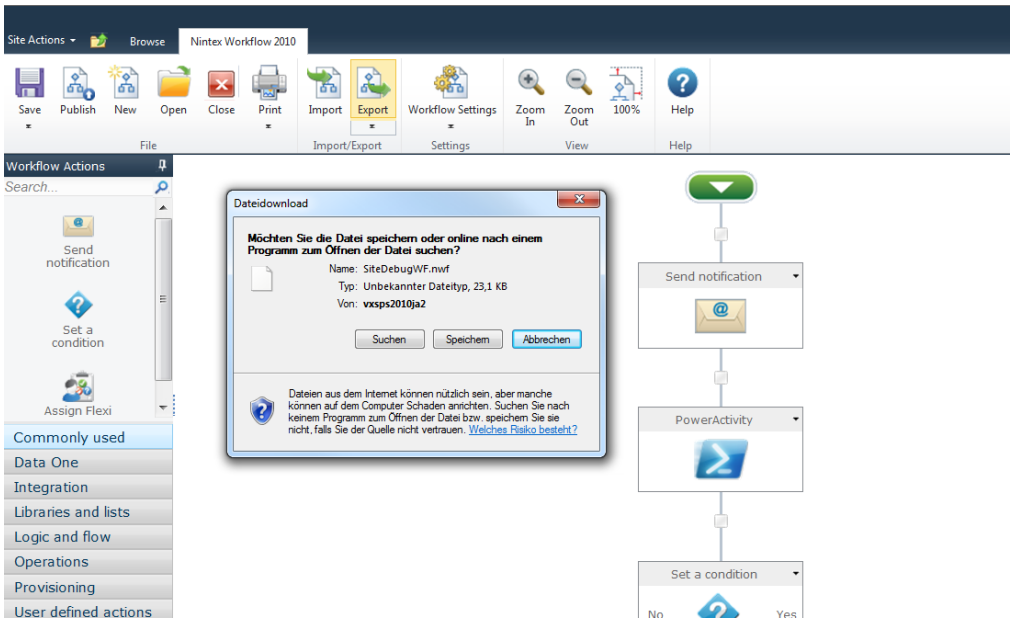


Figure 17: Export a Nintex Workflow which contains a PowerActivity action

If you want to import a workflow, just press the import button and choose the *.nwf file (export file of a Nintex Workflow).

2.3.7. Define PowerActivity Users

Not every Nintex workflow designer has the necessary rights to use the PowerActivity action correctly. Non-trusted users can drag the action on the workflow. Yet they are not able to make any configuration.

If you want to assign permissions to a user that enable this user to configure PowerActivity, you have to add this user to the Designers group. To do so, switch to *Central Administration* -> *Data One* -> *PowerActivity 2010* -> *Configuration*.

In the section Designer, add the respective user to the list of designers.

