Sitecore Installation Framework 2.3.0 Configuration Guide

February 24, 2021
Sitecore Experience Platform 10.1.0

Table of Contents

1. Introduction

This document describes Sitecore Install Framework (SIF) for Sitecore XP 10.0.0 - 10.0.1.

This document contains the following chapters:

- **Chapter 1 – Introduction**
  An introduction to the Sitecore Install Framework module.
- **Chapter 2 – Install the Module**
  How to install the Sitecore Install Framework module.
- **Chapter 3 – Customize Sitecore Install Framework**
  The extension points that allow you to customize SIF for your specific installation needs
- **Chapter 4 – Further guidance and troubleshooting**
  Extra information and common troubleshooting strategies.

1.1. Getting started

Sitecore Install Framework is a Microsoft® PowerShell module that supports local and remote installations of Sitecore XP. It is fully extensible and you can use it to install the entire Sitecore solution (XP), or the CMS-only mode (XM) solution.

This guide describes how to configure and customize the installation process with SIF.

1.1.1. How to use this guide

This guide describes how to configure your Sitecore installation with SIF, as well as various cmdlets and extensibility points.

This guide is intended to be a supplement all the Sitecore XP installation guides and to help you customize your installation. For more information about system requirements, prerequisites, the general installation process, and the additional configuration tasks that are necessary after you install Sitecore XP, see the appropriate installation guide.

You can download the Sitecore installation guides from the Sitecore Downloads page.

2. Install the Sitecore Install Framework module

This chapter contains information about installing, updating, validating, and importing the Sitecore Install Framework module.

This chapter contains the following sections:

- Install the Sitecore Install Framework Module
- Multiple Versions of Sitecore Install Framework

2.1. Install the Sitecore Install Framework Module

You can install the Sitecore Install Framework (SIF) module directly with Microsoft PowerShell®, or you can install it manually by downloading the module as a ZIP package.

2.1.1. Install SIF with Microsoft PowerShell

SIF is available from the Sitecore Gallery. The Sitecore Gallery is a public MyGet feed where you can download the PowerShell modules created by Sitecore.

To install SIF with PowerShell:

1. To add the repository, in Windows, open PowerShell as an administrator and run the following cmdlet:

   ```
   Register-PSRepository -Name SitecoreGallery -SourceLocation https://
   sitecore.myget.org/F/sc-powershell/api/v2
   ```

2. When you are prompted to add the repository, press **Y**, and then **Enter**.
3. To install the PowerShell module, run the following cmdlet:

   ```
   Install-Module SitecoreInstallFramework
   ```

4. When you are prompted to install the module, press **Y**, and then **Enter**.

Update the Sitecore Install Framework Module

As new features and bug fixes are periodically released, it is recommended that you update the Sitecore Install Framework.

> **NOTE**
> This procedure is optional.

- To update the Sitecore Install Framework module, run the following cmdlet:

  ```
  Update-Module SitecoreInstallFramework
  ```

2.1.2. Install SIF manually

Sitecore Install Framework is also provided as a ZIP package. You can download SIF from the Sitecore Downloads page – https://dev.sitecore.net. When you download the package, the ZIP package might be marked as *blocked* by Microsoft Windows. To continue the installation of SIF, you must first unblock the ZIP package.

Unblock a ZIP package

To unblock a ZIP package:

1. In Windows Explorer, navigate to the folder where you downloaded the packages, and right-click the relevant ZIP file.
2. Click **Properties**.
3. In the **Properties** dialog box, on the **General** tab, click **Unblock**.
4. Click **OK**.

Extract the Sitecore Install Framework

The installation path that you use depends on where you want to install Sitecore Install Framework. You can install it for all users (global path), for a specific user, or to a custom location.

Usage	Path
All users	`C:\Program Files\WindowsPowerShell\Modules`
Specific user	`C:\Users\<user>\Documents\WindowsPowerShell\Modules`
Custom location	Any path

For example, if you want to make SIF available to all users, extract the `Sitecore Install Framework.ZIP` package to the following folder:

```
C:\Program Files\WindowsPowerShell\Modules\SitecoreInstallFramework
```

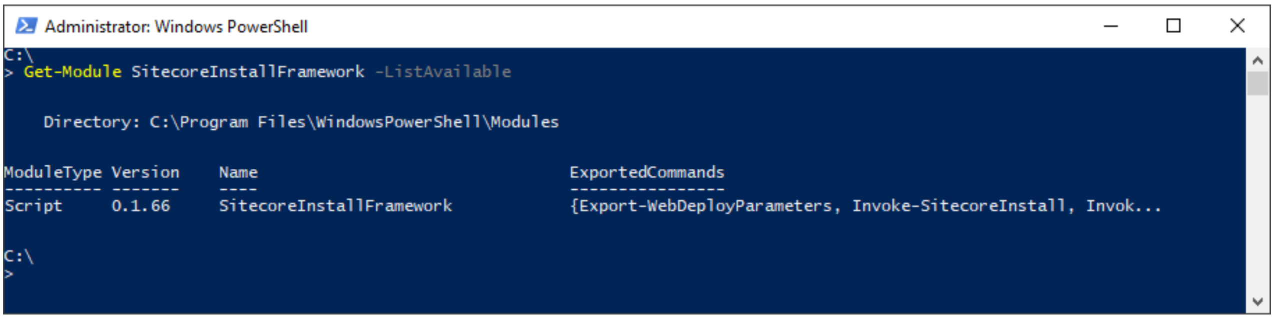
2.1.3. Validate the installation

To confirm that SIF is available for use after you install it, you can validate the installation. This procedure is optional.

> **NOTE**
> Validation only works if you have installed SIF for *All users* (global).

To validate the installation, run the following cmdlet:

```
Get-Module SitecoreInstallFramework –ListAvailable
```



```
C:\
> Get-Module SitecoreInstallFramework -ListAvailable


    Directory: C:\Program Files\WindowsPowerShell\Modules


ModuleType Version    Name                                ExportedCommands
---------- -------    ----                                ----------------
Script     0.1.66     SitecoreInstallFramework            {Export-WebDeployParameters, Invoke-SitecoreInstall, Invok...


C:\
>
```

2.1.4. Import Sitecore Install Framework into a PowerShell session

If you have made SIF available to either *All users* or a *Specific user*, you do not have to import it, as this is done automatically, and you can immediately use it in a session by running any cmdlet that is available in the PowerShell module.

However, if you have installed SIF to a custom location, you must run the following cmdlet.

```
Import-Module C:\<CustomLocation>\SitecoreInstallFramework
```

2.2. Multiple Versions of Sitecore Install Framework

PowerShell uses the latest available version of a module in a session by default.

If you want to install a version of Sitecore XP that uses an older version of SIF, you must install the appropriate version of SIF.

For example, if you want to install a 9.0.x version of Sitecore XP on the same computer as a Sitecore XP 10.0.0 installation, you must also have SIF 1.2.1 installed.

To install a specific version of SIF, run the following cmdlet:

```
Install-Module -Name SitecoreInstallFramework -RequiredVersion x.x.x
```

Enter the appropriate value in the `RequiredVersion` parameter. The following table lists the versions of SIF that are compatible with Sitecore XP 9.0.0 and later:

Sitecore XP Version	Compatible SIF Version
9.0.x	1.2.1
9.1	2.0.0
9.1.1	2.1.0 or later
9.2	2.1.0 or later
9.3	2.2.0
10.X	2.3.0

2.2.1. Run a specific version of SIF

To run a specific version of SIF, start a new PowerShell session and run the following cmdlet:

```
Import-Module -Name SitecoreInstallFramework -Force -RequiredVersion x.x.x
```

You will use the specified version for the remainder of the session.

The next time you start a PowerShell session, it will automatically use the latest available version.

3. Customize the Sitecore Install Framework

SIF lets you customize the installation process by using a standard configuration design that you can extend with custom PowerShell functions. The framework defines a configuration format that supports tasks, parameters, config functions, and variables. For example, you can configure a computer with one or more Sitecore instances, add services, or add custom applications.

SIF configurations are written as JSON (JavaScript Object Notation) files.

This chapter contains the following sections:

3.1. Create and customize configurations

You can use tasks, parameters, config functions, and variables to customize the installation process. Custom tasks and config functions can also be packaged as a module, and included in configurations.

