

Contents

[Contributor Guide overview and browser-based tasks](#)

[Set up and work locally](#)

- [1. Sign up for GitHub](#)
- [2. Install Git and Markdown tools](#)
- [3. Set up a local Git repository](#)

[Git and GitHub fundamentals](#)

[Full workflow](#)

[Writing essentials](#)

[Markdown](#)

[Markdown reference](#)

[Style and voice quick start](#)

[Links](#)

[Docs Authoring Pack](#)

[Documentation set-specific guidance](#)

[.NET docs](#)

[Contribute to .NET docs](#)

[.NET docs style conventions](#)

[Voice and tone guide](#)

[Pull request review process](#)

[Additional resources](#)

Microsoft Docs contributor guide overview

7/30/2019 • 5 minutes to read • [Edit Online](#)

Welcome to the docs.microsoft.com (Docs) Contributor Guide!

Several of the Microsoft documentation sets are open source and hosted on GitHub. Not all document sets are completely open source but many have public-facing repos where you can make suggested changes via pull requests. This open source approach streamlines and improves communication between product engineers, content teams, and customers, and has other advantages:

- Open source repos *plan in the open* to get feedback on what docs are most needed.
- Open source repos *review in the open* to publish the most helpful content on our first release.
- Open source repos *update in the open* to make it easier to continuously improve the content.

The user experience on docs.microsoft.com integrates [GitHub](#) workflows directly to make it even easier. Start by [editing the document you are viewing](#). Or, help by [reviewing new topics](#), or [create quality issues](#).

IMPORTANT

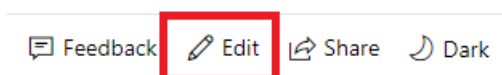
All repositories that publish to docs.microsoft.com have adopted the [Microsoft Open Source Code of Conduct](#) or the [.NET Foundation Code of Conduct](#). For more information, see the [Code of Conduct FAQ](#). Or contact opencode@microsoft.com, or conduct@dotnetfoundation.org with any questions or comments.

Minor corrections or clarifications to documentation and code examples in public repositories are covered by the [docs.microsoft.com Terms of Use](#). New or significant changes generate a comment in the pull request, asking you to submit an online Contribution License Agreement (CLA) if you are not an employee of Microsoft. We need you to complete the online form before we can review or accept your pull request.

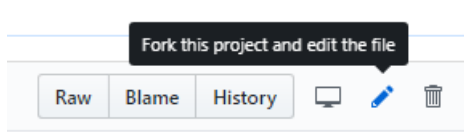
Quick edits to existing documents

Quick edits streamline the process to report and fix small errors and omissions in documents. Despite all efforts, small grammar and spelling errors *do* make their way into our published documents. While you can create issues to report mistakes, it's faster and easier to create a pull request (PR) to fix the issue, when the option is available.

1. Some docs pages allow you to edit content directly in the browser. If so, you'll see an **Edit** button like the one shown below. Clicking the **Edit** (or equivalently localized) button takes you to the source file on GitHub. If the **Edit** button (pencil icon without text) is missing, that means the documentation page is not available to be changed.

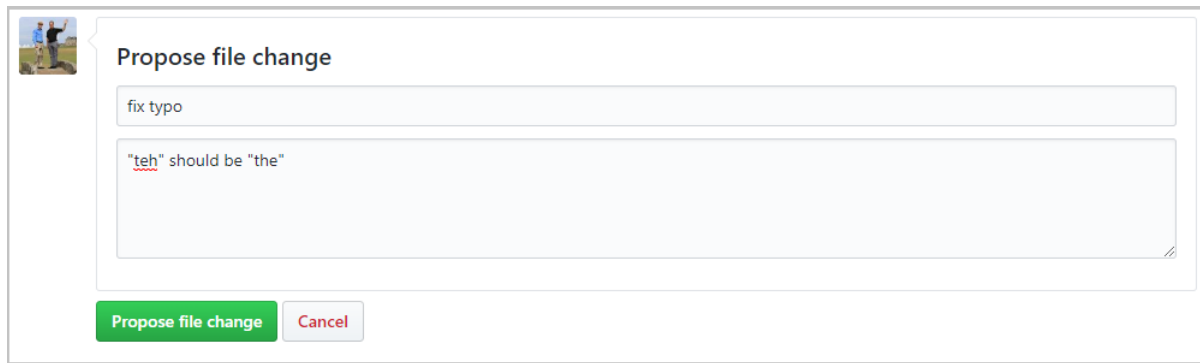


2. Next, click the pencil icon, to edit the article as shown. If the pencil icon is grayed out, you need to either login to your GitHub account or create a new account.



3. Make your changes in the web editor. Click the **Preview changes** tab to check the formatting of your change.

4. Once you have made your changes, scroll to the bottom of the page. Enter a title and description for your changes and click **Propose file change** as shown in the following figure:



5. Now that you've proposed your change, you need to ask the owners of the repository to "pull" your changes into their repository. This is done using something called a "pull request". When you clicked on **Propose file change** in the figure above, you should have been taken to a new page that looks like the following figure:

Comparing changes

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also [compare across forks](#).



Click **Create pull request**, enter a title (and optionally a description) for the pull request, and then click **Create pull request** again. (If you are new to GitHub, see [About Pull Requests](#) for more information.)

6. That's it! Content team members will review and merge your PR. You may get some feedback requesting changes if you made larger changes.

The GitHub editing UI responds to your permissions on the repository. The preceding images are accurate for contributors that do not have write permissions to the target repository. GitHub automatically creates a fork of the target repository in your account. If you have write-access to the target repository, GitHub creates a new branch in the target repo. The branch name has the form **<GitHubId>-patch-n** using your GitHub ID, and a numeric identifier for the patch branch.

We use pull requests for all changes, even for contributors that have write-access. Most repositories have the `master` branch protected so that updates must be submitted as pull requests.

The in-browser editing experience is best for minor or infrequent changes. If you make large contributions or use advanced Git features (such as branch management or advanced merge conflict resolution), you need to [fork the repo and work locally](#).

NOTE

If enabled, you can edit an article in **any language** and, based on the type of edit, the following will happen:

1. any linguistic change that is approved, will also help improve our Machine Translation engine
2. any edit that significantly modifies the content of the article will be handled internally to submit a change to the same article in English so that it will get localized in all languages if approved. So your suggested improvements will not only positively affect articles in your own language, but in all available languages.

Review open PRs

You can read new topics before they are published by checking the currently open PRs. Reviews follow the [GitHub flow](#) process. You can see proposed updates or new articles in public repositories. Review them and add your comments. Look at any of our docs repositories, and check the open pull requests (PRs) for areas that interest you. Community feedback on proposed updates helps the entire community.

Create quality issues

Our docs are a continuous work in progress. Good issues help us focus our efforts on the highest priorities for the community. The more detail you can provide, the more helpful the issue. Tell us what information you sought. Tell us the search terms you used. If you can't get started, tell us how you want to start exploring unfamiliar technology.

Many of Microsoft's documentation pages have a **Feedback** section at the bottom of the page where you can click to leave **Product feedback** or **Content feedback** to track issues that are specific to that article.

Issues start the conversation about what's needed. The content team will respond to these issues with ideas for what we can add, and ask for your opinions. When we create a draft, we'll ask you to [review the PR](#).

Get more involved

Other topics help you get started productively contributing to Microsoft Docs. They explain working with GitHub repositories, Markdown tools, and extensions used in the Microsoft Docs platform.

GitHub account setup

7/30/2019 • 2 minutes to read • [Edit Online](#)

Set up your GitHub account

To contribute to Docs technical content, you need to set up your own GitHub account. The good news is, you usually only have to perform these steps once.

1. Create a GitHub account and set up your profile

If you don't already have a GitHub account, [create one](#). Identify any affiliations in your GitHub profile. Contributions to docs.microsoft.com count toward [MVP award](#) consideration. Identification helps us build a complete profile of all your activities.

NOTE

Microsoft employees participating in Open Source projects always identify themselves as such in their GitHub profiles. Community contributors should ensure that their profile does not incorrectly imply an employment relationship.

Next step

- Continue to the [Tool installations](#) article to install Git Bash, a Markdown editor, and more.

Install content authoring tools

7/30/2019 • 3 minutes to read • [Edit Online](#)

This article describes the steps to interactively install Git client tools and Visual Studio Code.

- Install [Git](#)
- Install [Visual Studio Code](#)
- Install [Docs Authoring Pack](#)

IMPORTANT

If you're making only minor changes to an article, you *do not* need to complete the steps in this article and can continue directly to the [quick changes workflow](#).

Major contributors are encouraged to complete these steps, which enable you to use the [major/long-running changes workflow](#). Even if you have write permissions in the main repository, *we highly recommend (and this guide assumes) that you fork and clone the repository*, so that you have read/write permissions to store your proposed changes in your fork.

Install Git client tools

Install the latest version of [Software Freedom Conservancy's Git client tools](#) for your platform.

- [Git for Windows](#). This install includes the Git version control system and Git Bash, the command-line app that you use to interact with your local Git repository.
- Git for Mac is provided as part of the Xcode Command Line Tools. Simply run `git` from the command line. You will be prompted to install the command line tools if needed. You can also download [Git for Mac](#) from the Software Freedom Conservancy.
- [Git for Linux and Unix](#)

If you prefer a graphical user interface (GUI) over a command-line interface (CLI), see [Software Freedom Conservancy's available GUI Clients page](#), [GitHub's GitHub Desktop](#), or [Visual Studio Code](#) for some popular options.

Follow the instructions for your chosen client for installation and configuration.

In the next article, you will [Set up a local Git repository](#).

Additional Git resources are available here: [Git terminology](#) | [Git basics](#) | [Learning Git and GitHub](#)

Understand Markdown editors

Markdown is a lightweight markup language that is both easy to read and easy to learn. Therefore, it has rapidly become an industry standard. To write articles in Markdown, we recommend that you first download and install a Markdown editor. [Visual Studio Code](#) is the preferred tool for editing Markdown at Microsoft. [Atom](#) is another popular tool for editing Markdown.

Markdown text is saved into files with .md extension.

Additional details on how to write with Markdown, including Markdown basics and the features supported by Open Publishing Services (OPS) custom Markdown extensions, are covered in the [How to use Markdown for writing Docs](#) and [Markdown Reference for OPS](#) articles.

Visual Studio Code

[Visual Studio Code](#), also known as VS Code, is a lightweight editor that works on Windows, Linux, and Mac. It includes git integration, and support for extensions.

Download and install [VS Code](#). The VS Code home page should detect your operating system correctly.

- [Windows](#)
- [Mac](#)
- [Linux](#)

TIP

To launch VS Code and open the current folder, run the command `code .` in the command line or bash shell. If the current folder is part of a local git repo, the GitHub integration appears in Visual Studio Code automatically.

Docs Authoring Pack

Install the Docs Authoring Pack for Visual Studio Code. This set of extensions includes basic authoring assistance for help when writing Markdown, and a preview feature, so that you can see what the Markdown looks like in the style of the docs.microsoft.com site.

Visit this [marketplace page](#) and select **Install**, or search for `docsmsft.docs-authoring-pack` in your extensions list in the VS Code window.

The Docs Authoring Pack is accessible by pressing Alt+M inside of VS Code. The toolbar is hidden by default but can be shown. Edit the VS Code settings (Control+comma) and adding user setting `"markdown.showToolbar": true` to show the toolbar.

For more information, see the [Docs Authoring Pack](#) page.

Next steps

Now you are ready to [Set up a local Git repository](#).

Set up Git repository locally for documentation

7/30/2019 • 5 minutes to read • [Edit Online](#)

This article describes the steps to set up a git repository on your local machine, with the intent to contribute to Microsoft documentation. Contributors may use a locally cloned repository to add new articles, do major edits on existing articles, or change artwork.