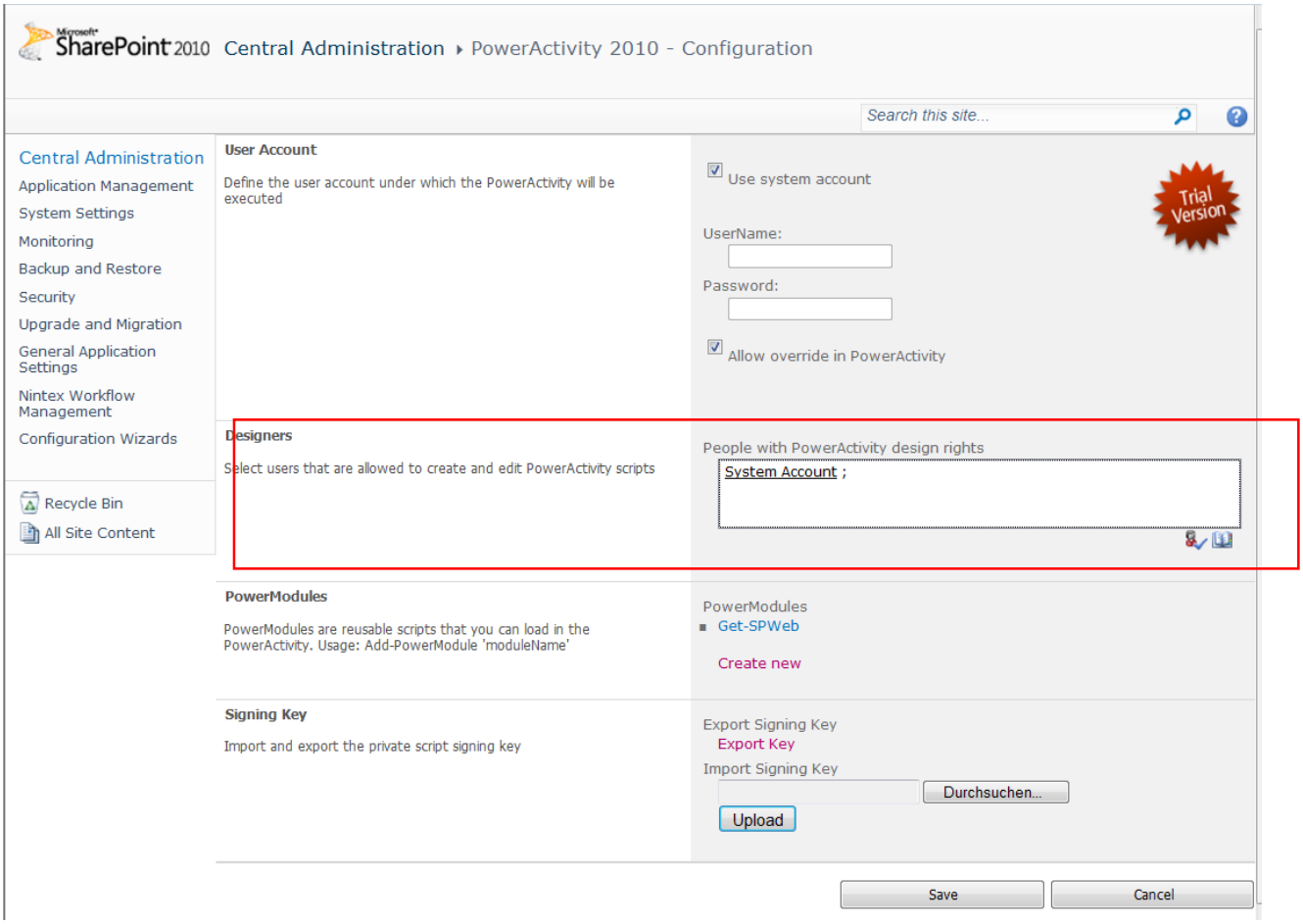


Figure 18: Granting permission for PowerActivity

2.3.8. Using Nintex Error Handling

Within PowerActivity for Nintex Workflow 2010, you can use Nintex Workflow error handling. To achieve this, you have to make the relevant configurations inside of the PowerActivity action. For more details, see Chapter 2.2.6.

2.3.9. Predefined Variables

The PowerActivity for Nintex Workflow 2010 was delivered with some predefined variables. Please check chapter 2.2.8 for a list of available variables. Always keep in mind that there are differences between list and site workflows due to the different structural levels and scopes.

3. Frequently Asked Questions

3.1. When Starting the Workflow, I Get an Error Message that Reads: Invalid Script Signature.

You get this error message if you migrate and start a Nintex Workflow that contains a PowerActivity component from one system to another such as for example from development to production or from 2010 to 2013. In so doing, you have probably forgotten to transfer the signing key from source to target system. For more information, see chapter 1.3.2.

There are different ways to solve this issue:

If there is currently NO OTHER workflow deployed on target system that contains PowerActivity:

If your target system already contains workflows with PowerActivity components and these workflows have already been published on the target system at an earlier point in time, you have to republish them after importing the new key. In this scenario, the number of signing keys, is limited to a single key.

Go to source system *Central Administration* -> *Data One* -> *Configure PowerActivity* -> *Signing Key* and click Export Key to save the signing key.

Go to target system *Central Administration* -> *Data One* -> *Configure PowerActivity* -> *Signing Key* and click Upload. Choose the source key and upload it.

