Sitecore Publishing Service Installation and Configuration Guide

Sitecore XP 8.2

How to install and configure the Sitecore Publishing Service

sitecore®
Own the experience™

Table of Contents

Chapter 1 Introduction 4
1.1 About the Publishing Service Module 5
1.1.1 Publishing Service Concepts 6
Chapter 2 Requirements 7
2.1 Prerequisites 8
2.1.1 Sitecore Publishing Service Requirements 8
2.1.2 Sitecore Publishing Module Requirements 8
Chapter 3 Installing the Sitecore Publishing Service 9
3.1 Manual Installation 10
3.2 Scripted Installation 12
Chapter 4 Configuring the Sitecore Publishing Service 14
4.1 Configuration Sources 15
4.2 Adding Configuration Values 16
4.3 Overriding Configuration Values 17
4.4 Referencing Configuration Values 18
4.5 Configuring Options 19
4.5.1 DatabaseConnectionOptions 19
4.5.2 PublishHostOptions 19
4.5.3 PublishJobHandlerOptions 20
4.5.4 PromoterOptions 21
4.6 Database Configuration 23
4.6.1 Source Connection 23
4.6.2 Target Connection 24
4.6.3 Custom Connection 25
4.6.4 Service Connection 25
4.6.5 SQL Connection Types 25
4.6.6 PublishTask 26
4.6.7 PublishJobCleanUpTask 26
4.6.8 The Task Scheduler 26
4.6.9 Defining a Task 26
4.6.10 Defining a Trigger 27
4.7 Schema Configuration 28
4.7.1 The Deployment Map 29
4.7.2 Schemas 29
4.7.3 Schema Update tool 29
4.7.4 Validating Schemas 31
4.8 Logging Configuration 32
4.9 Troubleshooting 33
Chapter 5 Installing and Configuring the Sitecore Publishing Module 34
5.1 Security 35
5.2 Installing the Sitecore Publishing Module 36
5.3 Post-installation Configuration 37
5.3.1 Service Endpoints 37
5.3.2 Content Delivery Servers 38
5.4 Recovery 39
5.5 Publisher Operations Service 40
5.6 Security 41
5.6.1 Granting Permission to Perform a Full Republish 41
5.7 Operation Emitter 42
5.8 Events 43
Chapter 6 Publishing with the Sitecore Publishing Module 44
6.1 The Sitecore Publishing Module 45
6.1.1 The Publishing Dashboard 45

6.1.2 Publishing Viewer 46
6.2 Publishing an Item 47
6.3 Publishing a Website 49
6.4 Publish all Items 51
6.5 The Sitecore Commerce Server Connect Publishing extension package 54

Chapter 1

Introduction

This document describes how to install and configure the Sitecore Publishing Service. It also describes how to install and work with the Sitecore Publishing module.

The document contains the following chapters:

- **Chapter 1 Introduction**
  An introduction to the Sitecore Publishing Service module.
- **Chapter 2 Requirements**
  An outline for the installation requirements for the Sitecore Publishing Service.
- **Chapter 3 Installing the Sitecore Publishing Service**
  Step-by-step instructions for installing the Sitecore Publishing Service manually or with a script.
- **Chapter 4 Configuring the Sitecore Publishing Service**
  Step-by-step instructions for configuring the Sitecore Publishing Service.
- **Chapter 5 Installing and Configuring the Sitecore Publishing Module**
  Installing the module as a Sitecore package and instructions for post-installation configuration.
- **Chapter 6 Publishing with the Sitecore Publishing Module**
  How to use the module to publish Sitecore.

1.1 About the Publishing Service Module

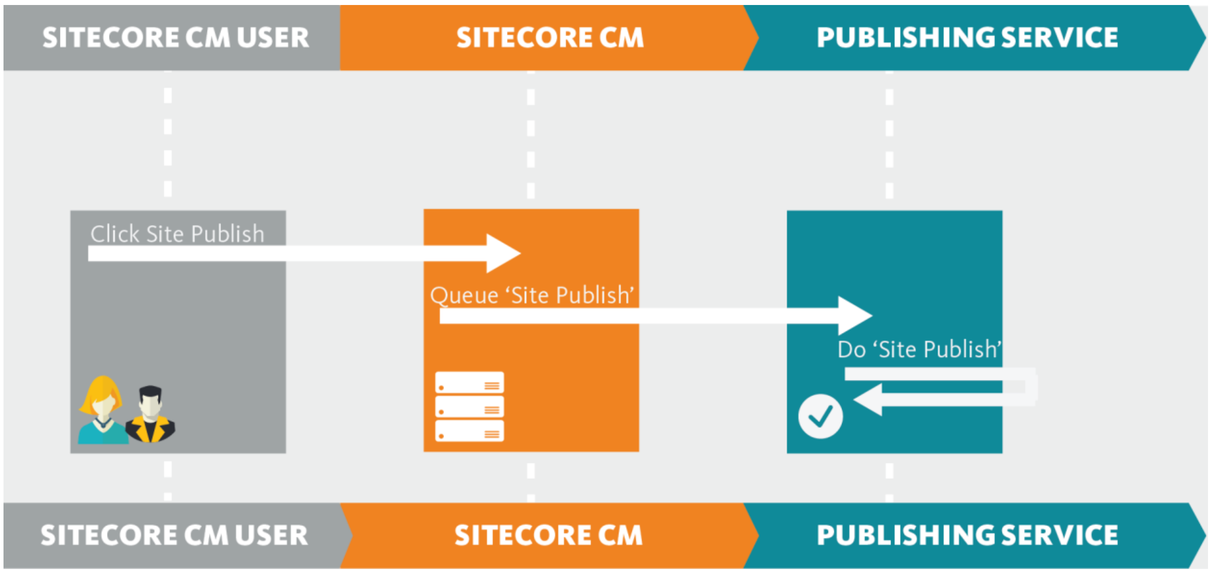
The Publishing Service module is an optional replacement for the existing Sitecore publishing methods. This module increases publishing throughput, reduces the amount of time spent publishing large volumes of items, and offers greater data consistency and reliability. The module also improves the user experience and provides better visual feedback to the user on the state of the publishing system.

The Publishing Service does not use any of the features, pipelines, and settings in the current publishing system. It is an entirely new way of publishing Sitecore items and media.

The Publishing Service runs a separate process to the Sitecore CM instance.

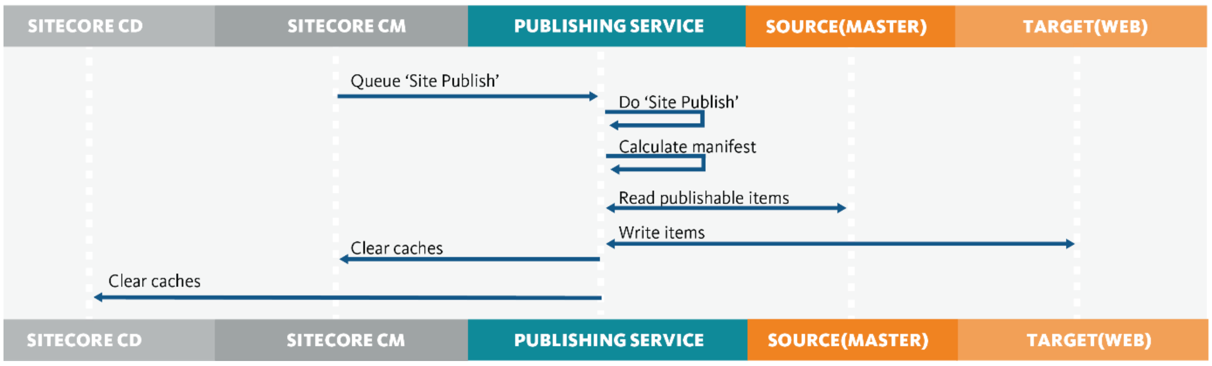
Installation involves:

1. Installation and configuration of the Publishing Service.
2. Installation of the integration module package on your Sitecore instance. The integration module ensures that every publishing action, such as triggering a site publish, is handed on to the publishing service.



When you have installed the Publishing Service, it manages the whole publishing process:

1. It queues and executes publishing jobs.
2. It connects to the Source and Target (SQL) databases directly – reading and writing items in bulk.
3. It issues events, such as cache clearing events, on Content Delivery servers.
4. It reports status information back to UI features, such as the Publishing Dashboard application.



1.1.1 Publishing Service Concepts

The Publishing Service introduces some new concepts for understanding how the different stages of the publishing work are handled:

- **Publishing jobs**

  Previously, when a user chose to publish something, the publishing dialog remained open for the duration of the publish process. This was awkward if the user needed to reboot or if their session ended because they could not see the status of the publishing job.

  The publishing service places all publishing jobs in a queue. When you request a publishing job of any kind, it is queued and then processed as soon as possible. You can see all the active, queued, and completed jobs in the **Publishing Dashboard** application.

- **Manifests**

  This is the collective name for all the tasks that a publishing job performs. The Publishing Service calculates the manifest at the beginning of the publishing job, before it moves any data.

  The Publishing Service looks at the items to see if there any restrictions that would prevent them from being published:

  - Valid dates/workflow states, and so on.
  - Evaluating whether or not the item might need to be deleted.
  - If it is a media file.
  - If extra data needs to be moved along with the item.

  Valid items are added to the manifest as a 'Manifest Step'. Each publishing target gets its own manifest. A publishing job can therefore consist of one or more manifests. The completed manifest is a list of all the items that will be used in the next stage of the process - the Promotion.

- **Promotion**

  This term describes the process of moving the items and data from the source, most often the *Master* database, to one or more publishing targets, such as the *Web* database.

  The Publishing Service creates a manifest and then moves it to one or more publishing targets.

- **Manifest results**

  A list of the changes that were made during the promotion of the manifest. This includes things like item name changes and template updates.

  At the end of the publishing job, the results are passed to the `publishEndResultBatch` pipeline in Sitecore. Developers can hook into this pipeline to work with these results and update any third-party systems or features that may need to know about the changes to items.

  If there is no work to do, that is, if an item is unchanged even though it was in the manifest, a manifest result is not generated.

Chapter 2

Requirements

The prerequisites for the Sitecore Publishing Service and the Publishing module

This chapter contains the following sections:

- Prerequisites

2.1 Prerequisites

2.1.1 Sitecore Publishing Service Requirements

The Sitecore Publishing Service comes in a single ZIP archive that you can be execute directly after you have unpacked it. However, you should run the service under IIS because this gives greater configurability of host addresses, port binding, and so on.

- Sitecore Publishing Service 1.1.0.zip

The prerequisites for the Sitecore Publishing Service 1.1.0 Initial release are:

- Sitecore XP 8.2 (8.2 rev.160729)
- .NET Core

  To enable the service to run under IIS, you must install the Windows (Server Hosting) package from `https://www.microsoft.com/net/download#core`

2.1.2 Sitecore Publishing Module Requirements

The Sitecore Publishing module is distributed as a Sitecore package. This package contains the UI for the publishing service.