You can base your customization on one of the configurations provided by Sitecore, or you can create your own configuration. This section describes the different components that you can use in a Sitecore Install Framework configuration.

3.1.1. Tasks

Tasks are actions that are conducted in sequence when you run the `Install-SitecoreConfiguration` cmdlet. A task is implemented as a PowerShell cmdlet.

Each task is identified by a unique name and must contain a *Type* property. A task can have parameters or a collection of parameters passed to it. Tasks map directly to PowerShell functions and are registered with the `Register-SitecoreInstallExtension -Type` task.

The following example, is a task of the `Copy` type that you can use in a configuration to copy files from one location to another:

```
{
    "Tasks": {
        "CopyFiles" : {
            "Description": "Copies files to the specified location",
            "Type": "Copy",
            "Params": {
                "Source": "c:\files",
                "Destination": "c:\newfiles"
            }
        }
    }
}
```

Skipping tasks

A task can include a *Skip* property that refers to a parameter, variable, or config function. If the return value is *true*, the task is not executed.

```
{
    "Parameters : {
        "Param1" : {
            "Type" : "String"
            "DefaultValue" : ""
        }
    },
    "Tasks" :{
        "Task1" :{
            "Type" : "Copy
            "Params" :{
                "Source" : "C:\\Source",
                "Destination : "C:\\Destination"
            }
            "Skip" : "[not(Parameter('Param1'))]"
        }
    }
}
```

> **NOTE**
>
> To ensure that your configuration is valid, each task must have a unique name. This means that the task can be directly executed, and the task can be identified in a log. It also lets you use the same type of task multiple times in a configuration.

In this configuration, if Param1 is set to any value, `Task1` is skipped.

Requires (prerequisites)

A task can include a `Requires` block that allows prerequisite checks to be performed prior to executing the task. The prerequisite checks are performed after the `Skip` section. Prerequisites should take the form of Config Functions which return a Boolean. If the prerequisite check returns *false* , SIF will try to enter a nested shell within the current host. You will then have an opportunity to correct the problem, either from the prompt provided or externally.

After you have corrected the problem, enter `exit` to return to SIF. The prerequisite checks are then performed again. Alternatively, you can enter the `SkipRequire` command to skip this prerequisite. If the current host does not support a nested shell, the requirements are skipped.

```
{
    "Variables": {
        "Requires.Success" : [TestPath(Path:'C:\\Windows')]
    },
    "Tasks": {
        "Task1":{
            "Type": "WriteOutput",
            "Params" :{
                "InputObject":"Simple Task, depends on a variable."
            },
            "Requires": "[variable('Requires.Success')]"
        }
    },
    "Settings" : {
        "AutoRegisterExtensions" : True
    }
}
```

In this configuration, `Task1` requires the output of the `Requires.Success` variable to check that the presence of `C:\Windows` is *true*.

3.1.2. Parameters

Parameters let users change values inside a configuration at runtime. Parameters must declare a type. They can also declare a default value and a description.

When you add a parameter to a configuration, it can be passed into a task using the `parameter` config function. The task then points to the value provided in the configuration or the value passed when `Install-SitecoreConfiguration` is called.

For example, to pass the `Source` and `Destination` parameters to the `CopyFiles` task:

```
{
    "Parameters": {
        "Source": { "Type": "string", "Description": "The source of files" },
        "Destination": { "Type": "string", "DefaultValue": "c:\newfiles" }
    },
    "Tasks": {
        "CopyFiles" : {
            "Type": "Copy",
            "Params": {
                "Source": "[parameter('Source')]",
                "Destination": "[parameter('Destination')]"
            }
        }
    }
}
```

The `Source` parameter does not contain the *DefaultValue* property, and therefore it is required when `Install-SitecoreConfiguration` is called.

However, the `Destination` parameter does have a default value. If the user does not provide the value, the `DefaultValue` from the configuration is used.

The values at runtime are then passed to the `CopyFiles` task you use the `parameter` config function.

To pass the values at the command line, you must use the name of the parameter with the standard PowerShell parameter syntax. For example:

```
Install-SitecoreConfiguration -Path .\configuration.json -Source c:\sourcefiles
```

Parameters validation

Parameter values can be validated before any tasks are started. Validation logic can be specified for each parameter using config functions:

```
{
    "Parameters": {
        "Source": {
            "Type": "string",
            "DefaultValue": "c:\\myfiles",
            "Validate": "[validatelength(4, 260, $_)]"
        },
    }
}
```

In this example, the `Source` parameter is validated by the `validatelength` config function, which ensures that the value is between 4 and 260 characters long.

The `Validate` property accepts any config function that returns `bool`.

The predefined functions that you can use for validation are:

- `ValidateCount` - validates that the array length of the parameter is within the specified range.
- `ValidateLength` - validates that the length of the parameter is within the specified range.
- `ValidateNotNull` - verifies that the argument is not null.
- `ValidateNotNullOrEmpty` - validates that the argument is not null and is not an empty string.
- `ValidatePattern` - validates that the parameter matches the RegexPattern.
- `ValidateRange` - validates that number parameter falls within the range specified by `Min` and `Max`.
- `ValidateSet` - validates that the parameter is present in a specified set.

When you call `Install-SitecoreConfiguration`, all the parameters with validation logic are checked. If any validation fails, the installation command is aborted. To bypass validation, call `Install-SitecoreConfiguration` and use the `-SkipValidation` option.

3.1.3. Config functions

Config functions allow elements of the configuration to be dynamic, and allow you to calculate values, invoke functions, and pass these values to tasks so that a configuration can be flexible.

A config function is written as a string enclosed in square brackets `[ ]`. The function is identified by a name and can receive function parameters. For example: `[functionName(param1,param2,param3)]`.

Each function maps directly to a PowerShell function that is registered with the following cmdlet:

```
Register-SitecoreInstallExtension -Type ConfigFunction
```

Named parameters

To ensure that the correct values are passed to the expected parameter, you can reference the parameters of a config function directly.

Named parameters should be in a comma separated list. The parameter name must be followed by a colon:

```
{
    "Variables": {
        "Variable1": "GetFunction(Number:1234,String:'Text',Switch:True)]"
    }
}
```

In this example, `Variable1` calls the `GetFunction` function and passes the `Number` parameter the value *1234*, the `string` parameter the value `'Text'`, and the `Switch` parameter the value `True`. This is the equivalent of typing `Get-Function -Number:1234 -String:"Text" -Switch:$True` in the command line.

In the configuration `Switch` parameters, if declared, must be set to `True` or `False`.

Nested functions are also permitted and may refer to parameters, variables, or other config functions. For example:

```
[FunctionName1(functionName2(Switch1:'value1',Switch2:'value2'),Switch3:'value3')]
```

3.1.4. Variables

Variables are values that are calculated within the configuration itself. Unlike parameters, variables can use config functions to determine their value. Variables cannot be overridden at runtime. However, when they are calculated, they can use the values from parameters and other variables.

In the following example, the `Destination` variable uses the `environment` and `concat` config functions to determine the destination path:

```
{
    "Parameters": {
        "Source": { "Type": "string", "Description": "The source of files" }
    },
    "Variables": {
        "Destination": "[concat(environment('SystemDrive'),'\\newfiles')]"
    },
    "Tasks": {
        "CopyFiles" : {
            "Type": "Copy",
            "Params": {
                "Source": "[parameter('Source')]",
                "Destination": "[variable('Destination')]"
            }
        }
    }
}
```

3.1.5. Modules

SIF provides many tasks and config functions. You can load additional tasks and config functions by including them in a configuration.

If the module is available on PSModulePath, you can load modules by name. Alternatively, you must load modules using an explicit path. Modules are loaded at the beginning of an installation.

If the additional features are packaged in a module on your computer, you can import them into the install session by adding them in the `Modules` section of a configuration. For example:

- To load a module by name –"`MyCustomModule`"
- To load a module by explicit path – "`C:\\extensions\\extensions.psm1`"

```
{
    "Modules": [
        "MyCustomModule",
        "C:\\extensions\\extensions.psm1"
    ]
}
```

3.1.6. Uninstall tasks

Configurations can contain an optional `UninstallTasks` section. This section has the same form as the `Tasks` section. It should contain a list of tasks that undo all the actions completed in the `Tasks` section.