You run these one-time setup activities to start contributing:

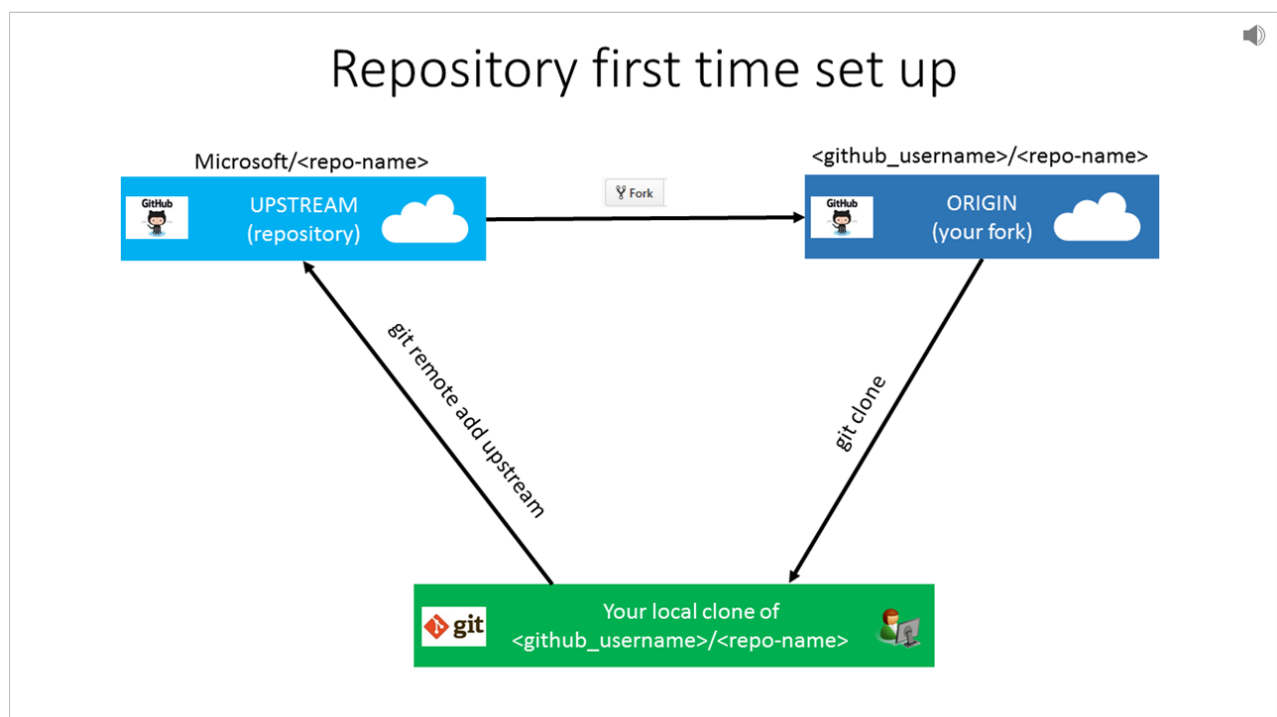
- Determine the appropriate repository
- Fork the repository to your GitHub account
- Choose a local folder for the cloned files
- Clone the repository to your local machine
- Configure the upstream remote value

IMPORTANT

If you're making only minor changes to an article, you *do not* need to complete the steps in this article. You can continue directly to the [quick changes workflow](#).

Overview

To contribute to Microsoft's documentation site, you can make and edit Markdown files locally by cloning the corresponding documentation repository. Microsoft requires you to fork the appropriate repository into your own GitHub account so that you have read/write permissions there to store your proposed changes. Then you use pull requests to merge changes into the read-only central shared repository.

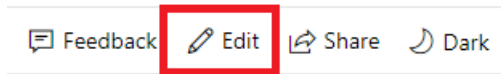


If you're new to GitHub, watch the following video for a conceptual overview of the forking and cloning process:

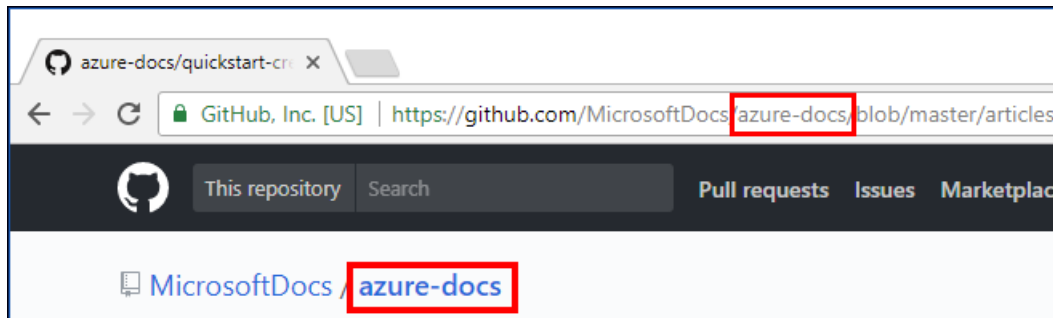
Determine the repository

Documentation hosted at docs.microsoft.com resides in several different repositories at github.com.

1. If you are unsure of which repository to use, then visit the article on docs.microsoft.com using your web browser. Select the **Edit** link (pencil icon) on the upper right of the article.



2. That link takes you to github.com location for the corresponding Markdown file in the appropriate repository. Note the URL to view the repository name.



For example, these popular repositories are available for public contributions:

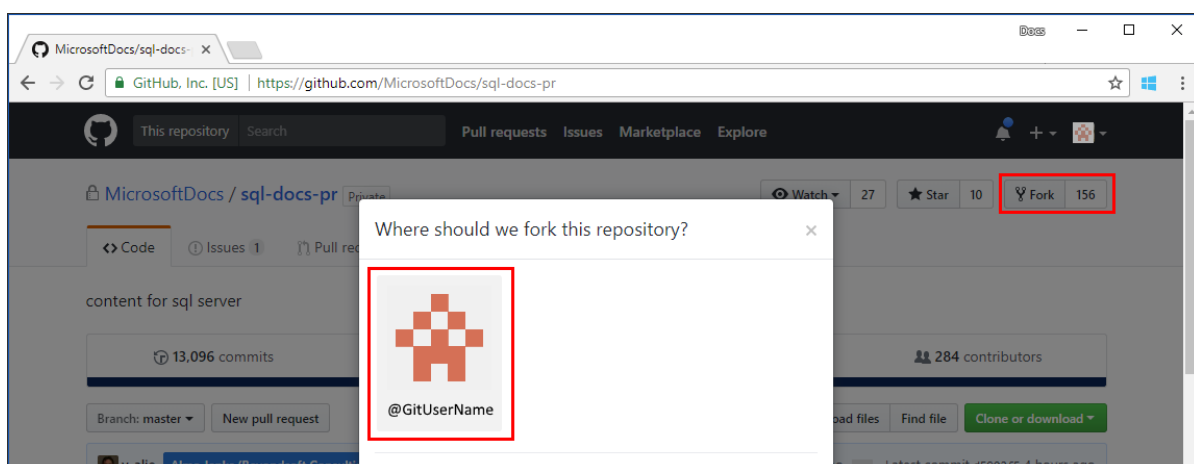
- Azure documentation <https://github.com/MicrosoftDocs/azure-docs>
- SQL Server documentation <https://github.com/MicrosoftDocs/sql-docs>
- Visual Studio documentation <https://github.com/MicrosoftDocs/visualstudio-docs>
- .NET Documentation <https://github.com/dotnet/docs>
- Azure .Net SDK documentation <https://github.com/azure/azure-docs-sdk-dotnet>
- ConfigMgr documentation [<https://github.com/MicrosoftDocs/SCCMdocs>]
(<https://github.com/MicrosoftDocs/SCCMdocs/>)

Fork the repository

Using the appropriate repository, create a fork of the repository into your own GitHub account by using the GitHub website.

A personal fork is required since all main documentation repositories provide read-only access. To make changes, you must submit a [pull request](#) from your fork into the main repository. To facilitate this process, you first need your own copy of the repository, in which you have write access. A GitHub *fork* serves that purpose.

1. Go to the main repository's GitHub page and click the **Fork** button on the upper right.



2. If you are prompted, select your GitHub account tile as the destination where the fork should be created.
This prompt creates a copy of the repository within your GitHub account, known as a fork.

Choose a local folder

Make a local folder to hold a copy of the repository locally. Some of the repositories can be large; up to 5 GB for azure-docs for example. Choose a location with available disk space.

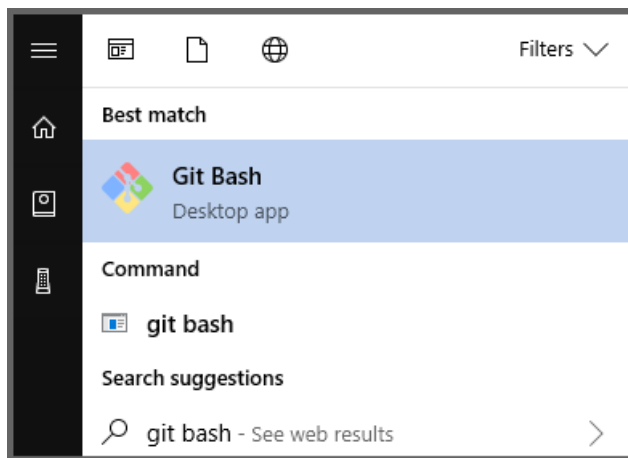
1. Choose a folder name should be easy for you to remember and type. For example, consider a root folder

`C:\docs\` or make a folder in your user profile directory `~/Documents/docs/`

IMPORTANT

Avoid choosing a local folder path that is nested inside of another git repository folder location. While it is acceptable to store the git cloned folders adjacent to each other, nesting git folders inside one another causes errors for the file tracking.

2. Launch Git Bash



The default location that Git Bash starts in is typically the home directory (~) or `/c/users/<Windows-user-account>/` on Windows OS.

To determine the current directory, type `pwd` at the \$ prompt.

3. Change directory (cd) into the folder that you created for hosting the repository locally. Note that Git Bash uses the Linux convention of forward-slashes instead of back-slashes for folder paths.

For example, `cd /c/docs/` or `cd ~/Documents/docs/`

Create a local clone

Using Git Bash, prepare to run the **clone** command to pull a copy of a repository (your fork) down to your device on the current directory.

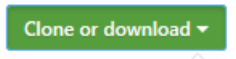
Authenticate by using Git Credential Manager

If you installed the latest version of Git for Windows and accepted the default installation, Git Credential Manager is enabled by default. Git Credential Manager makes authentication much easier because you don't need to recall your personal access token when re-establishing authenticated connections and remotes with GitHub.

1. Run the **clone** command, by providing the repository name. Cloning downloads (clone) the forked repository on your local computer.

TIP

You can get your fork's GitHub URL for the clone command from the **Clone or download** button in the GitHub UI:

A green button with the text "Clone or download" and a small downward-pointing arrow.

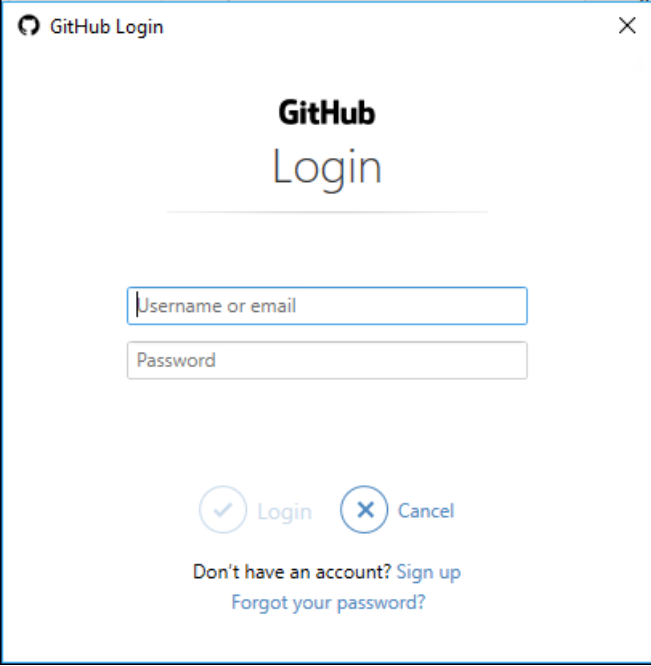
Be sure to specify the path to *your fork* during the cloning process, not the main repository from which you created the fork. Otherwise, you cannot contribute changes. Your fork is referenced through your personal GitHub user account, such as `github.com/<github-username>/<repo>`.

```
git clone https://github.com/<github-username>/<repo>.git
```

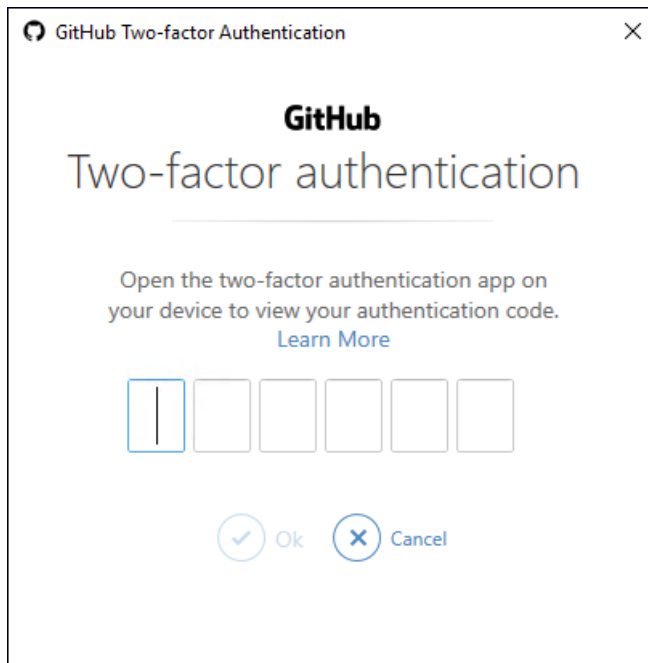
Your clone command should look similar to this example:

```
git clone https://github.com/smithj/azure-docs.git
```

2. When you're prompted, enter your GitHub credentials.

A screenshot of a "GitHub Login" dialog box. It has a title bar with the GitHub logo and the text "GitHub Login" and a close button. The main content area has the "GitHub Login" text centered. Below it are two input fields: "Username or email" and "Password". At the bottom, there are two buttons: "Login" (with a checkmark icon) and "Cancel" (with an 'X' icon). Below the buttons, there is a link "Don't have an account? Sign up" and another link "Forgot your password?".

