$curDir = Split-Path -parent $MyInvocation.MyCommand.Definition

$dotSource = $curDir + "\./FIMFunctionLibrary.ps1"

. $dotSource

if($args.count -ne 1) {throw "You need to specify your attribute values as parameter"}

$attributes = ($args[0]).split("|")

if(0 -ne [String]::Compare(($attributes[0]).split(":")[0],"displayname", $true)){

throw "You need to specify a display name"

}

$objectName = ($attributes[0]).split(":")[1]

$exportObject = export-fimconfig -uri $URI `

                                  –onlyBaseResources `

                                  -customconfig "/Person[DisplayName='$objectName']"

if($exportObject) {throw "L:User $objectName already exists"}

#Prepare the object

$object = Prepare-FimObject "Person"

#Populate attributes

foreach($attribute in $attributes)

 {

    $attrData = $attribute.split(":")

    Set-FimAttribute -object $object -attributeName $($attrData[0]) -attributeValue $($attrData[1])

 }

#Create the object in FIM

Commit-FimObject $object

**How to Use PowerShell as an FIM Function Library**

This page provides a set of PowerShell Functions which can be used as a shared library across PowerShell scripts or from the PowerShell prompt.

**Table of Contents**

* [Introduction](https://social.technet.microsoft.com/wiki/contents/articles/3985.how-to-use-powershell-as-an-fim-function-library.aspx#Introduction)
* [FIMFunctionLibrary](https://social.technet.microsoft.com/wiki/contents/articles/3985.how-to-use-powershell-as-an-fim-function-library.aspx#FIMFunctionLibrary)
* [The Functions Explained](https://social.technet.microsoft.com/wiki/contents/articles/3985.how-to-use-powershell-as-an-fim-function-library.aspx#The_Functions_Explained)
  + [Convert-FimExportToPSOjbect](https://social.technet.microsoft.com/wiki/contents/articles/3985.how-to-use-powershell-as-an-fim-function-library.aspx#Convert-FimExportToPSOjbect)
  + [Delete-FimObject](https://social.technet.microsoft.com/wiki/contents/articles/3985.how-to-use-powershell-as-an-fim-function-library.aspx#Delete-FimObject)
  + [Prepare-FimObject > Set-FimAttribute > Commit-FimObject](https://social.technet.microsoft.com/wiki/contents/articles/3985.how-to-use-powershell-as-an-fim-function-library.aspx#Prepare-FimObject_gt_Set-FimAttribute_gt_Commit-FimObject)
  + [Get-FimObjectID](https://social.technet.microsoft.com/wiki/contents/articles/3985.how-to-use-powershell-as-an-fim-function-library.aspx#Get-FimObjectID)
* [Examples](https://social.technet.microsoft.com/wiki/contents/articles/3985.how-to-use-powershell-as-an-fim-function-library.aspx#Examples)
  + [Create a Criteria-Based Set](https://social.technet.microsoft.com/wiki/contents/articles/3985.how-to-use-powershell-as-an-fim-function-library.aspx#Create_a_Criteria-Based_Set)
  + [Create a User](https://social.technet.microsoft.com/wiki/contents/articles/3985.how-to-use-powershell-as-an-fim-function-library.aspx#Create_a_User)
  + [Create an Outbound Synchronization Rule](https://social.technet.microsoft.com/wiki/contents/articles/3985.how-to-use-powershell-as-an-fim-function-library.aspx#Create_an_Outbound_Synchronization_Rule)
  + [Update a User Property](https://social.technet.microsoft.com/wiki/contents/articles/3985.how-to-use-powershell-as-an-fim-function-library.aspx#Update_a_User_Property)
  + [Creating a User From The Prompt](https://social.technet.microsoft.com/wiki/contents/articles/3985.how-to-use-powershell-as-an-fim-function-library.aspx#Creating_a_User_From_The_Prompt)
* [To do](https://social.technet.microsoft.com/wiki/contents/articles/3985.how-to-use-powershell-as-an-fim-function-library.aspx#Todo)
* [See Also](https://social.technet.microsoft.com/wiki/contents/articles/3985.how-to-use-powershell-as-an-fim-function-library.aspx#SeeAlso)

**Introduction**

Here on the TechNet Wiki quit some "How to Use PowerShell to ..." articles have been written in the past. A lot of them allow you to interact with the FIM Portal. You can manage users, groups, sets, MPRs, Schema, … They all use one or more of the following *operations*:

* Create
* Modify
* Delete

All these operations require some specific PowerShell code in order to be performed successfully. I don’t think there are a lot of people who can actually remember what code to use for what operation. So probably you just take your last script, copy paste it, and then start modifying it until it meet your requirements. Often this gives cluttered/dirty scripts in my opinion.

