SIM v6.1

Aug 23, 2019

Contents

Note: Documentation is still in development process. Please do not hesitate to contact us on support@silvermonkey.net for further information.

CHAPTER 1

Welcome

This document is meant to be a source for all information regarding the administration and installation of the Silver Monkey v6.1 engine.

Contents:

1.1 Requirements

1.1.1 Application Server (IIS)

- Microsoft Windows Server 2016 or higher
- Internet Information Server
- Microsoft .NET Core 2.x

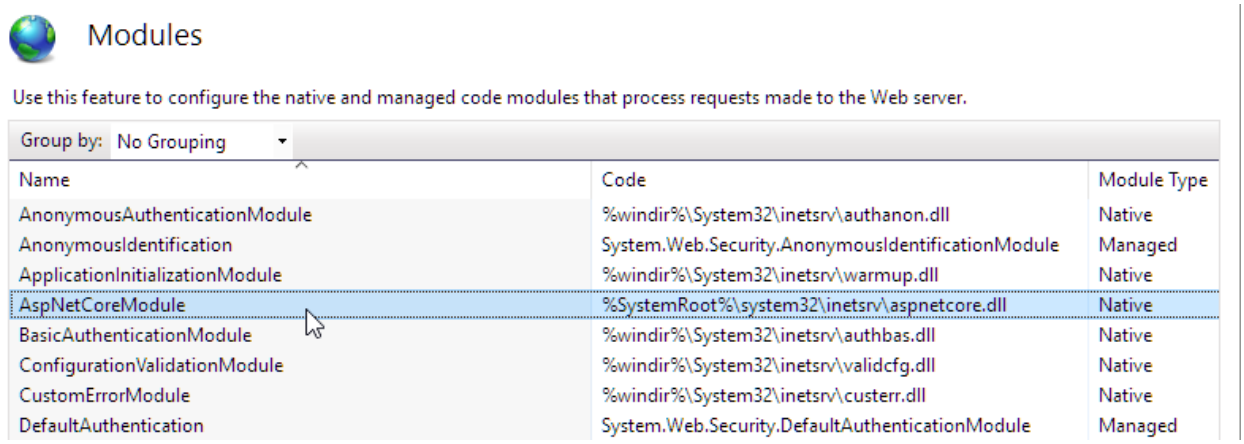
To Check if .Net Core is installed, run the following cmd command and check the output.

```
dotnet --version
```

Install the .NET Core Windows Server Hosting bundle

1. Install the .NET Core Windows Server Hosting bundle on the server. The bundle will install the .NET Core Runtime, .NET Core Library, and the ASP.NET Core Module. The module creates the reverse-proxy between IIS and the Kestrel server.
2. Restart the server or execute **net stop was /y** followed by **net start w3svc** from the command-line to pickup changes to the system PATH.

Important: Make sure that module is available in IIS/Modules:



1.1.2 Database Server (SQL)

- For Application: Microsoft SQL Server 2017 or higher
- Or Microsoft SQL Server Express with **Advanced Services**
- For SCCM Database: Microsoft SQL Server 2017 (a lower version causes limited functions)

1.1.3 Server Hardware Requirements (IIS+SQL)

The system requirements for processors, RAM and hard disk space depend on the size of the correspondig ConfigMgr environment and the number of users working at the same time. Anyway, there is always the option to easily move the application to a more powerful machine or to distribute it across several servers with load balancing.

In addition to the requirements of the operating system, the following conditions arise:

- 2 CPUs 2GHz+
- RAM 4GB
- Database size 500MB
- Website/Application files 150MB

(Valid for up to 10,000 systems and 20 concurrent users on the Web Application)

1.1.4 Workplace Systems

- Microsoft Internet Explorer (not supported but works)
- Microsoft Edge
- Mozilla FireFox
- Google Chrome

1.2 Installation

In this article:

1.2.1 Requirements

1. For general information on system requirements see *Requirements*.
2. Microsoft Active Directory Service Account for accessing SIM SQL DB (in this article `sim-svc-sql`)

Important: Please install all requirements before beginning with this guide!

1.2.2 IIS Features

Execute the following command to enable IIS features on the application server:

```
CMD.EXE /C DISM.EXE /enable-feature /all /online /featureName:IIS-WebServerRole /
↪featureName:IIS-WebServer /featureName:IIS-CommonHttpFeatures /featureName:IIS-
↪StaticContent /featureName:IIS-DefaultDocument /featureName:IIS-DirectoryBrowsing /
↪featureName:IIS-HttpErrors /featureName:IIS-HttpRedirect /featureName:IIS-
↪ApplicationDevelopment /featureName:IIS-ASPNET /featureName:IIS-NetFxExtensibility /
↪featureName:IIS-ASPNET45 /featureName:IIS-NetFxExtensibility45 /featureName:IIS-ASP␣
↪/featureName:IIS-CGI /featureName:IIS-ISAPIExtensions /featureName:IIS-ISAPIFilter /
↪featureName:IIS-ServerSideIncludes /featureName:IIS-HealthAndDiagnostics /
↪featureName:IIS-HttpLogging /featureName:IIS-LoggingLibraries /featureName:IIS-
↪RequestMonitor /featureName:IIS-HttpTracing /featureName:IIS-CustomLogging /
↪featureName:IIS-ODBCLogging /featureName:IIS-Security /featureName:IIS-
↪BasicAuthentication /featureName:IIS-WindowsAuthentication /featureName:IIS-
↪DigestAuthentication /featureName:IIS-ClientCertificateMappingAuthentication /
↪featureName:IIS-IISCertificateMappingAuthentication /featureName:IIS-
↪URLAuthorization /featureName:IIS-RequestFiltering /featureName:IIS-IPSecurity /
↪featureName:IIS-Performance /featureName:IIS-HttpCompressionStatic /featureName:IIS-
↪HttpCompressionDynamic /featureName:IIS-WebDAV /featureName:IIS-
↪WebServerManagementTools /featureName:IIS-ManagementScriptingTools /featureName:IIS-
↪ManagementService /featureName:IIS-IIS6ManagementCompatibility /featureName:IIS-
```

(continues on next page)

(continued from previous page)

For easy deploymnet: `Download the script.`

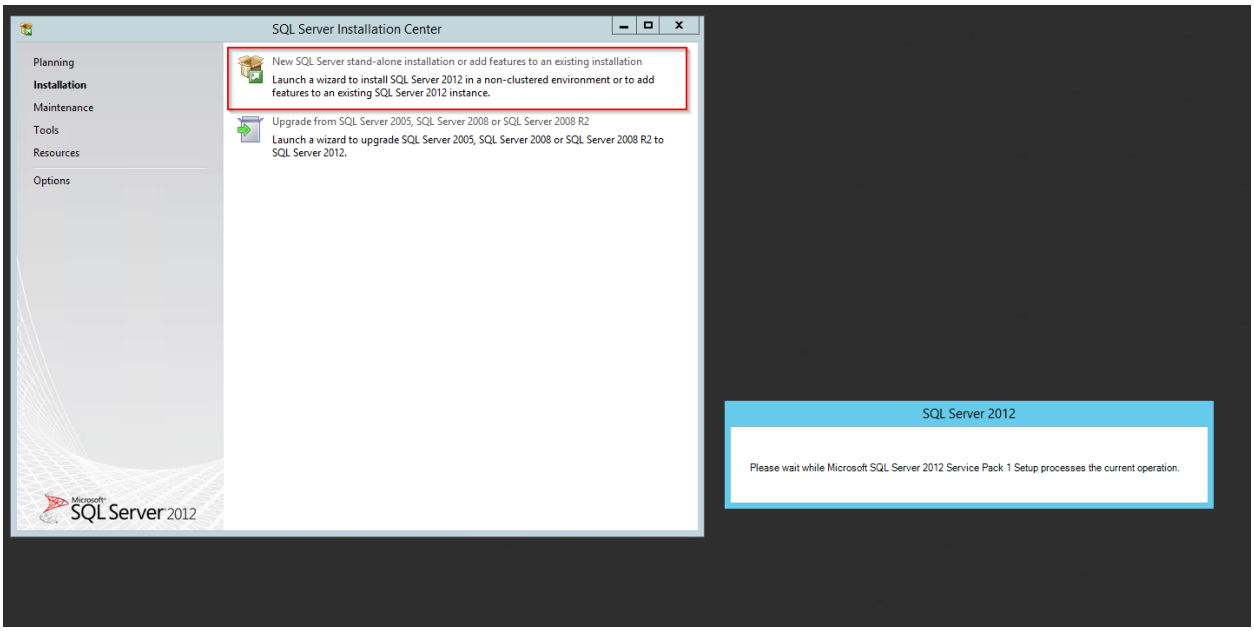
1.2.3 Microsoft SQL Server

For information about supported SQL Server versions see *Supported configurations*

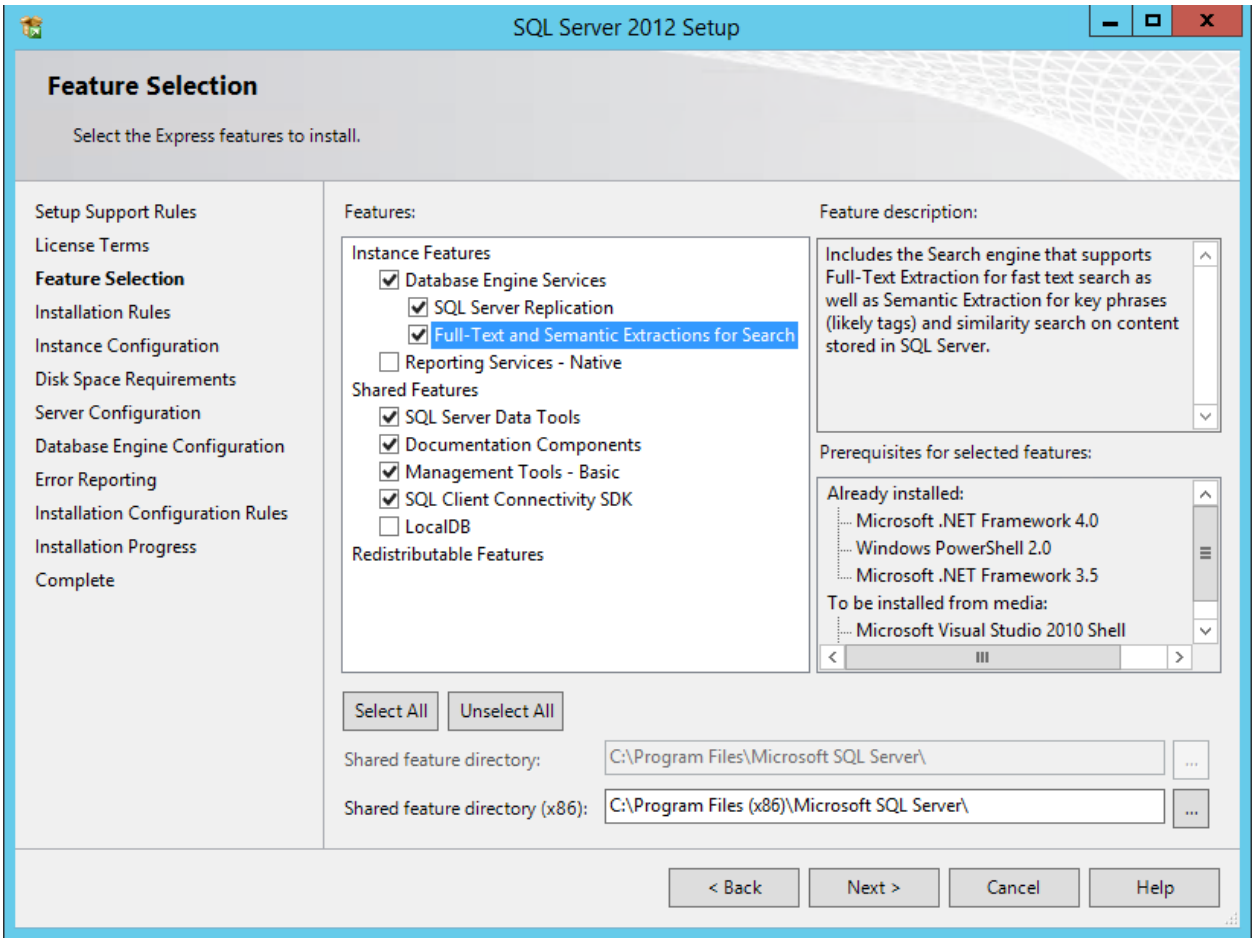
The installation of the SQL Server will be described in the following steps.

Installation Setup

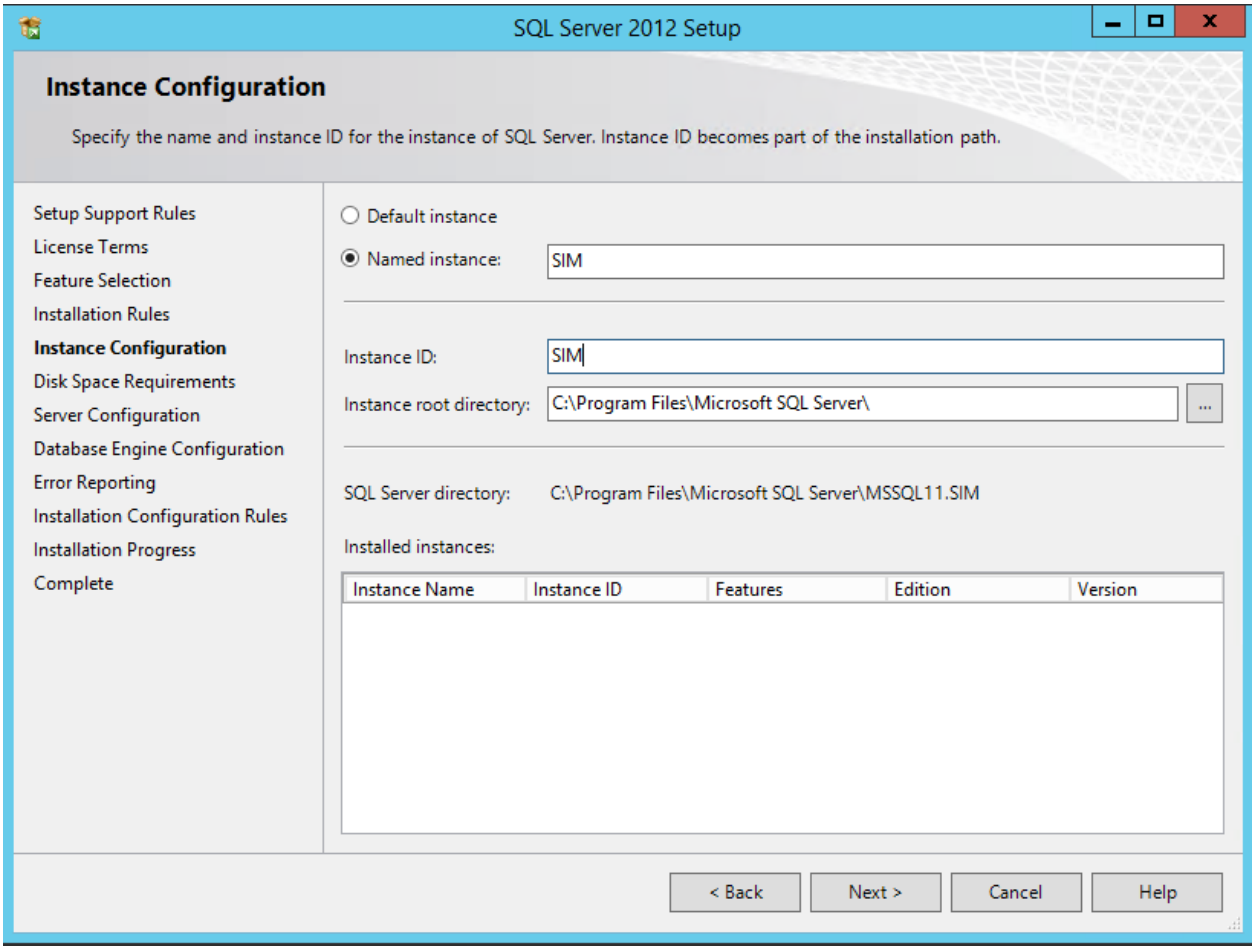
Start the SQL Server installation setup. Choose the "New SQL Server stand-alone installation…"-Option in the follwing Window:



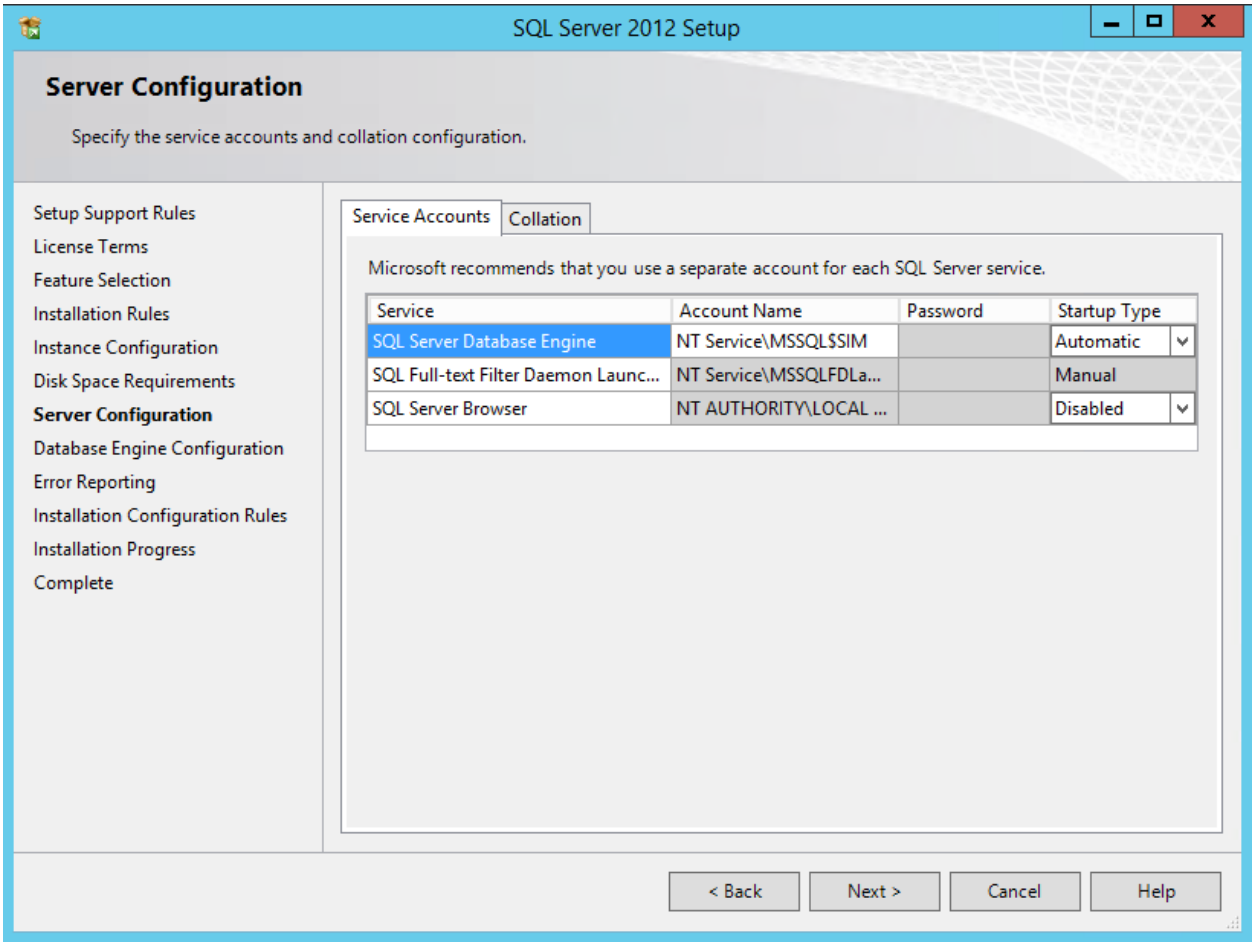
Throughout the installation, please choose the same features as shown below:



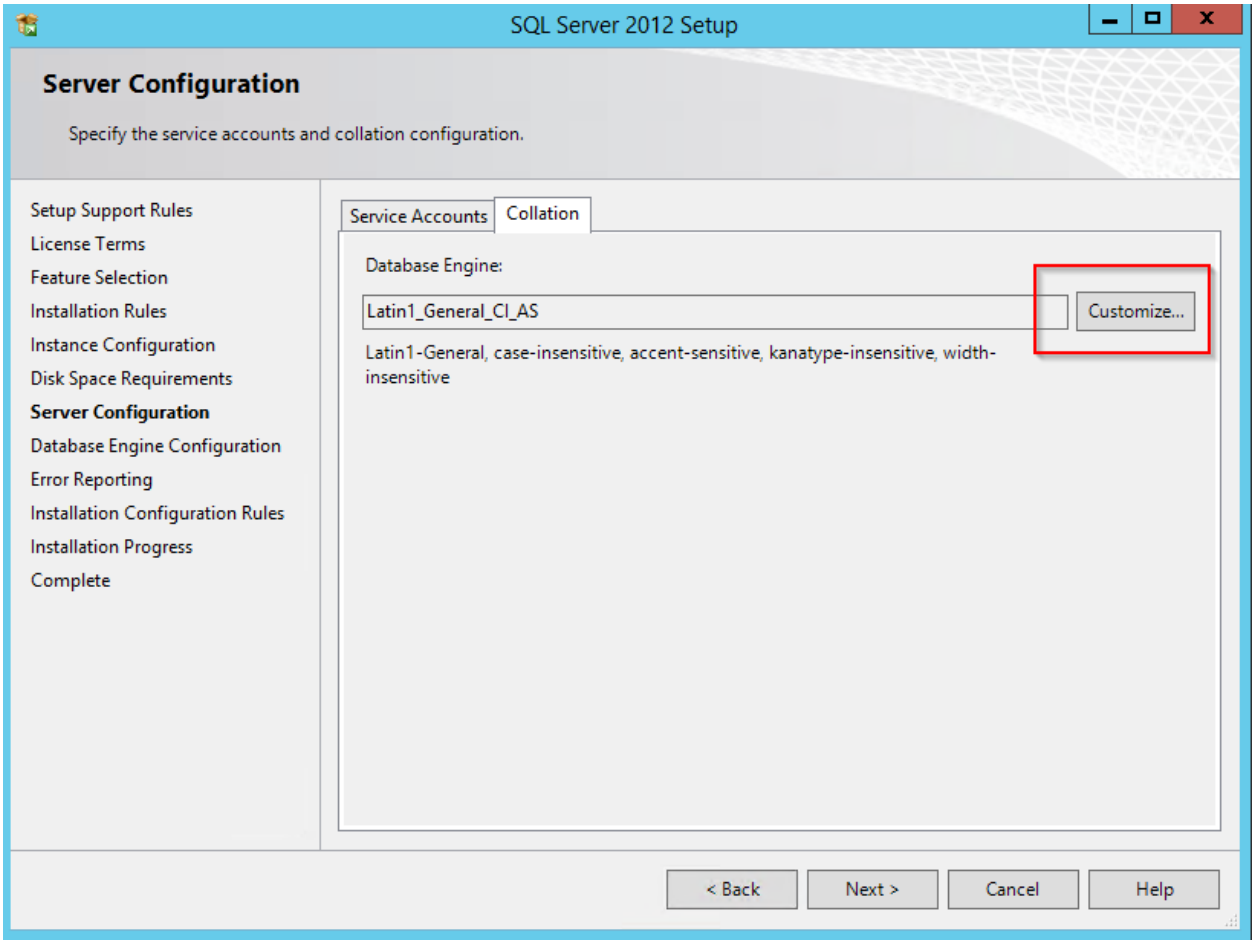
Name the instance SIM or choose another name:



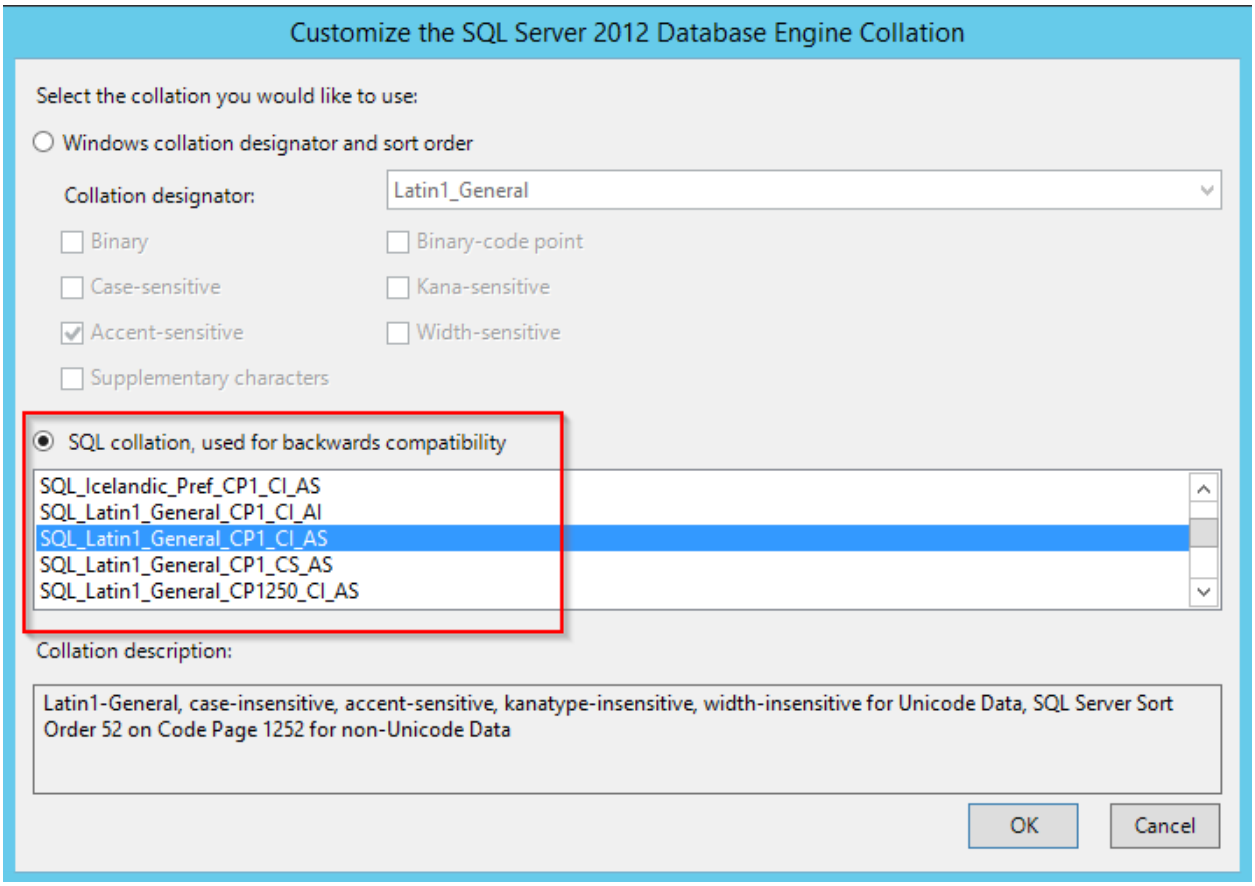
Configure the server as follows:



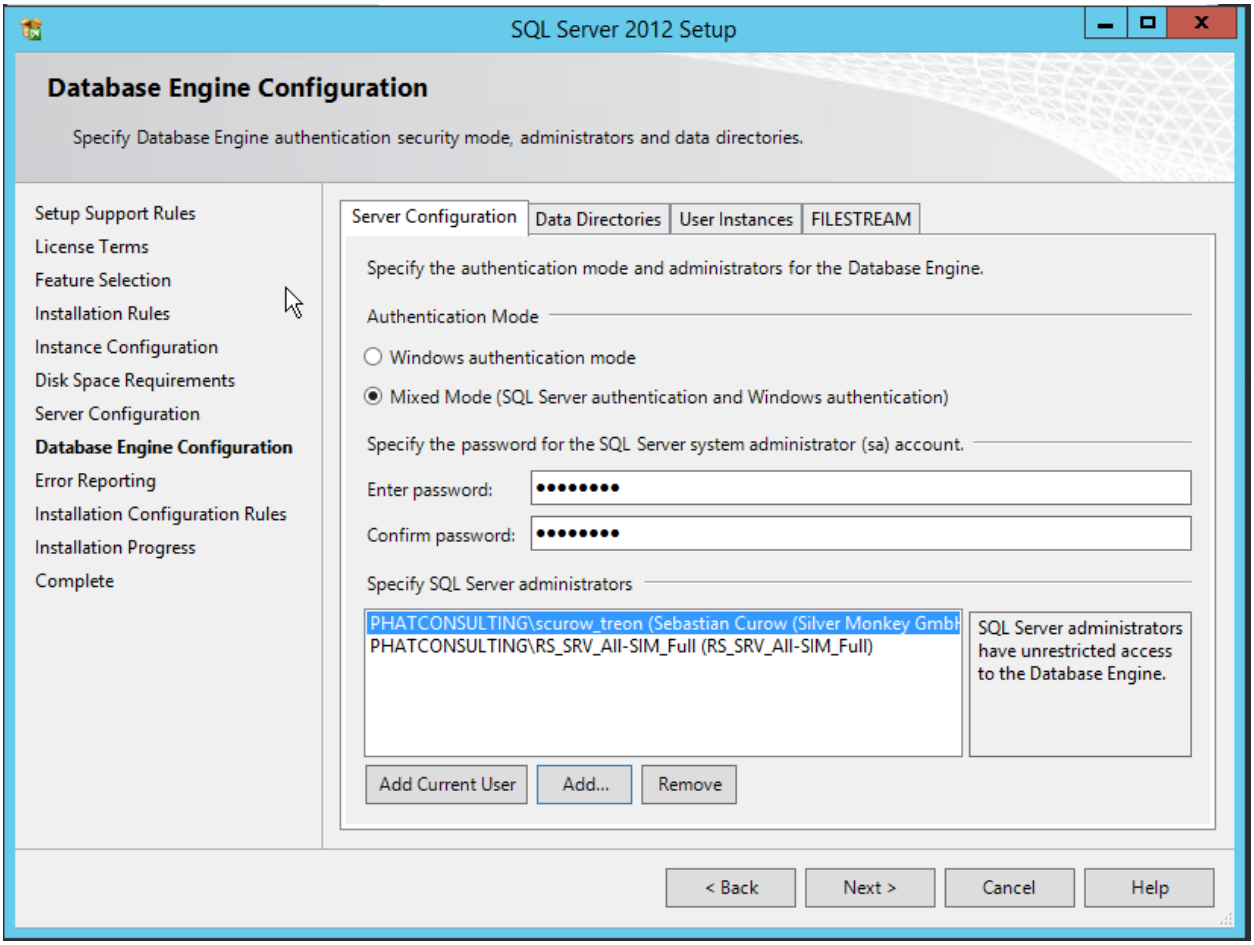
Customize the Database Engine



Choose the Database Engine called 'SQL_Latin_General_CP1_CI_AS':



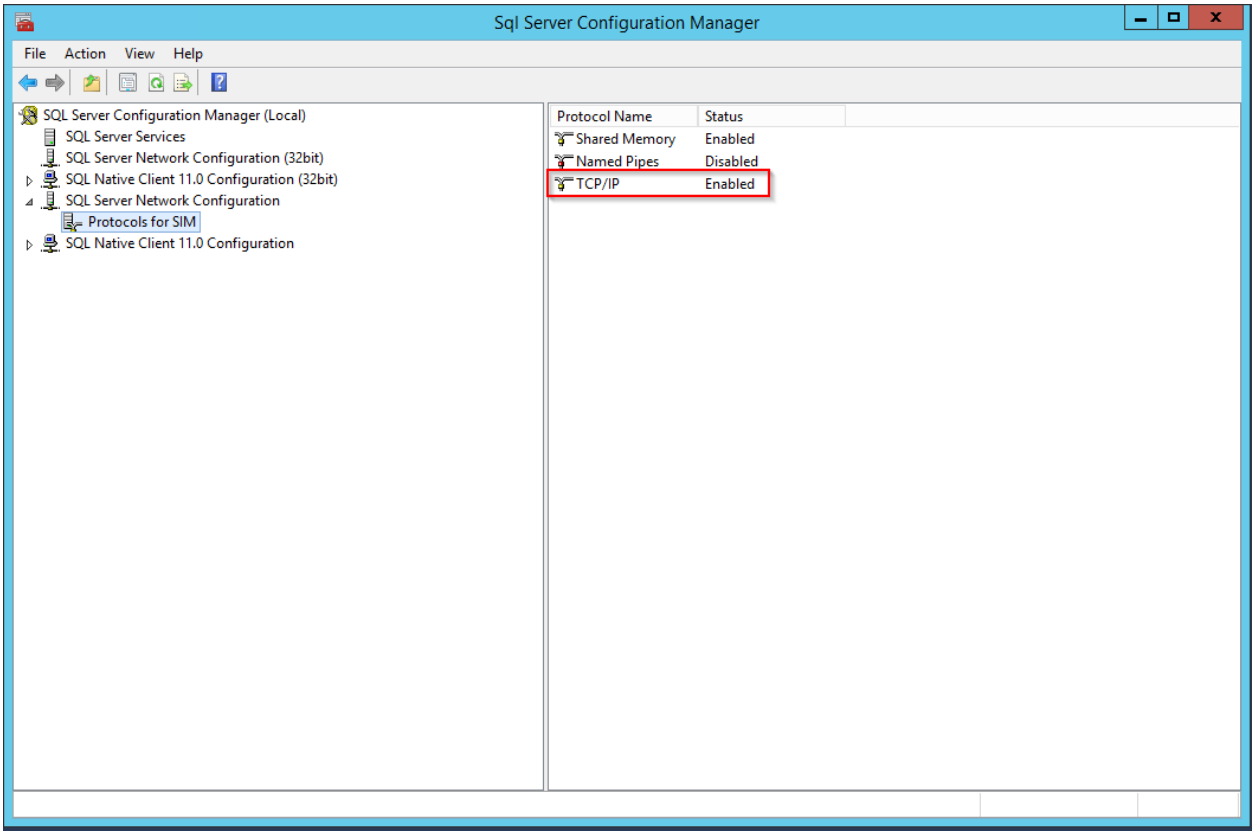
Select the 'mixed mode'-authentification and add your AD service account for SQL (`sim-svc-sql`) as SQL Server administrator:



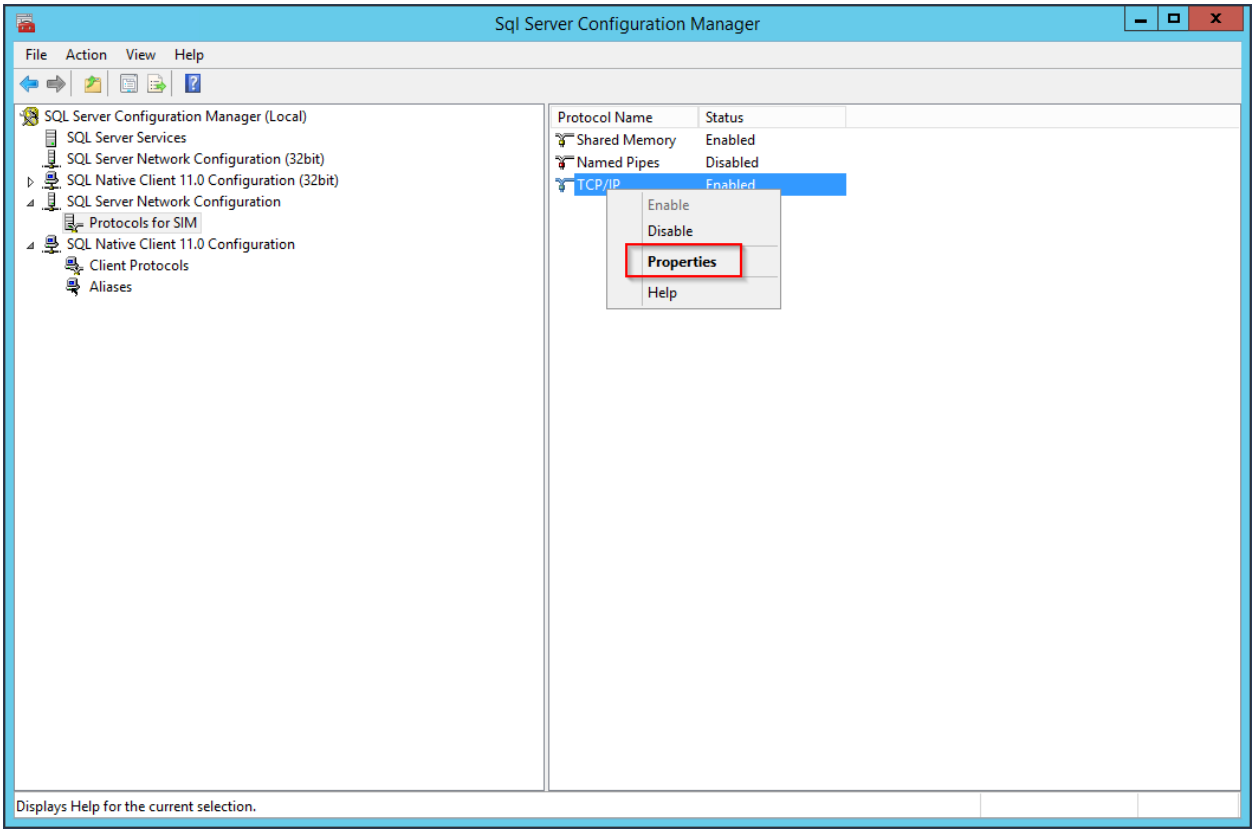
You have completed the setup!