3. When you're prompted, enter your two-factor authentication code.



NOTE

Your credentials will be saved and used to authenticate future GitHub requests. You only need to do this authentication once per computer.

4. The clone command runs and downloads a copy of the repository files from your fork into a new folder on the local disk. A new folder is made within the current folder. It may take a few minutes, depending on the repository size. You can explore the folder to see the structure once it is finished.

Configure remote upstream

After cloning the repository, set up a read-only remote connection to the main repository named **upstream**. You use the upstream URL to keep your local repository in sync with the latest changes made by others. The **git remote** command is used to set the configuration value. You use the **fetch** command to refresh the branch info from the upstream repository.

1. If you're using **Git Credential Manager**, use the following commands. Replace the `<repo>` and `<organization>` placeholders.

```
cd <repo>
git remote add upstream https://github.com/<organization>/<repo>.git
git fetch upstream
```

2. View the configured values and confirm the URLs are correct. Ensure the **origin** URLs point to your personal fork. Ensure the **upstream** URLs point to the main repository, such as MicrosoftDocs or Azure.

```
git remote -v
```

Example remote output is shown. A fictitious git account named MyGitAccount is configured with a personal access token to access the repo azure-docs:

```
origin https://github.com/MyGitAccount/azure-docs.git (fetch)
origin https://github.com/MyGitAccount/azure-docs.git(push)
upstream https://github.com/MicrosoftDocs/azure-docs.git (fetch)
upstream https://github.com/MicrosoftDocs/azure-docs.git (push)
```

3. If you made a mistake, you can remove the remote value. To remove the upstream value, run the command

```
git remote remove upstream .
```

Next steps

- To learn more about adding and updating content, continue to the [GitHub contribution workflow](#).

Git and GitHub essentials for Docs

7/30/2019 • 5 minutes to read • [Edit Online](#)

Overview

As a contributor to Docs content, you will interact with multiple tools and processes. You'll work in parallel with other contributors on the same project, potentially the exact same content, even at the same time. This is all enabled through Git and GitHub software.

Git is an open-source version control system. It facilitates this type of project collaboration through *distributed version control* of files that live in *repositories*. In essence, Git makes it possible to integrate streams of work done by multiple contributors over time, for a given repository.

GitHub is a web-based hosting service for Git repositories, such as those used to store docs.microsoft.com content. For any project, GitHub hosts the main repository, from which contributors can make copies for their own work.

Git

If you're familiar with centralized version control systems (such as Team Foundation Server, SharePoint, or Visual SourceSafe), you will notice that Git has a unique contribution workflow and terminology to support its distributed model. For instance, there is no file locking that is normally associated with check-out/check-in operations. As a matter of fact, Git is concerned about changes at an even finer level, comparing files byte by byte.

Git also uses a tiered structure to store and manage content for a project:

- **Repository:** Also known as a *repo*, this is the highest unit of storage. A repository contains one or more branches.
- **Branch:** A unit of storage that contains the files and folders that make up a project's content set. Branches separate streams of work (typically referred to as versions). Contributions are always made and scoped to a specific branch. All repositories contain a default branch (typically named "master") and one or more branches that are destined to be merged back into the master branch. The master branch serves as the current version and "single source of truth" for the project. It's the parent from which all other branches in the repository are created.

Contributors interact with Git to update and manipulate repositories at both the local and GitHub levels:

- Locally through tools such as the Git Bash console, which supports Git commands for managing local repositories and communicating with GitHub repositories.
- Via www.github.com, which integrates Git to manage the reconciliation of contributions that flow back into the main repository.

GitHub

NOTE

Although Docs guidance is based on using GitHub, some teams use Visual Studio Team Services to host Git repositories. The Visual Studio Team Explorer client provides a GUI for interacting with Team Services repositories, as an alternative to using Git commands through a command line.

Also, many of the following guidelines were developed as best practices from years of experience in hosting Azure service content in GitHub. They might be required in some Docs repositories.

All workflows begin and end at the GitHub level, where the main repository for any Docs project is stored. The copies that contributors create for their own use are distributed across multiple computers. These copies are eventually reconciled back into the project's main GitHub repository.

Directory organization

As mentioned earlier, a project's default/master branch serves as the current version of content for the project. The content in the master branch--and branches created from it--is loosely aligned with the organization of the articles on the corresponding Docs pages. Subdirectories are used for separation of like content (such as services), media content (such as image files), and "include" files (which enable reuse of content).

You can typically find a main `articles` directory off the root of the repository. The articles directory contains a set of subdirectories. Articles in the subdirectories are formatted as Markdown files that use an `.md` extension. Some repositories that support multiple services use a generic `/articles` subdirectory, such as the [Azure-Docs](#) repository. Others might use a service-specific name, such as the [IntuneDocs](#) repository, which uses `/IntuneDocs`.

Within the root of this directory, you can find general articles that relate to the overall service or product. And typically, you can then find another series of subdirectories that match the features/services or common scenarios. For instance, Azure "virtual machine" articles are in the `/virtual-machines` subdirectory, and Intune "understand and explore" articles are in the `/understand-explore` subdirectory.

Media subdirectory

Each article directory contains a `/media` subdirectory for corresponding media files. Media files contain images used by articles that have image references.

Includes subdirectory

Whenever we have reusable content that is shared across two or more articles, it is placed in an `/includes` subdirectory off the main `articles` directory. In a Markdown file that uses the include file, a corresponding "include" Markdown extension is placed in the location where the include file needs to be referenced.

See [How to use Markdown: Includes](#) for additional guidance.

Markdown file template

For convenience, the root directory of each repository typically contains a Markdown template file named `template.md`. You can use this template file as a "starter file" if you need to create a new article for submission to the repository. The file contains:

- A **metadata header** at the top of the file, delineated by two, 3-hyphen lines. It contains the various tags used for tracking information related to the article. Article metadata enables certain functionality, such as author attribution, contributor attribution, breadcrumbs, and article descriptions. It also includes SEO optimizations and reporting processes that Microsoft uses to evaluate the performance of the content. So the metadata is important!
- A **metadata section** that describes the various metadata tags and values. If you're unsure of the values to use for the metadata section, you can leave them blank or comment them with a leading hashtag (#), and they will be reviewed/completed by the pull request reviewer for the repository.
- Various **examples of using Markdown** to format the elements of an article.
- General **instructions on the use of Markdown extensions**, which you can use for various types of alerts.
- Examples of **embedding video** by using an `iframe`.
- General **instructions on the use of docs.microsoft.com extensions**, which you can use for special controls such as buttons and selectors.

Pull requests

A *pull request* provides a convenient way for a contributor to propose a set of changes that will be applied to the default branch. The changes (also known as *commits*) are stored in a contributor's branch, so GitHub can first

model the impact of *merging* them into the default branch. A pull request also serves as a mechanism to provide the contributor with feedback from a build/validation process, the pull request reviewer, to resolve potential issues or questions before the changes are merged into the default branch.

There are two ways to contribute by pull request, depending on the size of changes that you want to propose. We will cover this in detail later, in the [GitHub workflow](#) section of this guide.

GitHub contribution workflow for major or long-running changes

7/30/2019 • 9 minutes to read • [Edit Online](#)

IMPORTANT


All repositories that publish to docs.microsoft.com have adopted either the [Microsoft Open Source Code of Conduct](#) or the [.NET Foundation Code of Conduct](#). For more information, see the [Code of Conduct FAQ](#). Or contact opencode@microsoft.com, or conduct@dotnetfoundation.org with any questions or comments.

Minor corrections or clarifications to documentation and code examples in public repositories are covered by the [docs.microsoft.com Terms of Use](#). New or significant changes will generate a comment in the pull request, asking you to submit an online Contribution License Agreement (CLA) if you are not an employee of Microsoft. You will need to complete the online form before your pull request can be merged.

Overview

This workflow is suitable for a contributor who needs to make a major change or will be a frequent contributor to a repository. Frequent contributors typically have ongoing (long-running) changes, which go through multiple build/validation/staging cycles or span multiple days before pull request sign-off and merge.

Examples of these types of contributions include:

- **Making a large contribution.** For instance, you might make contributions (additions, changes, or deletions) that span multiple articles and need to be committed and tested as one unit of work in a single pull request.
- **Creating and publishing a new article**, which typically requires a more robust local editor.
- **Adding new images or updating images**, which typically requires simultaneous creation of a new  subdirectory, image files, updates to image links in articles, and previewing markdown files in a local editor to test image rendering.
- **Updating an article over a period of days before you publish.** In these cases, you typically need to do regular integration of other changes that occur in the master branch. This integration is easier via Git Bash and local editing. You also run the risk of losing your edits if you do this via the GitHub UI and wait before you commit the changes.
- **Making continual updates to the same article after a pull request has been opened** (unless you are comfortable doing this via the GitHub UI). Using the GitHub UI has the potential to create multiple outstanding pull requests for the same file, which may conflict with one another.

Terminology

Before you start, let's review some of the Git/GitHub terms and monikers used in this workflow. Don't worry about understanding them now. Just know that you will be learning about them, and you can refer back to this section when you need to verify a definition.

NAME	DESCRIPTION
------	-------------

NAME	DESCRIPTION
fork	Normally used as a noun, when referring to a copy of a main GitHub repository. In practice, a fork is just another repository. But it's special in the sense that GitHub maintains a connection back to the main/parent repository. It's sometimes used as a verb, as in "You must fork the repository first."
remote	A named connection to a remote repository, such as the "origin" or "upstream" remote. Git refers to this as remote because it is used to reference a repository that's hosted on another computer. In this workflow, a remote is always a GitHub repository.
origin	The name assigned to the connection between your local repository and the repository from which it was cloned. In this workflow, origin represents the connection to your fork. It's sometimes used as a moniker for the origin repository itself, as in "Remember to push your changes to origin."
upstream	Like the origin remote, upstream is a named connection to another repository. In this workflow, upstream represents the connection between your local repository and the main repository, from which your fork was created. It's sometimes used as a moniker for the upstream repository itself, as in "Remember to pull the changes from upstream."

Workflow

IMPORTANT

If you haven't already, you must complete the steps in the [Setup](#) section. This section walks you through setting up your GitHub account, installing Git Bash and a Markdown editor, creating a fork, and setting up your local repository. If you are unfamiliar with Git and GitHub concepts such as a repository or branch, please first review [Git and GitHub fundamentals](#).

In this workflow, changes flow in a repetitive cycle. Starting from your device's local repository, they flow back up to your GitHub fork, into the main GitHub repository, and back down locally again as you incorporate changes from other contributors.

Use GitHub flow

Recall from [Git and GitHub fundamentals](#) that a Git repository contains a master branch, plus any additional work-in-progress branches that have not been integrated into master. Whenever you introduce a set of logically related changes, it's a best practice to create a *working branch* to manage your changes through the workflow. We refer to it here as a working branch because it's a workspace to iterate/refine changes, until they can be integrated back into the master branch.

Isolating related changes to a specific branch allows you to control and introduce those changes independently, targeting them to a specific release time in the publishing cycle. In reality, depending on the type of work you do, you can easily end up with several working branches in your repository. It's not uncommon to be working on multiple branches at the same time, each representing a different project.

TIP

Making your changes in the master branch is *not* a good practice. Imagine that you use the master branch to introduce a set of changes for a timed feature release. You finish the changes and are waiting to release them. Then in the interim, you have an urgent request to fix something, so you make the change to a file in the master branch and then publish the change. In this example, you inadvertently publish both the fix *and* the changes that you were holding for release on a specific date.

Now let's create a new working branch in your local repository, to capture your proposed changes. Each git client is different, so consult the help for your preferred client. You can see an overview of the process in the GitHub Guide on [GitHub flow](#).

Pull request processing

The previous section walked you through the process of submitting proposed changes, by bundling them in a new pull request (PR) that is added to the destination repository's PR queue. A pull request enables GitHub's collaboration model, by asking for the changes from your working branch to be pulled and merged into another branch. In most cases, that other branch is the default/master branch in the main repository.

Validation

Before your pull request can be merged into its destination branch, it might be required to pass through one or more PR validation processes. Validation processes can vary depending on the scope of proposed changes and the rules of the destination repository. After your pull request is submitted, you can expect one or more of the following to happen:

- **Mergeability:** A baseline GitHub mergeability test occurs first, to verify whether the proposed changes in your branch are not in conflict with the destination branch. If the pull request indicates that this test failed, you must reconcile the content that is causing the merge conflict before processing can continue.
- **CLA:** If you are contributing to a public repository and are not a Microsoft employee, depending on the magnitude of the proposed changes, you might be asked to complete a short Contribution License Agreement (CLA) the first time you submit a pull request to that repository. After the CLA step is cleared, your pull request is processed.
- **Labeling:** Labels are automatically applied to your pull request, to indicate the state of your pull request as it passes through the validation workflow. For instance, new pull requests might automatically receive the "do-not-merge" label, indicating that the pull request has not yet completed the validation, review, and sign-off steps.
- **Validation and build:** Automated checks verify whether your changes pass validation tests. The validation tests might yield warnings or errors, requiring you to make changes to one or more files in your pull request before it can be merged. The validation test results are added as a comment in your pull request for your review, and they might be sent to you in e-mail.
- **Staging:** The article pages affected by your changes are automatically deployed to a staging environment for review upon successful validation and build. Preview URLs appear in a PR comment.
- **Auto-merge:** The pull request might be automatically merged, if it passes validation testing and certain criteria. In this case, you don't need to take any further action.