Now what if we could provide a library with most of the common operations already build in. Those operations could be available just like regular PowerShell cmdlets. Just as easy to use as Export-FIMConfig or Write-Host!

**FIMFunctionLibrary**

For this purpose the FIMFunctionLibrary was created. In fact it’s just a collection of PowerShell Functions. If you would execute it as a PowerShell script, it will do absolutely nothing. But we can make these functions available in another script or in the prompt by “*Dot-Sourcing*” it. If I would have to describe Dot-sourcing, I would say it's like adding extra DLLs in a .net project by using references.

Here's a sample piece of code which can be used to dot-source the library when it's located in the same directory as your script. You can put this at the top of your scripts:

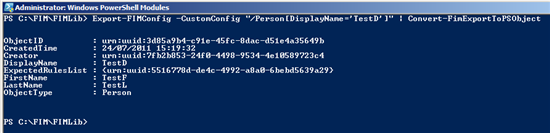
|  |  |
| --- | --- |
| 001  002  003 | $curDir = Split-Path -parent $MyInvocation.MyCommand.Definition  $dotSource = $curDir + "\./FIMFunctionLibrary.ps1"  . $dotSource |

On the other hand, if you just want to be able to use the Functions from the PowerShell command prompt like regular cmdlets, all you have to do is execute the following command:

[Administrator. Windows PowerSheU Modules 
PS c: NFIMĂFIMLib> 
. MFunct ionLibrary. PSI 
PS ](http://social.technet.microsoft.com/wiki/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-05/8686.01.DotSource.png)

**Mind the two dots and the space in between!** Now in the command prompt you can use “tab-completion” to use the Functions. Just like a regular cmdlet! Here’s an example:

Getting the properties of an object:

[](http://social.technet.microsoft.com/wiki/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-05/6886.02ConvertTo.png)

Deleting that object:

[C:\Users\west_\AppData\Local\Packages\Microsoft.Office.OneNote_8wekyb3d8bbwe\TempState\msohtmlclip\clip_image003.png](http://social.technet.microsoft.com/wiki/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-05/8360.03Delete.png)

Now how easy is that?! The FIMFunctionLibary.ps1 consists of the following functions:

* Interact with Export-FimConfig:
  + Convert-FimExportToPSObject
  + Delete-FimObject
* Interact with Import-FimConfig:
  + Prepare-FimObject
  + Set-FimAttribute
  + Commit-FimObject
* Helper Function:
  + Get-FimObjectID

And here is the library: Copy-Paste and save it as FIMFunctionLibrary.ps1

|  |  |
| --- | --- |
|  |  |

**The Functions Explained**

**Convert-FimExportToPSOjbect**

This function was written by Craig Martin. He provided it on his blog and gives a good explanation of how to use it: [Convert a FIM ExportObject to a PowerShell PSObject](http://www.identitytrench.com/2011/07/convert-fim-exportobject-to-powershell.html)

**Delete-FimObject**

The **Delete-FimObject** is pretty straightforward. All you have to do is feed it some output objects of the Export-FIMConfig cmdlet. Think very well about your xpath filter. If it's wrong you end up deleting the wrong object. As a general best practice consider executing Export-FIMConfig ... | Convert-FimExportToPSObject first so you can get an overview of all objects in the scope of your xpath filter.

**Prepare-FimObject > Set-FimAttribute > Commit-FimObject**

These are typically all used together. In the past you had to start by creating an “ImportObject”. This is what **Prepare-FimObject** will do. You feed it an ObjectType and it gives you a base object which you can use to create a new object in the Portal. On the other hand if you also provide it an existing ObjectID, it can be used to perform updates to an existing object in the Portal.