Uninstall tasks are invoked either by passing the `-Uninstall` switch to `Install-SitecoreConfiguration` or by calling the `Uninstall-SitecoreConfiguration` alias. If you call `Uninstall-SitecoreConfiguration`, you do not have to include the `-Uninstall` switch.

Uninstall tasks can be treated in the same way as normal tasks and support all the same functionality.

```
{
    "Parameters" :{
        "SolrService": {
            "Type": "string",
            "DefaultValue": "Solr-7.2.1",
            "Description": "The name of the Solr service."
        },
    },
    "UninstallTasks": {
        "RemoveSolrService": {
            "Type": "RemoveService",
            "Params": {
                "Name": "[parameter('SolrService')]"
            }
        },
    }
}
```

In this example, the `RemoveSolrService` uninstall task executes the `RemoveService` task and stops and removes the service specified in the `SolrService` parameter.

If you have referenced other configurations in the `Includes` section, the `Uninstall Tasks` for each of the included configurations are run, in reverse order, after the tasks in the first configuration.

3.1.7. Register

Each entry in the `Register` section allows you to expose PowerShell functions as SIF tasks or ConfigFunctions within the config file that you can use in the `Variables` or `Tasks` sections. They are registered automatically with the `Register-SitecoreInstallExtension` command.

The section takes the format:

```
{
    "Register": {
        "Tasks" : {
            "NewSMBShare" : "New-SMBShare",
            "Sleep": "Start-Sleep"
        },
        "ConfigFunction": {
            "GetRandom" : "Get-Random"
        }
    }
}
```

This configuration registers the `New-SMBShare` cmdlet as a task with the name `NewSMBShare` and registers `Start-Sleep` as a task with the name `Sleep`. The `Get-Random` cmdlet is registered as a config function with the name `GetRandom`.

Just like when you extend SIF, the `Tasks` section of `Register` is for functions that don't return anything and the `ConfigFunctions` section is for functions that return a value.

3.1.8. Automatic registration of extensions

Configuration files can optionally declare the `AutoRegisterExtensions` setting that allows you to dynamically register `Tasks` and `ConfigFunctions`. The default value is `false`.

Any PowerShell cmdlet can be referenced in a config file using a de-hyphenated version of its name.

```
{
    "Tasks": {
        "RandomSleep": {
            "Type": "StartSleep",
            "Params": {
                "Seconds": "[GetRandom(10)]"
            }
                }
    },
    "Settings" : {
        "AutoRegisterExtensions" : true
    }
}
```

In this configuration, the `RandomSleep` task automatically registers and executes `Start-Sleep` as a task and `Get-Random` as a `ConfigFunction`. It is the equivalent of running `Start-Sleep -Seconds (Get-Random 10)` on the command line.

If two or more cmdlets match, the cmdlets are listed and an error is thrown.

3.1.9. Includes

To reuse elements and reduce repetition, configurations can refer to other configuration files.

For example, the following config is a pseudo config which configures a site. This is saved as `CreateSite.json`:

```
{
  "Parameters": {
    "Destination":{
      "Type": "string"
    },
    "SourcePackage":{
      "Type": "string"
    },
    "SiteName":{
        "Type": "string"
    }
  },
  "Tasks":{
    "CreateSite":{
      "Type": "CreateSite",
      "Params":{
        "SiteName": "[parameter('SiteName')]",
        "Path": "[parameter('Destination')]"
      }
    },
    "InstallPackage":{
      "Type": "CreateSite",
      "Params":{
        "Source": "[parameter('SourcePackage')]",
        "Path": "[parameter('Destination')]"
      }
    }
  }
}
```

Another config can be created which includes the previous example:

```
{
    "Includes":{
        "CreateSite":{
            "Source": ".\\CreateSite.json"
        }
    }
}
```

You can override elements of the included config by using the namespace it was imported as (in this case 'CreateSite'):

```
{
    "Parameters": {
        "CreateSite:Destination":{
            "Type": "string",
            "DefaultValue": "C:\\inetpub"
        }
    },
    "Includes":{
        "CreateSite":{
            "Source": ".\\CreateSite.json"
        }
    }
}
```

This example overrides the default value of the destination parameter imported by the `CreateSite` config. A similar syntax is used to override `Variables` and `Tasks`.

You can also include the same config multiple times, as the name is used to namespace each feature:

```
{
  "Parameters": {
    "CreateSite:Destination":{
      "Type": "string",
      "DefaultValue": "C:\\inetpub"
    },
    "CreateBackupSite:Destination":{
      "Type": "string",
      "DefaultValue": "C:\\inetpubbackup"
    }
  },
  "Includes":{
    "CreateSite":{
      "Source": ".\\CreateSite.json"
    },
    "CreateBackupSite":{
      "Source": ".\\CreateSite.json"
    }
  }
}
```

3.1.10. Settings

Settings let you configure the default requirements of the installation process. Some settings can be overridden at runtime for the user by passing them as parameters to the `Install-SitecoreConfiguration` cmdlet.

For example:

```
Install-SitecoreConfiguration -WarningAction Stop -InformationAction SilentlyContinue
```

The values in the settings are applied from lowest to highest in the following order:

- Default value – contained in code.
- Configuration – set in the configuration file.
- Command line – passed at the command line.

Name	Default Value	Allowed Values	Command Line	Description
WarningAction	Continue	Continue Ignore Inquire SilentlyContinue Stop Suspend	Yes	The action to take when a warning occurs.
ErrorAction	Stop	Continue Ignore Inquire SilentlyContinue Stop Suspend	Yes	The action to take when an error occurs.
InformationAction	Continue	Continue Ignore Inquire SilentlyContinue Stop Suspend	Yes	The action to take when information is logged.
AutoRegisterExtensios	False	True False	No	Enables Dynamic registration of Tasks and Config Functions.

Settings within configuration files can be declared as:

```
{
    "Settings": {
        "AutoRegisterExtensions": true,
        "InformationAction": "SilentlyContinue"
    }
}
```

This configuration enables the `AutoRegisterExtensions` feature and sets the `InformationAction` preference to `SilentlyContinue`.

3.2. Create tasks

Tasks are PowerShell functions that you can invoke from a SIF configuration. When you invoke a configuration, each task performs an action. Because a task is implemented as a PowerShell function, it benefits from all the features that PowerShell offers.

3.2.1. The CmdletBinding Attribute

When you create a task, you must use the CmdletBinding attribute. This provides support for common PowerShell parameters, such as those that control error handling.

It is best practice to use the SupportsShouldProcess parameter so that users can test the actions that the task takes, without applying them. The following example uses the CmdletBinding attribute in the Invoke-UnpackTask cmdlet:

```
Function Invoke-UnpackTask {
    [CmdletBinding(SupportsShouldProcess=$true)]
    param(
        # Parameters
    )
    # function code
}
```

For more information about the CmdletBinding attribute, see About Functions CmdletBindingAttribute in the Microsoft PowerShell documentation.

3.2.2. Task parameters

Task parameters are declared as normal PowerShell parameters. You can use validation and types to restrict the values that can be passed to the cmdlet. This includes marking parameters as mandatory, and support for multiple parameter sets.

When a task is called from a configuration, the *Params* property is mapped to the parameters that are declared in the PowerShell function. For example, the Invoke-CopyTask cmdlet declares the following parameters:

```
Function Invoke-CopyTask {
    [CmdletBinding(SupportsShouldProcess=$true)]
    param(
      [Parameter(Mandatory=$true)]
      [ValidateScript({ Test-Path $_ })]
      [string]$Source,
      [Parameter(Mandatory=$true)]
      [ValidateScript({ Test-Path $_ -IsValid })]
      [string]$Destination
    )
# function code
}
```

- The Source parameter is mandatory and checks that the given value is a string that points to a path that exists.
- The Destination parameter is mandatory and checks that the given value is a string that is a valid file path.

Here is an example of how to declare the Copy task in a configuration where only the mandatory parameters are used:

```
{
    "Tasks": {
        "CopySomeFiles": {
        "Type": "Copy",
        "Params": {
          "Source":"c:\somefile\example.txt",
          "Destination":"c:\copied\"
        }
      }
    }
}
```

3.2.3. Return values from task

Tasks do not need return values. When you invoke tasks through a configuration, the values returned from a task are not captured or processed directly. However, any value that you return from a task is shown in the installation logs.

