Microsoft Script Explorer for Windows PowerShell

Repository Provider Guide

Microsoft Corporation

Abstract

This document describes how to create custom script repositories for Microsoft® Script Explorer for Windows PowerShell®.

Disclaimer: The sample code and scripts in this guide are not supported under any Microsoft standard support program or service. The sample code and scripts are provided AS IS without warranty of any kind. Microsoft further disclaims all implied warranties including, without limitation, any implied warranties of merchantability or of fitness for a particular purpose. The entire risk arising out of the use or performance of the sample scripts and documentation remains with you. In no event shall Microsoft, its authors, or anyone else involved in the creation, production, or delivery of the scripts be liable for any damages whatsoever (including, without limitation, damages for loss of business profits, business interruption, loss of business information, or other pecuniary loss) arising out of the use of or inability to use the sample scripts or documentation, even if Microsoft has been advised of the possibility of such damages.

Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted in examples herein are fictitious. No association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2011 Microsoft Corporation. All rights reserved.

Microsoft, Windows, Windows Server, and Windows PowerShell are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries/regions.

All other trademarks are property of their respective owners.

Table of Contents

[1. About Repositories 4](#_Toc332894283)

[2. Design Your Repository 4](#_Toc332894284)

[3. Develop Your Repository 7](#_Toc332894285)

[3.1 Implement the search repository 7](#_Toc332894286)

[3.2 Enable saving 10](#_Toc332894287)

[3.3 Configure the repository 11](#_Toc332894288)

[3.4 Deploy the repository 12](#_Toc332894289)

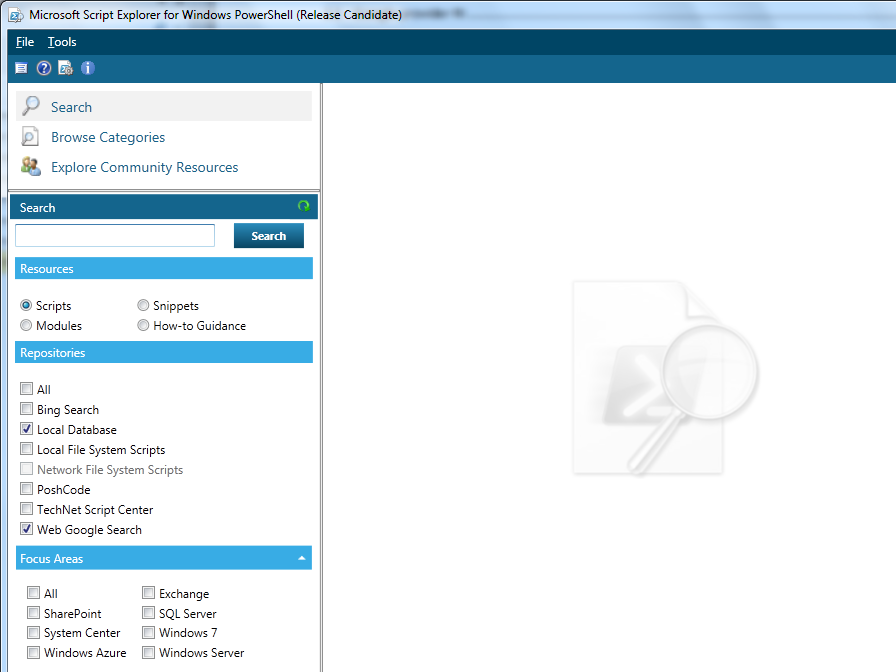
[3.5 Fine-tune repository functionality by external configuration 12](#_Toc332894290)

[4. Creating Preprocessor and Postprocessor Classes 13](#_Toc332894291)

# About Repositories

In Script Explorer, entities such as scripts are searched for and retrieved from repositories. Entities can also be saved to a storage location associated with a repository. Therefore, a repository is a data store that can be read from and written to by Script Explorer. When you create a new repository for a script resource, it appears as one of the repositories in Script Explorer.