If you want to use different signing keys and want to republish your workflow on target system:

If you want to use different signing keys for source and target system, you have to republish the workflow on the target system. This needs to be done once per import of this workflow from source to target system.

If you migrate from SharePoint 2010 to 2013 it is strongly recommended to first export the signature key from the source system (2010) and import it into the target system (2013) – after configuring PowerActivity and before importing or creating any workflow that contains PowerActivity components on the target system.

3.2. Windows User Access Control: Requested Registry Access is not Allowed

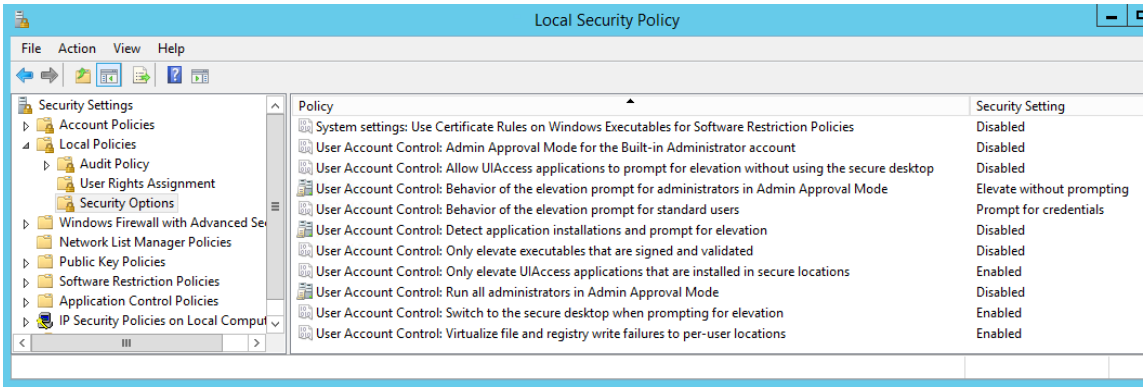
Reason 1 – Windows UAC

Depending on the CMDLETS you use, this error occurs. It is for example likely to do so for New-SPSite.

This message indicates that Windows UAC blocks the execution of the command.

If you want to get a distinct statements working without deactivating UAC, use the security policies:

Open the Local Security Policy Editor (secpol.msc) and set the UserAccountControl as displayed below:



Of course, it is also possible to deactivate UAC to get rid of this error. To deactivate the Windows UAC, set the registry key to 0 following the instruction below:

- Navigate to the following registry subkey:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System

- In the details pane (Right Pane), locate the EnableLUA key (REG_DWORD type). Double-click on Modify.
- In the data box Value, type 0 (zero). Click OK.
- Exit the registry editor.
- Restart the computer.

Should the problem still exist: Grant the WSS_Admin_WPG Group read access with regards to the following registry key:

HKEY_Users\<userid>\Environment

<userid> is the ID of the concerned application pool account.

Depending on the configuration of the farm, it can be necessary to assign read permissions to different users and/or for further keys as well. In this respect, the decisive factor the user context in which the workflow runs.

Reason 2 – ApplicatonPool Account deviates from FarmAdmin Account

The error message “Registry access is not allowed” can also happen if you use an ApplicationPoolAccount that is different from your farm admin. Generally, it is recommended that the two accounts are different from one another. As mentioned in 1.3.2, by default, PowerActivity actions are executed as ApplicationPoolAccount. Consequently, your ApplicationPoolAccount is not apt to execute CMDLETS. Make sure your ApplicationPool Account has enough privileges. Your scripts should be run with this user account if you do not use another account.

1) Add the SPShell Admin for content DB to your AppPoolAccount:

   Example: Add-SPShellAdmin -UserName -database #GUID_OF_YOUR_CONTENT_DBs#

2) Add the SPShell Admin to SP-Config DB to your AppPoolAccount:

   Get-SPDatabase | Where-Object {$_.WebApplication -like "SPAdministrationWebApplication"} | Add-SPShellAdmin DOMAIN\Username

In the example above, a new user named User1 is added to the SharePoint_Shell_Access role in both – the specified Central Administration content database and the configuration database.