Review and sign-off

After all PR processing is completed, you should review the results (PR comments, preview URLs, etc.) to determine if additional changes to its files are required before you sign off for merging. If a PR reviewer has reviewed your pull request, they can also provide feedback via comments if there are outstanding issues/questions to be resolved prior to merge.

Comment automation enables read-level users (users who don't have write permissions in a repo) to perform a write-level action, by assigning the appropriate label to a pull request. If you are working in a repository where

comment automation has been implemented, use the hashtag comments listed in the following table to assign labels, change labels, or close a pull request. Microsoft employees will also be notified via e-mail for review and sign-off of public repository PRs, whenever changes are proposed to articles for which you are the author.

HASHTAG COMMENT	WHAT IT DOES	REPO AVAILABILITY
<code>#sign-off</code>	<p>When the author of an article types the <code>#sign-off</code> comment in the comment stream, the ready-to-merge label is assigned. This label lets the reviewers in the repo know when a pull request is ready for review/merge.</p> <p>If a contributor who is <i>not</i> the listed author tries to sign off on a public repo pull request by using the <code>#sign-off</code> comment, a message is written to the pull request indicating that only the author can assign the label.</p>	Public and private
<code>#hold-off</code>	<p>Authors can type <code>#hold-off</code> in a PR comment to remove the ready-to-merge label--in case they change their mind or make a mistake. In the private repo, this assigns the do-not-merge label.</p>	Public and private
<code>#please-close</code>	<p>Authors can type <code>#please-close</code> in the comment stream to close the pull request if they decide not to have the changes merged.</p>	Public

When the pull request is issue-free and signed off, your changes are merged back into the parent branch and the pull request is closed.

Publishing

Remember, your pull request has to be merged by a PR reviewer before the changes can be included in the next scheduled publishing run. Pull requests are normally reviewed/merged in the order of submission. If your pull request requires merging for a specific publishing run, you will need to work with your PR reviewer ahead of time to ensure that merging happens prior to publishing.

After your contributions are approved and merged, the docs.microsoft.com publishing process picks them up. Depending on the team that manages the repository you are contributing to, publishing times can vary. Articles published under the following paths are normally deployed at approximately 10:30 AM and 3:30 PM Pacific Time, Monday-Friday:

- <https://docs.microsoft.com/azure/>
- <https://docs.microsoft.com/aspnet/>
- <https://docs.microsoft.com/dotnet/>
- <https://docs.microsoft.com/enterprise-mobility-security>

It can take up to 45 minutes for articles to appear online after publishing. After your article is published, you can verify your changes at the appropriate URL:

`https://docs.microsoft.com/<path-to-your-article-without-the-md-extension>`.

Next steps

That's it! You've made a contribution to docs.microsoft.com content!

- To learn more about topics such as Markdown and Markdown extensions syntax, continue to the "Writing essentials" section.

How to use Markdown for writing Docs

7/30/2019 • 10 minutes to read • [Edit Online](#)

[Docs.microsoft.com](https://docs.microsoft.com) articles are written in a lightweight markup language called [Markdown](#), which is both easy to read and easy to learn. Because of this, it has quickly become an industry standard. The docs site uses the [Markdig flavor](#) of Markdown.

Markdown basics

Headings

To create a heading, you use a hash mark (#), as follows:

```
# This is heading 1
## This is heading 2
### This is heading 3
#### This is heading 4
```

Headings should be done using atx-style, that is, use 1-6 hash characters (#) at the start of the line to indicate a heading, corresponding to HTML headings levels H1 through H6. Examples of first- through fourth-level headers are used above.

There **must** be only one first-level heading (H1) in your topic, which will be displayed as the on-page title.

If your heading finishes with a # character, you need to add an extra # character in the end in order for the title to render correctly. For example, # Async Programming in F# #.

Second-level headings will generate the on-page TOC that appears in the "In this article" section underneath the on-page title.

Bold and italic text

To format text as **bold**, you enclose it in two asterisks:

```
This text is **bold**.
```

To format text as *italic*, you enclose it in a single asterisk:

```
This text is *italic*.
```

To format text as both ***bold and italic***, you enclose it in three asterisks:

```
This is text is both ***bold and italic***.
```

Blockquotes

Blockquotes are created using the > character:

```
> The drought had lasted now for ten million years, and the reign of the terrible lizards had long since ended. Here on the Equator, in the continent which would one day be known as Africa, the battle for existence had reached a new climax of ferocity, and the victor was not yet in sight. In this barren and desiccated land, only the small or the swift or the fierce could flourish, or even hope to survive.
```

The preceding example renders as follows:

The drought had lasted now for ten million years, and the reign of the terrible lizards had long since ended. Here on the Equator, in the continent which would one day be known as Africa, the battle for existence had reached a new climax of ferocity, and the victor was not yet in sight. In this barren and desiccated land, only the small or the swift or the fierce could flourish, or even hope to survive.

Lists

Unordered list

To format an unordered/bulleted list, you can use either asterisks or dashes. For example, the following Markdown:

```
- List item 1  
- List item 2  
- List item 3
```

will be rendered as:

- List item 1
- List item 2
- List item 3

To nest a list within another list, indent the child list items. For example, the following Markdown:

```
- List item 1  
  - List item A  
  - List item B  
- List item 2
```

will be rendered as:

- List item 1
 - List item A
 - List item B
- List item 2

Ordered list

To format an ordered/stepwise list, you use corresponding numbers. For example, the following Markdown:

```
1. First instruction  
1. Second instruction  
1. Third instruction
```

will be rendered as:

1. First instruction
2. Second instruction
3. Third instruction

To nest a list within another list, indent the child list items. For example, the following Markdown:

```
1. First instruction
  1. Sub-instruction
    1. Sub-instruction
1. Second instruction
```

will be rendered as:

1. First instruction
 - a. Sub-instruction
 - b. Sub-instruction
2. Second instruction

Note that we use '1.' for all entries. It makes diffs easier to review when later updates include new steps or remove existing steps.

Tables

Tables are not part of the core Markdown specification, but GFM supports them. You can create tables by using the pipe (|) and hyphen (-) characters. Hyphens create each column's header, while pipes separate each column. Include a blank line before your table so it's rendered correctly.

For example, the following Markdown:

```
| Fun | With | Tables |
| :-----: | :-----: | :-----: |
| left-aligned column | right-aligned column | centered column |
| $100 | $100 | $100 |
| $10 | $10 | $10 |
| $1 | $1 | $1 |
```

Will be rendered as:

FUN	WITH	TABLES
left-aligned column	right-aligned column	centered column
\$100	\$100	\$100
\$10	\$10	\$10
\$1	\$1	\$1

For more information on creating tables, see:

- GitHub's [Organizing information with tables](#).
- The [Markdown Tables Generator](#) web app.
- [Adam Pritchard's Markdown Cheatsheet](#).
- [Michel Fortin's Markdown Extra](#).
- [Convert HTML tables to Markdown](#).

Links

The Markdown syntax for an inline link consists of the [link text] portion, which is the text that will be hyperlinked, followed by the (file-name.md) portion, which is the URL or file name that's being linked to:


```
[link text](file-name.md)
```

For more information on linking, see:

- The [Markdown syntax guide](#) for details on Markdown's base linking support.
- The [Links](#) section of this guide for details on the additional linking syntax that Markdig provides.

Code snippets

Markdown supports the placement of code snippets both inline in a sentence and as a separate "fenced" block between sentences. For details, see:

- [Markdown's native support for code blocks](#)
- [GFM support for code fencing and syntax highlighting](#)

Fenced code blocks are an easy way to enable syntax highlighting for your code snippets. The general format for fenced code blocks is:

```
```alias
...
your code goes in here
...
```
```

The alias after the initial three backtick (```) characters defines the syntax highlighting to be used. The following is a list of commonly used programming languages in Docs content and the matching label:

These languages have friendly name support and most have language highlighting.

| NAME | MARKDOWN LABEL |
|------------------|--------------------|
| .NET Console | dotnetcli |
| ASP.NET (C#) | aspx-csharp |
| ASP.NET (VB) | aspx-vb |
| AzCopy | azcopy |
| Azure CLI | azurecli |
| Azure PowerShell | azurepowershell |
| Bash | bash |
| C++ | cpp |
| C++/CX | cppcx |
| C++/WinRT | cppwinrt |
| C# | csharp |
| C# in browser | csharp-interactive |
| Console | console |

| NAME | MARKDOWN LABEL |
|-------------------------------------|-----------------|
| CSSHTML | cshtml |
| DAX | dax |
| Docker | dockerfile |
| F# | fsharp |
| Go | go |
| HTML | html |
| HTTP | http |
| Java | java |
| JavaScript | javascript |
| JSON | json |
| Kusto Query Language | kusto |
| Markdown | md |
| Objective-C | objc |
| OData | odata |
| PHP | php |
| PowerApps (dot decimal separator) | powerapps-dot |
| PowerApps (comma decimal separator) | powerapps-comma |
| PowerShell | powershell |
| Python | python |
| Q# | qsharp |
| R | r |
| Ruby | ruby |
| SQL | sql |
| Swift | swift |
| TypeScript | typescript |

| NAME | MARKDOWN LABEL |
|------|----------------|
| VB | vb |
| XAML | xaml |
| XML | xml |

The `csharp-interactive` name specifies the C# language, and the ability to run the samples from the browser. These snippets are compiled and executed in a Docker container, and the results of that program execution are displayed in the user's browser window.

Example: C#

Markdown

```
```csharp
// Hello1.cs
public class Hello1
{
 public static void Main()
 {
 System.Console.WriteLine("Hello, World!");
 }
}
```
```

Render

```
// Hello1.cs
public class Hello1
{
    public static void Main()
    {
        System.Console.WriteLine("Hello, World!");
    }
}
```

Example: SQL

Markdown

```
```sql
CREATE TABLE T1 (
 c1 int PRIMARY KEY,
 c2 varchar(50) SPARSE NULL
);
```
```

Render

```
CREATE TABLE T1 (
    c1 int PRIMARY KEY,
    c2 varchar(50) SPARSE NULL
);
```

OPS custom Markdown extensions

NOTE

Open Publishing Services (OPS) implements a Markdig Parser for Markdown, which is highly compatible with GitHub Flavored Markdown (GFM). Markdig adds some functionality through Markdown extensions. As such, selected articles from the full OPS Authoring Guide are included in this guide for reference. (For example, see "Markdig and Markdown extensions" and "Code snippets" in the table of contents.)

Docs articles use GFM for most article formatting, such as paragraphs, links, lists, and headings. For richer formatting, articles can use Markdig features such as:

- Note blocks
- Include files
- Selectors
- Embedded videos
- Code snippets/samples

For the complete list, refer to "Markdig and Markdown extensions" and "Code snippets" in the table of contents.

Note blocks

You can choose from four types of note blocks to draw attention to specific content:

- NOTE
- WARNING
- TIP
- IMPORTANT

In general, note blocks should be used sparingly because they can be disruptive. Although they also support code blocks, images, lists, and links, try to keep your note blocks simple and straightforward.

Examples:

```
> [!NOTE]
> This is a NOTE

> [!WARNING]
> This is a WARNING

> [!TIP]
> This is a TIP

> [!IMPORTANT]
> This is IMPORTANT
```

These render as follows:

NOTE

This is a NOTE

WARNING

This is a WARNING

TIP

This is a TIP

IMPORTANT

This is IMPORTANT

Include files

When you have reusable text or image files that need to be included in article files, you can use a reference to the "include" file via the Markdig file include feature. This feature instructs OPS to include the file in your article file at build time, making it part of your published article. Three types of include references are available to help you reuse content:

- Inline: Reuse a common text snippet inline with within another sentence.
- Block: Reuse an entire Markdown file as a block, nested within a section of an article.
- Image: This is how standard image inclusion is implemented in Docs.

An inline or block include file is just a simple Markdown (.md) file. It can contain any valid Markdown. All include Markdown files should be placed in a [common /includes subdirectory](#), in the root of the repository. When the article is published, the included file is seamlessly integrated into it.