As an example, Figure 1 shows a **Local Database** custom repository that was created using this guide and added to the list of available repositories in the appropriate configuration file.

  
*Figure 1 A* **Local Database** *custom repository*

# Design Your Repository

The following steps provide guidance for designing your repository. Make the following decisions:

1. Choose a name to be used as a moniker for the repository. Search results are aggregated across multiple repositories, and this moniker identifies the repository to use when retrieving an entity’s details. The name:
   1. Must be an ASCII uppercase alphanumeric string: [A-Z0-9]+
   2. Must be at least 1 character and less than 128 characters in length.
   3. Must not be used by other repositories in the same system.
2. Choose the entity type (e.g.: Microsoft.iX.AggregationService.Entities.Script) that will be returned from a search and how fields from your data store will be mapped to the entity’s properties. A Script entity has the following fields in the search results pane:

* **Title** The title of an entity (e.g.: script).
* **Summary** A brief summary of what the entity’s code will do. This appears just below the title.
* **Source** The display name of the repository that the entity came from.

1. Choose how fields from your data store will be mapped to the following properties in the Microsoft.iX.AggregationService.Entities.Constants.Script entity’s properties as the result of a search:

* ContentType = Microsoft.iX.AggregationService.Entities. Constants.Entity
* Name (for example, a filename)
* Rating (optional)
* ScriptId (optional, can be used as a placeholder when your unique key is a Guid)
* ScriptLocator (for example, “Moniker:UniqueKey:RepositoryName:HostEndpoint”)
* Summary
* Source (e.g.: repositoryProvider.DisplayName)
* Title

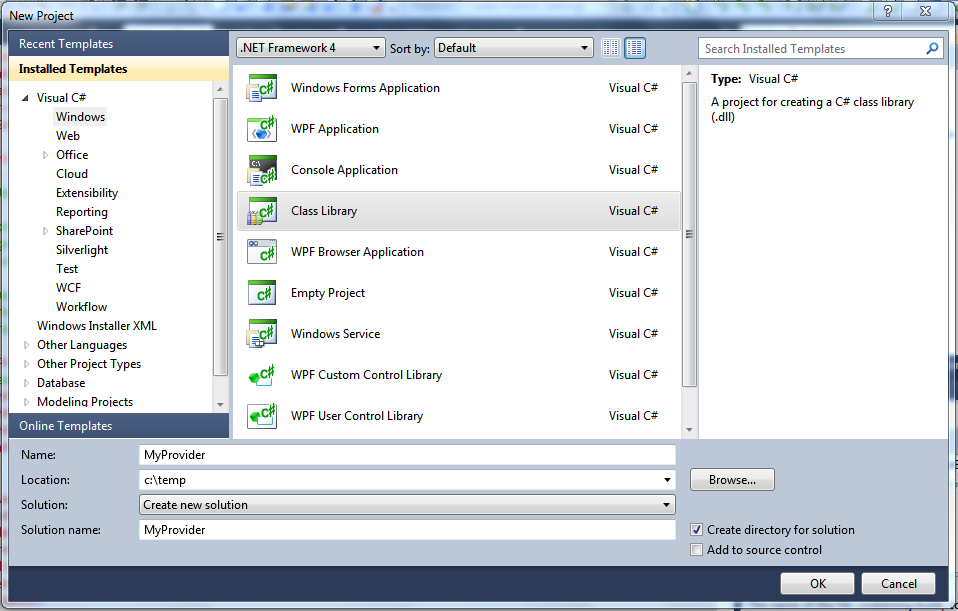
When getting a Microsoft.iX.AggregationService.Entities.Script entity for the details pane, set the following additional properties:

* Description
* ScriptCode
* Author
* PublishedDate
* Raters (optional)
* Tags
* Link
* Disclaimer (optional)
* Attachment (optional)

1. Choose a unique key. This should be a small subset of entity properties that uniquely identifies an entity (e.g.: Script) and distinguishes it from all other entities. This can be a URL, Guid, or primary key in a database. Note that this key must be unique across all instances of your repository class.
2. Other functional, operational, and practical decisions:
   1. Your repository should consistently return the same set of entities when presented with the same search terms.
   2. If you want to save content found via other repositories, the following conditions may apply:
   * Elements returned from other repositories may have properties that do not exist in your repository’s data store.
   * The repository you are retrieving entities from may require that a disclaimer be displayed to end users in the details pane. The Disclaimer property in script entity objects may be used to satisfy this requirement.