- Sitecore Publishing Module 1.8.0.zip

The prerequisites for the Sitecore Publishing Module 1.8.0 are:

- Sitecore XP 8.2 (8.2 rev.160729)
- The Sitecore Publishing Service 1.1.0 Initial release

Chapter 3

Installing the Sitecore Publishing Service

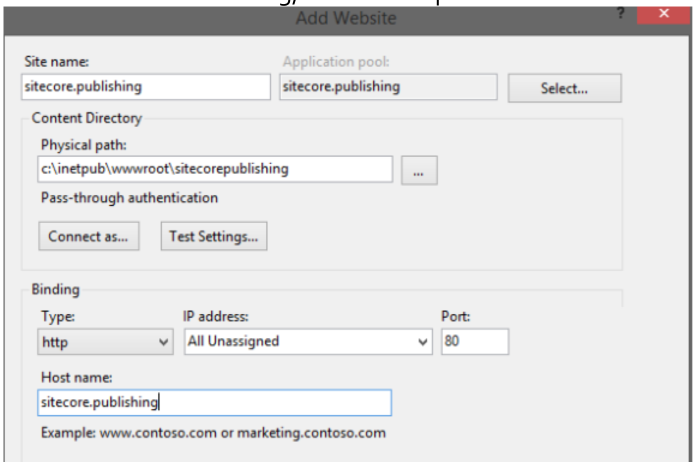
You can install the Sitecore Publishing Service manually or by using the utility scripts that come with the package. This chapter describes:

- Manual Installation
- Scripted Installation

3.1 Manual Installation

To install the Sitecore Publishing Service for Sitecore 8.2 manually:

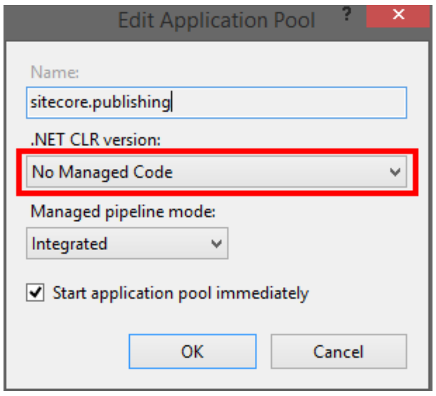
1. Download the Sitecore Publishing Service package from the Sitecore Developer Portal.
2. Extract the contents of the archive to a folder of your choice. For example: `C:\inetpub\wwwroot\sitecorepublishing`
3. In IIS, create a new site pointing to the folder.
4. Start the IIS Manager and in the Connections panel, expand *Sites*. Right-click *Sites* and then click **Add Website**.
5. In the **Add Website** dialog, fill in the required fields.



Site name: sitecore.publishing
Application pool: sitecore.publishing
Content Directory
Physical path: c:\inetpub\wwwroot\sitecorepublishing
Pass-through authentication
Binding
Type: http
IP address: All Unassigned
Port: 80
Host name: sitecore.publishing
Example: www.contoso.com or marketing.contoso.com

Note
If you add a custom host name, you must update your hosts file (`C:\Windows\System32\drivers\etc\`).

1. In the IIS Manager, right-click the application pool for the website that you created, click **Basic Settings**, and in the **.NET CLR version** field, select *No Managed Code*.



Name: sitecore.publishing
.NET CLR version: No Managed Code
Managed pipeline mode: Integrated
☑ Start application pool immediately

Note
The Application Pool user must have *Read*, *Execute*, and *Write* permissions to the site's physical path.

2. Configure the connection strings for the service along with any additional configuration values.
3. Run the schema tool from the extracted folder to upgrade the database schema. For example: `C:\inetpub\wwwroot\sitecorepublishing\schematool.exe upgrade`
4. To access your website, enter `http://<sitename>/api/publishing/operations/status` in your browser.

If you receive a value of `{ "Status" : 0 }`, the application is installed correctly. If you receive any other value, check the application logs for further details.

3.2 Scripted Installation

The Sitecore Publishing Service provides utility scripts to help you install.

Note
All commands must be run as an administrator in PowerShell.

To perform a scripted installation:

1. Extract the contents of the archive to a folder of your choice. For example:
   `c:\inetpub\wwwroot\publishingservice`

   This will be the location where IIS points to the service.

2. Configure a connection string that supports Multiple Active Result Sets, to enable the execution of multiple batches on a single connection.

   Note
   If the connection string does not support Multiple Active Result Sets (), it will be changed when you invoke the configuration command.

   If the provided connection string does not already exist, it will be added to the configuration when you invoke the configuration command. Otherwise, it replaces the connection string with the same key.

3. Run the following command in the location where you extracted the service:

   ```
   .\configure.ps1 -connectionstring -key 'master' -value
   'connectionstring'
   ```

4. For additional configuration settings, you must provide the full key to set or override an existing value.

   The following table lists the commands that you use to set the configuration file, the environment, and the instance:

Set	Command	Description
Configuration file	`-configFileName <path>`	By default, the settings are added to `sc.custom.json` Add the path to change the location.
Environment	`-environment <name>`	Configuration can be applied to specific start-up environments. To apply changes to a specific environment, pass the `–environment` command along with the name of the environment.
Log level	`.\configure.ps1 -key Sitecore:Publishing:Log Level -value Debug`	To change the log level of the service.

   Note
   If you have many settings that you want to apply at the same time and need to configure multiple values, provide a hash table that contains the changes.

   ```
   $changes = @{ Sitecore = @{ Publishing = @{ LogLevel = 'Debug'; TempFolder =
   '../temp2' } } } .\configure.ps1 -configuration $changes
   ```

5. Once you have configured the instance, you can run the install script to set it up in IIS:

   ```
   .\install.ps1 -sitename 'name' -appPoolName 'name' -port 80
   ```

The default values for each parameter are:

- `Sitename: sitecore.publishing`
- `appPoolName: sitecore.publishing`
- Port: 80

Note
If you want to re-run the script and remove an existing instance, add the `–Force` switch. For example: `.\install.ps1 -sitename 'sitecorepublising' -appPoolName 'sitecorepublishing' -port 80 -Force`

Chapter 4

Configuring the Sitecore Publishing Service

The Sitecore Publishing Service supports custom configurations.

This chapter contains the following sections:

- Configuration Sources
- Adding Configuration Values
- Overriding Configuration Values
- Referencing Configuration Values
- Configuring Options
- Database Configuration
- Task Configuration
- Schema Configuration
- Logging Configuration
- Troubleshooting

4.1 Configuration Sources

During application start up, the configuration sources are loaded in the following order:

- Environment variables
- Default Sitecore configuration:

  `<installationPath>\config\sitecore`
- Global configuration:

  `<installationPath>\config\global`
- Environment specific configuration:

  `<installationPath>\config\<environment>`

During each stage, the values that were previously loaded can be overwritten.

Note
You must restart the application after applying the configuration changes.

The Sitecore folder contains all the default configuration files provided by Sitecore. Review these files to learn what can be configured:

- Global: custom/module configuration files that extend or overwrite the Sitecore defaults.

  This is the location where developers should add their instance specific configuration files. For example, where a configuration file contains details of custom extensions and connection strings.
- <Environment>: you can add custom folders to support different environments.

  For example, if a *Development* folder exists and the `ASPNETCORE_ENVIRONMENT` variable is set to `'Development'`, the configuration files in this folder are loaded. The default environment `'Production'` will not load these files.
- Configuration File Naming: when you create a configuration file, it must be prefixed with `sc.` in order to be loaded. When you create a configuration file, it must be an `.xml`, `.json`, or `.ini` file in order to be loaded. All other files are ignored.

Important
Do not modify the files in the Sitecore folder. They are automatically overwritten during the upgrade process.

4.2 Adding Configuration Values

To add a configuration value, declare the value at the path you require.

For example, the default configuration contains an element called `<Sitecore><Publishing><ConnectionStrings>` that looks like this:

```
<Settings>
  <Sitecore>
    <Publishing>
      <ConnectionStrings>
        <!-- The Service connection is registered to map to the same connection string
as the master database by default. -->
        <Service>${Sitecore:Publishing:ConnectionStrings:Master}</Service>
      </ConnectionStrings>
      ...
    </Publishing>
  </Sitecore>
</Settings>
```

To add a new value, save the following in: `<installationPath>\config\global\sc.connectionstrings.xml`

```
<Settings>
    <Publishing>
      <ConnectionStrings>
        <Master>user id=sa;password=password;data
source=.\SQLEXPRESS;database=sitecore.Master;MultipleActiveResultSets=True;</Master>
      </ConnectionStrings>
    </Publishing>
</Settings>
```

The connection string is now defined at: `Sitecore:Publishing:ConnectionStrings:Master`

4.3 Overriding Configuration Values

To override a configuration value, you must re-declare the value.

For example, if the default configuration contained an element called `<Sitecore><Publishing><LogLevel>` that looks like this:

```
<Sitecore>
    <Publishing>
      <!-- The default Loglevel for the instance. -->
      <LogLevel>Information</LogLevel>
      ...
```

To set the log level to Debug only when running in Development, save the following as: `<installationPath>\config\development\sc.logging.json`

```
{
  "Sitecore": {
    "Publishing": {
      "LogLevel": "Debug"
    }
  }
}
```

Now when the application starts in a development environment, you get some additional logging information.

4.4 Referencing Configuration Values

If you have a configuration value that needs to be referenced elsewhere, you can use the syntax

`${ a:b:c }` to reference it.

This enables you to overwrite the value in a single location, and at the same time supports its use in multiple configuration files.

For example, the default configuration file contains a connection string entry for the service that is configured to point to the *Master* connection string by default:

```
<Settings>
  <Sitecore>
    <Publishing>
      <ConnectionStrings>
        <!-- The Service connection is registered to map to the same connection string
as the master database by default. -->
        <Service>${Sitecore:Publishing:ConnectionStrings:Master}</Service>
      </ConnectionStrings>
    </Publishing>
  </Sitecore>
</Settings>
```

By adding a configuration file that contains a value for `Sitecore:Publishing:ConnectionStrings:Master`, this connection string is used for both the *Master* database and the *Service* database.

Alternatively, the value at `<Sitecore><Publishing><ConnectionStrings><Service>` could be overwritten in another configuration file that provides an explicit connection string that should be used.

4.5 Configuring Options

Much of the configuration consists of registering object types, so that the service can replace default implementations with custom alternatives. Many of the object types that are registered support an optional configuration section called `Options`. When an object type supports `Options`, you can provide additional configuration values to change the behavior of the application.

4.5.1 DatabaseConnectionOptions

The `DatabaseConnectionOptions` class describes a connection to a data source and is used by: `Sitecore.Framework.Publishing.Sql.DatabaseConnection`:

```
namespace Sitecore.Framework.Publishing.Sql
{
    public class DatabaseConnectionOptions : ConnectionOptions
    {
        public string ConnectionString { get; set; }

        public int CommandTimeout { get; set; } = 180;
    }
}
```

This class also extends the `ConnectionOptions` class.

```
namespace Sitecore.Framework.Publishing.Data
{
    public class ConnectionOptions
    {
        public string ConnectionName { get; set; }

        public Guid Id { get; set; }
    }
}
```

The following configuration example provides an alternate value for the `CommandTimeout` setting of the Service connection:

```
<Sitecore>
    <Publishing>
      <Services>
        <DataStoreFactory>
          <Options>
            <Service>
              <Options>
                <CommandTimeout>30</CommandTimeout>
              </Options>
            </Service>
          </Options>
        </DataStoreFactory>
      </Services>
    </Publishing>
  </Sitecore>
</Settings>
```

4.5.2 PublishHostOptions

The `PublishHostOptions` class describes the main configuration options for the application and is used to get and set high-level options – LogLevel and Services – as well as the collection of services that must be registered. Services are all the types that are registered under the Services configuration option are added to the collection of services registered during start up.