Here are requirements and considerations for include files:

- Use an include file whenever you need the same text to appear in multiple articles.
- Use a block include reference for significant amounts of content--a paragraph or two, a shared procedure, or a shared section. Do not use them for anything smaller than a sentence.
- Include references won't be rendered in the GitHub rendered view of your article, because they rely on Markdig extensions. They'll be rendered only after publication.
- Ensure that all the text in an include file is written in complete sentences or phrases that do not depend on preceding text or following text in the article that references the include file. Ignoring this guidance creates an untranslatable string in the article that breaks the localized experience.
- Don't embed include references within other include files. They are not supported.
- Place media files in a media folder that's specific to the include subdirectory--for instance, the `<repo>/includes/media` folder. The media directory should not contain any images in its root. If the include file does not have images, a corresponding media directory is not required.
- As with regular articles, don't share media between include files. Use a separate file with a unique name for each include file and article. Store the media file in the media folder that's associated with the include file.
- Don't use an include file as the only content of an article. Include files are meant to be supplemental to the content in the rest of the article.

Example:

```
[!INCLUDE[sample include file](../includes/sampleinclude.md)]
```

Selectors

Use selectors in technical articles when you author multiple flavors of the same article, to address differences in implementation across technologies or platforms. This is typically most applicable to our mobile platform content for developers. There are currently two different types of selectors in Markdig, a single selector and a multi-selector.

Because the same selector Markdown goes in each article in the selection, we recommend placing the selector for your article in an include file. Then you can reference that include file in all your articles that use the same selector.

The following shows an example selector:

```
> [!div class="op_single_selector"]  
- [macOS](../docs/core/tutorials/using-on-macos.md)  
- [Windows](../docs/core/tutorials/with-visual-studio.md)
```

You can see an example of selectors in action at the [Azure docs](#).

Code include references

Markdig supports advanced inclusion of code in an article, via its code snippet extension. It provides advanced rendering that builds on GFM features such as programming language selection and syntax coloring, plus nice features such as:

- Inclusion of centralized code samples/snippets from an external repository.
- Tabbed UI to show multiple versions of code samples in different languages.

Gotchas and troubleshooting

Alt text

Alt text that contains underscores won't be rendered properly. For example, instead of using this:

```
![ADextension_2FA_Configure_Step4](../media/bogusfilename/ADextension_2FA_Configure_Step4.PNG)
```

Escape the underscores like this:

```
![ADextension\_2FA\_Configure\_Step4](../media/bogusfilename/ADextension_2FA_Configure_Step4.PNG)
```

Apostrophes and quotation marks

If you copy from Word into a Markdown editor, the text might contain "smart" (curly) apostrophes or quotation marks. These need to be encoded or changed to basic apostrophes or quotation marks. Otherwise, you end up with things like this when the file is published: Itâ€™s

Here are the encodings for the "smart" versions of these punctuation marks:

- Left (opening) quotation mark: `“`
- Right (closing) quotation mark: `”`
- Right (closing) single quotation mark or apostrophe: `’`
- Left (opening) single quotation mark (rarely used): `‘`

Angle brackets

It is common to use angle brackets to denote a placeholder. When you do this in text (not code), you must encode the angle brackets. Otherwise, Markdown thinks that they're intended to be an HTML tag.

For example, encode `<script name>` as `<script name>`

Markdown flavor

The docs.microsoft.com site backend uses Open Publishing Services (OPS) which supports [CommonMark](#) compliant markdown parsed through the [Markdig](#) parsing engine. This markdown flavor is mostly compatible with [GitHub Flavored Markdown \(GFM\)](#), as most docs are stored in GitHub and can be edited there. Additional

functionality is added through Markdown extensions.

See also:

Markdown resources

- [Introduction to Markdown](#)
- [Docs Markdown cheat sheet](#)
- [GitHub's Markdown Basics](#)
- [The Markdown Guide](#)

Markdown Reference

7/30/2019 • 13 minutes to read • [Edit Online](#)

Markdown is a lightweight markup language with plain text formatting syntax. The Docs platform supports the CommonMark standard for Markdown, plus some custom Markdown extensions designed to provide richer content on docs.microsoft.com. This article provides an alphabetical reference for using Markdown for docs.microsoft.com.

You can use any text editor to author Markdown. For an editor that facilitates inserting both standard Markdown syntax and custom Docs extensions, we recommend [VS Code](#) with the [Docs Authoring Pack](#) installed.

Docs uses the Markdig Markdown engine. You can test the rendering of Markdown in Markdig vs. other engines at <https://babelmark.github.io/>.

Alerts (Note, Tip, Important, Caution, Warning)

Alerts are a Docs Markdown extension to create block quotes that render on docs.microsoft.com with colors and icons that indicate the significance of the content. The following alert types are supported:

```
> [!NOTE]
> Information the user should notice even if skimming.

> [!TIP]
> Optional information to help a user be more successful.

> [!IMPORTANT]
> Essential information required for user success.

> [!CAUTION]
> Negative potential consequences of an action.

> [!WARNING]
> Dangerous certain consequences of an action.
```

These alerts look like this on docs.microsoft.com:

NOTE

Information the user should notice even if skimming.

TIP

Optional information to help a user be more successful.

IMPORTANT

Essential information required for user success.

Caution

Negative potential consequences of an action.

WARNING

Dangerous certain consequences of an action.

Code snippets

You can embed code snippets in your Markdown files:

```
[!code-<language>[<name>](<codepath><queryoption><queryoptionvalue> "<title>")]
```

Headings

Docs supports six levels of Markdown headings:

```
# This is a first level heading (H1)

## This is a second level heading (H2)

...

##### This is a sixth level heading (H6)
```

- There must be a space between the last `#` and heading text.
- Each Markdown file must have one and only one H1.
- The H1 must be the first content in the file after the YAML metadata block.
- H2s automatically appear in the right-hand navigating menu of the published file. Lower-level headings do not, so use H2s strategically to help readers navigate your content.
- HTML headings, such as `<h1>`, are not recommended and in some cases will cause build warnings.
- You can link to individual headings in a file via [bookmarks](#).

HTML

Although Markdown supports inline HTML, HTML is not recommended for publishing to Docs, and except for a limited list of values will cause build errors or warnings.

Images

The syntax to include an image is:

```
![[alt text]](<folderPath>)

Example:
![[alt text for image](../images/Introduction.png)]
```

Where `alt text` is a brief description of the image and `<folder path>` is a relative path to the image. Alternate text is required for screen readers for the visually impaired. It is also useful if there is a site bug where the image cannot render.

Images should be stored in a `/media` folder within your doc set. The following file types are supported by default for images:

- .jpg

- .png

You can add support for other image types by adding them as resources to the docfx.json file for your doc set.

Links

In most cases, Docs uses standard Markdown links to other files and pages. The types of links are described in subsections below.

TIP

The Docs Authoring Pack for VS Code can help insert relative links and bookmarks correctly without the tedium of figuring out the paths!

IMPORTANT

Do not include locale codes, such as en-us, in your links to Microsoft sites. Hard-coded locale codes prevent localized content from rendering, which is a bad customer experience for users in other locales and incurs significant localization costs. When you copy a URL from a browser, the locale code is included by default, so you need to manually delete it when you create your link. For example, use:

```
[Microsoft](https://www.microsoft.com)
```

Not:

```
[Microsoft](https://www.microsoft.com/en-us/)
```

Relative links to files in the same doc set

A relative path is the path to the target file relative to the current file. In Docs, you can use a relative path to link to another file within the same doc set. The syntax for a relative path is as follows:

```
[link text](../../folder/filename.md)
```

Where `../` indicates one level above in the hierarchy.

- The relative path will be resolved during the build, including removal of the .md extension.
- You can use `../` to link to a file in the parent folder, but that file has to be in the same doc set. You cannot use `../` to link to a file in another doc set folder.
- Docs also supports a special form of relative path that starts with `~` (for example, `~/foo/bar.md`). This syntax indicates a file relative to the root folder of a doc set. This kind of path is also validated and resolved during the build.

IMPORTANT

Include the file extension in the relative path. Build validates the existence of the target file of that relative path. If relative path does not include file extension, it is likely build will report a warning of broken link. For example, use:

```
[link text](../../folder/filename.md)
```

Not:

```
[link text](../../folder/filename)
```

Site relative links to other files on Docs

```
[Azure and Linux](/articles/virtual-machines/linux/overview)
```

Do not include the file extension (.md). This links to the Linux overview file from outside the Azure "articles" doc set.

Links to external sites

```
[Microsoft](https://www.microsoft.com)
```

URL-based link to another web page (must include https://).

Bookmark links

Bookmark link to a heading in another file in the same repo. For example:

```
[Managed Disks](../../linux/overview.md#managed-disks)
```

Bookmark link to a heading in the current file:

```
[Managed Disks](#managed-disks)
```

Use a hash mark `#` followed by the words of the heading. To change the heading text into link text:

- Use all lowercase characters
- Remove punctuation
- Replace spaces with dashes

For example, if the heading name is "2.2 Security concerns", then the bookmark link text will be "#22-security-concerns".

Explicit anchor links

Explicit anchor links using the `<a>` HTML tag are **not required or recommended** except in hub and landing pages. Use bookmarks as described above in general Markdown files. For hub and landing pages, use anchors as follows:

```
## <a id="AnchorText"> </a>Header text OR ## <a name="AnchorText"> </a>Header text
```

To link to explicit anchors, use the following syntax:

```
To go to a section on the same page:  
[text](#AnchorText)
```

```
To go to a section on another page.  
[text](FileName.md#AnchorText)
```

XREF (cross reference) links

To link to auto-generated API references pages in the current doc set or other doc sets, use XREF links with the unique ID (UID).

NOTE

To reference API reference pages in other doc sets, you need to add `xrefService` configuration in `docfx.json` file.

```
"build": {  
  ...  
  "xrefService": [ "https://xref.docs.microsoft.com/query?uid={uid}" ]  
}
```

The UID equates to the fully qualified class and member name. If you add a `*` after the UID, the link then represents the overload page and not a specific API. For example, use `List<T>.BinarySearch*` to link to the `BinarySearch` Method page instead of linking to a specific overload such as `List<T>.BinarySearch(T, IComparer<T>)`.

You can use one of the following syntaxes:

- Auto-link: `<xref:UID>` or `<xref:UID?displayProperty=nameWithType>`

The optional `displayProperty` query parameter produces a fully qualified link text. By default, link text shows only the member or type name.

- Markdown link: `[link text](xref:UID)`

Use when you want to customize the link text displayed.

Examples:

- `<xref:System.String>` renders as "String".
- `<xref:System.String?displayProperty=nameWithType>` renders as "System.String".
- `[String class](xref:System.String)` renders as "String class".

Right now, there is no easy way to find the UIDs. The best way to find the UID for an API is to view the source for the API page you want to link to and find the `ms.assetid` value. Individual overload values are not shown in the source. We're working on having a better system in the future.

When the UID contains the special characters ```, `#`, or `*`, the UID value needs to be HTML encoded as `%60`, `%23`, and `%2A`, respectively. You'll sometimes see parentheses encoded but it's not a requirement.

Examples:

- `System.Threading.Tasks.Task`1` becomes `System.Threading.Tasks.Task%601`
- `System.Exception.#ctor` becomes `System.Exception.%23ctor`
- `System.Lazy`1.#ctor(System.Threading.LazyThreadSafetyMode)` becomes `System.Lazy%601.%23ctor%28System.Threading.LazyThreadSafetyMode%29`

Lists (Numbered, Bulleted, Checklist)

Numbered list

To create a numbered list, you can use all 1s, which are rendered as a sequential list when published. For increased source readability, you can increment your lists.

Do not use letters in lists, including nested lists. They do not render correctly when published to Docs. Nested lists using numbers will render as lowercase letters when published. For example:

```
1. This is
1. a parent numbered list
  1. and this is
  1. a nested numbered list
1. (fin)
```

This renders as follows:

1. This is
2. a parent numbered list
 - a. and this is
 - b. a nested numbered list
3. (fin)

Bulleted list

To create a bulleted list, use `-` followed by a space at the beginning of each line:

```
- This is
- a parent bulleted list
  - and this is
  - a nested bulleted list
- All done!
```

This renders as follows:

- This is
- a parent bulleted list
 - and this is
 - a nested bulleted list
- All done!

Checklist

Checklists are available for use on docs.microsoft.com (only) via a custom Markdown extension:

```
> [!div class="checklist"]
> * List item 1
> * List item 2
> * List item 3
```

This example renders on docs.microsoft.com like this:

- List item 1
- List item 2
- List item 3

Use checklists at the beginning or end of an article to summarize "What will you learn" or "What have you learned" content. Do not add random checklists throughout your articles.

Next step action

You can use a custom extension to add a next step action button to pages on docs.microsoft.com (only).

The syntax is as follows:

```
> [!div class="nextstepaction"]
> [button text](link to topic)
```

For example:

```
> [!div class="nextstepaction"]
> [Learn about basic style](style-quick-start.md)
```

This renders as follows:

[Learn about basic style](#)

You can use any supported link in a next step action, including a Markdown link to another web page. In most cases, the next action link will be a relative link to another file in the same doc set.

Section definition

You might need to define a section. This syntax is mostly used for code tables. See the following example:

```
> [!div class="tabbedCodeSnippets" data-resources="OutlookServices.Calendar"]
> ```cs
> <cs code text>
> ```
> ```javascript
> <js code text>
> ```
```

The preceding blockquote Markdown text will be rendered as:

```
<cs code text>
```

```
<js code text>
```

Selectors

You can use a selector when you want to connect different pages for the same article. Readers can then switch between those pages.