# Develop Your Repository

To develop your repository, create a Visual C# class library project that targets .NET Framework 4 (Figure 2).

   
*Figure 2 Creating a Visual C# class library project that targets .NET Framework 4*

Add the following assembly references, which include common assets and interfaces:

* %ScriptExplorer\_InstallDir%/ Microsoft.iX.AggregationService.Configuration.dll
* %ScriptExplorer\_InstallDir%/ Microsoft.iX.AggregationService.Entities.dll
* %ScriptExplorer\_InstallDir%/ Microsoft.iX.AggregationService.Interfaces.dll
* %ScriptExplorer\_InstallDir%/ Microsoft.iX.AggregationService.Providers.dll
* %ScriptExplorer\_InstallDir%/ Microsoft.iX.AggregationService.Providers.Interfaces.dll
* System.Configuration.dll

## Implement the search repository

A repository is responsible for searching entities and optionally saving them to a specific storage location. The first step to follow is to inherit from **BaseSearchProvider**.

The **Script** entity is defined in the Microsoft.iX.AggregationService.Entities.dll assembly and corresponds to objects displayed in the search results pane.

The **BaseSearchProvider** is defined in the Microsoft.iX.AggregationService.Providers.dll assembly and provides common infrastructure needed for most search repositories.

Implement your search repository class:

public class MyProvider : BaseSearchProvider

{

public MyProvider(string name, RepositoryElement repositoryElement, IHost host, ILogger logger, ISettingsManager settings) :

base(name, repositoryElement, host, logger, settings)

{

}

}

**Constructor parameters**

1. **Name** The Name attribute of this repository element.
2. **RepositoryElement** The object containing information from the repository element in Microsoft.iX.ScriptExplorer.exe.config.
3. **Host** The object containing information about the instance of the host where this repository is located.
4. **Logger** The component responsible for logging.
5. **Settings** The settings that are saved in the %Users%\AppData\Roaming\Microsoft\Microsoft Script Explorer for Windows PowerShell

Specify the unique name you decided for your repository class (see [Design Your Repositor](#_Design_Your_Repository)y):

public override string Moniker

{

get { return "myprovider"; }

}

The **Moniker** property identifies the repository to use when retrieving an entity such as a script. It is used when composing a Script Locator, which also includes an entity identifier (e.g.: URL) and the aggregation host name used to retrieve the entity.

Next, implement the **Search** method (again, see [Design Your Repository](#choosefields)):

public override IEnumerable Search(string searchCriteria)

{

// Implement search logic

// Populate the Title, Summary and Source as you would

// like them to see

}

The **Search** method is the core of the repository. This is where to add the logic for performing a search and returning the results.

The **searchCriteria** passed to the **Search** method is the text entered in the **Search** text box, combined with the selected focus areas in the following format:

searchTextFromTextBox AND (FocusArea1 OR FocusArea2 OR ….)

**Note** Refer to the *Script Explorer Reference Implementation* for more details. Download it from <http://scriptexplorer.codeplex.com>.

Compose the Script Locator based on the unique repository key (see [Design Your Repository](#chooseprovidername)) and the unique key of the entity (see [Design Your Repositor](#chooseuniquekey)y). The script locator must be in the following format:

provider key: …[entity key].. ……:provider name: host name

You can combine the fields into a string in any way in the highlighted area in the preceding code, most commonly using a delimiter such as a colon (“:”). Note that the string must be passed around as a part of a URL, so you should avoid:

* Any characters that need special treatment in a URL (-, /, <>).
* Any characters that could occur in the key fields themselves.