SQL Server TCP/IP Configuration

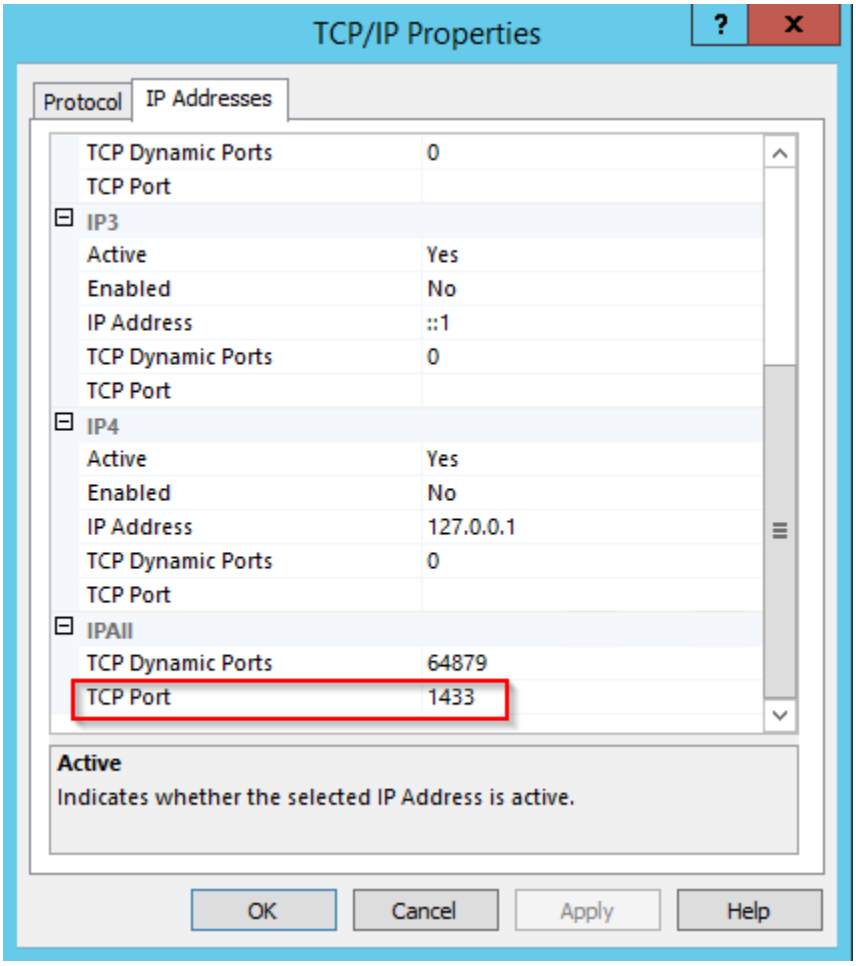
Open the SQL Server Configuration Manager, choose 'SQL Server Network Configuration' and then 'Protocols for [Database Name]'. Change the TCP/IP Status to *Enabled*:



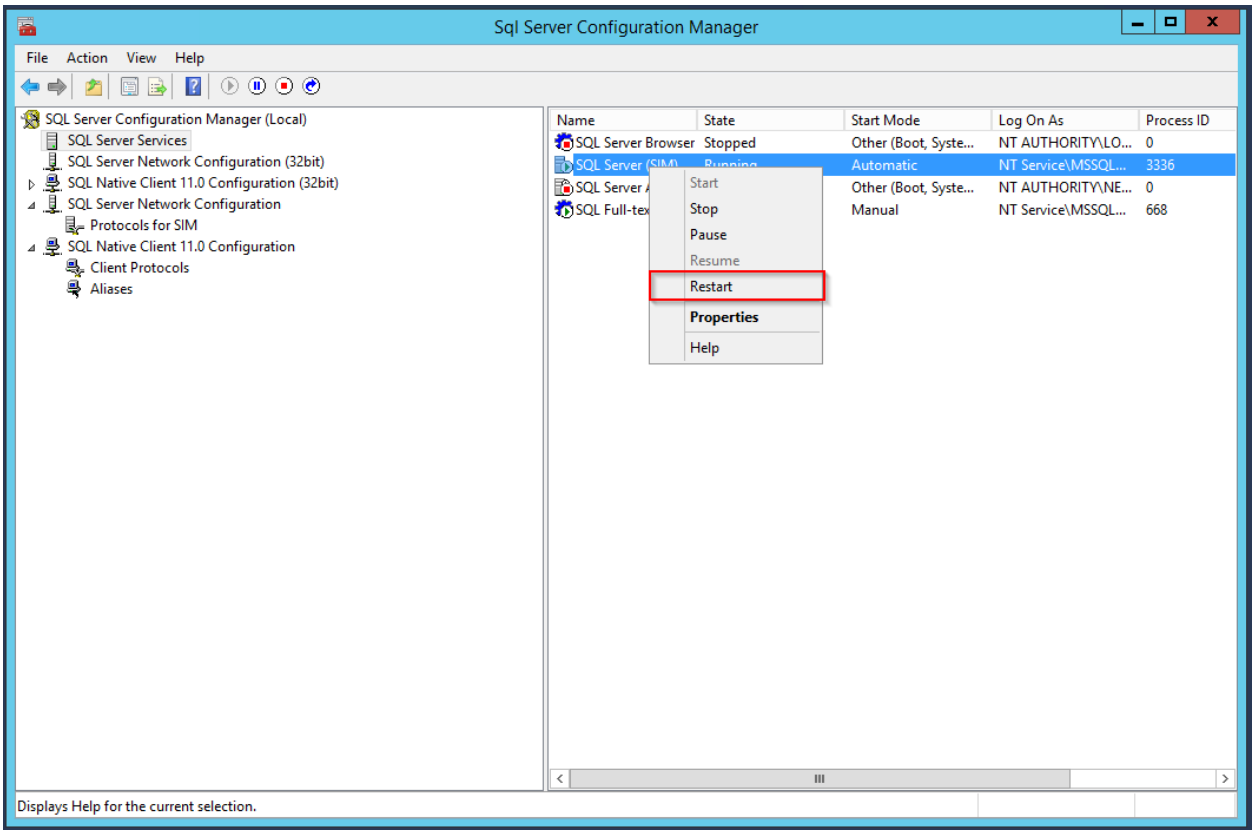
Right-click the TCP/IP line and choose ‘Properties’:



Choose the tab “IP Adresses” and change the ‘TCP Port’-entry to 1433:

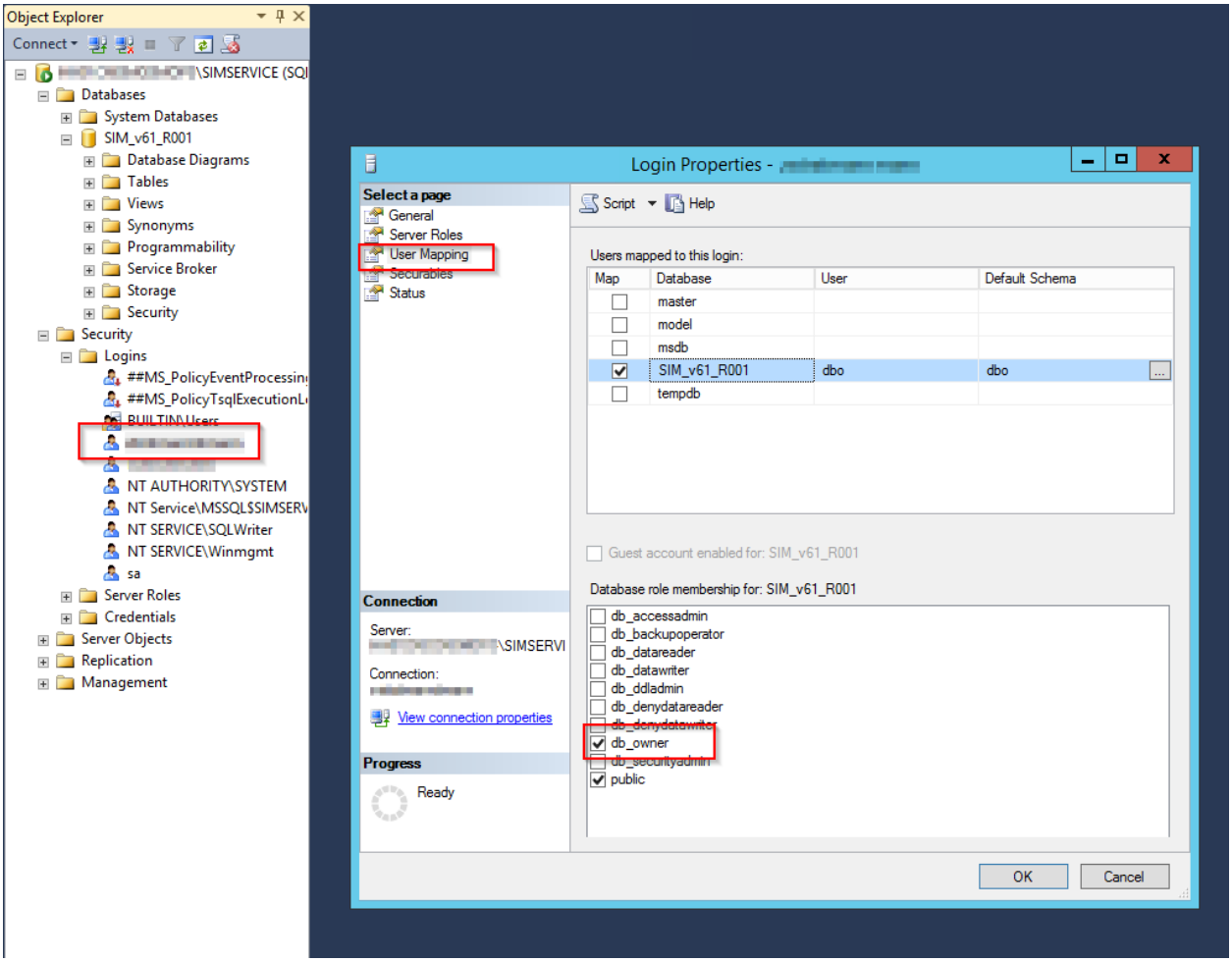


Afterwards, navigate to the SQL Server Services and restart the ‘SQL Server ([Database Name]):



SIM SQL DB Installation

1. Create database `SIM_v61_R001`
2. Grant SilverMonkey Service Account (`sim-svc-sql`) “db_owner” rights for the corresponding database

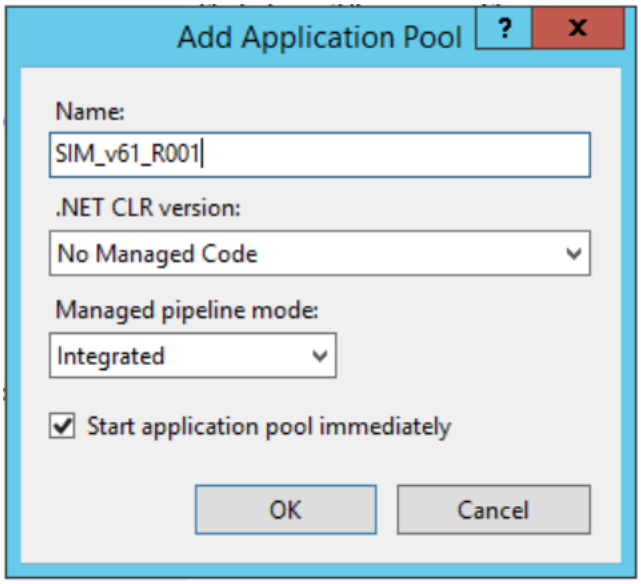


3. Import .SQL file from installation media (.\Database) into SQL Management Studio
4. Make sure the **USE** command aims to the correct database created above and execute script

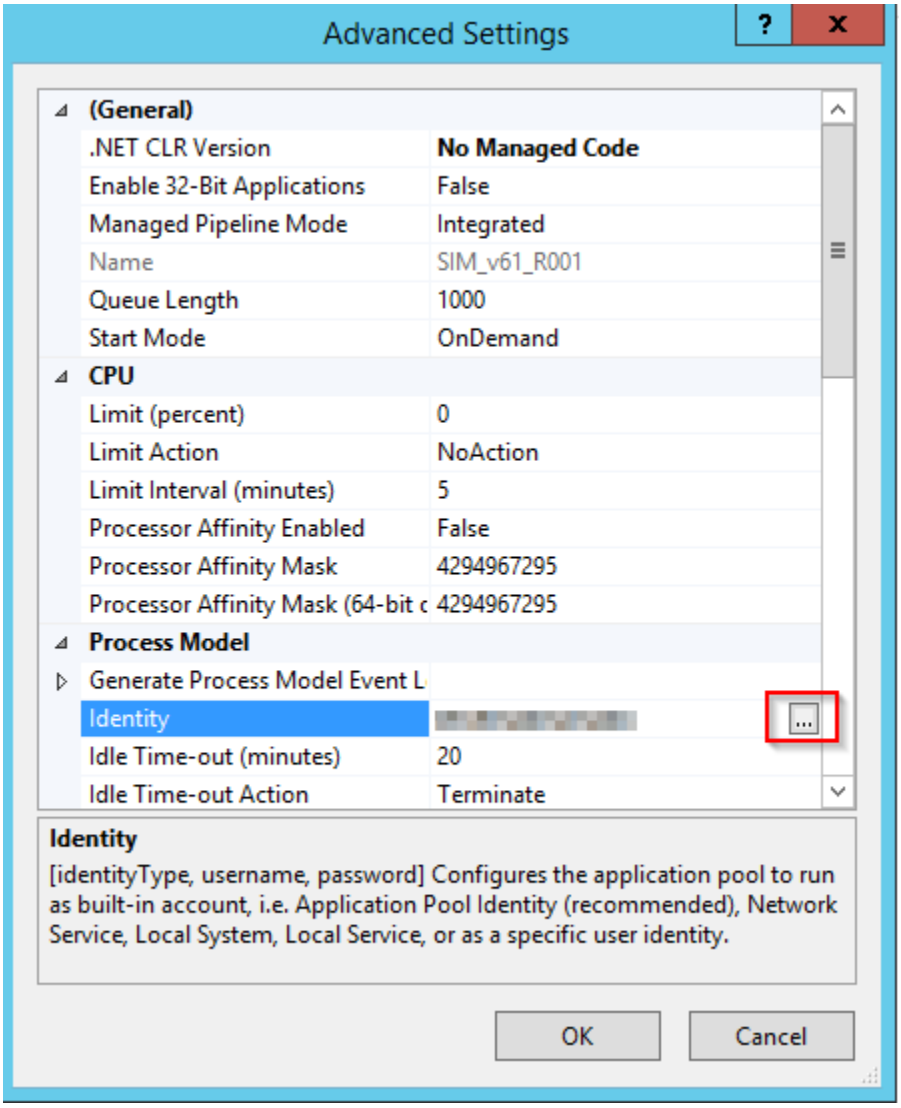
1.2.4 Configure IIS

Create IIS App Pool

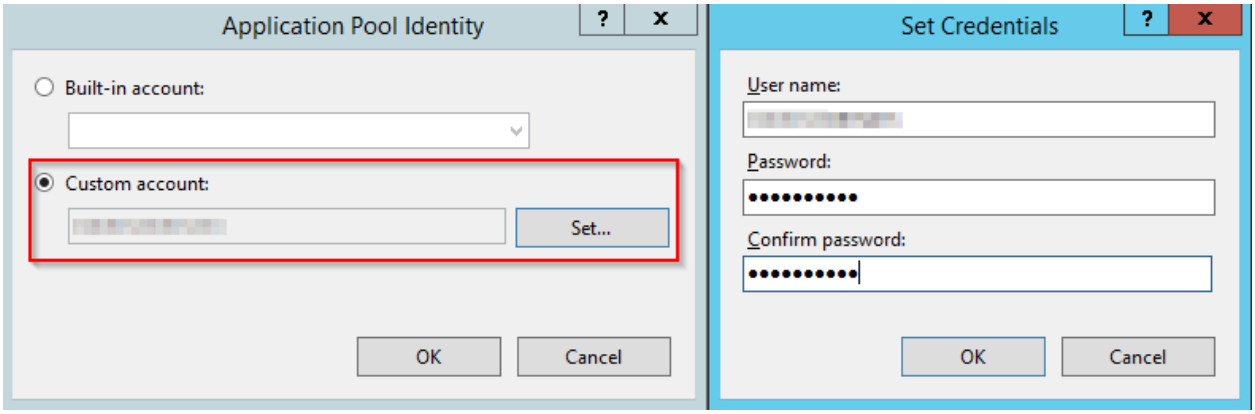
1. Go to IIS Manager and create an AppPool with .NET CLR version set to `No Managed Code` :



2. Go into the Advanced Settings of this AppPool and change the `Process Model - Identity`:



3. Make sure to use a custom **Active Directory** user account, i.e. the Service Account (`sim-svc-sql`) which has db_owner rights in the SIM v61 SQL database. This account is only used for accessing the own SQL database. Syntax: `DOMAIN\UserName`.