3.2.4. Write to the logs

Previous versions of SIF used the `Write-TaskInfo` function to log information. This function has been deprecated and will be removed in a future version. Use the `Write-Information` function instead.

```
Write-Information -MessageData task "[Info] Updated"
```

Silent output

You cab specify silent output by re-directing all the streams to `$null`.

```
Install-SitecoreConfiguration -Path MyConfig.json *> $null
```

Creating a log file

In SIF version 1.x, every time you used the built-in PowerShell transcript features to invoke `Install-SitecoreConfiguration`, a log file was automatically created.

This feature has been removed due to issues with different hosts and with the information that was logged to a log file. To create a log file for an installation, use the following syntax:

```
c:\> Install-SitecoreConfiguration <parameters> *>&1 | Tee-Object <logfile>
```

`*>&1` merges every stream - information, warning, and so on into the output stream.

`| Tee-Object <logfile>` outputs the stream to the console and to a log file.

3.2.5. Include tasks in a configuration

Once a task has been written, it must be registered with SIF. To include tasks in a configuration, you must package them as a PowerShell module and add them to the `Modules` section of a configuration, by directly registering them within a configuration, or enabling the `autoregisterextensions` function.

When you use the `Register-SitecoreInstallExtension` cmdlet, you can use the task in configurations. For example, to register the `Copy-CustomItems` cmdlet as the `CustomCopy` task, use the following cmdlet:

```
Register-SitecoreInstallExtension -Command Copy-CustomItems -As CustomCopy -Type Task
```

> **NOTE**
>
> You can replace an existing registered task by using the `-Force` parameter. The following custom cmdlet replaces the default copy task: `Register-SitecoreInstallExtension -Command Copy-CustomItems -As Copy -Type Task -Force`.

3.3. Create config functions

Config functions are PowerShell functions that you can invoke from within a SIF configuration to access and calculate values that you can then pass to a task.

Because a config function is implemented as a PowerShell function, it benefits from all the features that PowerShell offers. This includes parameter validation, strict mode, requires, and others.

Config functions must always return a value, and this value can be used by other config functions or by tasks within a configuration.

3.3.1. Config function parameters

Config function parameters are declared as normal PowerShell parameters. You can use validation and types to restrict the values that can be passed. This includes marking parameters as mandatory and support for multiple parameter sets.

When a config function is called from a configuration, the parameters are applied in the order that they are declared. If you want them to be applied in a different order, use the Position argument to specify the order.

The `Invoke-JoinConfigFunction` function declares the following parameters:

```
Function Invoke-JoinConfigFunction {
    param(
        [Parameter(Mandatory=$true)]
        [psobject[]]$Values = @(),
        [Parameter(Mandatory=$false)]
        [string]$Delimiter = ","
    )

    # function code
}
```

In a configuration, this can be used as follows:

```
{
    "Parameters": {
        "Values": { "Type": "string[]", "DefaultValue": [ 1,2,3,4,5 ]
    },
    "Variables": {
        "Joined": "[join(parameter('Values'), '-')]
    }
}
```

When the `Joined` variable is evaluated, it results in a value of: `1-2-3-4-5`.

3.3.2. Include config functions in a configuration

Once you have written a config function, it must be registered with SIF. You can include config functions in a configuration by packaging them as a PowerShell module and adding them to the `Modules` section of a configuration.

By using the `Register-SitecoreInstallExtension` cmdlet, the config function is made available for use in configurations. For example the following cmdlet registers the `Get-CustomJoin` cmdlet as the `CustomJoin` config function:

```
Register-SitecoreInstallExtension
-Command Get-CustomJoin -As CustomJoin -Type ConfigFunction
```

NOTE

You can also use the –`Force` parameter to replace an existing registered config function. In the following example, the default join config function is replaced with a custom cmdlet:

```
Register-SitecoreInstallExtension -Command Get-CustomJoin -As Join -Type
ConfigFunction –Force
```

3.4. Invoking an installation

To start a Sitecore installation, use the following syntax:

```
Install-SitecoreConfiguration [-Path] <String> [[-Tasks] <String[]>] [[-From] <String>] [[-To]
<String>]
[[-Skip] <String[]>] [[-WorkingDirectory] <String>] [-WhatIf] [-Confirm] [<CommonParameters>]
```

This starts the Sitecore installation and uses the given configuration loaded from the specified path.

The working directory is set as follows:

- If provided, the working directory is set to the given path.
- The current directory is used.

One or more tasks can also be passed to enable the execution of only a selection of tasks from the full configuration. If no tasks are passed (or an empty list is provided) all the tasks are executed. Task execution can be further restricted by using the `From` and `To` parameters to specify an inclusive subset of tasks.

The parameters contained in the configuration file can also be overridden at the command line.

3.4.1. Examples

Examples based on JSON configuration files.

Example 1

```
PS C:\> Install-SitecoreConfiguration -Path .\MyConfig.json
```

Starts an installation based on a JSON configuration file.

Example 2

```
PS C:\> Install-SitecoreConfiguration -Path .\MyConfig.json -Tasks Alpha,Beta,Epsilon
```

Starts an installation based on a JSON configuration file and executes only the named tasks.

Example 3

```
PS C:\> Install-SitecoreConfiguration -Path .\MyConfig.json -Skip Alpha,Beta
```

Starts an installation based on a JSON configuration file and executes all the tasks except the named tasks.

Example 4

```
PS C:\> Install-SitecoreConfiguration -Path .\MyConfig.json -From Beta
```

Starts an installation based on a JSON configuration file and executes from the specified task.

Example 5

```
PS C:\> Install-SitecoreConfiguration -Path .\MyConfig.json -From Alpha -To Beta
```

Starts an installation based on a JSON configuration file and executes all the tasks from the task named Alpha to the task named Beta.

Example 6

```
PS C:\> Install-SitecoreConfiguration -Path .\MyConfig.json -SiteName 'MySite'
```

Starts an installation based on a JSON configuration file and overrides the value for the SiteName parameter contained in that file.

Example 7

```
PS C:\> Install-SitecoreConfiguration -Path .\MyConfig.json -SkipValidation
```

Starts an installation based on a JSON configuration file and skips parameter validation.

4. Further guidance and troubleshooting

SIF contains embedded documentation about tasks and configuration functions that you can access directly from the PowerShell command line.

This chapter contains the following section:

- Further usage and help
- Troubleshooting

4.1. Further usage and help

This section contains the following additional information that might be useful when you are using Sitecore Install Framework:

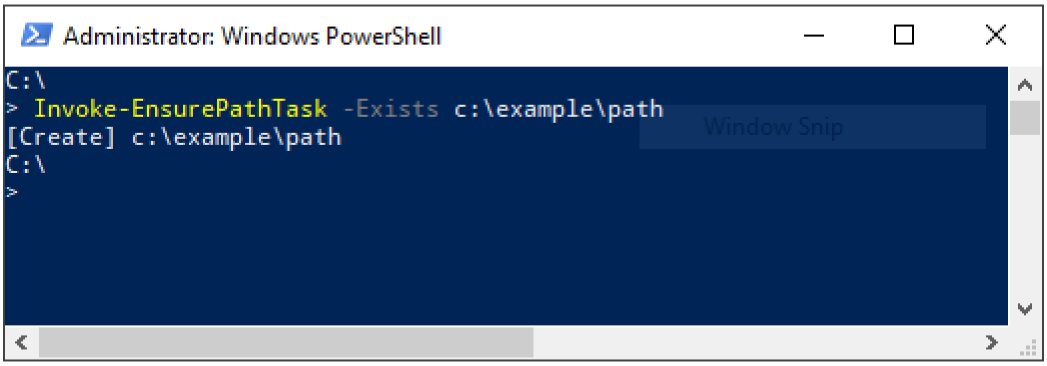
- Run tasks and config functions directly
- Execution policies
- Get help about Sitecore Install Framework

4.1.1. Run tasks and config functions directly

Tasks and config functions are implemented as PowerShell cmdlets. When SIF is installed, you can use standard PowerShell syntax to directly invoke these cmdlets.