**Note** Even if you choose to compose your entity key in a different way, the delimiters between the repository key and those with the fields at the end must be a colon (“:”), as shown.

The repository name and host name at the end of the locator string can be obtained from methods in **BaseSearchProvider**:

script.ScriptLocator = string.Format("{0}:…..:{n}:{n+1}", unique provider name (decision#1), …., this.Name, this.Host.GetHostname());

Next, implement the **GetStatus** method:

public override ProviderStatus GetStatus()

{

// Return status

}

The **GetStatus** method is responsible for returning the status of the repository. The status is composed of a code and a message. The codes map to HTTP status codes. For example, 200 is **OK**, and 400 is **BadRequest**. For more information, go to <http://msdn.microsoft.com/en-us/library/system.net.httpstatuscode.aspx>.

Next, implement the **GetDetails** method as follows:

public override object GetDetails(string moniker)

{

// Implement search logic

}

This method receives an input parameter named **moniker**, which contains the identifier of the script. This is really the script **Locator** string that was placed in the **ScriptLocator** field of the selected result whose details are being requested.

The **Disclaimer** field is special in two ways:

1. If it is null, on trying to save the **Entity**, no **Accept**/**Decline** dialog box will be displayed.
2. The text set in the **Disclaimer** field must be well-formed **System.Windows.Documents.FlowDocument** XML, such as:

<FlowDocument xmlns="<http://schemas.microsoft.com/winfx/2006/xaml/presentation>" xmlns:x="<http://schemas.microsoft.com/winfx/2006/xaml>"

x:Name="FlowDocumentDisclaimer" AutomationProperties.Name="FlowDocumentDisclaimer">

<Paragraph FontWeight="Normal" FontSize="10.667" Foreground="#FF58595B" FontFamily="Segoe UI">

           <Bold>Disclaimer:</Bold>

           <Run> Place Holder: Please provide appropriate disclaimer.

           </Run>

</Paragraph>

</FlowDocument>

When present, the Disclaimer is rendered by WPF as a FlowDocument in the details pane.

Next, implement the **Clone()** method as follows:

public override object Clone()

{

var cloned = new MyProvider (this.Name, this.ConfigurationElement, this.Host, this.Logger);

return cloned;

}

## Enable saving

You can decide whether the repository will support saving or not by implementing the **ISaveProvider** interface in the **Search** provider, which is defined in the Microsoft.iX.AggregationService.Interfaces.dll assembly as follows:

public class MyProvider : BaseSearchProvider, ISaveProvider

{

}

public bool CanSave()

{

throw new NotImplementedException();

}

The **CanSave** method is responsible for stating if the repository contains the necessary context for performing a **save** operation (for example, if a database is available or if we have permissions for writing a file):

public void Save(object entity, System.Dynamic.ExpandoObject context = null)

{

throw new NotImplementedException();

}

The **Save** method is where you implement the save operation.

Parameters:

1. **Entity** The entity to be saved.
2. **Context**  Optional save context

## Configure the repository

The repository is consumed by the aggregation service through configuration. You must register the repository in the **<repositories>** collection in the .config file:

<adm service="wcf">

<resource resourceSet="Scripts" name="Script" clrType="Microsoft.iX.AggregationService.Entities.Script, Microsoft.iX.AggregationService.Entities" key="ScriptId" informationResourceType="EntityType" namespace="AggregationServiceProvider">

<repositories>

<repository name="MyProvider" displayName="Xml provider" saveEnabled="true" xmlFilePath="c:\temp\storage.xml" type="[ assembly qualified name ]"/>

</repositories>

</resource>

</adm>

**Note** Depending on where you are hosting the aggregation service, you need to update the appropriate configuration file. For self-hosting, update the Microsoft.iX.ScriptExplorer.exe.config file. For IIS hosting, update the web.config file.

**Required attributes**

1. **name** The name of the repository.
2. **type** The assembly qualified name of the repository.