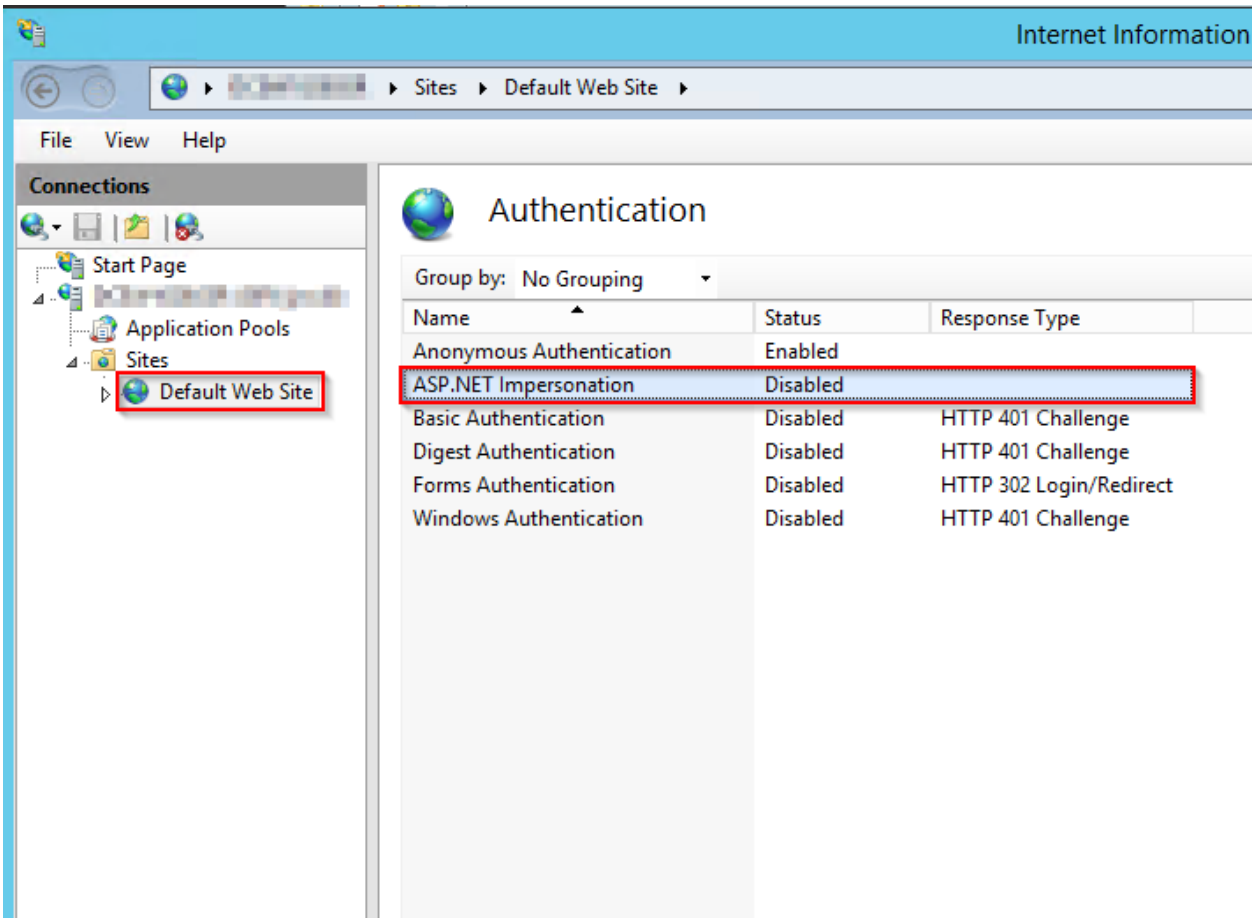


Create SilverMonkey folder

1. Create C:\SilverMonkey
2. Copy files from installation media
3. Change connection string in file **C:\SilverMonkey\v61\Config.xml** (XPath: //Configuration/DBConnection)

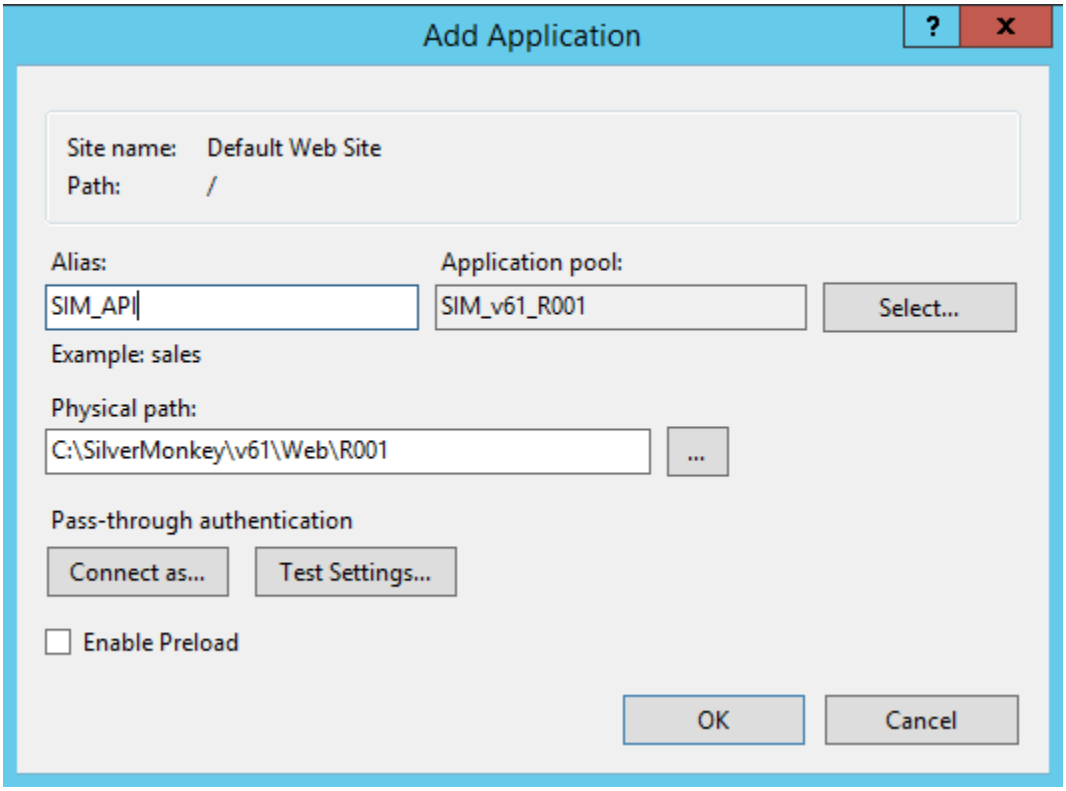
Create IIS Application

1. Go to IIS Manager, DefaultWebSite (or other Website, make sure to disable Impersonation). Impersonation is not supported for v61 and must not be inherited from Default Website to IIS application.



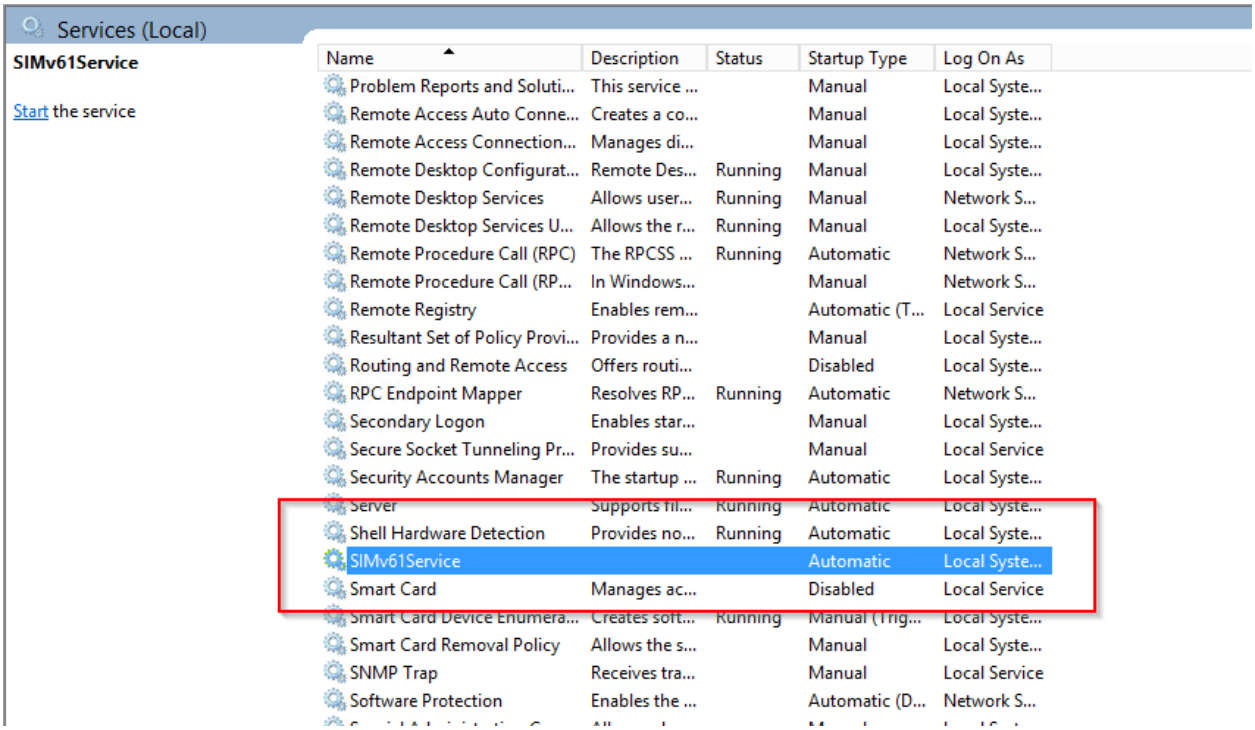
2. Add application, choose SIM AppPool (created above) and target to C:\SilverMonkey\Web\R001.

Hint: The alias defines the later URL: http://HOSTNAME/ALIAS

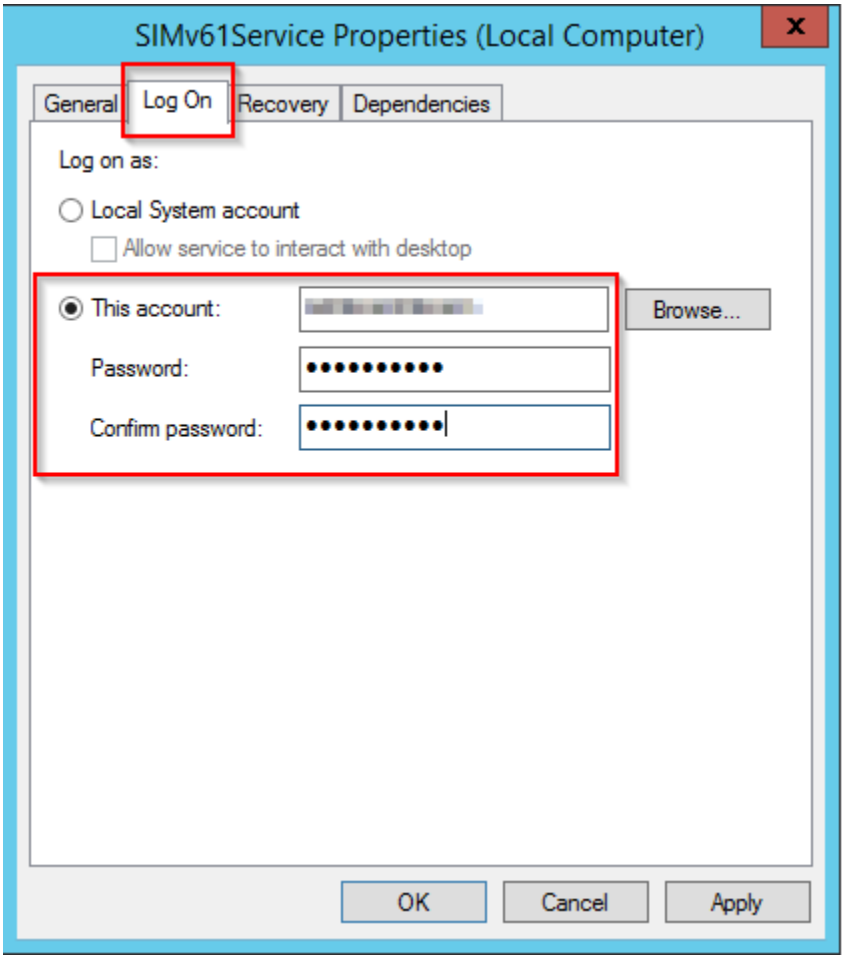


1.2.5 Install Windows Service

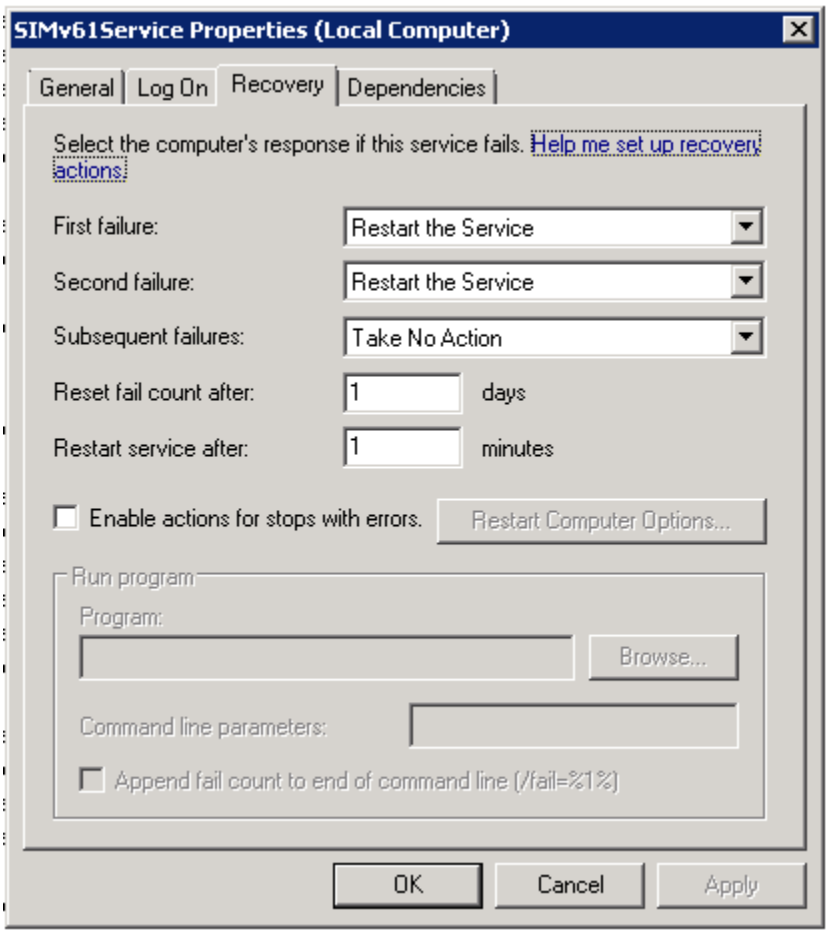
1. Go to C:\SilverMonkey\v61\WinService
2. Execute **Install.cmd** with administrative rights
3. Open services.msc and make sure that the Windows Service **SIMv61Service** is installed



4. Go into the properties of this service and change the Log On Account to the Service Account. This service account is used for the execution of every plugin run by the web service.



5. To optimize failure tollerance, please configure “Recovery” tab like the following:



1.2.6 Test Installation

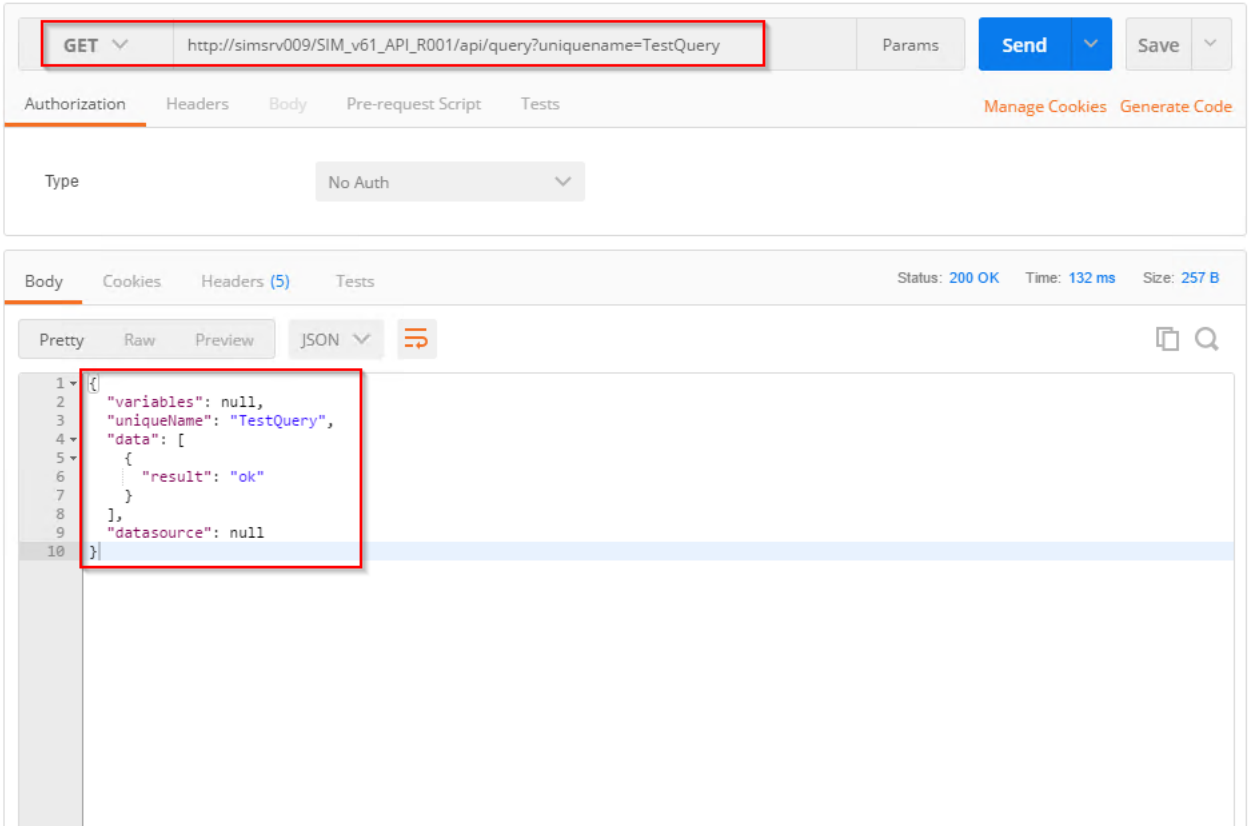
Note: For testing API download&install Postman: https://www.getpostman.com/apps

Test Query

Important: Try restarting the IIS Application/AppPool if you dont get the expected results!