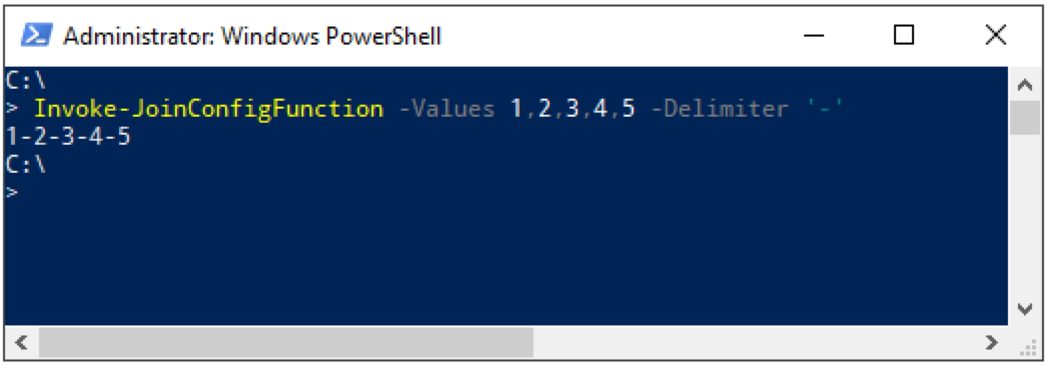
Running tasks or config functions directly allows you to test the results at the command line. You can also integrate the commands into your own PowerShell scripts.

For example, you can directly invoke the `EnsurePath` task by using its full PowerShell syntax:



```
C:\
> Invoke-EnsurePathTask -Exists c:\example\path
[Create] c:\example\path
C:\
>
```

Similarly, you can directly invoke the `Join` config function by using its full PowerShell syntax:



```
C:\
> Invoke-JoinConfigFunction -Values 1,2,3,4,5 -Delimiter '-'
1-2-3-4-5
C:\
>
```

4.1.2. Execution policies

The PowerShell execution policies let you restrict the conditions in which scripts and modules are loaded.

You can set policies for the computer, the user, or for the current session. You can also use a Group Policy to apply them. SIF is digitally signed. This means that the module can be imported and executed in a PowerShell session running under any execution policy, except **Restricted**.

4.1.3. Get help about Sitecore Install Framework

SIF contains information about each task and config function as well as general documentation about the framework:

`about_SitecoreInstallFramework` - general information about the framework.

`about_SitecoreInstallFramework_Extending` - information about extension points that let you customize the framework.

`about_SitecoreInstallFramework_Configurations` - examples of how to use tasks, scripts, and modules if you extend the framework.

There are three ways to view the help documentation:

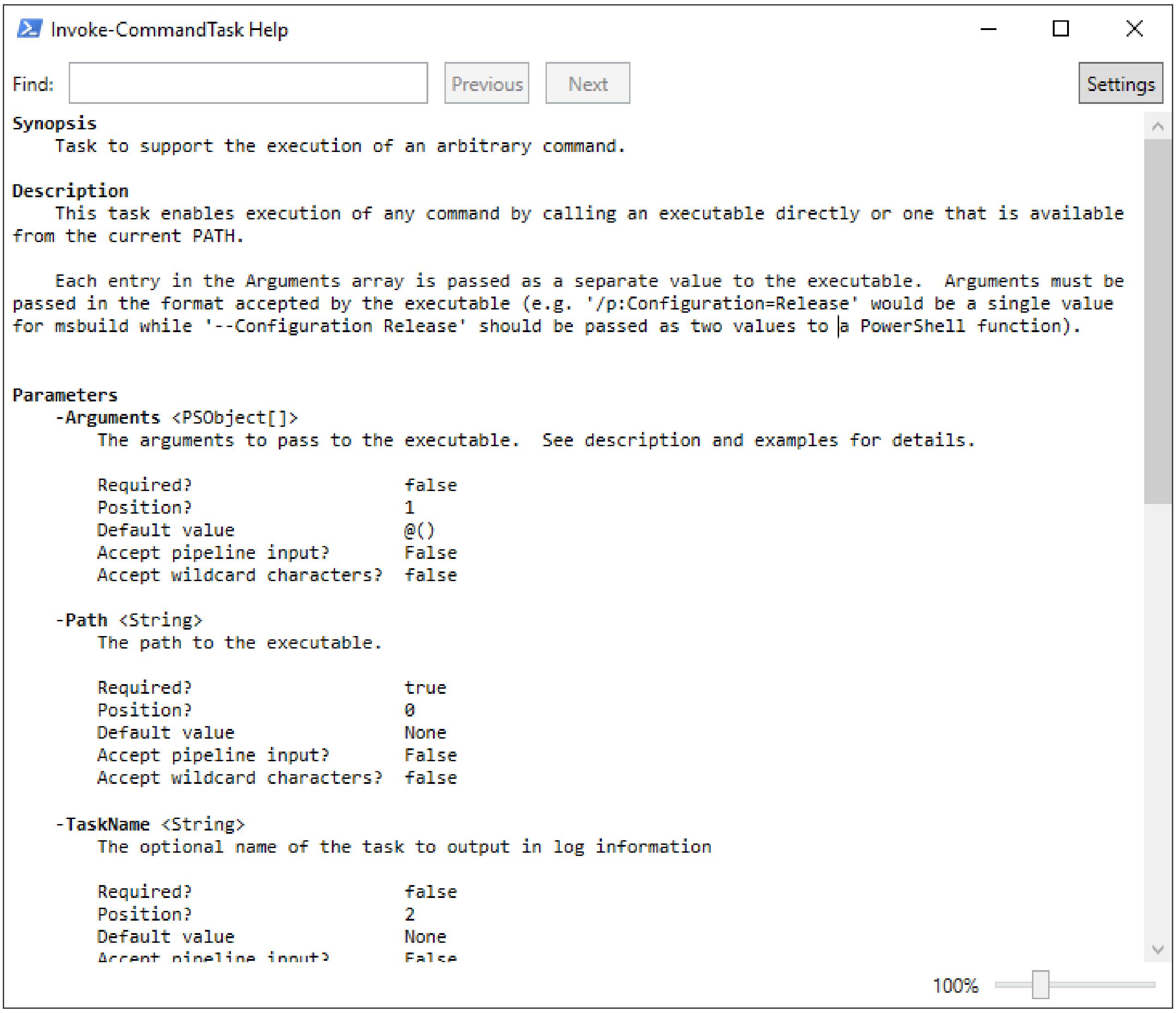
- In the PowerShell window.
- With a markdown reader.
- As HTML pages.

In addition to the help for individual tasks and functions, you can also use the `Get-Command` cmdlet to see a list of all the available tasks and config functions.

View help in the PowerShell window

The PowerShell module has embedded help documentation. To see the help, enter the `Get-Help` cmdlet and the help is displayed in the command line.

If you want to open the help for a specific PowerShell command or function in a separate window, add the `ShowWindow` parameter to the `Get-Help` cmdlet. For example: `Get-Help Invoke-CommandTask -ShowWindow`:

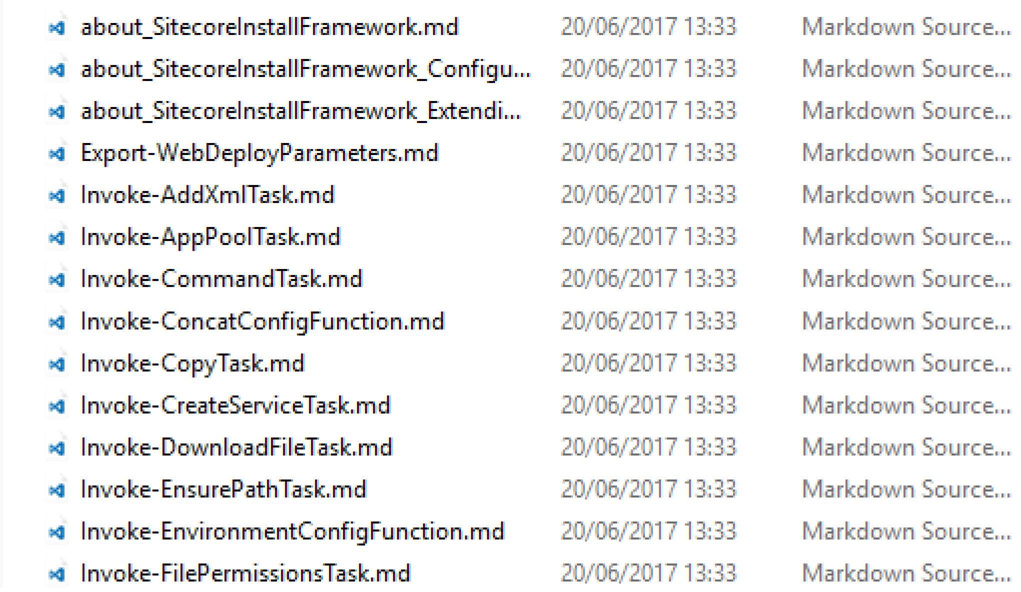


```
Synopsis
    Task to support the execution of an arbitrary command.

Description
    This task enables execution of any command by calling an executable directly or one that is available
from the current PATH.

    Each entry in the Arguments array is passed as a separate value to the executable.  Arguments must be
passed in the format accepted by the executable (e.g. '/p:Configuration=Release' would be a single value
for msbuild while '--Configuration Release' should be passed as two values to a PowerShell function).


Parameters
    -Arguments <PSObject[]>
        The arguments to pass to the executable.  See description and examples for details.

        Required?                    false
        Position?                    1
        Default value                @()
        Accept pipeline input?       False
        Accept wildcard characters?  false

    -Path <String>
        The path to the executable.

        Required?                    true
        Position?                    0
        Default value                None
        Accept pipeline input?       False
        Accept wildcard characters?  false

    -TaskName <String>
        The optional name of the task to output in log information

        Required?                    false
        Position?                    2
        Default value                None
        Accept pipeline input?       False
```