**Optional attributes**

1. **displayName** The name to be used when showing this repository in Script Explorer. If this attribute is missing, the repository name will be used instead.
2. **saveEnabled**  Determines whether the repository is enabled for saving. This attribute allows you to turn on and off the save feature. The default value is **false**.

**Custom attributes**

* Additional information for the repository that must be key/name attributes.

## Deploy the repository

In order to enable the repository, you must follow these steps:

* Self-hosting:

1. Copy the repository assembly to %ScriptExplorer\_InstallDir%\LocalService\v1.0\bin.

2. Configure the repository in the Microsoft.iX.ScriptExplorer.exe.config file located in %ScriptExplorer\_InstallDir%.

* Web hosting:
  1. Copy the repository assembly to the service’s bin folder (%ScriptExplorer\_InstallDir%\AggregationService\v1.0\bin).
  2. Configure the repository in the web.config file.

## Fine-tune repository functionality by external configuration

You can use configuration from the Microsoft.iX.ScriptExplorer.exe.config file (or web.config) file to modify your repository behavior. The search provider gets a **RepositoryElement** object in its constructor. This System.Configuration.ConfigurationSection object can be retrieved with the following code:

configurationElement.CurrentConfiguration.GetSection(“adm”)

# Creating Preprocessor and Postprocessor Classes

Script Explorer searches can be customized through the use of preprocessors and postprocessors associated with a Resource (e.g.: Scripts, Snippets, Modules in the <adm> section of Microsoft.iX.ScriptExplorer.exe.config) or a Repository provider. Preprocessors and Postprocessors are configured as part of a <resource> or <repository> element in Microsoft.iX.ScriptExplorer.exe.config.

Preprocessors are executed before calling the Search methods of providers, and can be used to access resource information and parameters supplied by Script Explorer. This information can be retained by providers for use during search operations.

Postprocessors are executed after calling the Search methods of providers, and can be used to filter entities (e.g.: Scripts) returned from providers.

Custom logic can be added at two levels, resource and provider respectively.

*Resource*

      <resource resourceSet="Scripts" name="Script">

<preProcessors>

<preProcessor name="ProviderSort" type="Microsoft.iX.AggregationService.DataServiceProvider.ProviderSortPreProcessor, Microsoft.iX.AggregationService.DataServiceProvider"/>

</preProcessors>

        <postProcessors>

          <postProcessor name="YourProcessorName" type="FullTypeName, AssemblyName"/>

        </postProcessors>

IResourcePreProcessor.Process accepts a cloned IResource object and a NameValueCollection of parameters supplied by Script Explorer.

public interface IResourcePreProcessor

{

void Process(IResource resource, NameValueCollection parameters);

}

IResourcePostProcessor.Process accepts the cloned IResource object used by the preprocessor and search operations and an IEnumerable containing the entities (e.g.: Script objects) returned by a search. It then returns a potentially filtered and/or reordered IEnumerable of the entities it received.

public interface IResourcePostProcessor

{

IEnumerable Process(IResource resource, IEnumerable input);

}

*Provider*

          <repository name="Posh">

            <preProcessors>

           <preProcessor name="YourProcessorName" type="FullTypeName, AssemblyName"/>

            </preProcessors>

            <postProcessors>

           <postProcessor name="YourProcessorName" type="FullTypeName, AssemblyName"/>

            </postProcessors>

          </repository>

IProviderPreProcessor.Process accepts a cloned IProvider object and a NameValueCollection of parameters supplied by Script Explorer.

public interface IProviderPreProcessor

{

void Process(IProvider provider, NameValueCollection parameters);

}

IProviderPostProcessor.Process accepts the cloned IProvider object used by the preprocessor and search operations and an IEnumerable containing the entities (e.g.: Script objects) returned by a search. It then returns a potentially filtered and/or reordered IEnumerable of the entities it received.

public interface IProviderPostProcessor

{

IEnumerable Process(IProvider provider, IEnumerable input);

}