`PublishHostOptions` is used by `Sitecore.Framework.Publishing.Host`:

```
namespace Sitecore.Framework.Publishing.Host
{
    public class PublishHostOptions
    {
        public List<ConfigurationServiceType> Services { get; set; } = new
List<ConfigurationServiceType>();

        public LogLevel LogLevel { get; set; } = LogLevel.Information;
    }
}
```

The following configuration example provides alternative values for the `LogLevel` setting and adds a custom service to the collection of services.

```
<Settings>
  <Sitecore>
    <Publishing>
      <LogLevel>Debug</LogLevel>
      <Services>
        <MyCustomService>
          <Type>MyCustom.Service, MyCustomType</Type>
          <As>MyCustom.IService, MyCustomType.Abstractions</As>
        </MyCustomService>
      </Services>
    </Publishing>
  </Sitecore>
</Settings>
```

4.5.3 PublishJobHandlerOptions

The `PublishJobHandlerOptions` class lets you configure various aspects of the Publish Job handler implementations to optimize performance and is used by:

`Sitecore.Framework.Publishing.PublishJobQueue.Handlers.IncrementalPublishHandler`

```
namespace Sitecore.Framework.Publishing.PublishJobQueue
{
    public class PublishJobHandlerOptions
    {
        public int RelatedItemBatchSize { get; set; } = 2000;

        public int ManifestBuilderBatchSize { get; set; } = 5000;

        public int UnpublishedOperationsLoadingBatchSize { get; set; } = 2000;

      public int DeletedItemsBatchSize { get; set; } = 2000;

        public int MediaBatchSize { get; set; } = 2000;


        public int IndexReadBatchSize { get; set; } = 500;

        public int IndexWriteBatchSize { get; set; } = 500;

        public int TargetOperationsBatchSize { get; set; } = 2000;
        public int SourceTreeReaderBatchSize { get; set; } = 2000;

        public bool TransactionalPromote { get; set; } = true;

        public bool ParallelPromote { get; set; } = true;

        public bool ContentTesting { get; set; } = true;
    }
}
```

The following configuration example provides alternative values for the default configuration:

```
<Settings>
  <Sitecore>
    <Publishing>
      <Services>
        <IncrementalPublishHandler>
          <Options>
            <ContentTesting>True</ContentTesting>
            <IndexReadBatchSize>500</IndexReadBatchSize>
            <IndexWriteBatchSize>500</IndexWriteBatchSize>
            <ManifestBuilderBatchSize>5000</ManifestBuilderBatchSize>
            <ParallelPromote>False</ParallelPromote>
            <RelatedItemBatchSize>2000</RelatedItemBatchSize>
            <SourceTreeReaderBatchSize>2000</SourceTreeReaderBatchSize>
            <TargetOperationsBatchSize>2000</TargetOperationsBatchSize>
            <TransactionalPromote>True</TransactionalPromote>

<UnpublishedOperationsLoadingBatchSize>2000</UnpublishedOperationsLoadingBatchS
ize>
          </Options>
        </IncrementalPublishHandler>
      </Services>
    </Publishing>
  </Sitecore>
</Settings>
```

4.5.4 PromoterOptions

The `PromoterOptions` class lets you configure various aspects of the Publish Job promoter implementations to optimize performance and is used by:

```
        Sitecore.Framework.Publishing.DataPromotion.DefaultItemCloneManifestPromoter
        Sitecore.Framework.Publishing.DataPromotion.DefaultItemManifestPromoter
        Sitecore.Framework.Publishing.DataPromotionDefaultMediaManifestPromoter
```

```
namespace Sitecore.Framework.Publishing.Abstractions.DataPromotion
{
    public class PromoterOptions
    {
        public int BatchSize { get; set; } = 500;
    }
}
```

The following configuration example provides an alternative `BatchSize` class for the registered `ItemCloneManifestPromoter`:

```
<Settings>
  <Sitecore>
    <Publishing>
      <Services>
        <ItemCloneManifestPromoter>
          <Options>
            <BatchSize>1000</BatchSize>
          </Options>
        </ItemCloneManifestPromoter>
      </Services>
    </Publishing>
  </Sitecore>
</Settings>
```

4.6 Database Configuration

Connection strings are configured under `<Sitecore><Publishing><ConnectionStrings>`.

Sitecore expects three default connection strings to be configured - *core*, *web*, and *master* and these are referenced elsewhere in the configuration.

```
<Settings>
    <Publishing>
      <ConnectionStrings>
        <Master>user id=sa;password=password;data
source=.\SQLEXPRESS;database=sitecore.Master;MultipleActiveResultSets=True;</Ma
ster>
        <Web>user id=sa;password=password;data
source=.\SQLEXPRESS;database=sitecore.Web;MultipleActiveResultSets=True;</Web>
        <Core>user id=sa;password=password;data
source=.\SQLEXPRESS;database=sitecore.Core;MultipleActiveResultSets=True;</Core
>
      </ConnectionStrings>
    </Publishing>
</Settings>
```

Currently, SQL connection strings require that they support Multiple Active Result Sets (MARS). When configuring a connection string, you must set `MultipleActiveResultSets` to `true`.

Use the following format or similar for connection strings:

```
Data Source = .\\SQLEXPRESS; Initial Catalog = publishing_master;
Integrated Security=True; MultipleActiveResultSets=True;
```

For more information, see https://www.connectionstrings.com/sqlconnection/

The connection factory configures the default targets, sources, and connection for the service database that is used throughout the lifetime of the service. You can also configure additional custom databases. Connections are split based on their type (Target, Source, Custom) with the Service connection explicitly defined.

Connection	Configuration entry
Source connection	Sitecore:Publishing:Services: DataStoreFactory:Options:Sources
Target connection	Sitecore:Publishing:Services: DataStoreFactory:Options:Targets
Custom connection	Sitecore:Publishing:Services: DataStoreFactory:Options:Custom
Service connection	Sitecore:Publishing:Services: DataStoreFactory:Options:Service

4.6.1 Source Connection

A source connection is a data connection to a Sitecore Content Database; usually the *Master* database where you do the content authoring.

Add the source connection to:

`<Sitecore><Publishing><Services><DataStoreFactory><Options><Sources>`

For example, the new source code `master2`:

```
 <Settings>
  <Sitecore>
    <Publishing>
      <Services>
        <DataStoreFactory>
```

```
          <Options>
            <Sources>
              <Master2>
                <Type>Sitecore.Framework.Publishing.Sql.DatabaseConnection,
Sitecore.Framework.Publishing.Sql</Type>

<As>Sitecore.Framework.Publishing.Connection.IDatabaseConnection,
Sitecore.Framework.Publishing.Abstractions</As>
                <Lifetime>Transient</Lifetime>
                <Options>
                  <CommandTimeout>180</CommandTimeout>

<ConnectionName>master2</ConnectionName><DbConnectionType>SqlConnection.Default
</DbConnectionType>
<ConnectionSting>${Sitecore:Publishing:ConnectionStrings:Master2}</ConnectionSt
ing>
                </Options>
              </Master2>
            </Sources>
```

The `master2` key under `<Sources>` identifies the connection in the system. The configuration registers the connection so that it is created using the `Sql.DatabaseConnection` object type and handled in the system as an `IDatabaseConnection`. The connection uses `master2` as its `<ConnectionName>` and also references a connection string that is defined somewhere else in the configuration.

4.6.2 Target Connection

A target connection is a data connection to a Sitecore content database that can be published to a publishing target, for example the *Web* database. The configuration of publishing targets is the same as a source, but for target connections, you must also define the ID in the connection options. You must also ensure that the name in the connection is the same as the name of the publishing target in Sitecore:

`<Sitecore><Publishing><Services><DataStoreFactory><Options><Targets><Intern et>`

- `Id`: the item ID of the publishing target in Sitecore.
- `ConnectionName`: the value in the **Target database** field in Sitecore.

Configuration entry:

`<Sitecore><Publishing><Services><DataStoreFactory><Options><Targets>`

For example:

```
<Settings>
  <Sitecore>
    <Publishing>
      <Services>
        <DataStoreFactory>
          <Options>
            <Targets>
               <Options>
                  <CommandTimeout>360</CommandTimeout>
                  <ConnectionName>web</ConnectionName>
                  <DbConnectionType>SqlConnection.Default</DbConnectionType>
<ConnectionString>${Sitecore:Publishing:ConnectionStrings:Web}</ConnectionStrin
g>
                  <!-- The id of the target item definition in Sitecore -->
                  <Id>8E080626-DDC3-4EF4-A1D1-F0BE4A200254</Id>
               </Options>
```

4.6.3 Custom Connection

You configure custom connections in exactly the same way as target connections. You should configure custom connections under:

`<Sitecore><Publishing><Services><DataStoreFactory><Options><Custom>`

4.6.4 Service Connection

This is the data connection to the Publishing Database.

Only one service connection can be configured under:

`<Sitecore><Publishing><Services><DataStoreFactory><Options><Service>`

4.6.5 SQL Connection Types

When you register a connection that handles communication with a SQL server instance, you can change how the connection is handled during error and retry scenarios.

The `DbConnectionTypeRegistry` section in the configuration file contains multiple registered factories for creating `DbConnections`. Each connection uses a different strategy to handle errors that may occur when making requests to a SQL instance.

`SqlConnection.Default` – The default connection implementation. Retries do not occur.

`SqlConnection.ReliableWithExponentialBackoff` – This configuration does not retry when an error occurs during automatic failover. There are a number of retries with an increased delay between each attempt.

`SqlConnection.ReliableWithFixedBackOff` – This configuration retries when an error occurs during automatic failover. There are a number of retries with a fixed delay between each attempt.

To configure the connection type used by a connection, set the `DbConnectionType` options to the name of the connection type.

For example, to set the service connection to use the `SqlConnection.ReliableWithFixedBackOff` connection type, provide the following override:

```
<Settings>
  <Sitecore>
    <Publishing>
      <Services>
        <DataStoreFactory>
          <Options>
            <Service>
              <Options>

<DbConnectionType>SqlConnection.ReliableWithFixedBackOff</DbConnectionType>
              </Options>
            </Service>
          </Options>
        </DataStoreFactory>
      </Services>
    </Publishing>
  </Sitecore>
</Settings>
```

Task Configuration

The Publishing Service lets you configure independent tasks in the system. It contains two task definitions by default:

`PublishTask` – The task that handles requests to publish items from sources to targets.

PublishJobCleanUpTask – The tasks that handles the periodic clean-up of historical publishing jobs.

The default task configuration is contained in the config\sitecore\sc.publishing.tasks.xml configuration file

4.6.6 PublishTask

The PublishTask task definition is configured with two triggers:

Interval: The interval trigger runs every few seconds to check for publishing jobs that were requested while the previous publishing job was running.

Event: The event based trigger causes a publishing job to start immediately after it is requested. If a publishing job is already being processed, the job is delayed until the next interval.

4.6.7 PublishJobCleanUpTask

The PublishJobCleanUpTask task definition removes old publishing jobs from the database to prevent the buildup of data over time. It has a single trigger raising on an infrequent schedule to remove jobs over a week old.

You can configure the task by changing its options:

JobAge: The time that must have passed since a publishing job's Stopped time. The default value is seven days. If a publishing job's Stopped time is older than the JobAge, it is eligible for clean-up.

BatchSize: This is the number of items in the batch that can be deleted together. The default value is 50.

4.6.8 The Task Scheduler

The task scheduler is a service that manages the creation of tasks at start up as well as enabling the addition and execution of tasks at runtime.

4.6.9 Defining a Task

Once a task has been implemented, it must be added to the configuration so that it can be created at startup:

```
<Settings>
  <Sitecore>
    <Publishing>
      <Services>
        <Scheduler>
          <Options>
            <Tasks>
              <CustomTask>
                <TaskDefinition Type="Custom.Task, Custom"
BindOptions="property">
                  <Options>
                    <Id>Custom Task</Id>
                    <Categories>
                      <Custom>Custom</Custom>
                      <Other>Other</Other>
                    </Categories>
                  </Options>
                </TaskDefinition>
              </CustomTask>
            </Tasks>
          </Options>
        </Scheduler>
      </Services>
    </Publishing>
  </Sitecore>
</Settings>
```

A task can expose additional parameters, such as ID and Categories, to help identify the task when the system is running.