The **Set-FimAttribute** allows to add a Replace operation for a given attribute to the prepared object. You can execute this one multiple times in a row in order to prepare multiple attributes.

And finally the object can be exported to the Portal by executing the **Commit-FimObject** Function.

**Get-FimObjectID**

The **Get-FimObjectID** function can be used to retrieve the ObjectID for an object which matches the xpath filter provided to the function. This can be used when creating or updating objects which use attributes that are in fact references to other objects.

**Examples**

These are typically all used together. In the past you had to start by creating an “ImportObject”. This is what will do. You feed it an ObjectType and it gives you a base object which you can use to create a new object in the Portal. On the other hand if you also provide it an existing ObjectID, it can be used to perform updates to an existing object in the Portal.The allows to add a Replace operation for a given attribute to the prepared object. You can execute this one multiple times in a row in order to prepare multiple attributes.And finally the object can be exported to the Portal by executing the Function.

**Create a Criteria-Based Set**

Here's the equivalent without the FIMFunctionLibrary: [How to Use PowerShell to Create a Criteria-Based Set](http://social.technet.microsoft.com/wiki/contents/articles/how-to-use-powershell-to-create-a-criteria-based-set.aspx)

|  |  |
| --- | --- |
| 001  002  003  004  005  006  007  008  009  010  011  012  013  014  015  016 | $curDir = Split-Path -parent $MyInvocation.MyCommand.Definition  $dotSource = $curDir + "\./FIMFunctionLibrary.ps1"  . $dotSource    #Prepare the object  $object = Prepare-FimObject "Set"    #Populate attributes  Set-FimAttribute -object $object -attributeName "DisplayName" -attributeValue "Test Set"    $xPathFilter = "/Person[EmployeeType = \.'home'/]"  $filter = $PREFILTER + $xPathFilter + $POSTFILTER  Set-FimAttribute -object $object -attributeName "Filter" -attributeValue $filter    #Create the object in FIM  Commit-FimObject $object |

**Create a User**

Here's the equivalent without the FIMFunctionLibrary: [How to Use PowerShell to Create a User in the FIM Portal](http://social.technet.microsoft.com/wiki/contents/articles/how-to-use-powershell-to-create-a-user-in-the-fim-portal.aspx)

|  |  |
| --- | --- |
| 001  002  003  004  005  006  007  008  009  010  011  012  013  014  015  016  017  018  019  020  021  022  023  024  025  026  027  028  029 |  |

**Create an Outbound Synchronization Rule**

|  |  |
| --- | --- |
| 001  002  003  004  005  006  007  008  009  010  011  012  013  014  015  016  017  018  019  020  021  022  023  024  025  026  027  028  029  030  031  032  033  034  035  036  037  038  039  040  041  042  043  044  045 | $curDir = Split-Path -parent $MyInvocation.MyCommand.Definition  $dotSource = $curDir + "\./FIMFunctionLibrary.ps1"  . $dotSource    #Create Outbound Synchronization Rule  $ObjectType = "SynchronizationRule"    #Required Variables  $DisplayName = "Test Sync Rule"  $FlowType = 1 #0 Inbound, 1 outbound, 2 Inbound/Outbound  #MV Resource Type  $ILMObjectType = "person"  #External System  $ConnectedSystem = "{09A22997-1E65-4745-9259-DE047EF3E524}"  #External System Resource Type  $ConnectedObjectType = "user"    #Optional Variables  #Dependency (other OSR)  #$Dependency = "urn:uuid:a91ecb11-d692-4bb1-ae9b-a2cd56956ce1"  #Relationship Criteria  $RelationshipCriteria = "<conditions><condition><ilmAttribute>accountName</ilmAttribute><csAttribute>sAMAccountName</csAttribute></condition></conditions>"  #Create Resource in FIM  $CreateILMObject = $false  #Create Resource in External System  $CreateConnectedSystemObject = $false  #Enable Deprovisioning  $DisconnectConnectedSystemObject = $false    #Attribute Flows  $PersistentFlow = "<export-flow><src><attr>accountName</attr></src><dest>sAMAccountName</dest><scoping></scoping></export-flow>"    #$ManagementAgentID = "urn:uuid:657745e6-8d15-4cfb-889e-1cca82d7d69d"    $object = Prepare-FimObject $ObjectType  Set-FimAttribute -Object $object -AttributeName "DisplayName" -AttributeValue $DisplayName  Set-FimAttribute -Object $object -AttributeName "FlowType" -AttributeValue $FlowType  Set-FimAttribute -Object $object -AttributeName "ILMObjectType" -AttributeValue $ILMObjectType  Set-FimAttribute -Object $object -AttributeName "ConnectedSystem" -AttributeValue $ConnectedSystem  Set-FimAttribute -Object $object -AttributeName "ConnectedObjectType" -AttributeValue $ConnectedObjectType  Set-FimAttribute -Object $object -AttributeName "RelationshipCriteria" -AttributeValue $RelationshipCriteria  Set-FimAttribute -Object $object -AttributeName "CreateILMObject" -AttributeValue $CreateILMObject  Set-FimAttribute -Object $object -AttributeName "CreateConnectedSystemObject" -AttributeValue $CreateConnectedSystemObject  Set-FimAttribute -Object $object -AttributeName "DisconnectConnectedSystemObject" -AttributeValue $DisconnectConnectedSystemObject  Commit-FimObject $object |