NOTE

This extension works differently between docs.microsoft.com and MSDN.

Single selector

```
> [!div class="op_single_selector"]
> - [Universal Windows](how-to-write-use-markdown.md)
> - [Windows Phone](how-to-write-use-markdown.md)
> - [iOS](how-to-write-use-markdown.md)
> - [Android](how-to-write-use-markdown.md)
> - [Kindle](how-to-write-use-markdown.md)
> - [Baidu](how-to-write-use-markdown.md)
> - [Xamarin.iOS](how-to-write-use-markdown.md)
> - [Xamarin.Android](how-to-write-use-markdown.md)
```

... will be rendered like this:

Multi-selector

```
> [!div class="op_multi_selector" title1="Platform" title2="Backend"]
> - [(iOS | .NET)](how-to-write-workflows-major.md)
> - [(iOS | JavaScript)](how-to-write-workflows-major.md)
> - [(Windows universal C# | .NET)](how-to-write-workflows-major.md)
> - [(Windows universal C# | Javascript)](how-to-write-workflows-major.md)
> - [(Windows Phone | .NET)](how-to-write-workflows-major.md)
> - [(Windows Phone | Javascript)](how-to-write-workflows-major.md)
> - [(Android | .NET)](how-to-write-workflows-major.md)
> - [(Android | Javascript)](how-to-write-workflows-major.md)
> - [(Xamarin iOS | Javascript)](how-to-write-workflows-major.md)
> - [(Xamarin Android | Javascript)](how-to-write-workflows-major.md)
```

... will be rendered like this:

Tables

The simplest way to create a table in Markdown is to use pipes and lines. To create a standard table with a header, follow the first line with dashed line:

```
|This is |a simple |table header|
|-----|-----|-----|
|table |data |here |
|it doesn't|actually |have to line up nicely!|
```

This renders as follows:

THIS IS	A SIMPLE	TABLE HEADER	
table	data	here	
it doesn't	actually	have to line up nicely!	

You can also create a table without a header. For example, to create a multiple-column list:

```
| | |
| - | - |
| This | table |
| has no | header |
```

This renders like this:

This	table
has no	header

You can align the columns by using colons:

```
|
|-----|
|   right aligned:|
|:left aligned   |
|:centered       :|
```

Renders as follows:

right aligned:

:left aligned

:centered :

TIP

The Docs Authoring Extension for VS Code makes it easy to add basic Markdown tables!

You can also use an [online table generator](#).

mx-tdBreakAll

IMPORTANT

This only works on the docs.microsoft.com site.

If you create a table in Markdown, the table might expand to the right navigation and become unreadable. You can solve that by allowing Docs rendering to break the table when needed. Just wrap up the table with the custom class `[!div class="mx-tdBreakAll"]`.

Here is a Markdown sample of a table with three rows that will be wrapped by a `div` with the class name `mx-tdBreakAll`.

```
> [!div class="mx-tdBreakAll"]
> |Name|Syntax|Mandatory for silent installation?|Description|
> |-----|-----|-----|-----|
> |Quiet|/quiet|Yes|Runs the installer, displaying no UI and no prompts.|
> |NoRestart|/norestart|No|Suppresses any attempts to restart. By default, the UI will prompt before restart.|
> |Help|/help|No|Provides help and quick reference. Displays the correct use of the setup command, including a
list of all options and behaviors.|
```

It will be rendered like this:

NAME	SYNTAX	MANDATORY FOR SILENT INSTALLATION?	DESCRIPTION
Quiet	/quiet	Yes	Runs the installer, displaying no UI and no prompts.
NoRestart	/norestart	No	Suppresses any attempts to restart. By default, the UI will prompt before restart.

NAME	SYNTAX	MANDATORY FOR SILENT INSTALLATION?	DESCRIPTION
Help	/help	No	Provides help and quick reference. Displays the correct use of the setup command, including a list of all options and behaviors.

mx-tdCol2BreakAll

IMPORTANT

This only works on the docs.microsoft.com site.

From time to time, you might have very long words in the second column of a table. To ensure they are broken apart nicely, you can apply the class `mx-tdCol2BreakAll` by using the `div` wrapper syntax as shown earlier.

HTML Tables

HTML tables are not recommended for docs.microsoft.com. They are not human readable in the source - which is a key principle of Markdown.

Videos

Embedding videos into a Markdown page

Currently, Docs can support videos published to one of three locations:

- YouTube
- Channel 9
- Microsoft's own 'One Player' system

You can embed a video with the following syntax, and Docs will render it.

```
> [!VIDEO <embedded_video_link>]
```

Example:

```
> [!VIDEO https://channel9.msdn.com/Series/Youve-Got-Key-Values-A-Redis-Jump-Start/03/player]
> [!VIDEO https://www.youtube.com/embed/iAtwVM-Z7rY]
> [!VIDEO https://www.microsoft.com/en-us/videoplayer/embed/RE1XVQS]
```

... will be rendered as:

```
<iframe src="https://channel9.msdn.com/Series/Youve-Got-Key-Values-A-Redis-Jump-Start/03/player" width="640" height="320" allowFullScreen="true" frameBorder="0"></iframe>

<iframe src="https://www.youtube-nocookie.com/embed/iAtwVM-Z7rY" width="640" height="320" allowFullScreen="true" frameBorder="0"></iframe>
<iframe src="https://www.microsoft.com/en-us/videoplayer/embed/RE1XVQS" width="640" height="320" allowFullScreen="true" frameBorder="0"></iframe>
```

And it will be displayed like this on published pages:

IMPORTANT

The CH9 video URL should start with `https` and end with `/player`. Otherwise, it will embed the whole page instead of the video only.

Uploading new videos

Any new videos should be uploaded using the following process:

1. Join the **docs_video_users** group on IDWEB.
2. Go to <https://aka.ms/VideoUploadRequest> and fill in the details for your video. You will need (note that none of these items will be visible to the public):
 - a. A title for your video.
 - b. A list of products/services that your video is related to.
 - c. The target page or (if you don't have the page yet) doc set that your video will be hosted on.
 - d. A link to the MP4 file for your video (if you don't have a location to put the file, you can put it here temporarily: `\\scratch2\scratch\apex`). MP4 files should be 720p or higher.
 - e. A description of the video.
3. Submit (save) that item.
4. Within two business days, the video will get uploaded. The link you need for embedding will be placed into the work item, and it will be resolved *back to you*.
5. Once you have grabbed the video link, close the work item.
6. The video link can then be added to your post, using this syntax:

```
> [!VIDEO https://www.microsoft.com/en-us/videoplayer/embed/RE1XVQS]
```

Docs style and voice quick start

7/30/2019 • 3 minutes to read • [Edit Online](#)

This quick start is a brief guide to writing technical content for publication on docs.microsoft.com. These guidelines apply whether you are creating new documentation or updating existing documentation.

Best practices:

- Check the spelling and grammar in your articles, even if you have to copy and paste into Microsoft Word to check.
- Use a casual and friendly voice—like you're talking to another person one-on-one.
- Use simple sentences. Easy-to-read sentences mean the reader can quickly use the guidance you share.

Use the Microsoft voice principles

We aspire to follow these principles when we write technical content for docs.microsoft.com. We might not always get there, but we need to keep trying!

- **Focus on the intent:** Customers have a specific purpose in mind when they consult our documentation. Before you begin writing, clearly determine who the customer is and what task he or she is trying to do. Then, write your article to help that specific customer do that specific task.
- **Use everyday words:** Try to use natural language, the words your customers use. Be less formal but not less technical. Provide examples that explain new concepts.
- **Write concisely:** Don't waste words. Be affirmative and don't use extra words or lots of qualifiers. Keep sentences short and concise. Keep your article focused. If a task has a qualifier, put it at the beginning of the sentence or paragraph. Also, keep the number of notes to a minimum. Use a screenshot when it can save words.
- **Make your article easy to scan:** Put the most important things first. Use sections to chunk long procedures into more manageable groups of steps. (Procedures with more than 12 steps are probably too long.) Use a screenshot when it adds clarity.
- **Show empathy:** Use a supportive tone in the article, and keep disclaimers to a minimum. Honestly call out areas that will be frustrating to customers. Make sure the article focuses on what matters to customers; don't just give a technical lecture.

Consider localization and machine translation

Our technical articles are translated into several languages, and some are modified for particular markets or geographies. People might also use machine translation on the web to read the technical articles. So, keep the following guidelines in mind when you're writing:

- **Make sure the article contains no grammar, spelling, or punctuation errors:** This is something we should do in general. Some Markdown editors (such as MarkdownPad 2.0) have a basic spell checker, but it's a good practice to paste the rendered HTML content from the article into Word, which has a more robust spell and grammar checker.
- **Make your sentences as short as possible:** Compound sentences or chains of clauses make translation difficult. Split up sentences if you can do it without being too redundant or sounding weird. We don't want articles written in unnatural language either.
- **Use simple and consistent sentence construction:** Consistency is better for translation. Avoid parentheticals and asides, and have the subject as near the beginning of the sentence as possible. Check out a few published articles. If an article has a friendly, easy-to-read style, use it as a model.

- **Use consistent wording and capitalization:** Again, consistency is key. Do not capitalize a word if it isn't at the start of a sentence or it isn't a proper noun.
- **Include the "small words":** Words that we consider small and unimportant in English because they are understood for context (such as "a," "the," "that," and "is") are crucial for machine translation. Be sure to include them.

Other style and voice issues to watch for

- Don't break up steps with commentary or asides.
- For steps that include code snippets, put additional information about the step into the code as comments. This reduces the amount of text that people have to read through. The key information gets copied into the code project to remind people of what the code is doing when they refer to it later.
- Use sentence case for all titles and headings.
- Use "sign in" and not "log in."
- For more guidelines, see the [Microsoft Writing Style Guide](#).

Localized documentation

- If you are contributing to localized documentation, refer to the [Microsoft Language Portal](#).
- For localization guidelines, information on language style and usage in technical publications, and information on market-specific data formats, download the [style guide](#) in your language.
- For Microsoft localized terminology, search for [product-specific approved terminology](#) or download the [Microsoft Terminology Collection](#) in your language.
- To learn more about localization, see "Global communications" in the [Microsoft Writing Style Guide](#).

Using links in documentation

7/30/2019 • 6 minutes to read • [Edit Online](#)

This article describes how to use hyperlinks from pages hosted at docs.microsoft.com. Links are easy to add into markdown with a few varying conventions. Links point users to content in the same page, point off into other neighboring pages, or point to external sites and URLs.

The docs.microsoft.com site backend uses Open Publishing Services (OPS) which supports [CommonMark](#) compliant markdown parsed through the [Markdig](#) parsing engine. This markdown flavor is mostly compatible with [GitHub Flavored Markdown \(GFM\)](#), as most docs are stored in GitHub and can be edited there. Additional functionality is added through Markdown extensions.

IMPORTANT

All links must be secure (`https` vs `http`) whenever the target supports it (which the vast majority should).

Link text

The words that you include in link text should be friendly. In other words, they should be normal English words or the title of the page that you're linking to.

IMPORTANT

Do not use "click here." It's bad for search engine optimization, and doesn't adequately describe the target.

Correct:

- For more information, see the [contributor guide index](https://github.com/Azure/azure-content/blob/master/contributor-guide/contributor-guide-index.md).
- For more details, see the [SET TRANSACTION ISOLATION LEVEL](https://msdn.microsoft.com/library/ms173763.aspx) reference.

Incorrect:

- For more details, see https://msdn.microsoft.com/library/ms173763.aspx.
- For more information, click [here](https://github.com/Azure/azure-content/blob/master/contributor-guide/contributor-guide-index.md).

Links from one article to another

To create an inline link from a Docs technical article to another Docs technical article within the same docset, use the following link syntax:

- An article in a directory links to another article in the same directory:

```
[link text](article-name.md)
```

- An article links from a subdirectory to an article in the root directory:

```
[link text](../article-name.md)
```

- An article in the root directory links to an article in a subdirectory:

```
[link text](./directory/article-name.md)
```

- An article in a subdirectory links to an article in another subdirectory:

```
[link text](../directory/article-name.md)
```

- An article linking across docsets (even if in the same repository): `[link text](./directory/article-name)`

IMPORTANT

None of the above examples use the `~/` as part of the link. If you are linking to a path at the root of the repository, start with the `/`. Including the `~/` produces invalid links when navigating the source repositories on GitHub. Starting the path with `/` resolves correctly.

Links to anchors

You do not have to create anchors. They're automatically generated at publishing time for all H2 headings. The only thing you have to do is create links to the H2 sections.

- To link to a heading within the same article:

```
[link](#the-text-of-the-H2-section-separated-by-hyphens) [Create cache](#create-cache)
```

- To link to an anchor in another article in the same subdirectory:

```
[link text](article-name.md#anchor-name)  
[Configure your profile](media-services-create-account.md#configure-your-profile)
```

- To link to an anchor in another service subdirectory:

```
[link text](../directory/article-name.md#anchor-name)  
[Configure your profile](../directory/media-services-create-account.md#configure-your-profile)
```

Links from includes

Because include files are located in another directory, you must use longer relative paths. To link to an article from an include file, use this format:

```
[link text](../articles/folder/article-name.md)
```

Links in selectors

A selector is a navigation component that appears in a docs article as a drop-down list. When a reader selects a value in the drop-down, the browser opens the selected article. Typically the selector list contains links to closely related articles, for example the same subject matter in multiple programming languages or a closely related series of articles.