4.6.10 Defining a Trigger

A task will not run if there are no triggers associated with it. Each trigger is a unique instance, so you can register multiple triggers of the same type. For example, two interval triggers could be registered that trigger a task at different polling intervals:

```
<Settings>
  <Sitecore>
    <Publishing>
      <Services>
        <Scheduler>
          <Options>
            <Tasks>
              <CustomTask>
                <TaskDefinition Type="Custom.Task, Custom"
BindOptions="property">
                  <Options>
                    <Id>Custom Task</Id>
                    <Categories>
                      <Custom>Custom</Custom>
                      <Other>Other</Other>
                    </Categories>
                  </Options>
                </TaskDefinition>
                <TriggerDefinitions>
                  <Interval1
Type="Sitecore.Framework.Scheduling.Triggers.IntervalTriggerDefinition,
Sitecore.Framework.Scheduling" BindOptions="property">
                    <Options Interval="00:10:00" /> <!-- Raise every ten
minutes -->
                  </Interval1>
                  <Interval2
Type="Sitecore.Framework.Scheduling.Triggers.IntervalTriggerDefinition,
Sitecore.Framework.Scheduling" BindOptions="property">
                    <Options Interval="00:00:10" /> <!-- Raise every ten
seconds -->
                  </Interval2>
                </TriggerDefinitions>
              </CustomTask>
            </Tasks>
          </Options>
        </Scheduler>
      </Services>
    </Publishing>
  </Sitecore>
</Settings>
```

4.7 Schema Configuration

During start up, the service checks whether the latest version of the schema is installed. If the schema needs updating, the service shuts down and you must run the schema update tool `schematool.exe`.

A schema is defined as a DLL that contains a set of resources for preparing a connection for its role in the service. The resources are organized into versions to support incremental schema upgrade and downgrade. This means that – in the example of a SQL schema – the DLL contains multiple scripts for dropping and recreating tables, stored procedures, and other requirements for accessing SQL data.

Schemas can be split, based on their feature set and/or their connection type and they are configured under `<Sitecore><Publishing><Services><SchemaInstaller><Options>`.

The following code sample is the default schema configuration that defines the suite of schemas that are installed by the schema update tool:

```
<Settings>
  <Sitecore>
    <Publishing>
      <Services>
        <SchemaInstaller>
          <Options>
            <!--
              The DeploymentMap defines which schemas are loaded into which
connection
            -->
            <DeploymentMap>
              <Custom>
                <Links>
                  <Common>Common</Common>
                  <Data-Common>Data-Common</Data-Common>
                  <Data-Links>Data-Links</Data-Links>
                </Links>
              </Custom>
              <Service>
                <Common>Common</Common>
                <Service>Service</Service>
              </Service>
              <Source>
                <Common>Common</Common>
                <Data-Common>Data-Common</Data-Common>
                <Data-Source>Data-Source</Data-Source>
              </Source>
              <Target>
                <Common>Common</Common>
                <Data-Common>Data-Common</Data-Common>
                <Data-Target>Data-Target</Data-Target>
              </Target>
            </DeploymentMap>
            <!--
              The schemas bind names from the DeploymentMap to a Type/Assembly
containing sql schemas to be loaded
            -->
            <Schemas>
              <Common>Sitecore.Framework.Publishing.Common.Sql.Schema,
Sitecore.Framework.Publishing.Common.Sql.Schema</Common>
              <Data-
Common>Sitecore.Framework.Publishing.Data.Common.Sql.Schema,
Sitecore.Framework.Publishing.Data.Common.Sql.Schema</Data-Common>
              <Data-Links>Sitecore.Framework.Publishing.Data.Links.Sql.Schema,
Sitecore.Framework.Publishing.Data.Links.Sql.Schema</Data-Links>
```

```
              <Data-
Source>Sitecore.Framework.Publishing.Data.Source.Sql.Schema,
Sitecore.Framework.Publishing.Data.Source.Sql.Schema</Data-Source>
              <Data-
Target>Sitecore.Framework.Publishing.Data.Target.Sql.Schema,
Sitecore.Framework.Publishing.Data.Target.Sql.Schema</Data-Target>
              <Service>Sitecore.Framework.Publishing.Service.Sql.Schema,
Sitecore.Framework.Publishing.Service.Sql.Schema</Service>
            </Schemas>
          </Options>
        </SchemaInstaller>
      </Services>
    </Publishing>
  </Sitecore>
</Settings>
```

4.7.1 The Deployment Map

The deployment map section maps the schemas to connection types. The following code sample binds the d Common, Data-Common, Data-Links schemas that should be installed on the custom Links connection. The Common and Service schemas are installed on the Service connection.

```
<DeploymentMap>
              <Custom>
                <Links>
                  <Common>Common</Common>
                  <Data-Common>Data-Common</Data-Common>
                  <Data-Links>Data-Links</Data-Links>
                </Links>
              </Custom>
              <Service>
                <Common>Common</Common>
                <Service>Service</Service>
              </Service>
```

4.7.2 Schemas

The Schemas section names all of the schemas that are installed. Each configuration value should point to a type in an assembly where the schemas can be discovered. The following code sample names the Sitecore.Framework.Publishing.Common.Sql.Schema assembly as Common and the Sitecore.Framework.Publishing.Data.Common.Sql.Schema assembly as Data-Common:

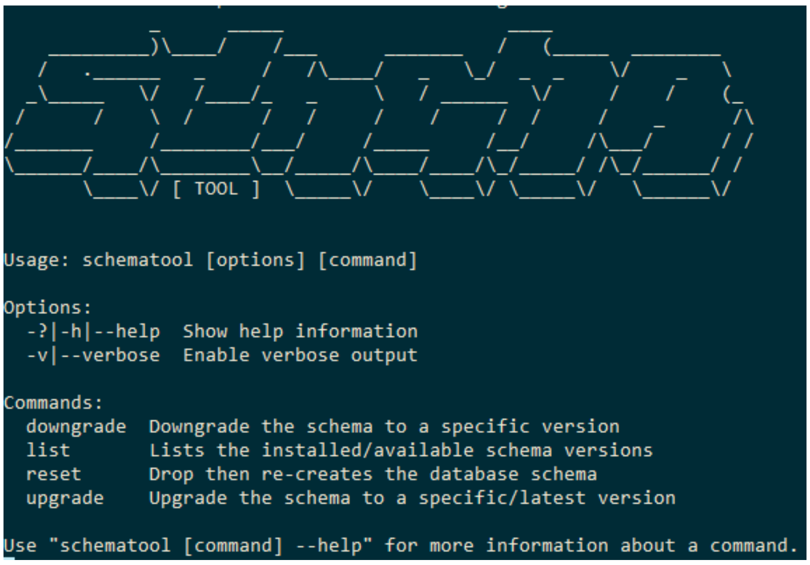
```
<Schemas>
              <Common>Sitecore.Framework.Publishing.Common.Sql.Schema,
Sitecore.Framework.Publishing.Common.Sql.Schema</Common>

              <Data-
Common>Sitecore.Framework.Publishing.Data.Common.Sql.Schema,
Sitecore.Framework.Publishing.Data.Common.Sql.Schema</Data-Common>
```

4.7.3 Schema Update tool

When the schematool.exe command is executed for the first time, it creates a __PublishingSchema table in each database that is configured in the system. The table keeps track of the installed schema versions.

Running the tool without any command line arguments displays a list of the available commandlets.



```
Usage: schematool [options] [command]

Options:
  -?|-h|--help  Show help information
  -v|--verbose  Enable verbose output

Commands:
  downgrade  Downgrade the schema to a specific version
  list       Lists the installed/available schema versions
  reset      Drop then re-creates the database schema
  upgrade    Upgrade the schema to a specific/latest version

Use "schematool [command] --help" for more information about a command.
```

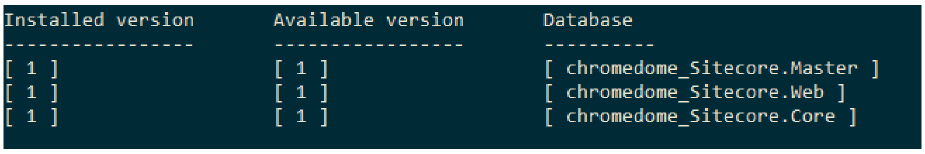
The tool supports the following commands:

- `Upgrade` (`schematool.exe upgrade`) – This command upgrades the schema to the latest version.

  Optionally, you can specify the schema version that you want to upgrade to by running the following command:

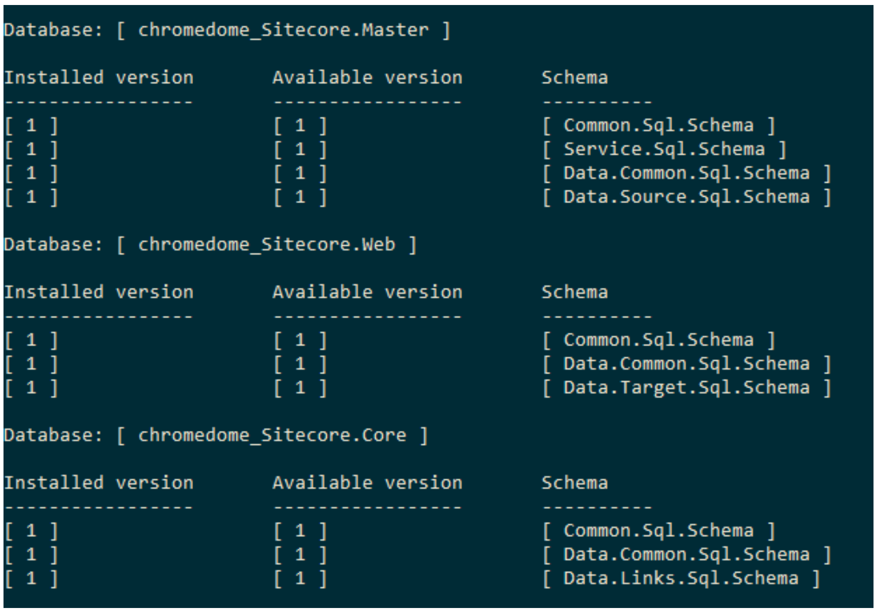
  `schematool.exe upgrade --version x`

- `List (schematool.exe list)` – This command lists the versions of the installed schemas along with the available schemas (from the resource files).



```
Installed version       Available version       Database
-----------------       -----------------       ----------
[ 1 ]                   [ 1 ]                   [ chromedome_Sitecore.Master ]
[ 1 ]                   [ 1 ]                   [ chromedome_Sitecore.Web ]
[ 1 ]                   [ 1 ]                   [ chromedome_Sitecore.Core ]
```

  Additionally, the tool can display more detailed information about the schema if the "`-d`" option is used. This lists the version of each schema resource in each database.



```
Database: [ chromedome_Sitecore.Master ]

Installed version       Available version       Schema
-----------------       -----------------       ----------
[ 1 ]                   [ 1 ]                   [ Common.Sql.Schema ]
[ 1 ]                   [ 1 ]                   [ Service.Sql.Schema ]
[ 1 ]                   [ 1 ]                   [ Data.Common.Sql.Schema ]
[ 1 ]                   [ 1 ]                   [ Data.Source.Sql.Schema ]

Database: [ chromedome_Sitecore.Web ]

Installed version       Available version       Schema
-----------------       -----------------       ----------
[ 1 ]                   [ 1 ]                   [ Common.Sql.Schema ]
[ 1 ]                   [ 1 ]                   [ Data.Common.Sql.Schema ]
[ 1 ]                   [ 1 ]                   [ Data.Target.Sql.Schema ]

Database: [ chromedome_Sitecore.Core ]

Installed version       Available version       Schema
-----------------       -----------------       ----------
[ 1 ]                   [ 1 ]                   [ Common.Sql.Schema ]
[ 1 ]                   [ 1 ]                   [ Data.Common.Sql.Schema ]
[ 1 ]                   [ 1 ]                   [ Data.Links.Sql.Schema ]
```

- `Downgrade` (`schematool.exe downgrade --force`) – This command removes all the installed schemas.