**Update a User Property**

|  |  |
| --- | --- |
| 001  002  003  004  005  006  007  008  009 | $curDir = Split-Path -parent $MyInvocation.MyCommand.Definition  $dotSource = $curDir + "\./FIMFunctionLibrary.ps1"  . $dotSource    ##Update Properties##  Get-FimObjectID "/Person[AccountName='vuylthom001']" | `  Prepare-FimObject "Person" | `  Set-FimAttribute -AttributeName "DisplayName" -AttributeValue "Updated T.V." | `  Commit-FimObject |

**Creating a User From The Prompt**

Now here's a lot of piping, but you can actually create a user from the commandprompt by just executing this command:

Prepare-FimObject "Person" | Set-FimAttribute -attributeName "DisplayName" -attributeValue "Thomas Vuylsteke Demo" |Set-FimAttribute -attributeName "AccountName" -attributeValue "vuylthom003" | Commit-FimObject

[C:\Users\west_\AppData\Local\Packages\Microsoft.Office.OneNote_8wekyb3d8bbwe\TempState\msohtmlclip\clip_image004.png](http://social.technet.microsoft.com/wiki/cfs-file.ashx/__key/communityserver-wikis-components-files/00-00-00-00-05/5355.04Shell.png)

**To do**

There are certainly other interesting functions to add:

* Add a value to a multivalued attribute function
* GetSingleResource function
* Logging function
* ...

*From <*[*https://social.technet.microsoft.com/wiki/contents/articles/3985.how-to-use-powershell-as-an-fim-function-library.aspx*](https://social.technet.microsoft.com/wiki/contents/articles/3985.how-to-use-powershell-as-an-fim-function-library.aspx)*>*