If you have selectors that are embedded in an include, use the following link structure:

```
> [AZURE.SELECTOR-LIST (Dropdown1 | Dropdown2 )]  
- [(Text1 | Example1 )](../articles/folder/article-name1.md)  
- [(Text1 | Example2 )](../articles/folder/article-name2.md)  
- [(Text2 | Example3 )](../articles/folder/article-name3.md)  
- [(Text2 | Example4 )](../articles/folder/article-name4.md) -->
```

Reference-style links

You can use reference-style links to make your source content easier to read. Reference-style links replace inline link syntax with simplified syntax that allows you to move the long URLs to the end of the article. Here's [Daring Fireball](#)'s example:

Inline text:

```
I get 10 times more traffic from [Google][1] than from [Yahoo][2] or [MSN][3].
```

Link references at the end of the article:

```
<!--Reference links in article-->
[1]: http://google.com/
[2]: http://search.yahoo.com/
[3]: http://search.msn.com/
```

Make sure that you include the space after the colon, before the link. When you link to other technical articles, if you forget to include the space, the link will be broken in the published article.

Links to pages that are not part of the technical documentation set

To link to a page on another Microsoft property (such as a pricing page, SLA page, or anything else that is not a documentation article), use an absolute URL, but omit the locale. The goal here is that links work in GitHub and on the rendered site:

```
[link text](https://azure.microsoft.com/pricing/details/virtual-machines/)
```

Links to third-party sites

The best user experience minimizes sending users to another site. So base any links to third-party sites, which we do sometimes need, on this info:

- **Accountability:** Link to third-party content when it's the third-party's information to share. For example, it's not Microsoft's place to tell people how to use Android developer tools--that is Google's story to tell. If we need to, we can explain how to use Android developer tools *with* Azure, but Google should tell the story of how to use their tools.
- **PM signoff:** Request that Microsoft sign off on third-party content. By linking to it, we are saying something about our trust in it and our obligation if people follow the instructions.
- **Freshness reviews:** Make sure that the third-party info is still current, correct, and relevant, and that the link hasn't changed.
- **Offsite:** Make users aware that they are going to another site. If the context does not make that clear, add a qualifying phrase. For example: "Prerequisites include the Android Developer Tools, which you can download on the Android Studio site."
- **Next steps:** It's fine to add a link to, say, an MVP blog in a "Next steps" section. Again, just make sure that users understand they'll be leaving the site.
- **Legal:** We are covered legally under **Links to Third Party Sites** in the **Terms of Use** footer on every ms.com page.

Links to MSDN or TechNet

When you need to link to MSDN or TechNet, use the full link to the topic, and remove the "en-us" language locale

from the link.

Links to Azure PowerShell reference content

The Azure PowerShell reference content has been through several changes since November 2016. Use the following guidelines for linking to this content from other articles on docs.microsoft.com.

Structure of the URL:

- For cmdlets:
 - `/powershell/module/<module-name>/<cmdlet-name>[?view=<moniker-name>]`
- For conceptual topics:
 - `/powershell/azure/<topic-file-name>[?view=<moniker-name>]`
 - `/powershell/azure/<service-name>/<topic-file-name>[?view=<moniker-name>]`

The `<moniker-name>` portion is optional. If it's omitted, you will be directed to the latest version of the content. The `<service-name>` portion is one of the examples shown in the following base URLs:

- Azure PowerShell (AzureRM) content: <https://docs.microsoft.com/powershell/azure/>
- Azure PowerShell (ASM) content: <https://docs.microsoft.com/powershell/azure/servicemanagement>
- Azure Active Directory (AzureAD) PowerShell content: <https://docs.microsoft.com/powershell/azure/active-directory>
- Azure Service Fabric PowerShell: <https://docs.microsoft.com/powershell/azure/service-fabric>
- Azure Information Protection PowerShell: <https://docs.microsoft.com/powershell/azure/aip>
- Azure Elastic DB Jobs PowerShell: <https://docs.microsoft.com/powershell/azure/elasticdbjobs>

When you use these URLs, you will be redirected to the latest version of the content. This way, you don't have to specify a version moniker. And you won't have links to conceptual content that must be updated when the version changes.

To create the correct link, find the page that you want to link to in your browser, and copy the URL. Then, remove `https://docs.microsoft.com` and the locale info.

When you're linking from a TOC, you must use the full URL without the locale information.

Example markdown:

```
[Get-AzureRmResourceGroup](/powershell/module/azurerm.resources/get-azurermresourcegroup)
[Get-AzureRmResourceGroup](/powershell/module/azurerm.resources/get-azurermresourcegroup?view=azurermps-4.1.0)
[New-AzureVM](/powershell/module/azure/new-azurevm?view=azuresmps-4.0.0)
[New-AzureRmVM](/powershell/module/azurerm.compute/new-azurermvm)
[Install Azure PowerShell for Service Management](/powershell/azure/servicemanagement/install-azurermps)
[Install Azure PowerShell](/powershell/azure/install-azurermps)
```


Docs Authoring Pack for VS Code

7/30/2019 • 7 minutes to read • [Edit Online](#)

The Docs Authoring Pack is a collection of Visual Studio Code extensions to aid with Markdown authoring for docs.microsoft.com. The pack is [available in the VS Code Marketplace](#) and contains the following extensions:

- [markdownlint](#): A popular Markdown linter by David Anson to help ensure your Markdown follows best practices.
- [Code Spell Checker](#): A fully offline spell checker by Street Side Software.
- [Docs Preview](#): Uses the docs.microsoft.com CSS for more accurate Markdown preview, including custom Markdown.
- [Docs Markdown](#): Provides Markdown authoring assistance for docs.microsoft.com content in the Open Publishing System (OPS), including basic Markdown support and support for custom Markdown syntax in OPS. The rest of this topic describes the Docs Markdown extension.
- [Docs Article Templates](#): Allows users to apply Markdown skeleton content to new files.

Prerequisites and assumptions

To accurately insert relative links, images, and other embedded content with the Docs Markdown extension, you must have your VS Code workspace scoped to the root of your cloned Open Publishing System (OPS) repo.

Some syntax supported by the extension, such as alerts and snippets, are custom Markdown for OPS, and will not render correctly unless published via OPS.

How to use the Docs Markdown extension

To access the Docs Markdown menu, type `ALT+M`. You can click or use up/down arrows to select the function you want, or type to start filtering, then hit `ENTER` when the function you want is highlighted in the menu. The following are available:

FUNCTION	DESCRIPTION
Preview	Preview the active topic in a side-by-side window using the Docs Preview extension. This option is only available if Docs Preview is installed.
Bold	Formats text bold .
Italic	Formats text <i>italic</i> .
Code	<p>If one line or less is selected, formats text as <code>inline code</code>.</p> <p>If multiple lines are selected, formats them as a fenced code block, and allows you to optionally select a programming language supported by OPS.</p>

FUNCTION	DESCRIPTION
Alert	<p>Inserts a Note, Important, Warning, or Tip.</p> <p>Select Alert from the menu, then select the alert type. If you have previously selected text, it will be surrounded with the selected alert syntax. If no text is selected, a new alert will be added with placeholder text.</p>
Numbered list	<p>Inserts a new numbered list.</p> <p>If multiple lines are selected, each will be a list item. Note that numbered lists show in the Markdown as all 1s, but will render on docs.microsoft.com as sequential numbers or, for nested lists, letters. To create a nested numbered list, tab from within the parent list.</p>
Bulleted list	Inserts a new bulleted list.
Table	<p>Inserts a Markdown table structure.</p> <p>After you select the table command, specify the number of columns and rows in the format columns:rows, such as 3:4. Note that the maximum number of columns you can specify via this extension is 5, which is the recommended maximum for readability.</p>
Link to file in repo	Inserts a relative link to another file in the current repo. After selecting this option, type in the command window to filter files by name, then select the file you want. If you have previously selected text, it will become the link text. Otherwise, the H1 of the target file will be used as link text.
Link to web page	Inserts a link to a web page. After selecting this option, paste or type the URI into the command window. <code>https://</code> is required. If you have previously selected text, it will become the link text. Otherwise, the URI will be used as link text.
Link to heading	<p>Links to a bookmark in the current file or another file in the repo.</p> <p>Bookmark in this file : Choose from a list of headings in the current file to insert a properly formatted bookmark.</p> <p>Bookmark in another file : First, filter by file name and select the file to link to, then choose the appropriate heading within the selected file.</p>
Image	Type alternate text (required for accessibility) and select it, then call this command to filter the list of supported image files in the repo and select the one you want. If you haven't selected alt text when you call this command, you will be prompted for it before you can select an image file.
Include	Find a file to embed in the current file.
Snippet	Find a code snippet in the repo to embed in the current file.
Video	Add an embedded video.

FUNCTION	DESCRIPTION
Template	Create a new file and apply a Markdown template. See Templates , below, for more information.

How to assign keyboard shortcuts

1. Type `CTRL+K` then `CTRL+S` to open the Keyboard Shortcuts list.
2. Search for the command, such as `formatBold`, for which you want to create a custom keybinding.
3. Click the plus that appears near the command name when you mouse over the line.
4. After a new input box is visible, type the keyboard shortcut you want to bind to that particular command. For example, to use the common shortcut for bold, type `ctrl+b`.
5. It's a good idea to insert a `when` clause into your keybinding, so it won't be available in files other than Markdown. To do this, open `keybindings.json` and insert the following line below the command name (be sure to add a comma between lines):

```
"when": "editorTextFocus && editorLangId == 'markdown'"
```

Your completed custom keybinding should look like this in `keybindings.json`:

```
// Place your key bindings in this file to overwrite the defaults
[
  {
    "key": "ctrl+b",
    "command": "formatBold",
    "when": "editorTextFocus && editorLangId == 'markdown'"
  }
]
```

6. Save `keybindings.json`.

See [Keybindings](#) in the VS Code docs for more information.

How to show the legacy toolbar

Users of the pre-release version of the extension will notice that the authoring toolbar no longer appears at the bottom of the VS Code window when the Docs Markdown Extension is installed. This is because the toolbar took up a lot of space on the VS Code status bar and did not follow best practices for extension UX, so it is deprecated in the new extension. However, you can optionally show the toolbar by updating your VS Code `settings.json` file as follows:

1. In VS Code, go to File -> Preferences -> Settings (`CTRL+Comma`).
2. Select User Settings to change the settings for all VS Code workspaces, or Workspace Settings to change them for just the current workspace.
3. In the Default Settings pane, find Docs Authoring Extension Configuration, and select the pencil icon next to the desired setting. Next, you will be prompted to select either `true` or `false`. Once you've made your selection, VS Code will automatically add the value to the `settings.json` file and you will be prompted to reload the window for the changes to take effect.

How to use Docs templates

The Docs Article Templates extension lets writers in VS Code pull a Markdown template from a centralized store

and apply it to a file. Templates can help ensure that required metadata is included in articles, that content standards are followed, and so on. Templates are managed as Markdown files in a public GitHub repository.

To apply a template in VS Code

1. If you don't have the Docs Markdown extension installed, hit F1 to open the command palette, start typing "template" to filter, then click `Docs: Template`. If you do have Docs Markdown installed, you can use either the command palette or click `Alt+M` to bring up the Docs Markdown QuickPick menu, then select `Template` from the list.
2. Select the desired template from the list that appears.

To add your GitHub ID and/or Microsoft alias to your VS Code settings

The Templates extension supports three dynamic metadata fields: author, ms.author, and ms.date. That means that if a template creator uses these fields in the metadata header of a Markdown template, they will be auto-populated in your file when you apply the template, as follows:

METADATA	VALUE
author	Your GitHub ID, if specified in your VS Code settings file.
ms.author	Your Microsoft alias, if specified in your VS Code settings file. If you are not a Microsoft employee, leave this unspecified.
ms.date	The current date in the Docs-supported format, MM/DD/YYYY. Note that the date is not automatically updated if you subsequently update the file. You must manually update the ms.date value to indicate the most recent date of publication on the docs.microsoft.com site.

To set author (GitHub ID) and/or ms.author (Microsoft alias)

1. In VS Code, go to File -> Preferences -> Settings (`CTRL+Comma`).
2. Select User Settings to change the settings for all VS Code workspaces, or Workspace Settings to change them for just the current workspace.
3. In the Default Settings pane on the left, find Docs Article Templates Extension Configuration, click the pencil icon next to the desired setting, then click Replace in Settings.
4. The User Settings pane will open side-by-side, with a new entry at the bottom.
5. Add your GitHub ID or Microsoft email alias, as appropriate, and save the file.
6. You might need to close and restart VS Code for the changes to take effect.
7. Now, when you apply a template that uses dynamic fields, your GitHub ID and/or Microsoft alias will be auto-populated in the metadata header.