  Optionally, you can downgrade to a specific schema version by running the following command:

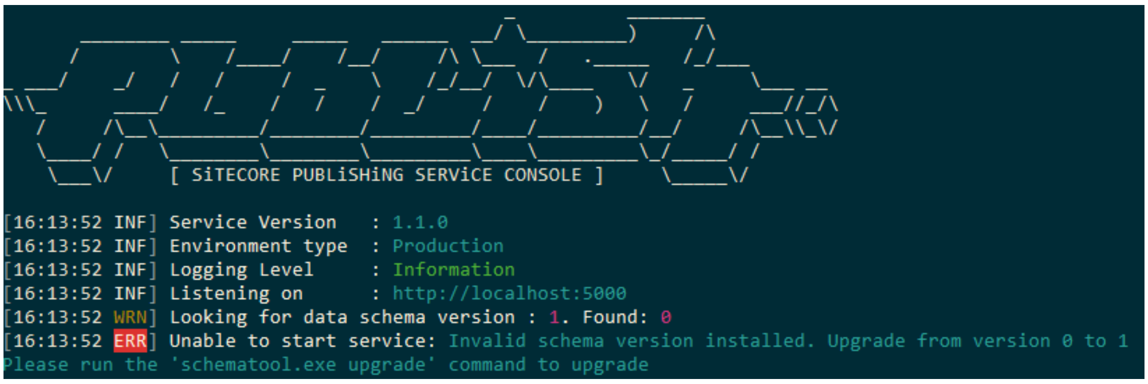
  ```
  schematool.exe downgrade --version x --force
  ```

- `Reset` (schematool.exe reset --force) – This command removes all the installed schemas and then installs the latest schema version. Optionally, you can choose to reset to a specific schema version by using the following command:

  ```
  schematool.exe reset --version x --force
  ```

4.7.4 Validating Schemas

When the publishing service starts, it checks whether the latest schema is installed. The version of the installed schemas retrieved from the `__PublishingSchema` table is compared to the schema version in the resource file. If a schema upgrade is needed, the service will shut down and display an error message telling you to upgrade the schema.



```
[ SiTECORE PUBLiSHiNG SERViCE CONSOLE ]

[16:13:52 INF] Service Version    : 1.1.0
[16:13:52 INF] Environment type   : Production
[16:13:52 INF] Logging Level      : Information
[16:13:52 INF] Listening on       : http://localhost:5000
[16:13:52 WRN] Looking for data schema version : 1. Found: 0
[16:13:52 ERR] Unable to start service: Invalid schema version installed. Upgrade from version 0 to 1
Please run the 'schematool.exe upgrade' command to upgrade
```

4.8 Logging Configuration

All log files that are generated by the publishing service are stored in a logs folder at the application installation path. Logs are treated as rolling files, and logging information is added to the file with the current date. If no log file exists, the publishing service creates one.

- General Logs: Log details are added to a log file called `Publishing-<date>.log` where `<date>` is the current date.
- Error Logging: All errors are added to a log file called `Publishing-Errors-<date>.log` where `<date>` is the current date.
- Log Level: The log level can be configured by overriding the <Sitecore><Publishing><LogLevel> setting

```
<Settings>
  <Sitecore>
    <Publishing>
      <LogLevel>Debug</LogLevel>
    </Publishing>
  </Sitecore>
</Settings>
```

  The log level supports the following values:

  - Trace
  - Debug
  - Information
  - Warning
  - Error
  - Critical
  - None

4.9 Troubleshooting

If you receive an error where the Internet Information Services (IIS) cannot read the application configuration, ensure you have installed all the prerequisites.

If you receive a *502 - Bad Gateway* error when you visit your site, check the logs for details.

After fixing any errors, restart your application pool and try again.

Chapter 5

Installing and Configuring the Sitecore Publishing Module

The Sitecore Publishing module is distributed as a standard Sitecore package. However, you must perform some manual configuration steps after the installation.

This chapter describes:

- Installing the
- Post-installation Configuration
- Recovery
- Publisher Operations Service

5.1 Security

When you install the Publishing Service module, it does not update the security system and does not grant any permissions to any security roles.

To perform a full publish, you must edit the configuration file and specifically grant permission to perform a full site publish to the appropriate security roles. You can only grant permission to perform a full site publish to a security role; you cannot grant it to an individual user.

Important
You cannot grant permission to perform a full site publish to a security role by editing the security roles in Sitecore.

- Granting Permission to Perform a Full Republish
- Operation Emitter
- Events

5.2 Installing the Sitecore Publishing Module

The Sitecore Publishing module is distributed as a standard Sitecore package. You can install it like any other Sitecore package.

The Sitecore Publishing module package is called:

- Sitecore Publishing Module 1.8.0.zip

To install the package:

1. On the Sitecore Developer Portal, download the installation package for the module.
2. On the Sitecore Launchpad, click **Control Panel**, and in the **Administration** section, click **Install a package**.

   The **Install a Package** wizard guides you through the installation process.
3. Before you close the wizard, select **Restart the Sitecore Client**.

5.3 Post-installation Configuration

After you have installed the Sitecore Publishing module, you must configure the module before you can use it.

5.3.1 Service Endpoints

To configure the service endpoints:

1. In the *Website root* folder, navigate to the `App_Config/Include/Sitecore.Publishing.Service.Config` configuration file.

   Note
   Do not modify the file directly. Use the Sitecore configuration Include features to change the values.

2. Add a configuration file which overrides the `PublishingServiceUrlRoot` setting to point to your service module:

```
==========================================================
***Important! Copy and save this information***
==========================================================
    BEFORE YOU CLICK NEXT:
    - Ensure you have installed and configured the Sitecore Publishing
Service (this module only enables integration with the service)
        Documentation detailing how to install the service is available
separately.

        [Warning] This module will not work without a properly configured
service instance. No items will be able to be published.

    AFTER YOU CLOSE THE WIZARD:
    After the package is installed, follow these steps to complete the
Sitecore Publishing Service installation:
    - Configure the service endpoints:
        Add a configuration file which overrides the
'PublishingServiceUrlRoot' setting to point to your service module
        Make sure the address contains a trailing slash
        e.g.
            <?xml version="1.0" encoding="utf-8"?>
            <configuration
xmlns:patch="http://www.sitecore.net/xmlconfig/">
              <sitecore>
                <settings>
                  <setting
name="PublishingServiceUrlRoot">http://sitecore.publishing/</setting>
                </settings>
              </sitecore>
            </configuration>
      - Configure the Content Delivery Servers:
            Rename the file
'Sitecore.Publishing.Service.Delivery.config.disabled' to
            'Sitecore.Publishing.Service.Delivery.config'.
            Ensure that the 'Sitecore.Publishing.Service.Delivery.dll'
exists in the website 'bin' directory
```

Important
Make sure that the URL ends with a trailing slash and that it is formatted correctly.

5.3.2 Content Delivery Servers

To configure the Content Delivery servers to use the publishing service:

1. Enable the `Sitecore.Publishing.Service.Delivery.config.disabled` file by removing the `.disabled` suffix.
2. Ensure that the following files are in the website 'bin' directory:
   - `Sitecore.Framework.Conditions.dll`
   - `Sitecore.Publishing.Service.dll`
   - `Sitecore.Publishing.Service.Abstractions.dll`
   - `Sitecore.Publishing.Service.Delivery.dll`

5.4 Recovery

When an item is modified during service downtime, a recovery process is activated and this process stores any modifications locally. When the service is available again, it recovers the stored changes and pushes them to the service.

The recovery strategy determines how often the recovery process attempts to recover the changes and push them to the publishing service.

To change the frequency of the recovery attempts, change the `interval` setting:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/">
  <sitecore>
    <publishing.service>
      <recoveryStrategy>
        <param name="interval">30</param>
      </recoveryStrategy>
    </publishing.service>
  </sitecore>
</configuration>
```

5.5 Publisher Operations Service

To capture the scenarios where the service is down, the `Publisher Operations` service executes requests inside a circuit breaker. When the circuit breaker detects a request error, it tracks the number of failures. When the service detects that the maximum allowed number of failures has been reached, it stops sending requests for a specified period of time.

You can configure the number of failures and the length of the timeout period:

```xml
<?xml version="1.0" encoding="utf-8" ?>
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/">
  <sitecore>
    <publishing.service>
      <publisherOpsService>
        <param name="circuitbreaker">

          <!-- The number of failed 'add event' requests to the service that
are allowed before determining a communication problem. -->
          <param name="exceptionsAllowedBeforeBreaking">3</param>

          <!-- The duration until communication with the service is attempted
again after a communication problem was last
               detected. -->
          <param name="secondsBeforRetrying">300</param>
        </param>

        <param name="recoveryStrategy"
ref="publishing.service/recoveryStrategy"/>
      </publisherOpsService>
    </publishing.service>
  </sitecore>
</configuration>
```

5.6 Security

When you install the Publishing Service module, it does not update the security system and does not grant any permissions to any security roles.

To perform a full publish, you must edit the configuration file and specifically grant permission to perform a full site publish to the appropriate security roles. You can only grant permission to perform a full site publish to a security role; you cannot grant it to an individual user.

Important
You cannot grant permission to perform a full site publish to a security role by editing the security roles in Sitecore.

5.6.1 Granting Permission to Perform a Full Republish

You can only explicitly grant permission to perform a full site publish to specific security roles in the configuration file. To enable the **Full republish** check box for a role, you must enter the `domain\name` for each role:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/">
  <sitecore>
    <publishing.service>
      <api>
        <services>
          <allowFullPublishRoles>
            <role>sitecore\Sitecore Client Publishing</role>
          </allowFullPublishRoles>
        </services>
      </api>
    </publishing.service>
  </sitecore>
</configuration>
```

5.7 Operation Emitter

The operation emitter buffers and streams item changes to the publishing service. You can configure it to stream content more frequently or in larger batches.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/">
  <sitecore>
    <publishing.service>
      <operationEmitter>
        <!-- The interval at which an event batch is created to be sent to the
service. -->
        <param name="eventBufferingWindowMaxMilliseconds">2000</param>