1. Start Postman
2. Select **GET** as option
3. Enter URL: http://SERVERNAME/APP_NAME/api/query?uniquename=TestQuery
4. Hit execute

The following result should appear:



Test Queue

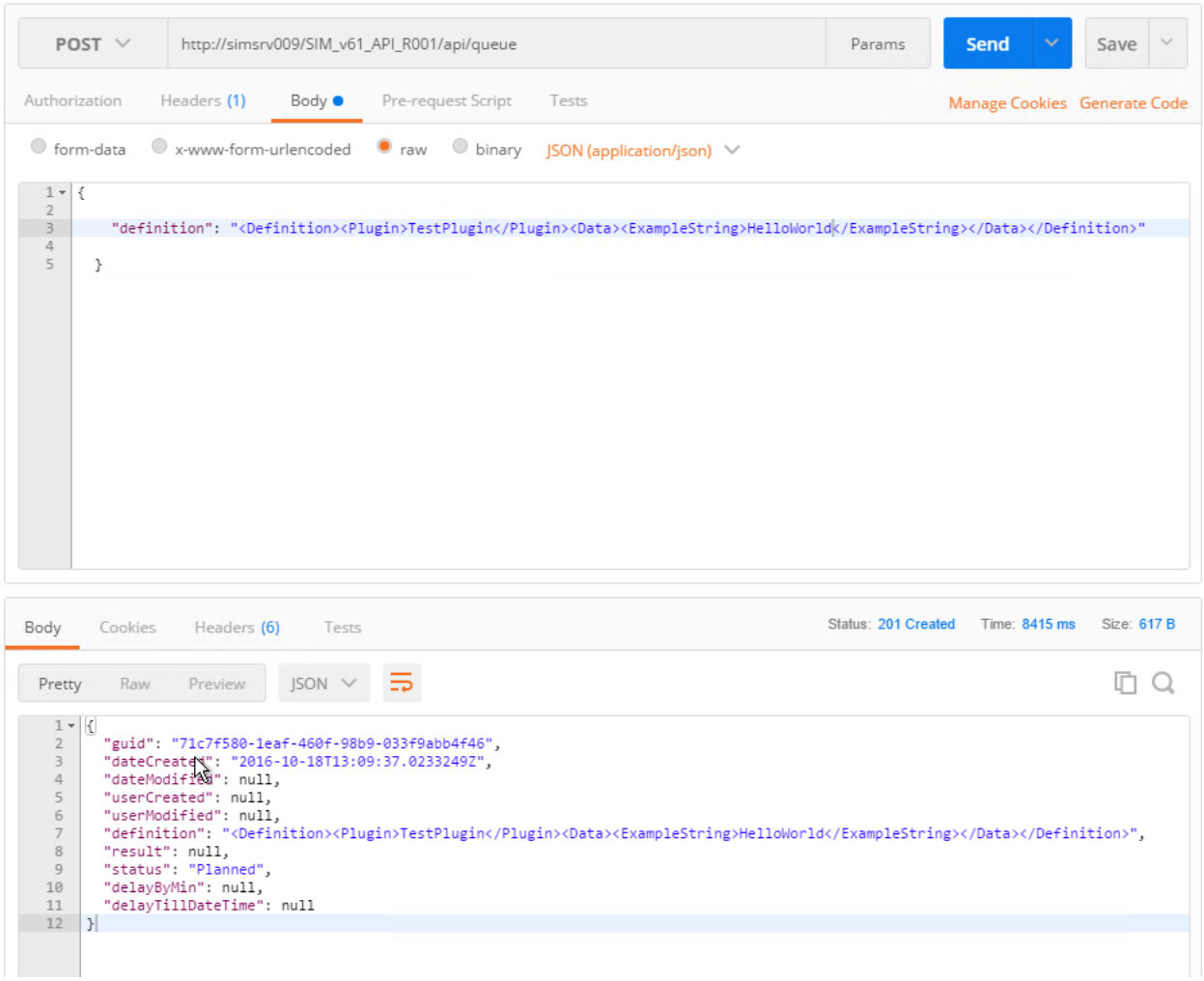
Important: Try restarting the IIS Application/AppPool if you dont get the expected results!

1. Start Postman
2. Select **POST** as option
3. Enter URL: http://SERVERNAME/APP_NAME/api/Queue
4. Add following code to body:

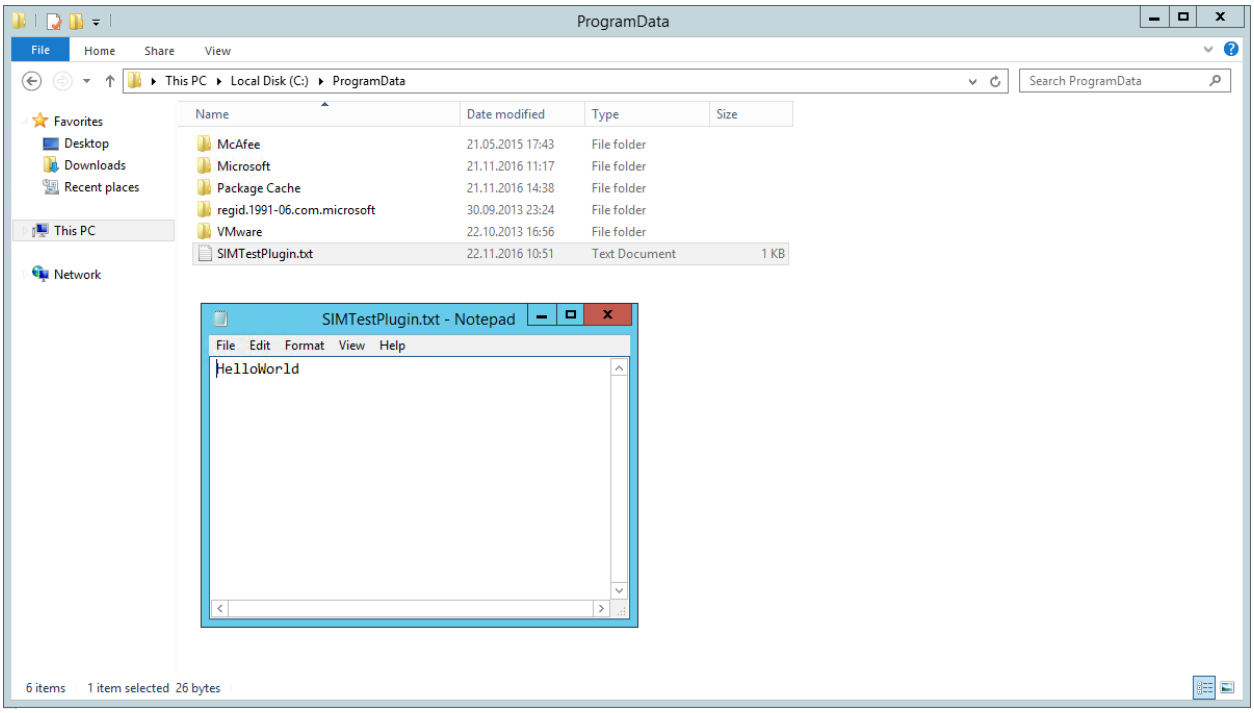
```
{"definition": "<Definition><Plugin>TestPlugin</Plugin><Data><ExampleString>HelloWorld
↪</ExampleString></Data></Definition>"}
```

5. Hit execute

The following result should appear:



Note: For testing Queue use the Test Plugin and Check in C:Programmdata for the testfile



1.3 Manual

Modules:

1.3.1 Manual for module “Webservice”

In this article:

Warning: This article is under construction! Please DO NOT use any of the instructions below, yet! You may cause damage to your system. This article will be finished soon.

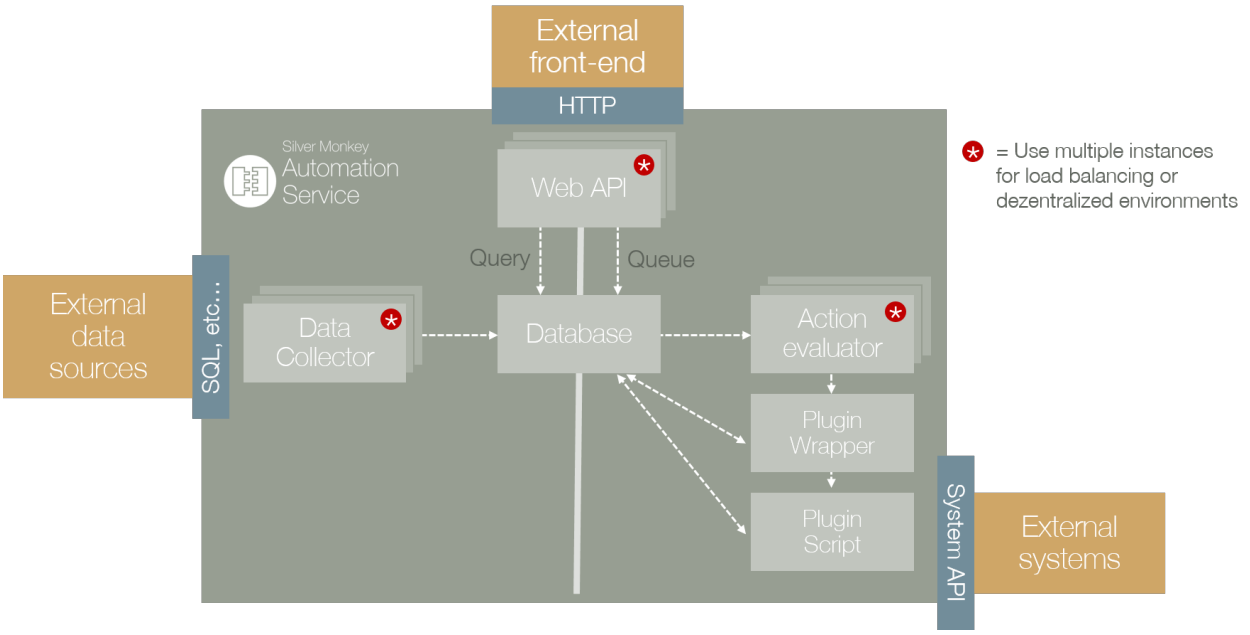
Authentication

Depending on the setting of the IIS application there are two possible authentication methods

a) Windows Authentication (recommended)

b) Authentication via firewall exception in IP base

Concept



The Webservice module consists of two main function: /queue and /query. Everything is accessable through a web api based on JSON format.

Queue For triggering and getting infos from (such as status) actions the /queue namespace have to be used.

All actions are created as planned actions in the SQL database table "queue". The "Action Evaluator" asks for planned actions. If an action is found, the Wrapper.ps1 is started with the information from `definition` XML and passes this data to the corresponding plugin PS1.

Query For retreiving dynamic data lists /query have to be used.

Queue

Adding a queue element for executing a powershell addon script

Creating Queue element via powershell

```
param(
[string]$definition,
[string]$url
)

Invoke-RestMethod -Uri "$url/api/queue?definition=$definition"
```

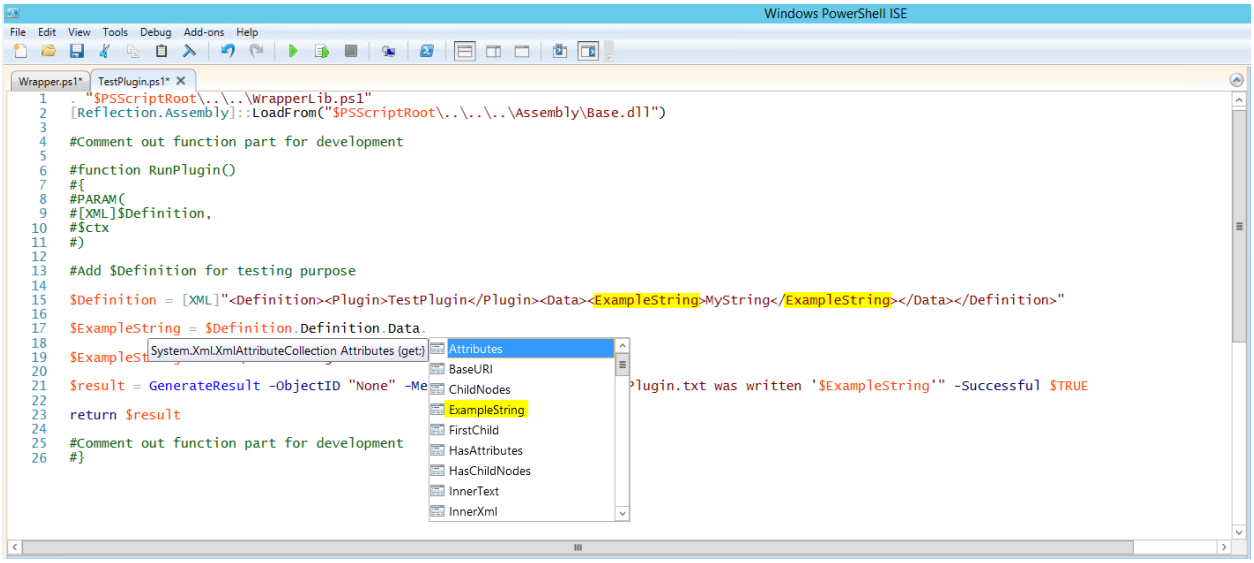
Creating a plugin

For creating plugins there are several rules:

1. Every plugin must consist of a main function (with specific parameters) which will be executed by the wrapper.ps1
2. Every plugin must return a specific class, which will be created by `GenerateResult`

```
. "$PSScriptRoot\..\..\WrapperLib.ps1"


function RunPlugin()
{
PARAM(
[XML]$Definition,
$ctx
)

    Try
    {

        #Place here general plugin functions


        #Generate a plugin result:
        $result = GenerateResult -ObjectID "None" -Message "Some return description␣
↪for queue result..." -Successful $TRUE

    }
    Catch
    {

        $ErrorMessage = $_.Exception.Message
        $result = GenerateResult -ObjectID "None" -Message "Unhandled exception␣
↪thrown while running plugin: $ErrorMessage" -Successful $FALSE

    }

    return $result

}
```

For development the wrapper behaviour can be simulated by commenting out the function part, and adding the XML string directly. Once the Powershell ISE run the XML variable declaration, the XML schema is avaiable through code completion:



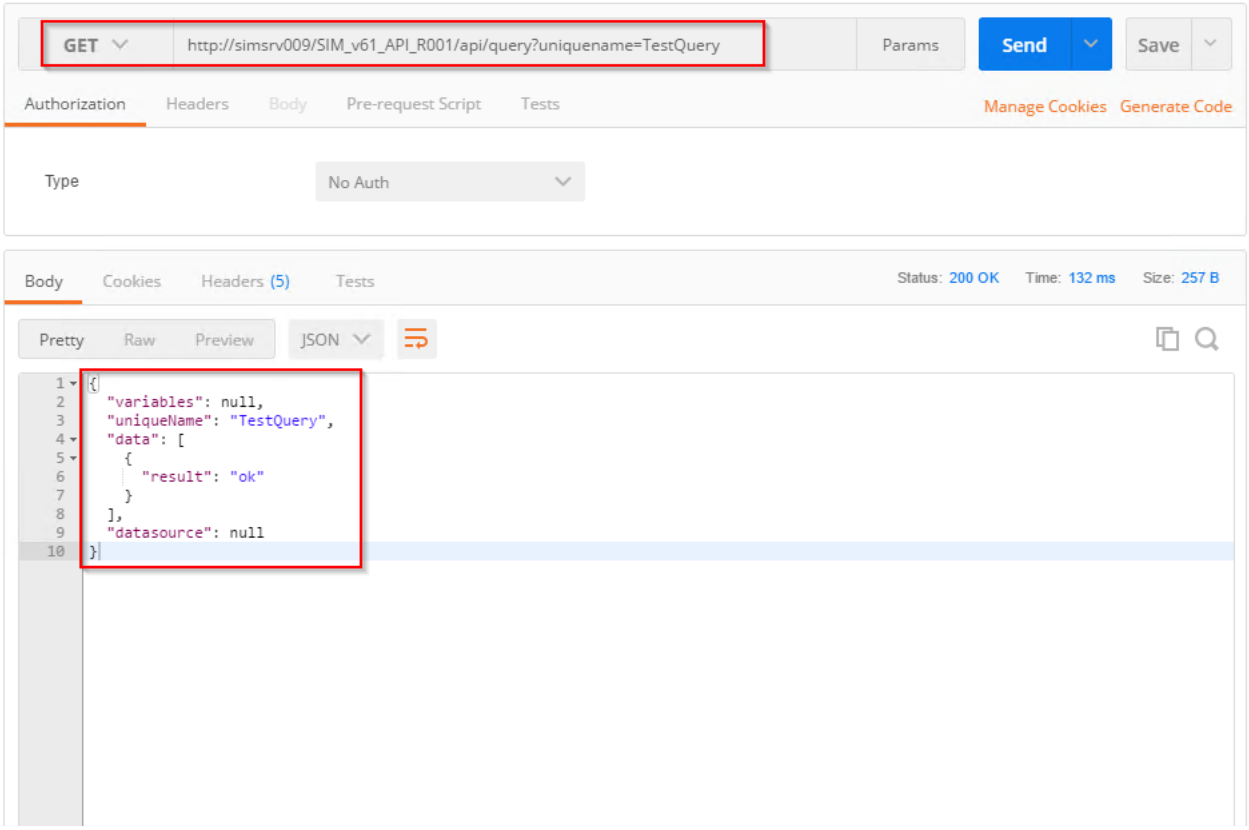

```
. "$PSScriptRoot\..\..\WrapperLib.ps1"

#Comment out function part for development

#function RunPlugin()
#{
#PARAM(
#[XML]$Definition,
#$ctx
#)

#Add $Definition for testing purpose

$Definition = [XML]"<Definition><Plugin>TestPlugin</Plugin><Data><ExampleString>
↪MyString</ExampleString></Data></Definition>"

$ExampleString = $Definition.Definition.Data.ExampleString

$ExampleString >> "C:\SIMTestPlugin.txt"

$result = GenerateResult -ObjectID "None" -Message "To file C:\SIMTestPlugin.txt was␣
↪written '$ExampleString'" -Successful $TRUE

return $result

#Comment out function part for development
#}
```

Query

Getting information from the web service.

To get data from the web service, simply the web service has the be called by GET with an URL like the following: http://SERVERNAME/APP_NAME/api/query?uniquename=TestQuery

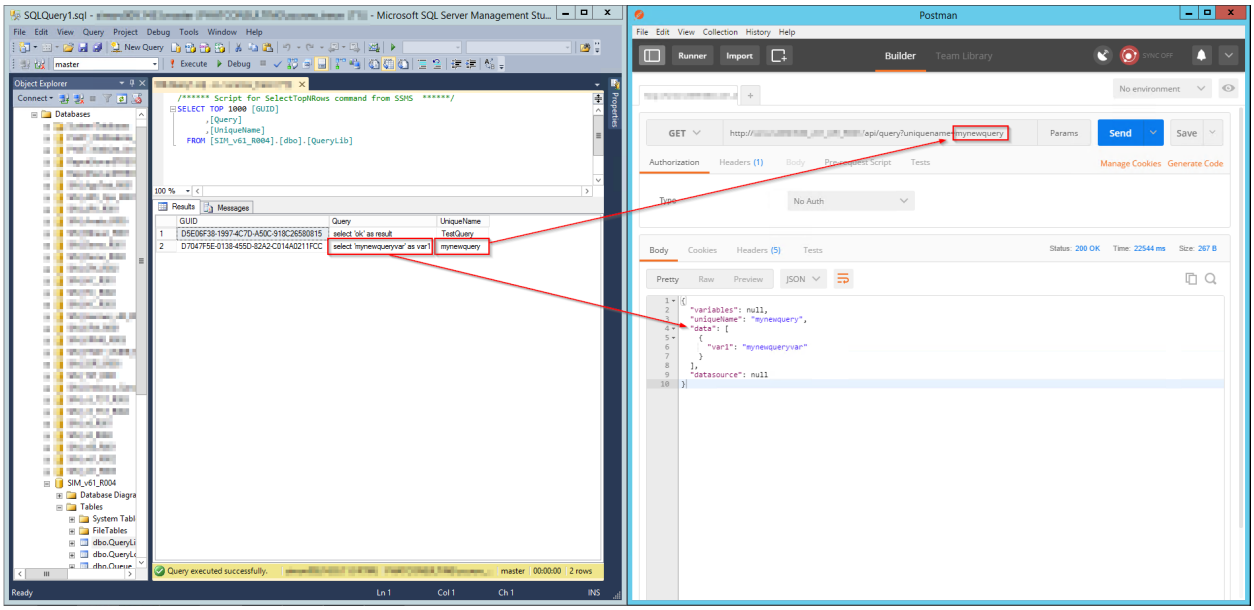
As a result, the configured query `TestQuery` is executed within the web service and is returned as a JSON array:



To create queries, the table `[QueryLib]` has to be extended by a new entry:

```
INSERT INTO [dbo].[QueryLib]
           ([Query]
           ,[UniqueName])
     VALUES
           ('select ''mynewqueryvar'' as var1'
           ,'mynewquery')
```

the query will be available by the supplied `[UniqueName]` value:



1.3.2 Extensions

Extensions for Silver Monkey v61 are .Net DLL based feature sets which can be imported in Powershell scripts.