001

002

003

004

005

006

007

008

009

010

011

012

013

014

015

016

017

018

019

020

021

022

023

024

025

026

027

028

029

030

031

032

033

034

035

036

037

038

039

040

041

042

043

044

045

046

047

048

049

050

051

052

053

054

055

056

057

058

059

060

061

062

063

064

065

066

067

068

069

070

071

072

073

074

075

076

077

078

079

080

081

082

083

084

085

086

087

088

089

090

091

092

093

094

095

096

097

098

099

100

101

102

103

104

105

106

107

108

109

110

111

112

113

114

115

116

117

118

119

120

121

122

123

124

125

126

127

128

129

130

131

132

133

134

135

136

137

138

139

140

141

142

143

144

145

146

147

148

149

150

151

152

153

154

155

156

157

158

159

160

161

162

163

164

165

166

167

168

169

170

171

172

173

174

175

176

177

178

179

180

181

182

183

184

185

186

187

188

189

190

191

192

193

194

195

196

197

198

199

200

201

202

203

204

205

206

207

208

209

210

211

212

213

214

215

216

217

218

219

220

221

222

223

224

225

226

227

228

229

230

231

232

233

234

235

236

237

238

239

240

241

242

243

244

245

246

247

248

249

250

251

252

253

254

255

256

257

258

259

260

261

262

263

264

265

266

267

268

269

270

271

272

273

274

275

276

277

278

279

280

281

282

283

284

285

286

287

288

289

290

291

292

293

294

295

296

297

298

299

300

301

302

303

304

305

306

307

308

309

310

311

312

313

314

315

316

317

318

319

320

321

322

323

324

###########################################################################

#

# Library Version 0.5

#

#

#

# Library usage: The Functions can be used by "dot-sourcing" this .ps1 file.

# Scripts should be executed on a server with the FIM Service installed.

#

# From within Scripts, requires the FIMFunctionLibrary.ps1 to be in the same directory as the script

#

# $curDir = Split-Path -parent $MyInvocation.MyCommand.Definition

# $dotSource = $curDir + "\./FIMFunctionLibrary.ps1"

# . $dotSource

#

# From within a PowerShell Prompt

#

# . ./FIMFunctionLibrary.ps1

# or when referencing a specific path:

# . d:\directory\./FIMFunctionLibrary.ps1

#

#

#----------------------------------------------------------------------------------------------------------

set-variable -name URI -value "http://localhost:5725/resourcemanagementservice' " -option constant

# Can be used to create filter criteria for sets (prepend)

set-variable -name PREFILTER -value "<Filter xmlns:xsi=""http://www.w3.org/2001/XMLSchema-instance"" xmlns:xsd=""http://www.w3.org/2001/XMLSchema"" Dialect=""http://schemas.microsoft.com/2006/11/XPathFilterDialect"" xmlns=""http://schemas.xmlsoap.org/ws/2004/09/enumeration"">" -option constant

# Can be used to create filter criteria for sets (append)

set-variable -name POSTFILTER -value "</Filter>"  -option constant

#----------------------------------------------------------------------------------------------------------

# Convert-FimExportToPSObject

# Author: Craig Martin

# Source: http://www.identitytrench.com/2011/07/convert-fim-exportobject-to-powershell.html

#

# Parameters:

# ExportObject: an object coming from the Export-FimConfig cmdlet

#

# Outcome:

# An PSobject which contains all properties from the inputobject available as . formation

# e.g. $psObject.DisplayName

#

Function Convert-FimExportToPSObject

{

    Param

    (

        [parameter(Mandatory=$true, ValueFromPipeline = $true)]

        [Microsoft.ResourceManagement.Automation.ObjectModel.ExportObject]

        $ExportObject

    )

    Process

    {

        $psObject = New-Object PSObject

        $ExportObject.ResourceManagementObject.ResourceManagementAttributes | ForEach-Object{

            if ($\_.Value -ne $null)

            {

                $value = $\_.Value

            }

            elseif($\_.Values -ne $null)

            {

                $value = $\_.Values

            }

            else

            {

                $value = $null

            }

            $psObject | Add-Member -MemberType NoteProperty -Name $\_.AttributeName -Value $value

        }

        Write-Output $psObject

    }

}

# Delete-FimObject

# Author: Community

# Source: TechNet Wiki

#

# Parameters:

# Object: an object coming from the Export-FimConfig cmdlet

#

# Outcome:

# The Object provided in the command is deleted from the Portal

#

Function Delete-FimObject

{

    Param

    (

        [parameter(Mandatory=$true, ValueFromPipeline = $true)]

[Microsoft.ResourceManagement.Automation.ObjectModel.ExportObject]

        $Object

    )

    Process

    {

if(@(get-pssnapin | where-object {$\_.Name -eq "FIMAutomation"} ).count -eq 0) {add-pssnapin FIMAutomation}

clear-host

$psObject = $Object | Convert-FimExportToPSObject

        $objectId = (($psObject.ObjectID).split(":"))[2]

$objectType = $psObject.ObjectType

$DeleteObject = New-Object Microsoft.ResourceManagement.Automation.ObjectModel.ImportObject