3) Your ApplicationPool Account should be local admin on SharePoint-Farm servers.

4) To make these changes pervasive, open shell and execute an iisreset.

3.3. Using PowerShell with a Remote Access

- Configure Windows Remote Management:

http://msdn.microsoft.com/en-us/library/aa384372%28v=VS.85%29.aspx.

- Ensure that the remoting works as expected with the PowerShell console.
- PowerActivity has to run as the System Account (Central Administration -> Data One -> Configure PowerActivity -> User system account (checked). Impersonation does not work. Besides, you have to use explicit credentials to open a remote session. The following example opens a remote session to myserver, invokes a commando and closes the session.

```
# generate a secure password from the credentias variable
$password = convertto-securestring -String $credentials.Password -AsPlainText -force


# create the PowerShell credentials that you need for New-PSSession
$remoteCred = new-object System.Management.Automation.PSCredential($credentials.UserName,
$password)


# open a session to myserver using the credentials from above
$session = New-PSSession -ComputerName myserver -Credential $remoteCred


# invoke a command on myserver and pass in the website title as argument
$result = Invoke-Command -Session $session -ArgumentList $web.Title -ScriptBlock {
    param($param1)

   # simply returns hello website title
    return "Hello " + $param1
}

# log the result to the workflow history log
$result | out-host


# close the session
Remove-PSSession $session
```

The variables that you pass to and return from the remote commands have to be serializable.

The credentials are defined here:

Figure 19: $credentials

3.4. Best Practice: How to Update List Items

Avoid updating list items in the following way:

```
$item["Title"] = "Test"
$item.SystemUpdate()
```

This script could cause an infinite loop because updating (even SystemUpdate()) will trigger the event receivers. Using the script from above in a workflow that automatically starts as soon as an item is changed is likely to give rise to an endless loop. To safely update a list item, use the following script instead:

$this.UpdateListItem($item, @{"Title" = "Test"; "Second Field Name"="Second Value"})

For the field name you could either use the title or the internal name of the field.

3.5. Using Insert Reference Values

If you use Insert Reference values such as {Common:WebURL}, be aware of the fact that they are replaced by the respective content before a PowerActivity component starts.

Example: *$web=Get-SPWeb "{Common:WebUrl}"* will be replaced and executed as *$web=Get-SPWeb "http://.."*

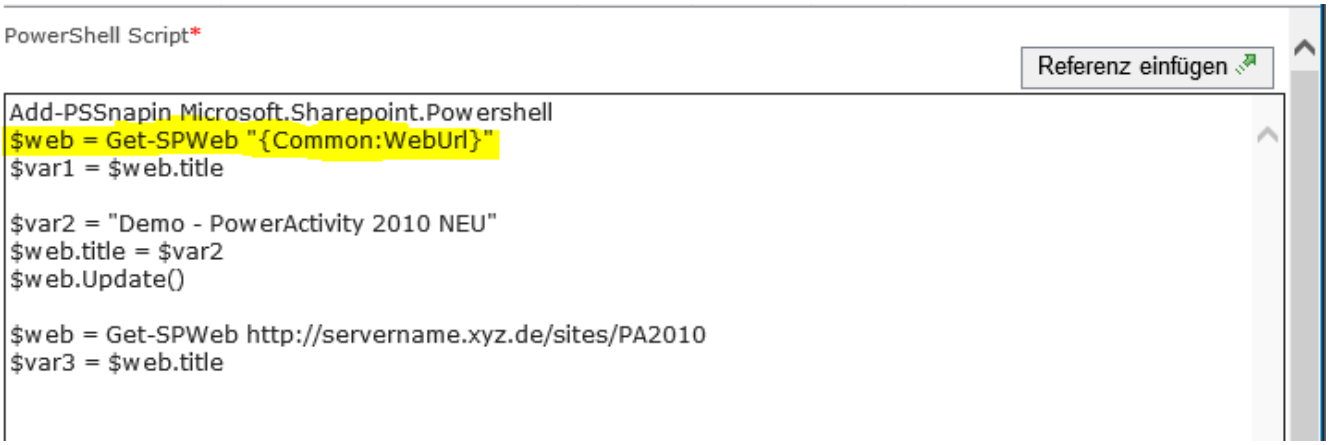


Figure 20: Using insert reference values

3.6. Best Practice: How to Access SharePoint Lists

Avoid using *$web.Lists['Listname']* to access list columns because this method will not reach all existing columns.