        <!-- The maximum size of an event batch that will be sent to the
service. -->
        <param name="eventBufferingMaxCount">50</param>
      </operationEmitter>
    </publishing.service>
  </sitecore>
</configuration>
```

5.8 Events

The publishing service module exposes a new event in Sitecore:

publishingservice:publishend

When the publishing service completes a publishing job, this event is triggered once for the entire job.

Chapter 6

Publishing with the Sitecore Publishing Module

This chapter is for content authors that need to know how to publish a website or an item with the Sitecore Publishing module.

This chapter contains the following sections:

- The Sitecore Publishing Module
- Publishing an Item
- Publishing a Website
- Publish all Items
- The Sitecore Commerce Server Connect Publishing extension package

6.1 The Sitecore Publishing Module

In the Sitecore Publishing module, you can choose to publish the entire website or a single item:

- **Item publishing** – publishes the item you select in either the Content Editor or the Experience Editor. The item can only be published if all its ancestors have been published.

  With you publish an item, you can choose to include all of the selected item's subitems and related items.

- **Site publishing** – publishes all the changes that have been made on your entire website since the last time the website was published. You can publish a site from the Content Editor or from the Sitecore Desktop.

- **Publish all items** – when you publish all items, you have the following two publishing options:

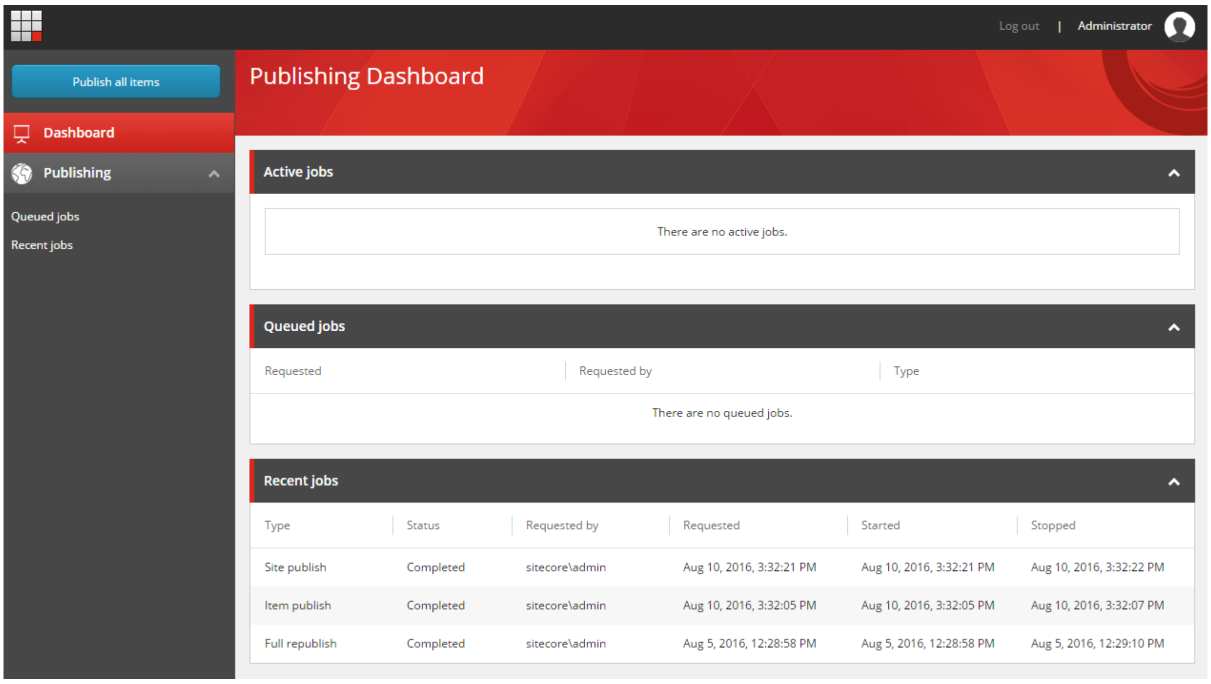
  - Publish the items that in the Master database are different from the equivalent item in the target database.
  - Publish all the items in your Sitecore installation regardless of when the items were last published. This requires a considerable amount of time and resources.

Note
Only users with sufficient access rights have the permission to perform this type of publishing.

6.1.1 The Publishing Dashboard

The Publishing Dashboard gives you an overview of all the active, queued, and recent publishing jobs:

- **Active jobs** – the publishing jobs that are currently being published.
- **Queued jobs** – the publishing jobs that are waiting to be published.
- **Recent jobs** – the most recent publishing jobs that have been processed.



Publishing Dashboard

Active jobs

There are no active jobs.

Queued jobs

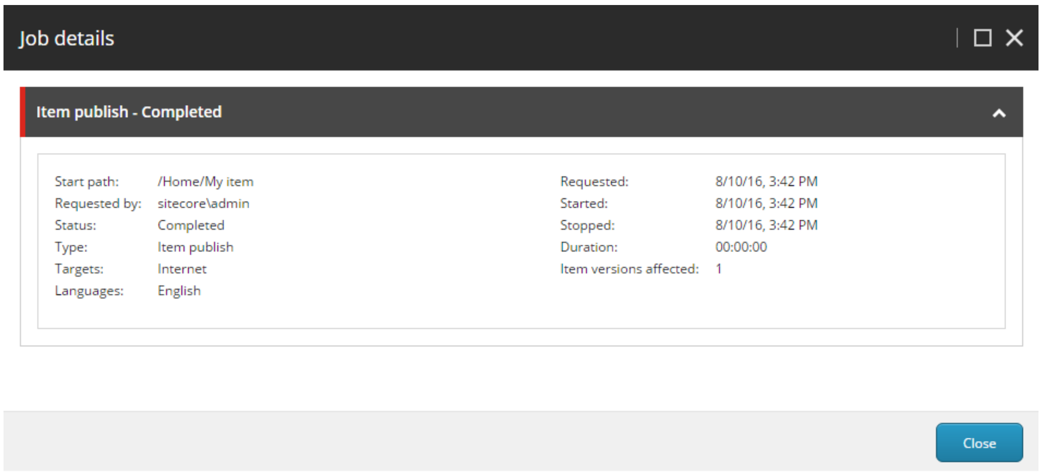
Requested	Requested by	Type

There are no queued jobs.

Recent jobs

Type	Status	Requested by	Requested	Started	Stopped
Site publish	Completed	sitecore\admin	Aug 10, 2016, 3:32:21 PM	Aug 10, 2016, 3:32:21 PM	Aug 10, 2016, 3:32:22 PM
Item publish	Completed	sitecore\admin	Aug 10, 2016, 3:32:05 PM	Aug 10, 2016, 3:32:05 PM	Aug 10, 2016, 3:32:07 PM
Full republish	Completed	sitecore\admin	Aug 5, 2016, 12:28:58 PM	Aug 5, 2016, 12:28:58 PM	Aug 5, 2016, 12:29:10 PM

To see the details of a recent publishing job or a queued publishing job, click the relevant job in one of the lists.


Job details

Item publish - Completed

Start path:	/Home/My item	Requested:	8/10/16, 3:42 PM
Requested by:	sitecore\admin	Started:	8/10/16, 3:42 PM
Status:	Completed	Stopped:	8/10/16, 3:42 PM
Type:	Item publish	Duration:	00:00:00
Targets:	Internet	Item versions affected:	1
Languages:	English		

Close

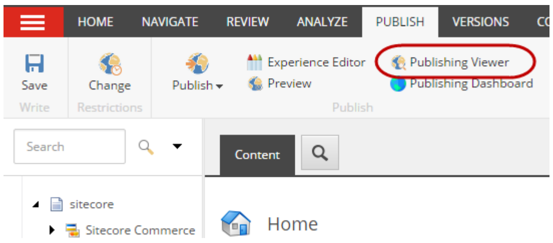
Note
There are no details to view for active publishing jobs.

6.1.2 Publishing Viewer

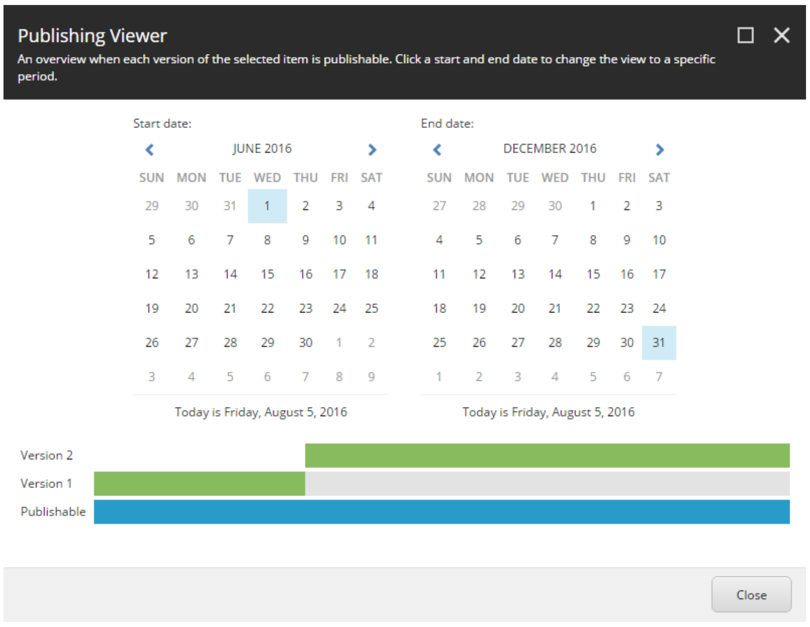
Use the Publishing Viewer to get an overview of when the various versions of an item are publishable.

To see the Publishing Viewer for an item:

1. In the Content Editor, click the relevant item.
2. On the **Publish** tab, in the **Publish** group, click **Publishing Viewer**.



3. In the **Publishing Viewer** dialog box, specify a start and end date to see if the items' versions are publishable during that period.


Publishing Viewer
An overview when each version of the selected item is publishable. Click a start and end date to change the view to a specific period.
Start date:
JUNE 2016
End date:
DECEMBER 2016
Today is Friday, August 5, 2016
Today is Friday, August 5, 2016
Version 2
Version 1
Publishable
Close

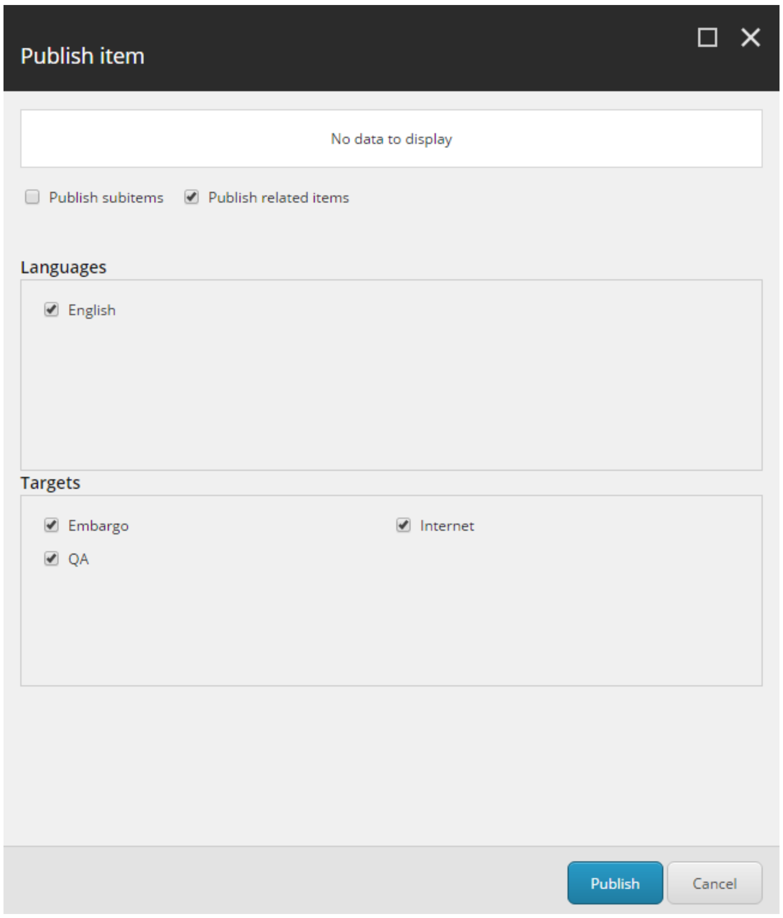
6.2 Publishing an Item