$DeleteObject.ObjectType = $objectType

$DeleteObject.TargetObjectIdentifier = $objectId

$DeleteObject.SourceObjectIdentifier = $objectId

$DeleteObject.State = 'Delete'

$DeleteObject | Import-FIMConfig -uri $URI

trap

{

    $exMessage = $\_.Exception.Message

    if($exMessage.StartsWith("L:"))

    {write-host "`n" $exMessage.substring(2) "`n" -foregroundcolor white -backgroundcolor darkblue}

    else {write-host "`nError: " $exMessage "`n" -foregroundcolor white -backgroundcolor darkred}

    #Exit

}

    }

}

# Prepare-FimObject

# Author: Community

# Source: TechNet Wiki

#

# Parameters:

# ObjectType: A String representing an object type in the Portal [Required]

# e.g. Person, Set, SynchronizationRule

# FimObjectID: A String representing the ObjectID of an existing object in the Portal [Optional]

#

# If only the ObjectType is specified a new object will be prepared. If a FimObjectID is specified

# a change to an existing object is prepared

#

# Outcome:

# An empty object which can be used in the Set-FimAttribute or Create-FimObject Functions

#

Function Prepare-FimObject

{

    Param

    (

        [parameter(Position=0, Mandatory=$true)]

[String]

$ObjectType,

[parameter(Position=1, Mandatory=$false, ValueFromPipeline = $true)]

[ValidateNotNullOrEmpty()]

[String]

$ObjectID

    )

    Process

    {

$Object = New-Object Microsoft.ResourceManagement.Automation.ObjectModel.ImportObject

       $Object.ObjectType = $objectType

#New Object

if(!$ObjectID){

$Object.SourceObjectIdentifier = [System.Guid]::NewGuid().ToString()

       }

#Existing Object

else{

$Object.TargetObjectIdentifier = $ObjectID

     $Object.SourceObjectIdentifier = $ObjectID

     $Object.State = 'Put'

}

return $Object

    }

}

# Set-FimAttribute

# Author: Community

# Source: TechNet Wiki

#

# Parameters:

# Object: A valid import object, typically coming from the Prepare-FimObject Function

# AttributeName: The name of the attribute to populate

# AttributeValue: A String value (empty string is allowed)

#

# Outcome:

# The input object with the change appended which in turn can be used in the Set-FimAttribute

# or Create-FimObject Functions

#

Function Set-FimAttribute

 {

    Param

    (

        [parameter(Position=0, Mandatory=$true, ValueFromPipeline = $true)]

        [ValidateNotNull()]

[Microsoft.ResourceManagement.Automation.ObjectModel.ImportObject]

$object,

[parameter(Position=1, Mandatory=$true)]

[String]

$attributeName,

[parameter(Position=2, Mandatory=$true)]

[AllowEmptyString()]

[String]

$attributeValue

    )

    Process

    {

        $importChange = New-Object Microsoft.ResourceManagement.Automation.ObjectModel.ImportChange

        $importChange.Operation = 'Replace'

        $importChange.AttributeName = $attributeName

        $importChange.AttributeValue = $attributeValue

        $importChange.FullyResolved = 1

        $importChange.Locale = "Invariant"

        if ($object.Changes -eq $null) {

$object.Changes = (,$importChange)

}

        else {

$object.Changes += $importChange

}

return $object

    }

}

# Create-FimObject

# Author: Community

# Source: TechNet Wiki

#

# Parameters:

# Object: A valid import object, typically coming from the Prepare-FimObject or Set-FimAttribute Function

#

# Outcome:

# The object is created in the Portal

#

Function Commit-FimObject

{

    Param

    (

        [parameter(Mandatory=$true, ValueFromPipeline = $true)]

        [Microsoft.ResourceManagement.Automation.ObjectModel.ImportObject]

$object

    )

    Process

    {

 if(@(get-pssnapin | where-object {$\_.Name -eq "FIMAutomation"} ).count -eq 0) {add-pssnapin FIMAutomation}

 clear-host

 $object | Import-FIMConfig -uri $URI

 trap

{

    $exMessage = $\_.Exception.Message

    if($exMessage.StartsWith("L:"))