Instead you should use a CAML-Query:

```
$web=Get-SPWeb -Identity
$spWeb = Get-SPWeb -Identity '###YOURWEB###'
$spList = $spWeb.Lists['###YOURLIST###']
$spQuery = New-Object Microsoft.SharePoint.SPQuery
$spViewFields= '<FieldRef Name="ID" /><FieldRef Name="Keywords" />'
$spQuery.ViewFields=$spViewFields
$camlQuery = '<Where><Eq><FieldRef Name="ID"/><Value Type="Integer">####VALUE#####</Value></Eq></Where>'
$spQuery.Query = $camlQuery
$spListItems = $spList.GetItems($spQuery)
```

3.7. Errors When Saving or Publishing a Workflow

3.7.1. Key set does not exists / Schlüsselsatz nicht vorhanden

PowerActivity uses the keys store of the machine to save the signing key which is necessary to sign the power shell scripts. The error message shown in figure 21 is received, if the permissions for the registry entry are wrong.

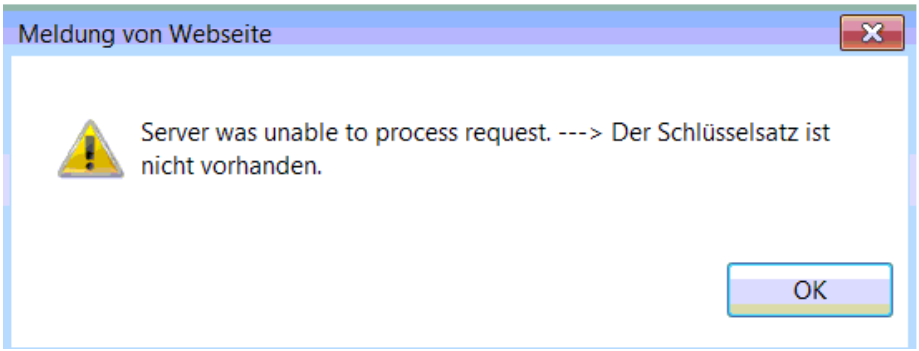


Figure 21: Key set error

If you get this message, take the following steps:

(1) Go to the folder: **c:\Programm Data\Microsoft\Crypto\RSA\MachineKeys**

(2) Assign the permissions **read, write, modify** to the group **Everyone**.

3.7.2. Bad Version of Provider

PowerActivity uses the keys store of the machine to save the signing key which is necessary to sign the power shell scripts. The error “Bad version of provider” occurs if the signing key does not meet the respective requirements and/or is ill-defined such as chosen too long.

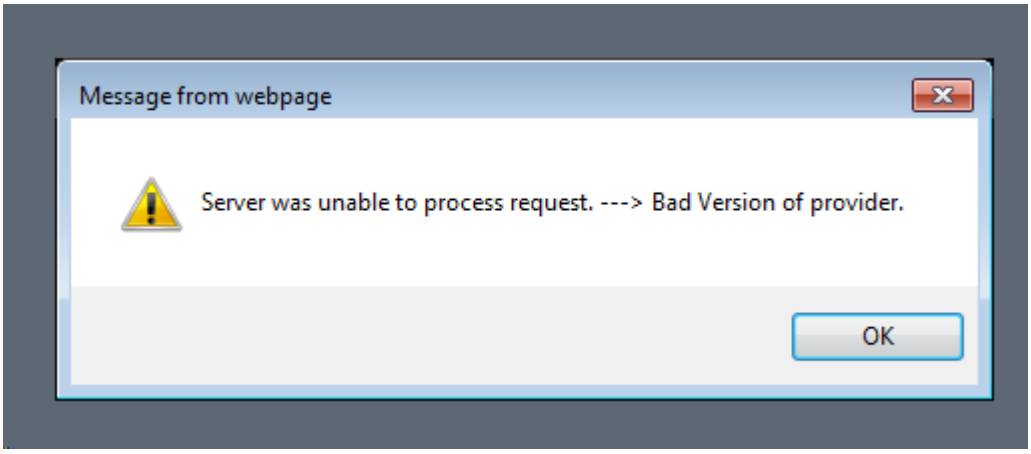


Figure 22: Bad Version of provider

If you get this message, take the following steps:

(1) Send a mail to the Data One Customer Care Center. Indicate the exported signing key.