Contents:

Manual for plugin "ConfigMgr"

In this article:

General assumptions:

1. Actions with `Workflow` prefix combine multiple actions. Workflow actions exist to standardize frequent used action combos.
2. Every action returns `Base.Result`. For more info go to Base.Result article.
3. Most of the functions accept parameters via a specific parameter class

Connection

In the Connection namespace all connection relevant parameters are stored.

Authentication Methods (or integrated authentication)

a) Windows Auth: Do not set the corresponding credentials attributes `UserName`, `UserPassword`.

Example:

```
$Assembly = [Reflection.Assembly]::LoadFile("$PSScriptRoot\Base.dll")
$Assembly = [Reflection.Assembly]::LoadFile("$PSScriptRoot\ConfigMgr.dll")

$ConfigMgrConnectionSettings = New-Object SIM.ConfigMgr.ConnectionSettings
$ConfigMgrConnectionSettings.WMIHostName = 'MyConfigMgrHost'
$ConfigMgrConnectionSettings.SiteCode = 'P01'
$ConfigMgrConnectionSettings.SQLHostname = $ConfigMgrConnectionSettings.WMIHostName
```

b) Clear text: Set the corresponding credentials attributes `UserName`, `UserPassword`.

Example:

```
$Assembly = [Reflection.Assembly]::LoadFile("$PSScriptRoot\Base.dll")
$Assembly = [Reflection.Assembly]::LoadFile("$PSScriptRoot\ConfigMgr.dll")

$ConfigMgrConnectionSettings = New-Object SIM.ConfigMgr.ConnectionSettings
$ConfigMgrConnectionSettings.WMIHostName = 'MyConfigMgrHost'
$ConfigMgrConnectionSettings.SiteCode = 'P01'
$ConfigMgrConnectionSettings.WMIUserDomainName = 'MyDomain'
```

(continues on next page)

(continued from previous page)

```
$ConfigMgrConnectionSettings.WMIUserName = 'administrator'
$ConfigMgrConnectionSettings.WMIUserPassword = 'Password123'
$ConfigMgrConnectionSettings.SQLHostname = $ConfigMgrConnectionSettings.WMIHostName;
$ConfigMgrConnectionSettings.SQLUserName = "administrator";
$ConfigMgrConnectionSettings.SQLUserPassword = "Password123";
```

c) Powershell Secure String: Will be supported in a later version.

Examples

Create collection, assign app, create membership

```
$Assembly = [Reflection.Assembly]::LoadFile("$PSScriptRoot\Base.dll")
$Assembly = [Reflection.Assembly]::LoadFile("$PSScriptRoot\ConfigMgr.dll")

$ConfigMgrConnectionSettings = New-Object SIM.ConfigMgr.ConnectionSettings
$ConfigMgrConnectionSettings.WMIHostName = 'localhost'
$ConfigMgrConnectionSettings.SiteCode = 'SIM'
$ConfigMgrConnectionSettings.SQLHostname = $ConfigMgrConnectionSettings.WMIHostName;

$ConfigMgrConnection = New-Object SIM.ConfigMgr.Connection(
↪$ConfigMgrConnectionSettings)

$res = New-Object Base.Result("Starting ConfigMgr script...")

$res.ChildAdd($ConfigMgrConnection.Connect())

if ($res.Successful -eq $true)

{
    $ApplicationName = "Test App 1"
    $CollectionName = "Test Collection 1"
    $ComputerName = "TestComputer1"


    $ConfigMgrComputersObject = New-Object SIM.ConfigMgr.Computers.Computer(
↪$ConfigMgrConnection)

    $objParameterCollection = New-Object SIM.ConfigMgr.ParameterCollection;
    $objParameterCollection.CollectionName = $CollectionName;
    $objParameterCollection.CollectionType = [SIM.ConfigMgr.
↪ParameterCollectionTypes]::Device
    $objParameterCollection.Folder = New-Object SIM.ConfigMgr.
↪ParameterFolder(16777217);

    #Check if collection exists

    [Base.Result] $resFindColletion = $ConfigMgrComputersObject.
↪GetWMIPredefinedClassProperty([SIM.ConfigMgr.
↪SuperClass+ConfigMgrQueries]::CollectionId_By_CollectionName,
↪$objParameterCollection.CollectionName)

    if ($resFindColletion.ExitCode.Code -eq [Base.
↪ResultSuperClass+ExitCodeType]::ElementFound)
```

(continues on next page)

(continued from previous page)

```
    {
        $objParameterCollection.CollectionId = $resFindColletion.ReturnObj;
    }
    else
    {

        #If not exists, create collection

        $res.ChildAdd($ConfigMgrComputersObject.CollectionCreate(
↪$objParameterCollection))

        if ($res.Successful -eq $true)

        {

            #Create assignment to application, if application does not exist, the␣
↪action will fail with corresponding message

            $objParameterAssignment = New-Object SIM.ConfigMgr.ParameterAssignment;
            $objParameterAssignment.Collection = $objParameterCollection;
            $objParameterAssignment.Application = New-Object SIM.ConfigMgr.
↪ParameterApplication($ApplicationName);
            $objParameterAssignment.AssignmentType = [SIM.ConfigMgr.
↪ParameterAssignment+AssignmentTypes]::Install
            $objParameterAssignment.OfferType = [SIM.ConfigMgr.
↪ParameterAssignment+OfferTypes]::Optional

            $res.ChildAdd($ConfigMgrComputersObject.AssignmentCreate(
↪$objParameterAssignment))

        }
    }

    #If everthing before was executed without any errors, a collection membership is␣
↪created

    if ($res.Successful -eq $true)
    {


        $objParametersCollectionMembership = New-Object SIM.ConfigMgr.
↪ParametersCollectionMembership;

        $objParametersCollectionMembership.CollectionMembershipType = [SIM.ConfigMgr.
↪ParametersCollectionMembership+CollectionMembershipTypes]::RuleDirectComputer;
        $objParametersCollectionMembership.Collection = $objParameterCollection
        $objParametersCollectionMembership.ResourceName = $ComputerName;

        $res.ChildAdd($ConfigMgrComputersObject.CollectionMembershipAdd(
↪$objParametersCollectionMembership));

        $res.ChildAdd($ConfigMgrComputersObject.CollectionMembershipRequestRefresh(
↪$objParametersCollectionMembership,10,20000,20000));


    }
```

(continues on next page)

(continued from previous page)

```
}

$res.Dump()
```

Example “reseting” existing computer

This example shows how to

1. Check whether a computer is found by Hostname
2. Delete every direct collection membership
3. Reset cmputer variables
4. Add new variables
5. Reset PXE flags
6. Add to specific collection

```
$Assembly = [Reflection.Assembly]::LoadFile("$PSScriptRoot\Base.dll")
$Assembly = [Reflection.Assembly]::LoadFile("$PSScriptRoot\ConfigMgr.dll")

$ConfigMgrConnectionSettings = New-Object SIM.ConfigMgr.ConnectionSettings
$ConfigMgrConnectionSettings.WMIHostName = 'localhost'
$ConfigMgrConnectionSettings.SiteCode = 'SIM'
$ConfigMgrConnectionSettings.SQLHostname = $ConfigMgrConnectionSettings.WMIHostName;

$ConfigMgrConnection = New-Object SIM.ConfigMgr.Connection(
↪$ConfigMgrConnectionSettings)

$res = New-Object Base.Result("Starting ConfigMgr script...")

$res.ChildAdd($ConfigMgrConnection.Connect())

if ($res.Successful -eq $true)

{

    $ConfigMgrComputersSettings = New-Object SIM.ConfigMgr.Computers.
↪ComputerParameters

    #Define Target Computer by Name
    $ConfigMgrComputersSettings.ComputerName = 'TestComputer1'

    $res.ChildAdd($ConfigMgrComputersSettings.TryResolve($ConfigMgrConnection))

    $ResourceId = $ConfigMgrComputersSettings.ResourceId

    If ($ResourceId)
    {


        # REMOVE FROM COLLECTIONS
```

(continues on next page)

(continued from previous page)

```
        $Command = New-Object System.Data.SQLClient.SQLCommand
        $Command.Connection = $ConfigMgrConnection.SQLConnection
        $Command.CommandText = "SELECT [CollectionID] FROM [v_CollectionRuleDirect]␣
↪WHERE [ResourceID] = '$ResourceId' AND [ResourceType] = 5"

        $DataAdapter = new-object System.Data.SqlClient.SqlDataAdapter $Command
        $Dataset = new-object System.Data.Dataset

        $DataAdapter.Fill($Dataset)

        $QueryResultCount = $Dataset.Tables[0].Rows.ToString()

        if ($QueryResultCount -gt 0)
        {

            Foreach ($Row in $Dataset.Tables[0])

            {

                $ConfigMgrComputersObject = New-Object SIM.ConfigMgr.Computers.
↪Computer($ConfigMgrConnection)
                $objParametersCollectionMembership = New-Object SIM.ConfigMgr.
↪ParametersCollectionMembership;

                $objParametersCollectionMembership.CollectionMembershipType = [SIM.
↪ConfigMgr.
↪ParametersCollectionMembership+CollectionMembershipTypes]::RuleDirectComputer;
                $objParametersCollectionMembership.CollectionId = $Row["CollectionID"]
                $objParametersCollectionMembership.ResourceId = $ResourceId

                $res.ChildAdd($ConfigMgrComputersObject.CollectionMembershipRemove(
↪$objParametersCollectionMembership));

            }

        }

        # ADD RES VARS

        $res.ChildAdd($ConfigMgrComputersObject.DeleteVariables(
↪$ConfigMgrComputersSettings))

        $MyVar1 = New-Object SIM.ConfigMgr.Parameter("MyVar1", "Value1")
        $MyVar2 = New-Object SIM.ConfigMgr.Parameter("MyVar2", "Value2")

        $ConfigMgrComputersSettings.Variables.Add($MyVar1)
        $ConfigMgrComputersSettings.Variables.Add($MyVar2)

        $res.ChildAdd($ConfigMgrComputersObject.AddResourceVariables(
↪$ConfigMgrComputersSettings))

        # ClearPxeAdvertisementResource

        $res.ChildAdd($ConfigMgrComputersObject.ClearPxeAdvertisementResource(
↪$ConfigMgrComputersSettings))

        # ADD TO COLL
```

(continues on next page)

(continued from previous page)

```
        $objParametersCollectionMembership = New-Object SIM.ConfigMgr.
↪ParametersCollectionMembership;

        $objParametersCollectionMembership.CollectionMembershipType = [SIM.ConfigMgr.
↪ParametersCollectionMembership+CollectionMembershipTypes]::RuleDirectComputer;
        $objParametersCollectionMembership.CollectionName = "DeployColl";
        $objParametersCollectionMembership.ResourceName = $ConfigMgrComputersSettings.
↪ComputerName

        $res.ChildAdd($ConfigMgrComputersObject.CollectionMembershipAdd(
↪$objParametersCollectionMembership));



    }

}

$res.Dump()
```

Generic

GetWMIPredefinedClassProperty

Get for predefined WMI classes specific properties. Thise queries can either be used to lookup data or to determine of the object exists.

Definition:

```
public Base.Result GetWMIPredefinedClassProperty(ConfigMgrQueries Query, string␣
↪FindValue, string FindValue2 = "")
```

a) Get data

Example:

b) Check if object exists.

Example:

Computers

Everthing concering computer management is stored in the `Computers` namespace.

AddResourceVariables

Adds variables to a specific computer system.

Example:

```
$Assembly = [Reflection.Assembly]::LoadFile("$PSScriptRoot\Base.dll")
$Assembly = [Reflection.Assembly]::LoadFile("$PSScriptRoot\ConfigMgr.dll")

$ConfigMgrConnectionSettings = New-Object SIM.ConfigMgr.ConnectionSettings
$ConfigMgrConnectionSettings.WMIHostName = 'localhost'
$ConfigMgrConnectionSettings.SiteCode = 'SIM'
$ConfigMgrConnectionSettings.SQLHostname = $ConfigMgrConnectionSettings.WMIHostName;

$ConfigMgrConnection = New-Object SIM.ConfigMgr.Connection(
↪$ConfigMgrConnectionSettings)

$res = New-Object Base.Result("Starting ConfigMgr script...")

$res.ChildAdd($ConfigMgrConnection.Connect())

if ($res.Successful -eq $true)

{

    $ConfigMgrComputersObject = New-Object SIM.ConfigMgr.Computers.Computer(
↪$ConfigMgrConnection)

    $ConfigMgrComputersSettings = New-Object SIM.ConfigMgr.Computers.
↪ComputerParameters

    #Define Target Computer by ResourceId
    $ConfigMgrComputersSettings.ResourceId = '16777221'
    #Define Target Computer by ComputerName
    $ConfigMgrComputersSettings.ComputerName = 'TestComputer478'

    $MyVar1 = New-Object SIM.ConfigMgr.Parameter("MyVar1", "Value1")
    $MyVar2 = New-Object SIM.ConfigMgr.Parameter("MyVar2", "Value2")

    $ConfigMgrComputersSettings.Variables.Add($MyVar1)
    $ConfigMgrComputersSettings.Variables.Add($MyVar2)

    $res.ChildAdd($ConfigMgrComputersObject.AddResourceVariables(
↪$ConfigMgrComputersSettings))

}

$res.Dump()
```

AdvertisementCreate

Creates an package advertisement for a specific collection

Example:

```
$Assembly = [Reflection.Assembly]::LoadFile("$PSScriptRoot\Base.dll")
$Assembly = [Reflection.Assembly]::LoadFile("$PSScriptRoot\Database.dll")
$Assembly = [Reflection.Assembly]::LoadFile("$PSScriptRoot\ConfigMgr.dll")

$ConfigMgrConnectionSettings = New-Object SIM.ConfigMgr.ConnectionSettings
$ConfigMgrConnectionSettings.WMIHostName = 'localhost'
$ConfigMgrConnectionSettings.SiteCode = 'P01'
```

(continues on next page)

(continued from previous page)

```
$ConfigMgrConnectionSettings.SQLHostname = $ConfigMgrConnectionSettings.WMIHostName;

$ConfigMgrConnection = New-Object SIM.ConfigMgr.Connection(
↪$ConfigMgrConnectionSettings)

$res = New-Object Base.Result("Starting ConfigMgr script...")

$res.ChildAdd($ConfigMgrConnection.Connect())

if ($res.Successful -eq $true)

{

    $ConfigMgrComputersObject = New-Object SIM.ConfigMgr.Computers.Computer(
↪$ConfigMgrConnection)

    $objParameterCollection = New-Object SIM.ConfigMgr.ParameterCollection;
    $objParameterCollection.CollectionName = "TestCollection";
    $objParameterCollection.CollectionType = [SIM.ConfigMgr.
↪ParameterCollectionTypes]::Device

    $objParameterAdvertisement = New-Object SIM.ConfigMgr.ParameterAdvertisement;
    $objParameterAdvertisement.Collection = $objParameterCollection;
    $objParameterAdvertisement.Package = New-Object SIM.ConfigMgr.ParameterPackage(
↪"P0100050");
    $objParameterAdvertisement.Package.ProgramName = "Install"
    $objParameterAdvertisement.OfferType = [SIM.ConfigMgr.
↪ParameterAssignment+OfferTypes]::Optional

    $res.ChildAdd($ConfigMgrComputersObject.AdvertisementCreate(
↪$objParameterAdvertisement))

}

$res.Dump()
```

AssignmentCreate

Creates an application assignment for users or computers

Example:

```
$Assembly = [Reflection.Assembly]::LoadFile("$PSScriptRoot\Base.dll")
$Assembly = [Reflection.Assembly]::LoadFile("$PSScriptRoot\ConfigMgr.dll")

$ConfigMgrConnectionSettings = New-Object SIM.ConfigMgr.ConnectionSettings
$ConfigMgrConnectionSettings.WMIHostName = 'localhost'
$ConfigMgrConnectionSettings.SiteCode = 'SIM'
$ConfigMgrConnectionSettings.SQLHostname = $ConfigMgrConnectionSettings.WMIHostName;

$ConfigMgrConnection = New-Object SIM.ConfigMgr.Connection(
↪$ConfigMgrConnectionSettings)

$res = New-Object Base.Result("Starting ConfigMgr script...")
```

(continues on next page)

(continued from previous page)

```
$res.ChildAdd($ConfigMgrConnection.Connect())

if ($res.Successful -eq $true)

{

    $ConfigMgrComputersObject = New-Object SIM.ConfigMgr.Computers.Computer(
↪$ConfigMgrConnection)

    $objParameterCollection = New-Object SIM.ConfigMgr.ParameterCollection;
    $objParameterCollection.CollectionName = "Test Collection 1";
    $objParameterCollection.CollectionType = [SIM.ConfigMgr.
↪ParameterCollectionTypes]::Device

    $objParameterAssignment = New-Object SIM.ConfigMgr.ParameterAssignment;
    $objParameterAssignment.Collection = $objParameterCollection;
    $objParameterAssignment.Application = New-Object SIM.ConfigMgr.
↪ParameterApplication("Test App 1");
    $objParameterAssignment.AssignmentType = [SIM.ConfigMgr.
↪ParameterAssignment+AssignmentTypes]::Install
    $objParameterAssignment.OfferType = [SIM.ConfigMgr.
↪ParameterAssignment+OfferTypes]::Optional

    $res.ChildAdd($ConfigMgrComputersObject.AssignmentCreate($objParameterAssignment))

}

$res.Dump()
```

ClearPxeAdvertisementResource

Clears the PXE advertisements for a computer.