View help with a markdown reader

When you unpack SIF, it contains a folder of markdown documentation that you can read with any text editor or markdown reader.

To read the documentation, in Windows Explorer, go to the `SitecoreInstallFramework\docs` folder and use a text editor or markdown reader to open the relevant topic:



about_SitecoreInstallFramework.md	20/06/2017 13:33	Markdown Source...
about_SitecoreInstallFramework_Configu...	20/06/2017 13:33	Markdown Source...
about_SitecoreInstallFramework_Extendi...	20/06/2017 13:33	Markdown Source...
Export-WebDeployParameters.md	20/06/2017 13:33	Markdown Source...
Invoke-AddXmlTask.md	20/06/2017 13:33	Markdown Source...
Invoke-AppPoolTask.md	20/06/2017 13:33	Markdown Source...
Invoke-CommandTask.md	20/06/2017 13:33	Markdown Source...
Invoke-ConcatConfigFunction.md	20/06/2017 13:33	Markdown Source...
Invoke-CopyTask.md	20/06/2017 13:33	Markdown Source...
Invoke-CreateServiceTask.md	20/06/2017 13:33	Markdown Source...
Invoke-DownloadFileTask.md	20/06/2017 13:33	Markdown Source...
Invoke-EnsurePathTask.md	20/06/2017 13:33	Markdown Source...
Invoke-EnvironmentConfigFunction.md	20/06/2017 13:33	Markdown Source...
Invoke-FilePermissionsTask.md	20/06/2017 13:33	Markdown Source...

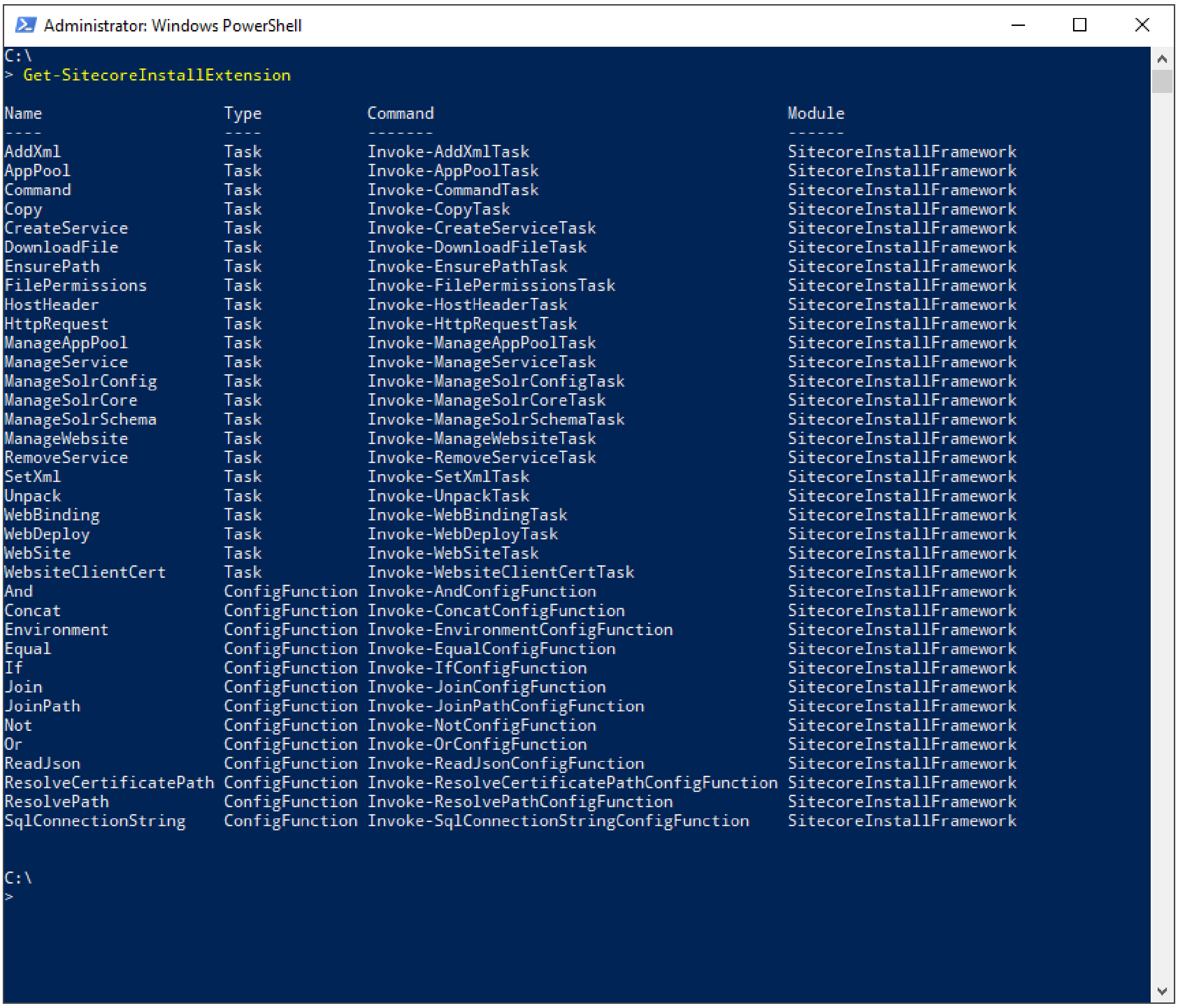
View help as HTML pages

SIF also contains a folder of HTML pages that you can read with any browser. To read the documentation:

1. In Windows Explorer, go to the `SitecoreInstallFramework\docs\html` folder.
2. To open a topic in the default browser, double-click it.