    {write-host "`n" $exMessage.substring(2) "`n" -foregroundcolor white -backgroundcolor darkblue}

    else {write-host "`nError: " $exMessage "`n" -foregroundcolor white -backgroundcolor darkred}

    #Exit

}

    }

}

# Get-FimObjectID

# Author: Thomas Vuylsteke

# Source: /

#

# Parameters:

# Filter: A string representing an Xpath Filter

#

# Outcome:

# The objectID of the object matching the filter.

# An empty string is returned when multiple objects match the filter

#

Function Get-FimObjectID

{

    Param

    (

        [parameter(Mandatory=$true, ValueFromPipeline = $true)]

        [String]

$filter

    )

    Process

    {

 if(@(get-pssnapin | where-object {$\_.Name -eq "FIMAutomation"} ).count -eq 0) {add-pssnapin FIMAutomation}

 clear-host

 $FimObject = export-fimconfig -uri $URI `

                                –onlyBaseResources `

                                -customconfig $filter;

 if(($FimObject -ne $null) -and ($FimObject.Count -lt 2)){

  $psObject = $FimObject |Convert-FimExportToPSObject

  $objectID = (($psObject.ObjectID).split(":"))[2]

return $ObjectID

 }

 elseif($FimObject.Count -gt 1){

  throw "L1:Get-FimObjectID: More than one object is returned from the filter"

 }

 else{

  return $null

 }

 trap

{

    $exMessage = $\_.Exception.Message

    if($exMessage.StartsWith("L1:")){

write-host "`n" $exMessage.substring(3) "`n" -foregroundcolor white -backgroundcolor darkblue

return ""

}

    else {

write-host "`nError: " $exMessage "`n" -foregroundcolor white -backgroundcolor darkred

}

    #Exit

}

    }

}

# PowerShell Knowledge bit #1:

# The Mandatory attribute blocks null and empty values and prompts you for a value.

# To allow empty values (including null) add the AllowEmptyString parameter attribute

# ValidateNotNullOrEmpty is redundant when Mandatory is specified

# PowerShell Knowledge bit #2:

# From http://technet.microsoft.com/en-us/library/ff394140(WS.10).aspx [Craig Martin]

# Powershell does a pretty good job using Enums, and there are two cases for them in the FIM cmdlets:

# [Enum]::GetValues('Microsoft.ResourceManagement.Automation.ObjectModel.ImportOperation')

# [Enum]::GetNames('Microsoft.ResourceManagement.Automation.ObjectModel.ImportState')

# Many of the FIM PowerShell samples here in TechNet could be more readable if they took advantage of this. For example, instead of passing the number for the Object State, or Change.Operation, you can specify the string, or the value directly from the enum. In the case of 'Add':

# [Microsoft.ResourceManagement.Automation.ObjectModel.ImportOperation]::Add or just

# 'Add'

# http://technet.microsoft.com/en-us/library/ff720152(WS.10).aspx

# States

# 0 = Create

# 1 = Put

# 2 = Delete

# 3 = Resolve

# 4 = None

# Operations

# 0 = Add

# 1 = Replace

# 2 = Delete

$curDir = Split-Path -parent $MyInvocation.MyCommand.Definition

$dotSource = $curDir + "\./FIMFunctionLibrary.ps1"

. $dotSource

if($args.count -ne 1) {throw "You need to specify your attribute values as parameter"}

$attributes = ($args[0]).split("|")

if(0 -ne [String]::Compare(($attributes[0]).split(":")[0],"displayname", $true)){

throw "You need to specify a display name"

}

$objectName = ($attributes[0]).split(":")[1]

$exportObject = export-fimconfig -uri $URI `

                                  –onlyBaseResources `

                                  -customconfig "/Person[DisplayName='$objectName']"

if($exportObject) {throw "L:User $objectName already exists"}

#Prepare the object

$object = Prepare-FimObject "Person"

#Populate attributes

foreach($attribute in $attributes)

 {

    $attrData = $attribute.split(":")

    Set-FimAttribute -object $object -attributeName $($attrData[0]) -attributeValue $($attrData[1])

 }

#Create the object in FIM

Commit-FimObject $object