Example:

```
$Assembly = [Reflection.Assembly]::LoadFile("$PSScriptRoot\Base.dll")
$Assembly = [Reflection.Assembly]::LoadFile("$PSScriptRoot\ConfigMgr.dll")

$ConfigMgrConnectionSettings = New-Object SIM.ConfigMgr.ConnectionSettings
$ConfigMgrConnectionSettings.WMIHostName = 'localhost'
$ConfigMgrConnectionSettings.SiteCode = 'SIM'
$ConfigMgrConnectionSettings.SQLHostname = $ConfigMgrConnectionSettings.WMIHostName;

$ConfigMgrConnection = New-Object SIM.ConfigMgr.Connection(
↪$ConfigMgrConnectionSettings)

$res = New-Object Base.Result("Starting ConfigMgr script...")

$res.ChildAdd($ConfigMgrConnection.Connect())

if ($res.Successful -eq $true)

{

    $ConfigMgrComputersObject = New-Object SIM.ConfigMgr.Computers.Computer(
↪$ConfigMgrConnection)
```

(continues on next page)

(continued from previous page)

```
    $ConfigMgrComputersSettings = New-Object SIM.ConfigMgr.Computers.
↪ComputerParameters
    $ConfigMgrComputersSettings.ResourceId = "16777221"

    $res.ChildAdd($ConfigMgrComputersObject.ClearPxeAdvertisementResource(
↪$ConfigMgrComputersSettings))

}

$res.Dump()
```

CollectionCreate

Create a collection.

Example:

```
$Assembly = [Reflection.Assembly]::LoadFile("$PSScriptRoot\Base.dll")
$Assembly = [Reflection.Assembly]::LoadFile("$PSScriptRoot\ConfigMgr.dll")

$ConfigMgrConnectionSettings = New-Object SIM.ConfigMgr.ConnectionSettings
$ConfigMgrConnectionSettings.WMIHostName = 'localhost'
$ConfigMgrConnectionSettings.SiteCode = 'SIM'
$ConfigMgrConnectionSettings.SQLHostname = $ConfigMgrConnectionSettings.WMIHostName;

$ConfigMgrConnection = New-Object SIM.ConfigMgr.Connection(
↪$ConfigMgrConnectionSettings)

$res = New-Object Base.Result("Starting ConfigMgr script...")

$res.ChildAdd($ConfigMgrConnection.Connect())

if ($res.Successful -eq $true)

{

    $ConfigMgrComputersObject = New-Object SIM.ConfigMgr.Computers.Computer(
↪$ConfigMgrConnection)


    $objParameterCollection = New-Object SIM.ConfigMgr.ParameterCollection;

    $objParameterCollection.CollectionName = "Test Collection 1";
    $objParameterCollection.CollectionType = [SIM.ConfigMgr.
↪ParameterCollectionTypes]::Device
    $objParameterCollection.Folder = New-Object SIM.ConfigMgr.
↪ParameterFolder(16777217); #If FolderId is defined, the collection will be moved on␣
↪creation
    $objParameterCollection.LimitingCollectionId = "SMS00001";

    $res.ChildAdd($ConfigMgrComputersObject.CollectionCreate($objParameterCollection))

}

$res.Dump()
```

CollectionMembershipAdd

Currently the following methods are supported by this action:

- RuleDirectUser
- RuleDirectComputer
- RuleInclude

Example for RuleDirectComputer:

```
$Assembly = [Reflection.Assembly]::LoadFile("$PSScriptRoot\Base.dll")
$Assembly = [Reflection.Assembly]::LoadFile("$PSScriptRoot\ConfigMgr.dll")

$ConfigMgrConnectionSettings = New-Object SIM.ConfigMgr.ConnectionSettings
$ConfigMgrConnectionSettings.WMIHostName = 'localhost'
$ConfigMgrConnectionSettings.SiteCode = 'SIM'
$ConfigMgrConnectionSettings.SQLHostname = $ConfigMgrConnectionSettings.WMIHostName;

$ConfigMgrConnection = New-Object SIM.ConfigMgr.Connection(
↪$ConfigMgrConnectionSettings)

$res = New-Object Base.Result("Starting ConfigMgr script...")

$res.ChildAdd($ConfigMgrConnection.Connect())

if ($res.Successful -eq $true)

{

    $ConfigMgrComputersObject = New-Object SIM.ConfigMgr.Computers.Computer(
↪$ConfigMgrConnection)

    $objParametersCollectionMembership = New-Object SIM.ConfigMgr.
↪ParametersCollectionMembership;

    $objParametersCollectionMembership.CollectionMembershipType = [SIM.ConfigMgr.
↪ParametersCollectionMembership+CollectionMembershipTypes]::RuleDirectComputer;
    $objParametersCollectionMembership.CollectionName = "ParentColl";
    $objParametersCollectionMembership.ResourceName = "TestComputer478";

    $res.ChildAdd($ConfigMgrComputersObject.CollectionMembershipAdd(
↪$objParametersCollectionMembership));
}

$res.Dump()
```

Example for RuleInclude:

```
$Assembly = [Reflection.Assembly]::LoadFile("$PSScriptRoot\Base.dll")
$Assembly = [Reflection.Assembly]::LoadFile("$PSScriptRoot\ConfigMgr.dll")

$ConfigMgrConnectionSettings = New-Object SIM.ConfigMgr.ConnectionSettings
$ConfigMgrConnectionSettings.WMIHostName = 'localhost'
$ConfigMgrConnectionSettings.SiteCode = 'SIM'
$ConfigMgrConnectionSettings.SQLHostname = $ConfigMgrConnectionSettings.WMIHostName;

$ConfigMgrConnection = New-Object SIM.ConfigMgr.Connection(
↪$ConfigMgrConnectionSettings)
```

(continues on next page)

(continued from previous page)

```
$res = New-Object Base.Result("Starting ConfigMgr script...")

$res.ChildAdd($ConfigMgrConnection.Connect())

if ($res.Successful -eq $true)

{

    $ConfigMgrComputersObject = New-Object SIM.ConfigMgr.Computers.Computer(
↪$ConfigMgrConnection)

    $objParametersCollectionMembership = New-Object SIM.ConfigMgr.
↪ParametersCollectionMembership;

    $objParametersCollectionMembership.CollectionMembershipType = [SIM.ConfigMgr.
↪ParametersCollectionMembership+CollectionMembershipTypes]::RuleInclude;
    $objParametersCollectionMembership.CollectionName = "ParentColl";
    $objParametersCollectionMembership.ResourceName = "ChildColl";

    $res.ChildAdd($ConfigMgrComputersObject.CollectionMembershipAdd(
↪$objParametersCollectionMembership));
}

$res.Dump()
```

CollectionMembershipRequestRefresh

Requests a refresh of the effective collection memberships. The request will be performed after the corresponding resource was found in the limiting collection.

Definition:

```
  Base.Result CollectionMembershipRequestRefresh(ParametersCollectionMembership␣
↪objParams, int RetriesMax = 45, int RetryWaitMilisec = 20000, int SleepAfterFound =␣
↪60000)
```

Example:

```
$Assembly = [Reflection.Assembly]::LoadFile("$PSScriptRoot\Base.dll")
$Assembly = [Reflection.Assembly]::LoadFile("$PSScriptRoot\ConfigMgr.dll")

$ConfigMgrConnectionSettings = New-Object SIM.ConfigMgr.ConnectionSettings
$ConfigMgrConnectionSettings.WMIHostName = 'localhost'
$ConfigMgrConnectionSettings.SiteCode = 'SIM'
$ConfigMgrConnectionSettings.SQLHostname = $ConfigMgrConnectionSettings.WMIHostName;

$ConfigMgrConnection = New-Object SIM.ConfigMgr.Connection(
↪$ConfigMgrConnectionSettings)

$res = New-Object Base.Result("Starting ConfigMgr script...")

$res.ChildAdd($ConfigMgrConnection.Connect())

if ($res.Successful -eq $true)
```

(continues on next page)

(continued from previous page)

```
{

    $ConfigMgrComputersObject = New-Object SIM.ConfigMgr.Computers.Computer(
↪$ConfigMgrConnection)
    $objParametersCollectionMembership = New-Object SIM.ConfigMgr.
↪ParametersCollectionMembership;

    $objParametersCollectionMembership.CollectionMembershipType = [SIM.ConfigMgr.
↪ParametersCollectionMembership+CollectionMembershipTypes]::RuleDirectComputer;
    $objParametersCollectionMembership.CollectionName = "Test Collection"
    $objParametersCollectionMembership.ResourceName = "Testcomputer1"

    $res.ChildAdd($ConfigMgrComputersObject.CollectionMembershipRequestRefresh(
↪$objParametersCollectionMembership,10,20000,20000));

}

$res.Dump()
```

CollectionMembershipRemove

Currently the following methods are supported by this action:

- RuleDirectUser
- RuleDirectComputer

Example for RuleDirectComputer:

```
$Assembly = [Reflection.Assembly]::LoadFile("$PSScriptRoot\Base.dll")
$Assembly = [Reflection.Assembly]::LoadFile("$PSScriptRoot\ConfigMgr.dll")

$ConfigMgrConnectionSettings = New-Object SIM.ConfigMgr.ConnectionSettings
$ConfigMgrConnectionSettings.WMIHostName = 'localhost'
$ConfigMgrConnectionSettings.SiteCode = 'SIM'
$ConfigMgrConnectionSettings.SQLHostname = $ConfigMgrConnectionSettings.WMIHostName;

$ConfigMgrConnection = New-Object SIM.ConfigMgr.Connection(
↪$ConfigMgrConnectionSettings)

$res = New-Object Base.Result("Starting ConfigMgr script...")

$res.ChildAdd($ConfigMgrConnection.Connect())

if ($res.Successful -eq $true)

{

    $ConfigMgrComputersObject = New-Object SIM.ConfigMgr.Computers.Computer(
↪$ConfigMgrConnection)
    $objParametersCollectionMembership = New-Object SIM.ConfigMgr.
↪ParametersCollectionMembership;

    $objParametersCollectionMembership.CollectionMembershipType = [SIM.ConfigMgr.
↪ParametersCollectionMembership+CollectionMembershipTypes]::RuleDirectComputer;
```

(continues on next page)

(continued from previous page)

```
    $objParametersCollectionMembership.CollectionId = "SIM00017"
    $objParametersCollectionMembership.ResourceId = "16777228"

    $res.ChildAdd($ConfigMgrComputersObject.CollectionMembershipRemove(
↪$objParametersCollectionMembership));

}

$res.Dump()
```

ComputerExists

Checks wether a computer exists (by Name, MAC or SMBIOSGUID). Found computers can be deleted directly via secondary function parameter `DeleteSystem`.

Example:

```
$Assembly = [Reflection.Assembly]::LoadFile("$PSScriptRoot\Base.dll")
$Assembly = [Reflection.Assembly]::LoadFile("$PSScriptRoot\ConfigMgr.dll")

$ConfigMgrConnectionSettings = New-Object SIM.ConfigMgr.ConnectionSettings
$ConfigMgrConnectionSettings.WMIHostName = 'localhost'
$ConfigMgrConnectionSettings.SiteCode = 'SIM'
$ConfigMgrConnectionSettings.SQLHostname = $ConfigMgrConnectionSettings.WMIHostName;

$ConfigMgrConnection = New-Object SIM.ConfigMgr.Connection(
↪$ConfigMgrConnectionSettings)

$res = New-Object Base.Result("Starting ConfigMgr script...")

$res.ChildAdd($ConfigMgrConnection.Connect())

if ($res.Successful -eq $true)

{

    $ConfigMgrComputersObject = New-Object SIM.ConfigMgr.Computers.Computer(
↪$ConfigMgrConnection)

    $ConfigMgrComputersSettings = New-Object SIM.ConfigMgr.Computers.
↪ComputerParameters
    $ConfigMgrComputersSettings.ComputerName = "SIMTestPC"
    $ConfigMgrComputersSettings.MACAddress = "00-00-00-00-00-12"

    $DeleteSystem = $true

    [Base.Result]$existsResult = $ConfigMgrComputersObject.ComputerExists(
↪$ConfigMgrComputersSettings,$DeleteSystem)
    [string]$ResourceId = $existsResult.ReturnObj
    $res.ChildAdd($existsResult)

    if ($existsResult.ExitCode.Code -eq 'ElementFound')
    {

        Write-Host "Computer was found and has following ResourceId: $ResourceId"
```

(continues on next page)

(continued from previous page)

```
    }

}

$res.Dump()
```

Create

Creates a computer.

Example:

```
$Assembly = [Reflection.Assembly]::LoadFile("$PSScriptRoot\Base.dll")
$Assembly = [Reflection.Assembly]::LoadFile("$PSScriptRoot\ConfigMgr.dll")

$ConfigMgrConnectionSettings = New-Object SIM.ConfigMgr.ConnectionSettings
$ConfigMgrConnectionSettings.WMIHostName = 'localhost'
$ConfigMgrConnectionSettings.SiteCode = 'SIM'
$ConfigMgrConnectionSettings.SQLHostname = $ConfigMgrConnectionSettings.WMIHostName;

$ConfigMgrConnection = New-Object SIM.ConfigMgr.Connection(
↪$ConfigMgrConnectionSettings)

$res = New-Object Base.Result("Starting ConfigMgr script...")

$res.ChildAdd($ConfigMgrConnection.Connect())

if ($res.Successful -eq $true)

{

    $ConfigMgrComputersObject = New-Object SIM.ConfigMgr.Computers.Computer(
↪$ConfigMgrConnection)

    $ConfigMgrComputersSettings = New-Object SIM.ConfigMgr.Computers.
↪ComputerParameters
    $ConfigMgrComputersSettings.ComputerName = "SIMTestPC"
    $ConfigMgrComputersSettings.MACAddress = "00-00-00-00-00-11"

    $resCreate = $ConfigMgrComputersObject.Create($ConfigMgrComputersSettings)
    [string]$ResourceId = $resCreate.ReturnObj
    $res.ChildAdd($resCreate)

    if ($res.Successful -eq $true -And $ResourceId -ne $null)
        {
        "Computer was created with ResourceId: $ResourceId"
    }

}

$res.Dump()
```

Delete

Deletes a computer. Can only be deleted with ResourceId. If deleting via Computername, MAC or SMBIOSGUID is needed, use `ComputerExists` function.

Example:

```
$Assembly = [Reflection.Assembly]::LoadFile("$PSScriptRoot\Base.dll")
$Assembly = [Reflection.Assembly]::LoadFile("$PSScriptRoot\ConfigMgr.dll")

$ConfigMgrConnectionSettings = New-Object SIM.ConfigMgr.ConnectionSettings
$ConfigMgrConnectionSettings.WMIHostName = 'localhost'
$ConfigMgrConnectionSettings.SiteCode = 'SIM'
$ConfigMgrConnectionSettings.SQLHostname = $ConfigMgrConnectionSettings.WMIHostName;

$ConfigMgrConnection = New-Object SIM.ConfigMgr.Connection(
↪$ConfigMgrConnectionSettings)

$res = New-Object Base.Result("Starting ConfigMgr script...")

$res.ChildAdd($ConfigMgrConnection.Connect())

if ($res.Successful -eq $true)

{

    $ConfigMgrComputersObject = New-Object SIM.ConfigMgr.Computers.Computer(
↪$ConfigMgrConnection)

    $ConfigMgrComputersSettings = New-Object SIM.ConfigMgr.Computers.
↪ComputerParameters
    $ConfigMgrComputersSettings.ResourceId = "16777225"

    $res.ChildAdd($ConfigMgrComputersObject.Delete($ConfigMgrComputersSettings))

}

$res.Dump()
```

DeleteVariables

Deletes the variables on the corresponding system.

Example:

```
$Assembly = [Reflection.Assembly]::LoadFile("$PSScriptRoot\Base.dll")
$Assembly = [Reflection.Assembly]::LoadFile("$PSScriptRoot\ConfigMgr.dll")

$ConfigMgrConnectionSettings = New-Object SIM.ConfigMgr.ConnectionSettings
$ConfigMgrConnectionSettings.WMIHostName = 'localhost'
$ConfigMgrConnectionSettings.SiteCode = 'SIM'
$ConfigMgrConnectionSettings.SQLHostname = $ConfigMgrConnectionSettings.WMIHostName;

$ConfigMgrConnection = New-Object SIM.ConfigMgr.Connection(
↪$ConfigMgrConnectionSettings)
```

(continues on next page)

(continued from previous page)

```
$res = New-Object Base.Result("Starting ConfigMgr script...")

$res.ChildAdd($ConfigMgrConnection.Connect())

if ($res.Successful -eq $true)

{

    $ConfigMgrComputersObject = New-Object SIM.ConfigMgr.Computers.Computer(
↪$ConfigMgrConnection)

    $ConfigMgrComputersSettings = New-Object SIM.ConfigMgr.Computers.
↪ComputerParameters
    $ConfigMgrComputersSettings.ResourceId = "16777221"

    $res.ChildAdd($ConfigMgrComputersObject.DeleteVariables(
↪$ConfigMgrComputersSettings))

}

$res.Dump()
```

PrimaryUserAdd

Add a user to a computer as a PrimaryUser reference.