When you want to publish a single item to your website in one or more language versions, you perform an item publish from the Content Editor or the Experience Editor.

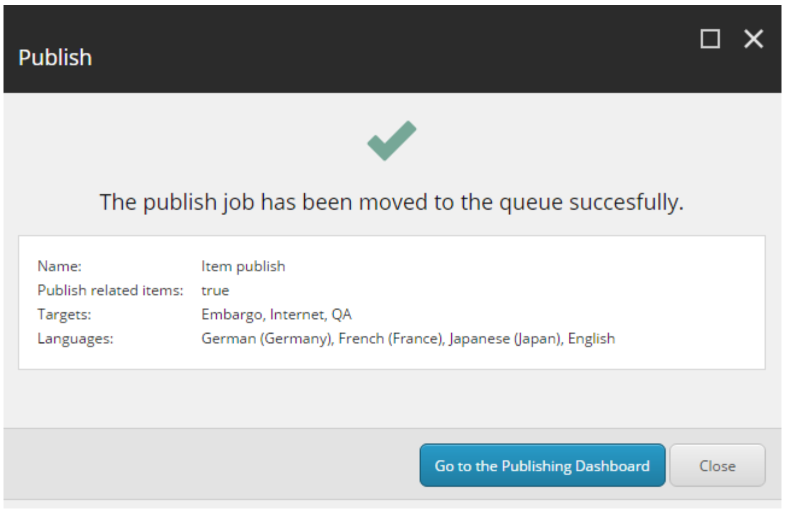
Note
To get an overview of when the different versions of an item are publishable, in the Content Editor, on the **Publish** tab, click **Publishing Viewer**.

To publish a single item:

1. Open the **Publish** dialog box:
   - In the Content Editor, select the item that you want to publish. On the **Publish** tab, in the **Publish** group, click the **Publish** drop-down arrow, and then select **Publish item**.
   - In the Experience Editor, navigate to the page that you want to publish, and then on the **Home** tab, in the **Publish** group, click **Publish**.
2. In the **Publish item** dialog box, verify the item details and select:
   - **Publish subitems** to publish the current item and all its subitems.
   - **Publish related items** to publish the current item and all its related items, such as clone references, media references, and alias references.



3. Select the language versions of the item that you want to publish and the targets that you want to publish the item to.
4. To move the publishing job to the publishing queue, click **Publish**.



The publish job has been moved to the queue succesfully.

Name:	Item publish
Publish related items:	true
Targets:	Embargo, Internet, QA
Languages:	German (Germany), French (France), Japanese (Japan), English

5. To get an overview of the active, queued, and recent publishing jobs, click **Go to the Publishing Dashboard**.

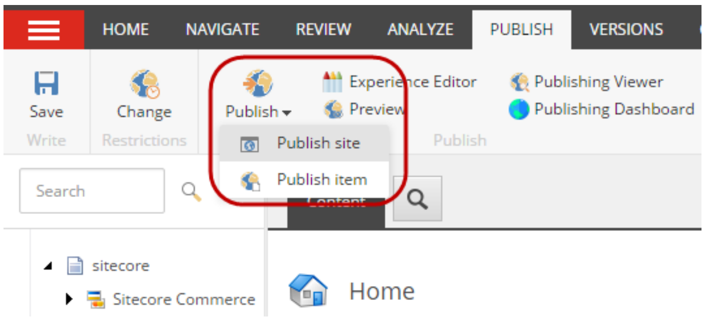
6.3 Publishing a Website

When you perform a site publish, you only publish the items that have changed since the site was last published.

Note
Users with sufficient access rights can publish all the items in your Sitecore installation at the same time regardless of when they were last published. This requires a considerable amount of time and resources. To publish all items, in the **Publishing Dashboard** click **Publish all items**.

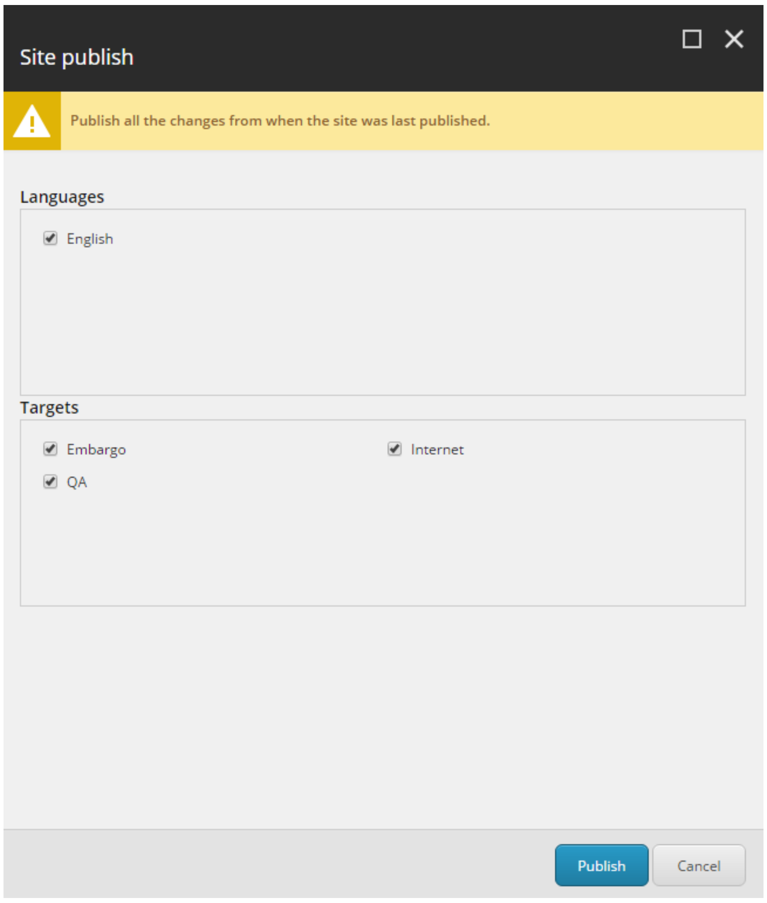
To publish the changes made to your website:

1. In the Content Editor, on the **Publish** tab, in the **Publish** group, click the **Publish** drop-down arrow, and click **Publish site**.



Note
You can also perform a site publish from the Sitecore Start menu.

2. In the **Publish** dialog, select the language versions that you want to publish and the targets that you want to publish the site to.

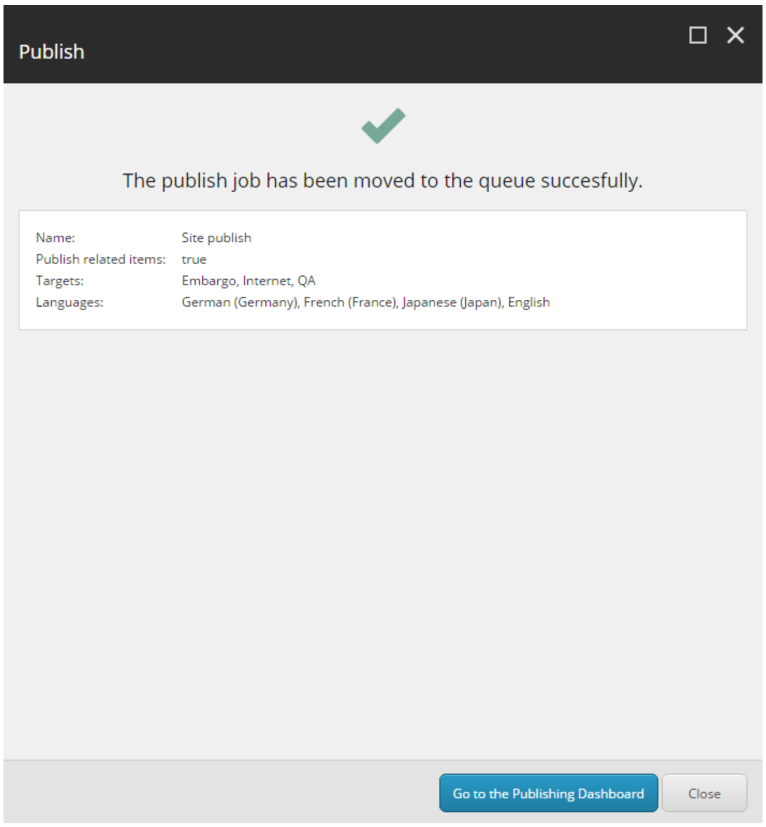


Publish all the changes from when the site was last published.

Languages
- [✓] English

Targets
- [✓] Embargo
- [✓] Internet
- [✓] QA

3. To move the publishing job to the publishing queue, click **Publish**.



The publish job has been moved to the queue succesfully.

Name:	Site publish
Publish related items:	true
Targets:	Embargo, Internet, QA
Languages:	German (Germany), French (France), Japanese (Japan), English

4. To see an overview of the active, queued, and recent publishing jobs, click **Go to the Publishing Dashboard**.

6.4 Publish all Items

From the Publishing Dashboard, users that have sufficient access rights have additional two publishing options:

- Publish all the items in the Master database that are different from the equivalent item in the target database.
- Publish all the items in your Sitecore installation regardless of when the items were last published. This requires a considerable amount of time and resources.

When you publish all the items, the database and the publishing service will be subject to a much larger load than usual. Therefore, this action requires a considerable amount of time and resources.

Note
To grant a specific security role permission to publish all items, you must configure the `Sitecore.Publishing.Service.Config` file. For more information, see the section *Security*

When you install the Publishing Service module, it does not update the security system and does not grant any permissions to any security roles.

To perform a full publish, you must edit the configuration file and specifically grant permission to perform a full site publish to the appropriate security roles. You can only grant permission to perform a full site publish to a security role; you cannot grant it to an individual user.

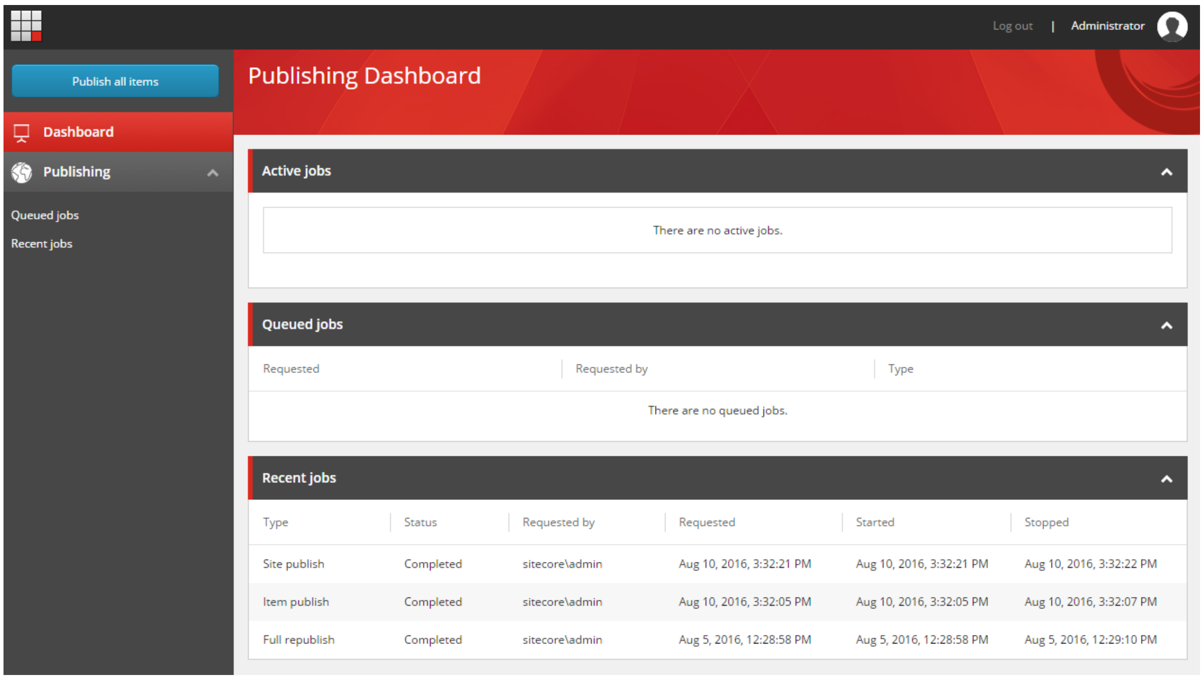
Important
You cannot grant permission to perform a full site publish to a security role by editing the security roles in Sitecore.