Get a list of the available tasks and config functions

To see the tasks and config functions that are available, run the `Get-SitecoreInstallExtension` cmdlet.



```
C:\
> Get-SitecoreInstallExtension

Name                   Type           Command                                     Module
----                   ----           -------                                     ------
AddXml                 Task           Invoke-AddXmlTask                           SitecoreInstallFramework
AppPool                Task           Invoke-AppPoolTask                          SitecoreInstallFramework
Command                Task           Invoke-CommandTask                          SitecoreInstallFramework
Copy                   Task           Invoke-CopyTask                             SitecoreInstallFramework
CreateService          Task           Invoke-CreateServiceTask                    SitecoreInstallFramework
DownloadFile           Task           Invoke-DownloadFileTask                     SitecoreInstallFramework
EnsurePath             Task           Invoke-EnsurePathTask                       SitecoreInstallFramework
FilePermissions        Task           Invoke-FilePermissionsTask                  SitecoreInstallFramework
HostHeader             Task           Invoke-HostHeaderTask                       SitecoreInstallFramework
HttpRequest            Task           Invoke-HttpRequestTask                      SitecoreInstallFramework
ManageAppPool          Task           Invoke-ManageAppPoolTask                    SitecoreInstallFramework
ManageService          Task           Invoke-ManageServiceTask                    SitecoreInstallFramework
ManageSolrConfig       Task           Invoke-ManageSolrConfigTask                 SitecoreInstallFramework
ManageSolrCore         Task           Invoke-ManageSolrCoreTask                   SitecoreInstallFramework
ManageSolrSchema       Task           Invoke-ManageSolrSchemaTask                 SitecoreInstallFramework
ManageWebsite          Task           Invoke-ManageWebsiteTask                    SitecoreInstallFramework
RemoveService          Task           Invoke-RemoveServiceTask                    SitecoreInstallFramework
SetXml                 Task           Invoke-SetXmlTask                           SitecoreInstallFramework
Unpack                 Task           Invoke-UnpackTask                           SitecoreInstallFramework
WebBinding             Task           Invoke-WebBindingTask                       SitecoreInstallFramework
WebDeploy              Task           Invoke-WebDeployTask                        SitecoreInstallFramework
WebSite                Task           Invoke-WebSiteTask                          SitecoreInstallFramework
WebsiteClientCert      Task           Invoke-WebsiteClientCertTask                SitecoreInstallFramework
And                    ConfigFunction Invoke-AndConfigFunction                    SitecoreInstallFramework
Concat                 ConfigFunction Invoke-ConcatConfigFunction                 SitecoreInstallFramework
Environment            ConfigFunction Invoke-EnvironmentConfigFunction            SitecoreInstallFramework
Equal                  ConfigFunction Invoke-EqualConfigFunction                  SitecoreInstallFramework
If                     ConfigFunction Invoke-IfConfigFunction                     SitecoreInstallFramework
Join                   ConfigFunction Invoke-JoinConfigFunction                   SitecoreInstallFramework
JoinPath               ConfigFunction Invoke-JoinPathConfigFunction               SitecoreInstallFramework
Not                    ConfigFunction Invoke-NotConfigFunction                    SitecoreInstallFramework
Or                     ConfigFunction Invoke-OrConfigFunction                     SitecoreInstallFramework
ReadJson               ConfigFunction Invoke-ReadJsonConfigFunction               SitecoreInstallFramework
ResolveCertificatePath ConfigFunction Invoke-ResolveCertificatePathConfigFunction SitecoreInstallFramework
ResolvePath            ConfigFunction Invoke-ResolvePathConfigFunction            SitecoreInstallFramework
SqlConnectionString    ConfigFunction Invoke-SqlConnectionStringConfigFunction    SitecoreInstallFramework


C:\
>
```

You can filter the list by passing a value to the `Type` parameter.

The following command returns the tasks:

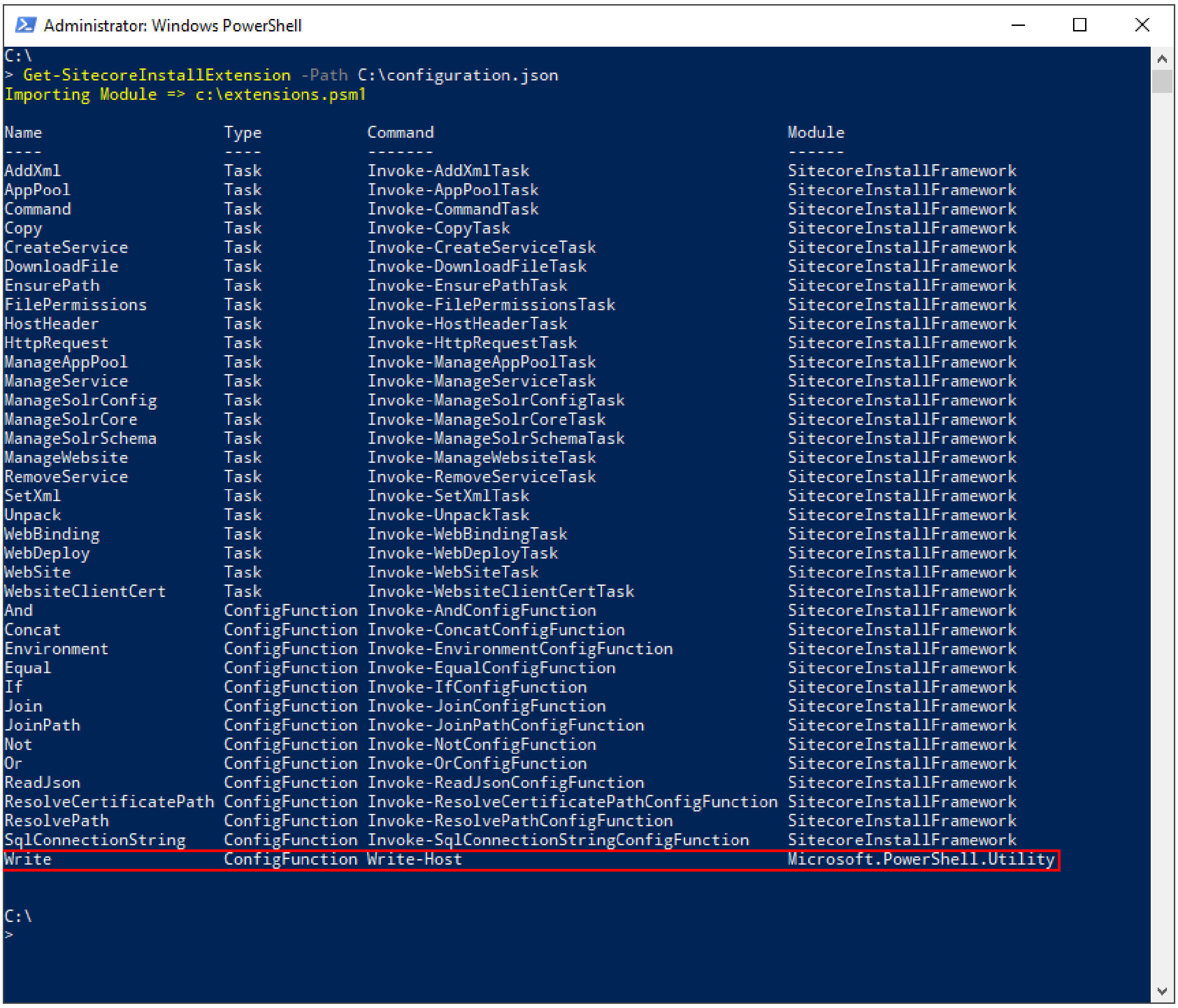
```
Get-SitecoreInstallExtension -Type Task
```

To return the tasks and config functions that are available when a particular configuration is run, you must pass the path to the configuration file to the `Path` parameter.