(2) You will get a new signing key. Import this key.

3.7.3. Namespace DataOne could not be Found

If you receive the error message below, check the web.config file.

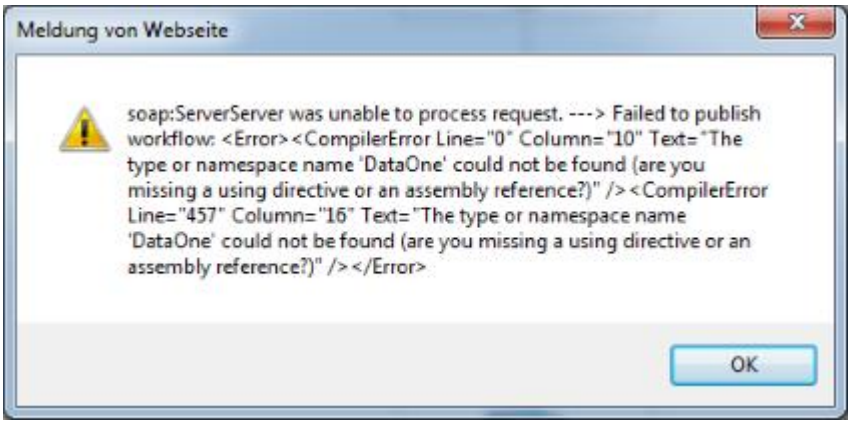


Figure 23: DataOne namespace is missing

If the following entry is missing in the section <authorizedTypes>, insert it:

<authorizedType Assembly="DataOne.Nintex.PowerActivity, Version=1.0.0.0, Culture=neutral, PublicKeyToken=7d3c850701dacb81" Namespace="DataOne.Nintex" TypeName="PowerActivity" Authorized="True" />

3.8. The Local Farm is not Accessible

You can receive this error message even if you start PowerActivity with an account that belongs into the group of farm Administrators. This error occurs because you do not have the required rights to execute commands. In the past, you have probably changed the user that executes PowerActivity scripts (1.3.2 Settings).

You change these rights by using the *Add-SPShellAdmin* cmdlet.

Syntax: *Add-SPShellAdmin -UserName Domain\User -database DBGUID*

You find additional information at: https://technet.microsoft.com/en-us/library/ff607596.aspx

Again, keep in mind that, by default, PowerActivity is executed as ApplicationPoolAccount and not as FarmAdmin.

4. PowerActivity 2010 – Known Limitations

4.1. Get-SPWebApplication

You cannot directly use this CMDLET within PowerActivity. You will get a message that says that you need FarmAdmin access. Even if you have these access privileges, you will still get this message.

4.2. CMDLET - Enable-SPFeature - How to Activate SharePoint Features

It is impossible use *Enable-SPFeature* in order to activate features. You will receive a message about missing privileges.

Instead, activate a feature by using this code:

```
## Activate WEB-Feature
$site_api = New-Object Microsoft.SharePoint.SPSite("{Common:WebUrl}")
$web_api  = $site_api.OpenWeb("{WorkflowVariable:WebName}")
$web_api.Features.Add("###YOUR FEATURE ID###")
$web_api.Update()
```

4.3. Write-Host / Out-Host / Read-Host

PowerActivity isn't a ConsoleHost instance of PowerShell, so it is not possible to use CMDLETs like Write-Host, Out-Host or Read-Host. Doing so will result in an error: "Cannot invoke this function because the current host does not implement it"

To get current value you can use "Get-Host" CMDLET.