Granting Permission to Perform a Full Republish.

To publish items from the Publishing Dashboard:

1. In the Publishing Dashboard, click **Publish all items**.

   Note
   Users that do not have the proper access rights cannot see the **Publish all items** button.



Active jobs

There are no active jobs.

Queued jobs

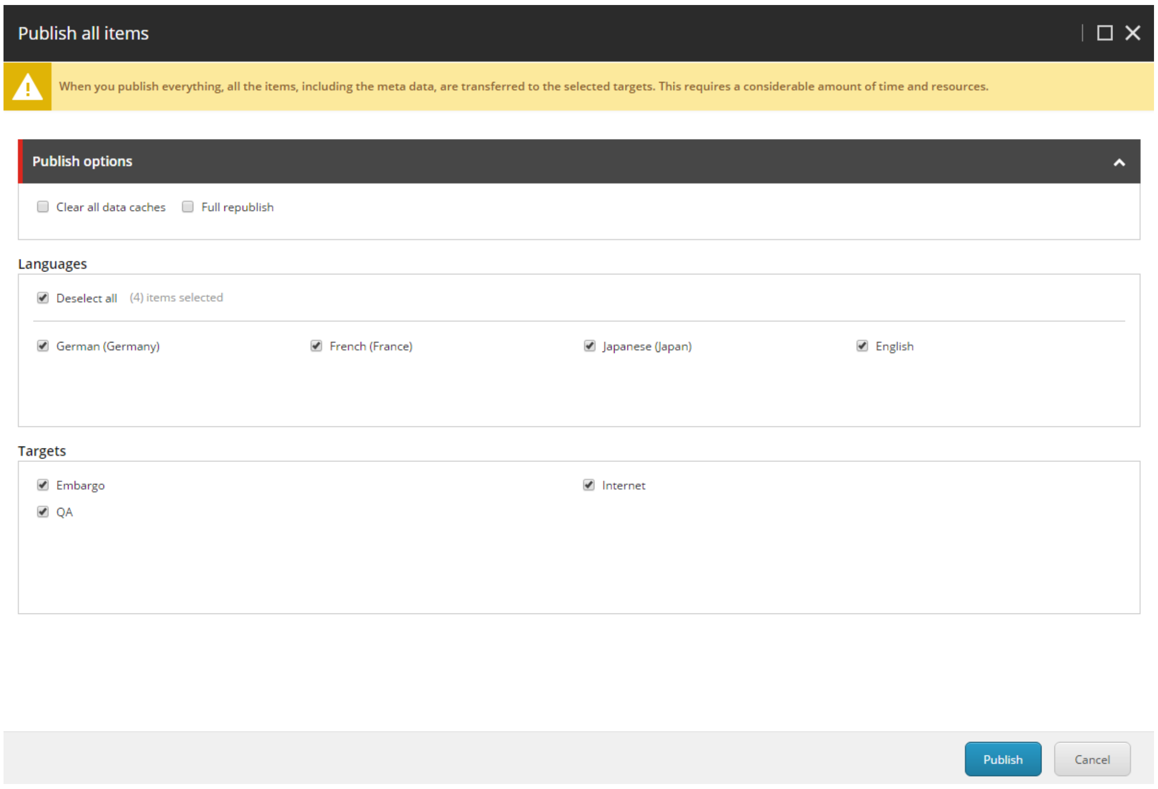
Requested	Requested by	Type

There are no queued jobs.

Recent jobs

Type	Status	Requested by	Requested	Started	Stopped
Site publish	Completed	sitecore\admin	Aug 10, 2016, 3:32:21 PM	Aug 10, 2016, 3:32:21 PM	Aug 10, 2016, 3:32:22 PM
Item publish	Completed	sitecore\admin	Aug 10, 2016, 3:32:05 PM	Aug 10, 2016, 3:32:05 PM	Aug 10, 2016, 3:32:07 PM
Full republish	Completed	sitecore\admin	Aug 5, 2016, 12:28:58 PM	Aug 5, 2016, 12:28:58 PM	Aug 5, 2016, 12:29:10 PM

2. In the **Publish all items** dialog box:
   - Select the **Full republish** check box to publish all the items in your installation regardless of when they were last published.
   - Clear the **Full republish** check box to publish all the items in the Master database that are different from the equivalent item in the selected target database.
3. If you want to publish a large amount of items or if you want to publish to a new target, select the **Clear all data caches** check box, to clear the data level caches that contain a reference to the items that are published.
4. Select the language versions that you want to publish and the targets that you want to publish the items to.


Publish all items

When you publish everything, all the items, including the meta data, are transferred to the selected targets. This requires a considerable amount of time and resources.

Publish options

☐ Clear all data caches ☐ Full republish

Languages

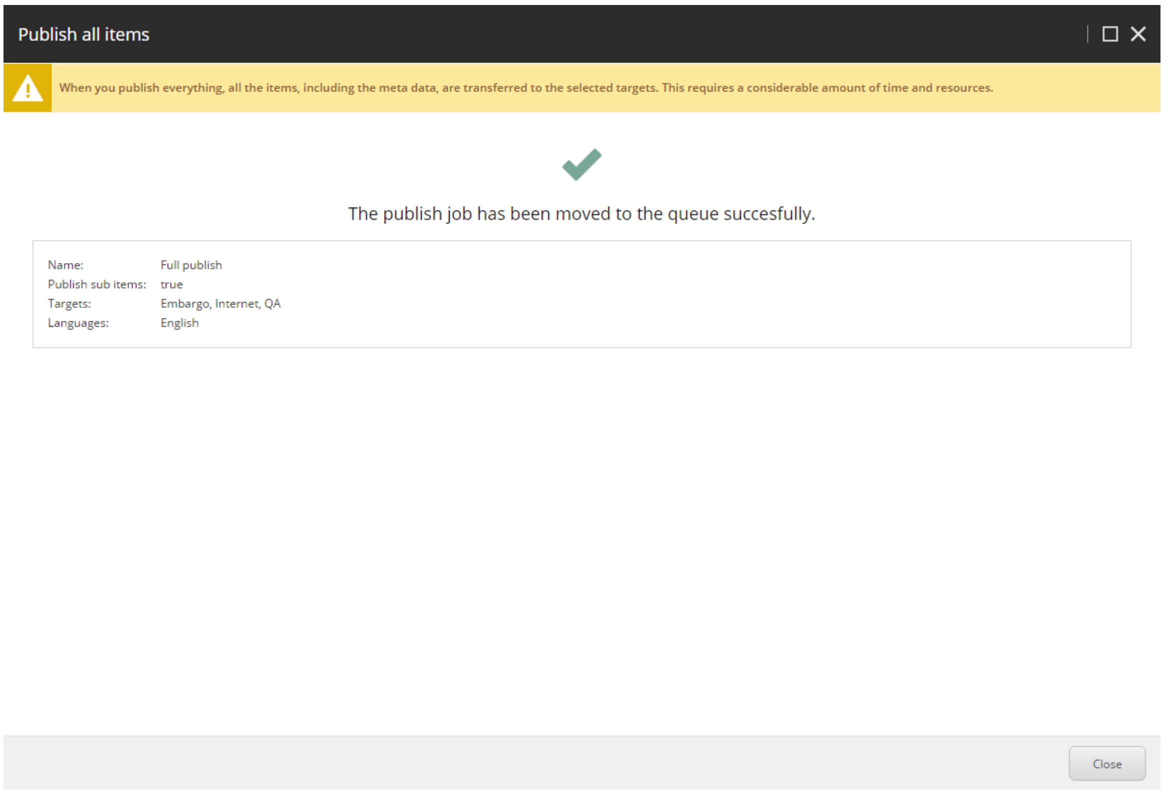
☑ Deselect all (4) items selected

☑ German (Germany) ☑ French (France) ☑ Japanese (Japan) ☑ English

Targets

☑ Embargo ☑ Internet
☑ QA

5. To move the publishing job to the publish queue, click **Publish**.



When you publish everything, all the items, including the meta data, are transferred to the selected targets. This requires a considerable amount of time and resources.

The publish job has been moved to the queue succesfully.

Name:	Full publish
Publish sub items:	true
Targets:	Embargo, Internet, QA
Languages:	English

6. To see an overview of the active, queued, and recent publishing jobs, close the **Publish all items** dialog box.

6.5 The Sitecore Commerce Server Connect Publishing extension package

If you have the Sitecore Commerce Server Connect Publishing extension package installed, you can start staging projects directly from the **Site publish** dialog or the **Item publish** dialog and execute staging and publishing in parallel.

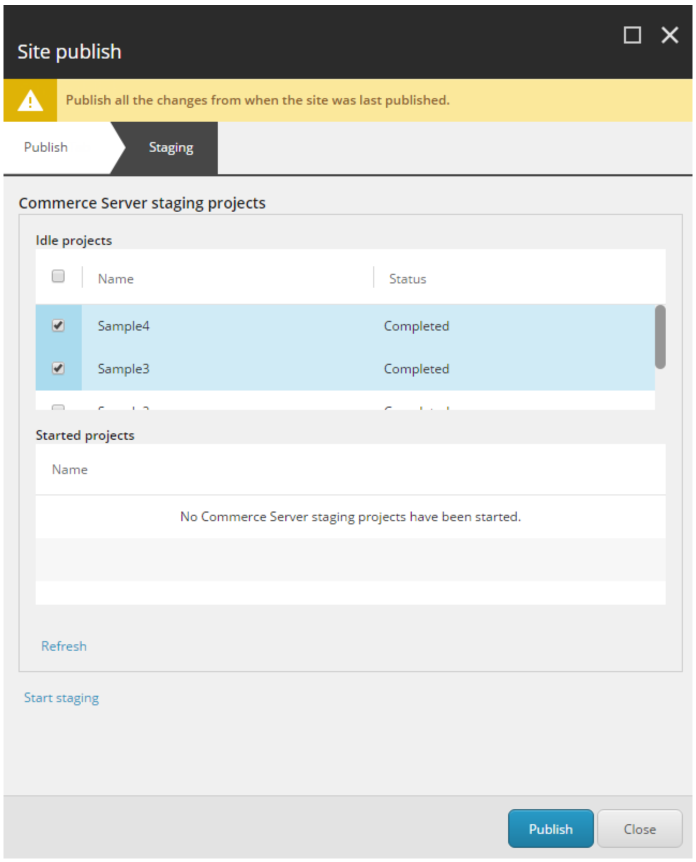
To install the extension package:

1. From http://dev.sitecore.net/ download the `Sitecore Commerce Server Connect Publishing Extensions.8.2.update` package.
2. In a browser, navigate to `http://<your site>/sitecore/admin/UpdateInstallationWizard.aspx` and then follow the steps in the installation wizard.

Note
When the installation has finished, you see a summary of the installation. The potential problems that are listed can generally be ignored.

To start a staging project when you publish:

1. In the **Site publish** or **Item publish** dialog, click the **Staging** tab.
2. In the **Idle projects** list, select the projects you would like to start and then click **Start staging**.


Site publish
Publish all the changes from when the site was last published.
Publish
Staging
Commerce Server staging projects
Idle projects
Name
Status
Sample4
Completed
Sample3
Completed
Started projects
Name
No Commerce Server staging projects have been started.
Refresh
Start staging
Publish
Close

A notification appears to indicate that the projects have started successfully and the *Idle projects* list and the *Started projects* list are updated accordingly.

3. If you want to move the publishing job to the publishing queue, click **Publish**, otherwise click **Close**.
4. To view the status of the started staging projects, refresh the lists in the publishing dialog.
5. If you have closed the publishing dialog, just open the dialog again to see the updated status of the projects. Alternatively, you can open the Commerce Server Staging Manager.