For example,

```
Get-SitecoreInstallExtension -Path c:\configuration.json
```

displays the default tasks and config functions, as well as the extra registrations brought in by the configuration.



```
C:\
> Get-SitecoreInstallExtension -Path C:\configuration.json
Importing Module => c:\extensions.psm1

Name                   Type           Command                                     Module
----                   ----           -------                                     ------
AddXml                 Task           Invoke-AddXmlTask                           SitecoreInstallFramework
AppPool                Task           Invoke-AppPoolTask                          SitecoreInstallFramework
Command                Task           Invoke-CommandTask                          SitecoreInstallFramework
Copy                   Task           Invoke-CopyTask                             SitecoreInstallFramework
CreateService          Task           Invoke-CreateServiceTask                    SitecoreInstallFramework
DownloadFile           Task           Invoke-DownloadFileTask                     SitecoreInstallFramework
EnsurePath             Task           Invoke-EnsurePathTask                       SitecoreInstallFramework
FilePermissions        Task           Invoke-FilePermissionsTask                  SitecoreInstallFramework
HostHeader             Task           Invoke-HostHeaderTask                       SitecoreInstallFramework
HttpRequest            Task           Invoke-HttpRequestTask                      SitecoreInstallFramework
ManageAppPool          Task           Invoke-ManageAppPoolTask                    SitecoreInstallFramework
ManageService          Task           Invoke-ManageServiceTask                    SitecoreInstallFramework
ManageSolrConfig       Task           Invoke-ManageSolrConfigTask                 SitecoreInstallFramework
ManageSolrCore         Task           Invoke-ManageSolrCoreTask                   SitecoreInstallFramework
ManageSolrSchema       Task           Invoke-ManageSolrSchemaTask                 SitecoreInstallFramework
ManageWebsite          Task           Invoke-ManageWebsiteTask                    SitecoreInstallFramework
RemoveService          Task           Invoke-RemoveServiceTask                    SitecoreInstallFramework
SetXml                 Task           Invoke-SetXmlTask                           SitecoreInstallFramework
Unpack                 Task           Invoke-UnpackTask                           SitecoreInstallFramework
WebBinding             Task           Invoke-WebBindingTask                       SitecoreInstallFramework
WebDeploy              Task           Invoke-WebDeployTask                        SitecoreInstallFramework
WebSite                Task           Invoke-WebSiteTask                          SitecoreInstallFramework
WebsiteClientCert      Task           Invoke-WebsiteClientCertTask                SitecoreInstallFramework
And                    ConfigFunction Invoke-AndConfigFunction                    SitecoreInstallFramework
Concat                 ConfigFunction Invoke-ConcatConfigFunction                 SitecoreInstallFramework
Environment            ConfigFunction Invoke-EnvironmentConfigFunction            SitecoreInstallFramework
Equal                  ConfigFunction Invoke-EqualConfigFunction                  SitecoreInstallFramework
If                     ConfigFunction Invoke-IfConfigFunction                     SitecoreInstallFramework
Join                   ConfigFunction Invoke-JoinConfigFunction                   SitecoreInstallFramework
JoinPath               ConfigFunction Invoke-JoinPathConfigFunction               SitecoreInstallFramework
Not                    ConfigFunction Invoke-NotConfigFunction                    SitecoreInstallFramework
Or                     ConfigFunction Invoke-OrConfigFunction                     SitecoreInstallFramework
ReadJson               ConfigFunction Invoke-ReadJsonConfigFunction               SitecoreInstallFramework
ResolveCertificatePath ConfigFunction Invoke-ResolveCertificatePathConfigFunction SitecoreInstallFramework
ResolvePath            ConfigFunction Invoke-ResolvePathConfigFunction            SitecoreInstallFramework
SqlConnectionString    ConfigFunction Invoke-SqlConnectionStringConfigFunction    SitecoreInstallFramework
Write                  ConfigFunction Write-Host                                  Microsoft.PowerShell.Utility


C:\
>
```

4.2. Troubleshooting

This section describes some of the issues that you can encounter when using SIF and how to resolve them.

4.2.1. Internal server error

After a successful installation, the CM instance cannot be started and the following error is displayed:

```
HTTP Error 500.19 - Internal Server Error
The requested page cannot be accessed because the related configuration data for the page is
invalid.
Error Code 0x8007000d
```

To resolve this error, install the *URL Rewrite* module for IIS:

- Install URL Rewrite 2.1 using the Web Platform Installer.

4.2.2. Error when you invoke the WebDeploy task

When you run `Invoke-WebDeployTask`, you might see the following error:

```
ERROR_SCRIPTDOM_NEEDED_FOR_SQL_PROVIDER
```

This error appears when the web deploy package uses the SQL DACFx framework to install the databases and the provider has not been registered.

To resolve this error, ensure that you have the following components installed:

- SQL Server System CLR Types (2016 version)
- SQL Server Transact-SQL ScriptDom (2016 version)
- SQL Server Data-Tier Application (2016 version)

> **NOTE**
> If you are running the Sitecore installation from a 64-bit computer (x64), you must install both the 32-bit (x86) and 64-bit versions of the SQL Server components.

If you still receive the error after installing the SQL Server components, you must directly register the *ScriptDom* components.

To register the *ScriptDom* component:

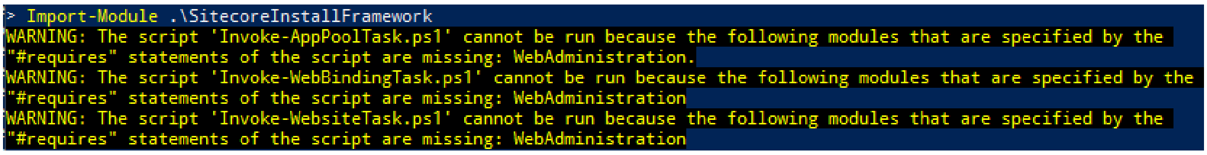
1. In Windows Explorer, go to the `C:\Program Files (x86)\Microsoft SQL Server` folder.
2. The *Microsoft SQL Server* folder contains several subfolders. Click the subfolders (`\90, \100, \110, \120, \130`) and find the `\DAC\bin\Microsoft.SqlServer.TransactSql.ScriptDom.dll` file.
3. Copy or write down the path to the DLL file.
4. Launch PowerShell and go to the `C:\Program Files (x86)\Microsoft SDKs\Windows\v8.1A\bin\NETFX 4.5.1 Tools` folder.
5. Invoke the `gacutil` application and enter the path to the DLL file. For example:

```
gacutil.exe /i C:\Program Files (x86)\Microsoft SDKs\Windows\v8.1A\bin\NETFX 4.5.1 Tools
\120\DAC\bin\Microsoft.SqlServer.TransactSql.ScriptDom.dll
```

4.2.3. Missing modules

Some SIF features require that other modules are loaded as well. You might see warnings that certain tasks cannot be loaded when importing the module.

You can continue to use other features in the module, however, the features that displayed the warnings cannot be executed. For example, if the `WebAdministration` module is not available, you see the following warnings:



```
> Import-Module .\SitecoreInstallFramework
WARNING: The script 'Invoke-AppPoolTask.ps1' cannot be run because the following modules that are specified by the
"#requires" statements of the script are missing: WebAdministration.
WARNING: The script 'Invoke-WebBindingTask.ps1' cannot be run because the following modules that are specified by the
"#requires" statements of the script are missing: WebAdministration
WARNING: The script 'Invoke-WebsiteTask.ps1' cannot be run because the following modules that are specified by the
"#requires" statements of the script are missing: WebAdministration
```

When this happens, the module is loaded but the following tasks are not available:

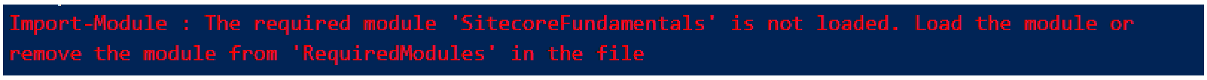
- `Invoke-AppPoolTask`
- `Invoke-WebBindingTask`
- `Invoke-WebsiteTask`

> **NOTE**
>
> To install the *WebAdministration* module, you must first configure IIS on your computer. For more information about how to configure IIS, see the appropriate Sitecore XP Installation Guide.

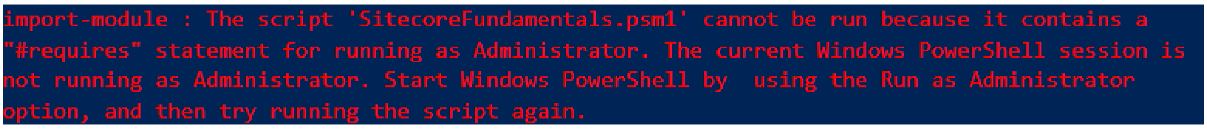
4.2.4. Administrator permissions

To run SIF, you must run PowerShell as an administrator.

> **IMPORTANT**
>
> If you try to use SIF in a non-administration window, you might see one of the following error messages and you will not be able to install Sitecore.



```
Import-Module : The required module 'SitecoreFundamentals' is not loaded. Load the module or
remove the module from 'RequiredModules' in the file
```



```
import-module : The script 'SitecoreFundamentals.psm1' cannot be run because it contains a
"#requires" statement for running as Administrator. The current Windows PowerShell session is
not running as Administrator. Start Windows PowerShell by  using the Run as Administrator
option, and then try running the script again.
```

4.2.5. Sitecore installation failed while using Skype

If you are using Skype or another communication tool when you are installing Sitecore, the xConnect installation might fail. This happens because Skype and Sitecore xConnect both use port 443. If this happens, you must change your Skype configuration.

Another way to solve this issue is to update the default HTTP port, and if needed the HTTPS port.

Updating the default HTTP port

Updating the default HTTP port can be applied to the following Sitecore configurations:

- All Sitecore configurations
- All xConnect configurations

To update the default HTTP port:

1. In a text editor, open the relevant configuration file, for example: `sitecore-xp1.json`.
2. In the `CreateWebsite` task, add a `Port` property to the `Params` collection with the new value. For example: `"CreateWebsite": {`