Example:

```
$Assembly = [Reflection.Assembly]::LoadFile("$PSScriptRoot\Base.dll")
$Assembly = [Reflection.Assembly]::LoadFile("$PSScriptRoot\ConfigMgr.dll")

$ConfigMgrConnectionSettings = New-Object SIM.ConfigMgr.ConnectionSettings
$ConfigMgrConnectionSettings.WMIHostName = 'localhost'
$ConfigMgrConnectionSettings.SiteCode = 'SIM'
$ConfigMgrConnectionSettings.SQLHostname = $ConfigMgrConnectionSettings.WMIHostName;

$ConfigMgrConnection = New-Object SIM.ConfigMgr.Connection(
↪$ConfigMgrConnectionSettings)

$res = New-Object Base.Result("Starting ConfigMgr script...")

$res.ChildAdd($ConfigMgrConnection.Connect())

if ($res.Successful -eq $true)

{

    $ConfigMgrComputersObject = New-Object SIM.ConfigMgr.Computers.Computer(
↪$ConfigMgrConnection)

    $ConfigMgrComputersSettings = New-Object SIM.ConfigMgr.Computers.
↪ComputerParameters

    #Define Target Computer by ResourceId
    $ConfigMgrComputersSettings.ResourceId = '16777222'
```

(continues on next page)

(continued from previous page)

```
    $UserParameters = New-Object SIM.ConfigMgr.Users.UserParameters(
↪$ConfigMgrConnection)
    $UserParameters.ResourceId = "2063597568"

    $ParameterPrimaryUser = New-Object SIM.ConfigMgr.ParameterPrimaryUser(
↪$ConfigMgrComputersSettings, $UserParameters)

    $res.ChildAdd($ConfigMgrComputersObject.PrimaryUserAdd($ParameterPrimaryUser))

}

$res.Dump()
```

WorkflowCreate

This worklow combines the following actions:

1. Check if computer exists (Delete if `ComputerParameters.ComputerOverwriteExistingObject` is true)
2. Create computer
3. Add variables to computer object
4. Add collection memberships to computer

Example (complete with connection):

```
$Assembly = [Reflection.Assembly]::LoadFile("$PSScriptRoot\Base.dll")
$Assembly = [Reflection.Assembly]::LoadFile("$PSScriptRoot\ConfigMgr.dll")

$ConfigMgrConnectionSettings = New-Object SIM.ConfigMgr.ConnectionSettings
$ConfigMgrConnectionSettings.WMIHostName = 'MyConfigMgrHost'
$ConfigMgrConnectionSettings.SiteCode = 'P01'
$ConfigMgrConnectionSettings.SQLHostname = $ConfigMgrConnectionSettings.WMIHostName

$ConfigMgrConnection = New-Object SIM.ConfigMgr.Connection(
↪$ConfigMgrConnectionSettings)

$ConfigMgrComputersSettings = New-Object SIM.ConfigMgr.Computers.ComputerParameters
$ConfigMgrComputersSettings.ComputerName = "TestComputer"
$ConfigMgrComputersSettings.MACAddress = "E4-F8-9C-5D-DE-39"
$ConfigMgrComputersSettings.ComputerOverwriteExistingObject = $true

$val = New-Object SIM.ConfigMgr.Parameter("Var1", "Val1")

$ConfigMgrComputersSettings.Variables.Add($val)

$ParametersCollectionMembership = New-Object SIM.ConfigMgr.
↪ParametersCollectionMembership
$ParametersCollectionMembership.IsComputerMembership = $true
$ParametersCollectionMembership.CollectionName = "Windows Server 2012 Deploy"
$ParametersCollectionMembership.CreateCollectionIfNotExist = $false

$ConfigMgrComputersSettings.Collections.Add($ParametersCollectionMembership);
```

(continues on next page)

(continued from previous page)

```
$ConfigMgrComputersWorkflowCreate = New-Object SIM.ConfigMgr.Computers.Computer(
↪$ConfigMgrConnection)

$ResCreate = $ConfigMgrComputersWorkflowCreate.WorkflowCreate(
↪$ConfigMgrComputersSettings)

$ResCreate.Dump()
```

Expected output:

```
Loaded assembly: ConfigMgr, Version=6.1.0.6, Culture=neutral, PublicKeyToken=null
+ Ok | Starting Workflow for creating computer... (ComputerOverwriteExistingObject:␣
↪True) | 11/18/2016 9:19:10 AM
++ Ok | Connecting to ConfigMgr Site with connection settings: Hostname: "192.168.42.
↪193" SiteCode: "P01" UserName: "administrator" UserDomainName: "SCCM12". | 11/18/
↪2016 9:19:10 AM
+++ Ok | Connected to scope (Path: \\192.168.42.193\root\SMS\site_P01). | 11/18/2016␣
↪9:19:11 AM
+++ Ok | Connecting to ConfigMgr SQL database "Data Source= 192.168.42.193; Initial␣
↪Catalog= CM_P01;" | 11/18/2016 9:19:11 AM
++ Ok | Validating Computers.Settings... | 11/18/2016 9:19:11 AM
+++ Ok | Valiation ok! | 11/18/2016 9:19:11 AM
++ ElementFound | Checking if computer already exists (1: SMBIOSGUID: "" / 2.␣
↪MacAddress: "E4:F8:9C:5D:DE:39" / 3. NetBiosname: TestComputer4")... | 11/18/2016␣
↪9:19:11 AM
+++ ElementFound | Executing "SELECT System_MAC_Addres_ARR.ItemKey FROM System_MAC_
↪Addres_ARR JOIN v_R_System ON v_R_System.ResourceId = System_MAC_Addres_ARR.ItemKey␣
↪WHERE System_MAC_Addres_ARR.[MAC_Addresses0] = 'E4:F8:9C:5D:DE:39' ORDER BY v_R_
↪System.Creation_Date0 DESC" | 11/18/2016 9:19:11 AM
+++ Ok | Computer was found (ResourceId: "16777736"). | 11/18/2016 9:19:11 AM
++ Ok | Deleting computer with ResourceID "16777736"... | 11/18/2016 9:19:11 AM
++ Ok | Creating computer... (NetbiosName = TestComputer4, MACAddress =␣
↪E4:F8:9C:5D:DE:39, SMBIOSGUID = ) | 11/18/2016 9:19:11 AM
+++ Ok | Computer was created with ResourceId: '16777737' | 11/18/2016 9:19:11 AM
++ Ok | Adding variables to resouce with ResourceID "16777737"... (Variable count: 1)␣
↪| 11/18/2016 9:19:11 AM
+++ Ok | Adding "Var1" (Value: "Val1222")... | 11/18/2016 9:19:11 AM
+++ Ok | Done! | 11/18/2016 9:19:12 AM
++ Ok | Starting AddCollectionMembership...(CollectionId: "P01000A2", CollectionName:
↪"Windows Server 2012 Deploy", ResourceName: "TestComputer4", ResourceId: "16777737",
↪ CreateCollectionIfNotExist: "False", IsComputerMembership: "True") | 11/18/2016␣
↪9:19:12 AM
+++ Ok | Validating CollectionMembershipParameters... | 11/18/2016 9:19:12 AM
+++ Ok | Done! | 11/18/2016 9:19:12 AM
```

Applications

Everthing concering application management is stored in the `Application` namespace.

CreateApplication

Creates an Application with a XML definition provided.

Example (complete with connection):

Security

Everthing concering computer management is stored in the `Computers` namespace.

SecurityScopeAdd

Adds a existing security scope (by name) to a ConfigMgr object.

Example (complete with connection):

```
$Assembly = [Reflection.Assembly]::LoadFile("$PSScriptRoot\Base.dll")
$Assembly = [Reflection.Assembly]::LoadFile("$PSScriptRoot\Database.dll")
$Assembly = [Reflection.Assembly]::LoadFile("$PSScriptRoot\ConfigMgr.dll")

$ConfigMgrConnectionSettings = New-Object SIM.ConfigMgr.ConnectionSettings
$ConfigMgrConnectionSettings.WMIHostName = 'localhost'
$ConfigMgrConnectionSettings.SiteCode = 'P01'
$ConfigMgrConnectionSettings.SQLHostname = $ConfigMgrConnectionSettings.WMIHostName;

$ConfigMgrConnection = New-Object SIM.ConfigMgr.Connection(
↪$ConfigMgrConnectionSettings)

$res = New-Object Base.Result("Starting ConfigMgr script...")

$res.ChildAdd($ConfigMgrConnection.Connect())

if ($res.Successful -eq $true)
{

    $SecurityScopeManagement = New-Object SIM.ConfigMgr.Security.
↪SecurityScopeManagement($ConfigMgrConnection);

    $securityScopeList = New-Object SIM.ConfigMgr.Security.SecurityScopeList
    $securityScopeList.SecurityScopes.Add( (New-Object SIM.ConfigMgr.Security.
↪SecurityScope ("","GERMANY Scope","") ));

    $res.ChildAdd( ($SecurityScopeManagement.SecurityScopeAdd($securityScopeList,
↪"P0100052", [SIM.ConfigMgr.ObjectTypeID]::SMS_Package) ));

}

$res.Dump()
```

Possible values for ObjectTypeId:

```
    public enum ObjectTypeID
    {
        SMS_Package = 2,
        SMS_OperatingSystemInstallPackage = 14,
        SMS_ImagePackage = 18,
        SMS_BootImagePackage = 19,
        SMS_DriverPackage = 23,
        SMS_SoftwareUpdatesPackage = 24,
        SMS_Application = 31
    }
```

SecurityScopeRemove

Removes a security scope (by name) from a ConfigMgr object.

Example (complete with connection):

```
$Assembly = [Reflection.Assembly]::LoadFile("$PSScriptRoot\Base.dll")
$Assembly = [Reflection.Assembly]::LoadFile("$PSScriptRoot\Database.dll")
$Assembly = [Reflection.Assembly]::LoadFile("$PSScriptRoot\ConfigMgr.dll")

$ConfigMgrConnectionSettings = New-Object SIM.ConfigMgr.ConnectionSettings
$ConfigMgrConnectionSettings.WMIHostName = 'localhost'
$ConfigMgrConnectionSettings.SiteCode = 'P01'
$ConfigMgrConnectionSettings.SQLHostname = $ConfigMgrConnectionSettings.WMIHostName;

$ConfigMgrConnection = New-Object SIM.ConfigMgr.Connection(
↪$ConfigMgrConnectionSettings)

$res = New-Object Base.Result("Starting ConfigMgr script...")

$res.ChildAdd($ConfigMgrConnection.Connect())

if ($res.Successful -eq $true)
{

    $SecurityScopeManagement = New-Object SIM.ConfigMgr.Security.
↪SecurityScopeManagement($ConfigMgrConnection);

    $securityScope = New-Object SIM.ConfigMgr.Security.SecurityScope ("","GERMANY␣
↪Scope","")

    $res.ChildAdd( ($SecurityScopeManagement.SecurityScopeRemove($securityScope,
↪"P0100052", [SIM.ConfigMgr.ObjectTypeID]::SMS_Package) ));

}

$res.Dump()
```

Manual for library “Base”

In this article:

Base.Result

The Base.Result class is used to nest every action result of your scripts to get strong and reliable error handling.

Example:

```
 $Assembly = [Reflection.Assembly]::LoadFile("$PSScriptRoot\Base.dll")

$res = New-Object Base.Result("My first script with error handling.")

#------ First part will work ------#

Try
{
    $DevisionByTwo = 1 / 2
    "Result ByTwo: $DevisionByTwo"

    $resTemp = New-Object Base.Result("Result of DevisionByTwo: $DevisionByTwo")
    $res.ChildAdd($resTemp)

}
Catch
{

    $ErrorMessage = $_.Exception.Message
    $ExitCodeClass = New-Object Base.ResultSuperClass+ExitCodeClass([Base.
↪ResultSuperClass+ExitCodeCategory]::Error,[Base.
↪ResultSuperClass+ExitCodeType]::SystemException)
    $resError = New-Object Base.Result("Exception thrown: $ErrorMessage",
↪$ExitCodeClass)

    $res.ChildAdd($resError)

}


#------ Check if everything went good so far ------#
if ($res.Successful -eq $true)

{

    #------ Second part will not work ------#

    Try
    {
        $DevisionByZero = 1 / 0
        "Result ByZero: $DevisionByZero"

        $resTemp = New-Object Base.Result("Result of DevisionByZero: $DevisionByZero")
        $res.ChildAdd($resTemp)

    }
    Catch
    {

        $ErrorMessage = $_.Exception.Message
        $res.ChildAdd((New-Object Base.Result("Exception thrown: $ErrorMessage", (New-
↪Object Base.ResultSuperClass+ExitCodeClass([Base.
↪ResultSuperClass+ExitCodeCategory]::Error,[Base.
↪ResultSuperClass+ExitCodeType]::SystemException)))))

    }
}
```

(continues on next page)

(continued from previous page)

```
$res.Dump()
```

Expected output:

```
Result ByTwo: 0.5
+ Ok | My first script with error handling. | 21.04.2017 08:19:28
++ Ok | Result of DevisionByTwo: 0.5 | 21.04.2017 08:19:28
++ SystemException | Exception thrown: Attempted to divide by zero. | 21.04.2017␣
↪08:19:28
```

Powershell functions returning Base.Result

To make sure, that self created powershell functions return a Base.Result object, please stick to the following structure:

```
function MyResultFunction()
{

    #EVEY CONSOLE OUTPUT IS PIPED TO $NULL
    $Null = @(

        $res = New-Object Base.Result("Starting module X...")

        #HERE YOUR FUNCTION CONTENT


    )

    #return , $res: Comma is important character. Do not delete.
    return , $res

}


[Base.Result]$res = MyResultFunction

$res.Dump()
```

Manual for library “Database”

In this article:

SIM.Tools.Database

The SIM.Tools.Database class is used easily handle database actions to MSSQL servers.

Example with ConfigMgr:

```
$Assembly = [Reflection.Assembly]::LoadFile("$PSScriptRoot\Base.dll")
$Assembly = [Reflection.Assembly]::LoadFile("$PSScriptRoot\System.Data.SqlClient.dll")
$Assembly = [Reflection.Assembly]::LoadFile("$PSScriptRoot\Database.dll")
$Assembly = [Reflection.Assembly]::LoadFile("$PSScriptRoot\ConfigMgr.dll")

$ConfigMgrConnectionSettings = New-Object SIM.ConfigMgr.ConnectionSettings
$ConfigMgrConnectionSettings.WMIHostName = 'localhost'
$ConfigMgrConnectionSettings.SiteCode = 'SIM'
$ConfigMgrConnectionSettings.SQLHostname = $ConfigMgrConnectionSettings.WMIHostName;

$ConfigMgrConnection = New-Object SIM.ConfigMgr.Connection(
↪$ConfigMgrConnectionSettings)

$res = New-Object Base.Result("Starting ConfigMgr script...")

$res.ChildAdd($ConfigMgrConnection.Connect())

if ($res.Successful -eq $true)

{

    # ************************************
    # Create new instance of Database with
    # (A) an existing SQLConnection object
    # (B) with a path to the SIMV61 Config file: $SIM_DB = New-Object SIM.Tools.
↪Database("C:\SilverMonkey\v61\Config.xml")
    # (C) with a connection string: $SIM_DB = New-Object SIM.Tools.Database("data␣
↪source = server1,1433; initial catalog = DB1; integrated security = True;␣
↪MultipleActiveResultSets=True;App=EntityFramework", $true)
    # ************************************
    $SIM_DB = New-Object SIM.Tools.Database($ConfigMgrConnection.SQLConnection)


    # ************************************
    # EXAMPLE with single row return:
    # ************************************
    $row = $SIM_DB.SQLQueryFirstRow("SELECT * FROM [v_CollectionRuleDirect] WHERE␣
↪[ResourceType] = 5")
    $row['RuleName']

    # ************************************
    # EXAMPLE with multiple rows (in a table) return:
    # ************************************
    $table = $SIM_DB.SQLQueryAll("SELECT * FROM [v_CollectionRuleDirect] WHERE␣
↪[ResourceType] = 5")

    if ($table.Rows.Count -gt 0)
    {
        Foreach ($Row in $table)
        {
           $Row['RuleName']
        }
    }

    # ************************************
    # EXAMPLE to fire a command:
    # ************************************
```

(continues on next page)

(continued from previous page)

```
    $intReturn = $SIM_DB.SQLCommand("use testdb;  create table testtable(bla␣
↪varchar(10));")

    "Affected items: $intReturn"

}

$res.Dump()
```

Manual for library "Tools"

In this article:

SIM.Tools CheckCondition

You can evaluate a string describing a condition into a boolean using an `SIM.Tools.ICondition`. `SIM.Tools.Condition` class is its main implementation. It takes a string with the condition as an input parameter in the constructor. You can check the result of the condition using the property `Result`.

Check the next example:

```
Add-Type -Path ("$PSScriptRoot\bin\debug\CheckCondition.dll")

#Here are 2 examples of use of the class SIM.Tools.Condition, with different inputs.

#Given some inputs (be sure to use simple qutations for string literals: 'text'):
$numInput = "3 * (2 + 1) = 9"
$stringInput = "'M' ! 'R'"

#We create the Condition:
$numCondition = New-Object SIM.Tools.Condition($numInput)
$stringCondition = New-Object SIM.Tools.Condition($stringInput)

#And so we can access the condition result:
$numCondition.get_Result()
$stringCondition.get_Result()

#We could also check the original condition from input:
#$numCondition.Condition
#$stringCondition.Condition
```

This are the accepted operators:

1. < -> less than
2. > -> greater than
3. = -> equals
4. ! -> not equals

5. `AND` -> 'and' logical operator
6. `OR` -> 'or' logical operator
7. `NOT` -> 'not' logical operator
8. `()` -> Parentheses to modify operators preference

IMPORTANT: String literals must be enclosed by simple quotation marks: 'literal'

SIM.Tools ResolveName

You can replace "keys" in a string for its corresponding "values" using a `SIM.Tools.IResolvedText` and a `SIM.Tools.IResolver`. `SIM.Tools.ResolvedText` class is the main implementation for `IResolvedText`. It takes the string with the keys and a `IResolver` as input parameters in the constructor. You can check the result using the property `Text`. For `IResolver` there is an implementation for v6 and v61 taking 2 parameter:

1. An open connection with a SQL DB.
2. An SQL Query containing the columns that will be used as keys. The column name must match the key name. The column value will be the value we will use.

You can check the result of the replacement using the property `Text`.

Check the next example:

```
Add-Type -Path ("$PSScriptRoot\bin\debug\ResolveName.dll")
Add-Type -Path ("$PSScriptRoot\..\SIMv61Database\bin\Debug\SIMv61Database.dll")

#Given a sample input
$input = '"<br>The software package: "{PackagingPackageName}" has been hand over to␣
↪packaging factory.
<br>Manufacturer: <b>{RequestManufacturerName}</b>
<br>Product:<b>{RequestProductName}</b>
<br>Version: <b>{RequestProductVersion}</b>
<br>Architecture: <b>{HC_Architecture}</b>"'

#We obtain a connection with the corresponding table:
$connV61 = (New-Object SIMv61Database.SIMv61Database).Database.Connection
$connV61.Open()
$connV61.ChangeDatabase("SIM_HC_R003")

#We create the Resolved Text, using either connection.
#We can use different Resolvers. Here we use v6 resolver:
$inputResolvedText = New-Object SIM.Tools.ResolvedText(
    $input,
    (New-Object SIM.Tools.V6SqlDbResolver(
        $connV61,
        "SELECT * FROM PackagingJob WHERE Id=101"
    ))
)

#And so we can access its resolved text (use of property accessor method to expose␣
↪exceptions in PS):
$inputResolvedText.get_Text()

#We could also check the original text before resolving:
#$inputResolvedText.OriginText
```

Error handling:

1. If braces do not match on the input text, a `FormatException` will be thrown.
2. If a key value is not found among the columns returned by the query, or no entries are returned, an `ApplicationException` will be thrown. You can opt out of this error by adding a 3rd parameter to the `ResolvedText` constructor with the value `false`, like this:

```
$resolvedText = New-Object SIM.Tools.ResolvedText($input, $resolver, false)
```

1.4 Changelog

Version	TicketId	Product	Description
6.1.0	None	Initial Version	

1.5 Supported configurations

1.5.1 Supported Microsoft SQL Server Versions

Product	Version	Supported
SQL Server 2012	11.0	Yes
SQL Server 2014	12.0	Yes
SQL Server 2016	13.0	Yes

1.5.2 Supported Microsoft Windows Server Versions

Product	Version	Supported
Windows Server 2012	NT 6.2	Yes
Windows Server 2012 R2	NT 6.3	Yes
Windows Server 2016	NT 10.0	Yes

1.5.3 Supported .Net Framework Versions

- Hence the code was written in .Net Core 1.0 only this version is supported

1.6 Support

If you have further questions regarding out products or the documentation contact us:

- Tel. : +49 40 - 226 383 160
- E-Mail : Support@SilverMonkey.net

If you need general Information about our Products visit: http://www.SilverMonkey.net