Learn how to contribute to the .NET docs repositories

7/30/2019 • 2 minutes to read • [Edit Online](#)

Thank you for your interest in contributing to the .NET documentation!

This document covers the process for contributing to the articles and code samples that are hosted on the [.NET documentation site](#). Contributions may be as simple as typo corrections or as complex as new articles.

The .NET documentation site is built from multiple repositories:

- [.NET conceptual articles](#)
- [Code samples and snippets](#)
- [.NET Standard, .NET Core, .NET Framework API reference](#)
- [.NET Compiler Platform SDK reference](#)
- [ML.NET API reference](#)

Issues for all these repositories are tracked at the [dotnet/docs](#) repository.

Process for contributing to .NET docs

7/30/2019 • 11 minutes to read • [Edit Online](#)

We appreciate community contributions to docs. The following list shows some guiding rules that you should keep in mind when you're contributing to the .NET documentation:

- **DON'T** surprise us with large pull requests. Instead, file an issue and start a discussion so we can agree on a direction before you invest a large amount of time.
- **DO** follow these instructions and [voice and tone](#) guidelines.
- **DO** use the [template](#) file as the starting point of your work.
- **DO** create a separate branch on your fork before working on the articles.
- **DO** follow the [GitHub Flow workflow](#).
- **DO** blog and tweet (or whatever) about your contributions, frequently!

Following these guidelines will ensure a better experience for you and for us.

Make a contribution to .NET docs

Step 1: Skip this step for small changes. If you're interested in writing new content or in thoroughly revising existing content, open an [issue](#) describing what you want to do.

The content inside the **docs** folder is organized into sections that are reflected in the Table of Contents (TOC). Define where the topic will be located in the TOC. Get feedback on your proposal.

-or-

You can also choose from existing issues for which community contributions are welcome. [Projects for .NET Community contributors](#) lists many of the issues that are available for community contributors. Depending on your interests and level of commitment, you can choose from issues in the following categories:

- **Maintenance.** This category includes fairly simple contributions, such as fixing broken or incorrect links, adding missing code examples, or addressing limited content issues. In some cases, these issues may concern large numbers of files. In that case, you should let us know what you'd like to work on before you begin. Add a comment on the issue to tell us that you are working on it.
- **Content updates.** Given the enormity of the doc set, content easily becomes outdated and requires revision. In addition, for a variety of reason, some content has been duplicated or even triplicated. Updating content involves making sure that individual topics are current or revising content in a feature area to eliminate duplication and ensure that all unique content is preserved in the smaller documentation set.
- **New content authoring.** If you're interested in authoring your own topic, these issues list topics that we know we'd like to add to our doc set. Let us know before you begin working on a topic, though. If you're interested in writing a topic that isn't listed here, open an issue.

You can also look at our [open issues](#) list and volunteer to work on the ones you're interested in. We use the [up-for-grabs](#) label to tag issues identified for community contribution. These usually require minimal context and are well-suited for first issues. We encourage experienced contributors in our community to look at any issues of interest. When you find one, add a comment to ask if it's open.

Once you've picked a task to work on, follow the [get started](#) guide to create a GitHub account and setup your environment.

Step 2: Fork the `/dotnet/docs`, `dotnet/samples`, `dotnet/dotnet-api-docs`, `dotnet/roslyn-api-docs`, or

`dotnet/ml-api-docs` repos as needed and create a branch for your changes.

For small changes, see the instructions on editing in GitHub on the [home page](#) of the contributor guide.

Step 3: Make the changes on this new branch.

If it's a new topic, you can use this [template file](#) as your starting point. It contains the writing guidelines and also explains the metadata required for each article, such as author information.

Navigate to the folder that corresponds to the TOC location determined for your article in step 1. That folder contains the Markdown files for all articles in that section. If necessary, create a new folder to place the files for your content. The main article for that section is called *index.md*. For images and other static resources, create a subfolder called **media** inside the folder that contains your article, if it doesn't already exist. Inside the **media** folder, create a subfolder with the article name (except for the index file). Sample code should be in the

`dotnet/samples` repo, as described in the section on [samples](#).

Be sure to follow the proper Markdown syntax. For examples of common , see the [template and markdown cheatsheet](#).

Example structure

```
docs
  /about
  /core
    /porting
      porting-overview.md
    /media
      /porting-overview
        portability_report.png
```

Step 4: Submit a Pull Request (PR) from your branch to the master branch.

IMPORTANT

The [comment automation](#) functionality is not available on any of the .NET docs repositories at this time. Members of the .NET docs team will review and merge your PR.

Each PR should usually address one issue at a time. The PR can modify one or multiple files. If you're addressing multiple fixes on different files, separate PRs are preferred. If you are creating samples as well as updating markdown, you'll need to create a separate PR for samples.

If your PR fixes an existing issue, add the `Fixes #Issue_Number` keyword to the commit message or PR description. That way, the issue is automatically closed when the PR is merged. For more information, see [Closing issues via commit messages](#).

The .NET team will review your PR and let you know if there are any other updates/changes necessary in order to approve it.

Step 5: Make any necessary updates to your branch as discussed with the team.

The maintainers will merge your PR into the master branch once feedback has been applied and your change is approved.

We regularly push all commits from master branch into the live branch and then you'll be able to see your contribution live at <https://docs.microsoft.com/dotnet/>. We typically publish daily during the work week. Maintenance activities may delay publishing for a few days.

Contributing to samples

The [dotnet/samples](#) repo contains all the sample code that is part of any topic under the .NET documentation. There are several different projects that are organized in sub-folders. These sub-folders are organized similarly to the organization of the docs for .NET.

We make the following distinction for code that exists in our repository:

- **Samples:** readers can download and run the samples. All samples should be complete applications or libraries. Where the sample creates a library, it should include unit tests or an application that lets readers run the code. They often use more than one technology, feature, or toolkit. The `readme.md` file for each sample will refer to the article so that you can read more about the concepts covered in each sample.
- **Snippets:** illustrate a smaller concept or task. They compile but they are not intended to be complete applications. They should run correctly, but aren't an example application for a typical scenario. Instead, they are designed to be as small as possible to illustrate a single concept or feature. These should be no more than a single screen of code.

Code all lives in the [dotnet/samples](#) repository. We are working toward a model where our samples folder structure matches our docs folder structure. Standards that we follow are:

- The top level *snippets* folder contains snippets for small, focused samples.
- API reference samples are placed in a folder following this pattern:
`snippets/<language>/api/<namespace>/<apiname>`.
- Other top-level folders match the top level folders in the *docs* repository. For example, the docs repository has a *machine-learning/tutorials* folder, and the samples for machine learning tutorials are in the *samples/machine-learning/tutorials* folder.

In addition, all samples under the *core* and *standard* folders should build and run on all platforms supported by .NET Core. Our CI build system will enforce that. The top level *framework* folder contains samples that are only built and validated on Windows.

Sample projects should build and run on the widest set of platforms possible for the given sample. In practice, that means building .NET Core-based console applications where possible. Samples that are specific to the web or a UI framework should add those tools as needed. Examples include web applications, mobile apps, WPF or WinForms apps, and so on.

We are working toward having a CI system in place for all code. When you make any updates to samples, make sure each update is part of a buildable project. Ideally, add tests for correctness on samples as well.

Each complete sample that you create should contain a *readme.md* file. This file should contain a short description of the sample (one or two paragraphs). Your *readme.md* should tell readers what they will learn by exploring this sample. The *readme.md* file should also contain a link to the live document on the [.NET documentation site](#). To determine where a given file in the repository maps to that site, replace `/docs` in the repository path with `http://docs.microsoft.com/dotnet`.

Your topic will also contain links to the sample. Link directly to the sample's folder on GitHub.

Writing a new snippet or sample

1. Your sample **must be part of a buildable project**. Where possible, the projects should build on all platforms supported by .NET Core. Exceptions to this are samples that demonstrate a platform specific feature or platform specific tool.
2. Your sample should conform to the [corefx coding style](#) to maintain consistency.
 - Additionally, we prefer the use of `static` methods rather than instance methods when demonstrating something that doesn't require instantiating a new object.

3. Your sample should include **appropriate exception handling**. It should handle all exceptions that are likely to be thrown in the context of the sample. For example, a sample that calls the [Console.ReadLine](#) method to retrieve user input should use appropriate exception handling when the input string is passed as an argument to a method. Similarly, if your sample expects a method call to fail, the resulting exception must be handled. Always handle the specific exceptions thrown by the method, rather than base class exceptions such as [Exception](#) or [SystemException](#).
4. If your sample builds a standalone package, you must include the runtimes used by our CI build system, in addition to any runtimes used by your sample:

- win7-x64
- win8-x64
- win81-x64
- ubuntu.16.04-x64

We will have a CI system in place to build these projects shortly.

To create a sample:

1. File an [issue](#) or add a comment to an existing one that you are working on it.
2. Write the topic that explains the concepts demonstrated in your sample (example: `docs/standard/linq/where-clause.md`).
3. Write your sample (example: `WhereClause-Sample1.cs`).
4. Create a Program.cs with a Main entry point that calls your samples. If there is already one there, add the call to your sample:

```
public class Program
{
    public void Main(string[] args)
    {
        WhereClause1.QuerySyntaxExample();

        // Add the method syntax as an example.
        WhereClause1.MethodSyntaxExample();
    }
}
```

You build any .NET Core snippet or sample using the .NET Core CLI, which can be installed with [the .NET Core SDK](#). To build and run your sample:

1. Go to the sample folder and build to check for errors:

```
dotnet build
```

2. Run your sample:

```
dotnet run
```

3. Add a readme.md to the root directory of your sample.

This should include a brief description of the code, and refer people to the article that references the sample.

Except where noted, all samples build from the command line on any platform supported by .NET Core. There are a few samples that are specific to Visual Studio and require Visual Studio 2017 or later. In addition, some samples

show platform specific features and will require a specific platform. Other samples and snippets require the .NET Framework and will run on Windows platforms, and will need the Developer Pack for the target Framework version.

The C# interactive experience

All samples included in an article use a [language tag](#) to indicate the source language. Short code samples in C# can use the `csharp-interactive` language tag to specify a C# sample that runs in the browser. (Inline code samples use the `csharp-interactive` tag, for snippets included from source, use the `code-csharp-interactive` tag.) These code samples display a code window and an output window in the article. The output window displays any output from executing the interactive code once the user has run the sample.

The C# interactive experience changes how we work with samples. Visitors can run the sample to see the results. A number of factors help determine if the sample or corresponding text should include information about the output.

When to display the expected output without running the sample

- Articles intended for beginners should provide output so that readers can compare the output of their work with the expected answer.
- Samples where the output is integral to the topic should display that output. For example, articles on formatted text should show the text format without running the sample.
- When both the sample and the expected output is short, consider showing the output. It saves a bit of time.
- Articles explaining how current culture or invariant culture affect output should explain the expected output. The interactive REPL (Read Evaluate Print Loop) runs on a Linux-based host. The default culture, and the invariant culture produce different output on different operating systems and machines. The article should explain the output in Windows, Linux, and Mac systems.

When to exclude expected output from the sample

- Articles where the sample generates a larger output should not include that in comments. It obscures the code once the sample has been run.
- Articles where the sample demonstrates a topic, but the output isn't integral to understanding it. For example, code that runs a LINQ query to explain query syntax and then display every item in the output collection.

NOTE

You might notice that some of the topics are not currently following all the guidelines specified here. We're working towards achieving consistency throughout the site. Check the list of [open issues](#) we're currently tracking for that specific goal.

Contributor license agreement

You must sign the [.NET Foundation Contribution License Agreement \(CLA\)](#) before your PR is merged. This is a one-time requirement for projects in the .NET Foundation. You can read more about [Contribution License Agreements \(CLA\)](#) on Wikipedia.

The agreement: [net-foundation-contribution-license-agreement.pdf](#)

You don't have to sign the agreement up-front. You can clone, fork, and submit your PR as usual. When your PR is created, it is classified by a CLA bot. If the change is small (for example, you fixed a typo), then the PR is labeled with `cla-not-required`. Otherwise, it's classified as `cla-required`. Once you signed the CLA, the current and all future pull requests are labeled as `cla-signed`.

Metadata and Markdown template for .NET docs

7/30/2019 • 8 minutes to read • [Edit Online](#)

This dotnet/docs template contains examples of Markdown syntax, as well as guidance on setting the metadata.

When creating a Markdown file, you should copy the included template to a new file, fill out the metadata as specified below, and set the H1 heading above to the title of the article.

Metadata

The required metadata block is in the following sample metadata block:

```
---
title: [ARTICLE TITLE]
description: [usually a summary of your first paragraph. It gets displayed in search results, and can help
drive the correct traffic if well written.]
author: [GITHUB USERNAME]
ms.date: [CREATION/UPDATE DATE - mm/dd/yyyy]
---
# The H1 should not be the same as the title, but should describe the article contents
```

- You **must** have a space between the colon (:) and the value for a metadata element.
- Colons in a value (for example, a title) break the metadata parser. In this case, surround the title with double quotes (for example, `title: "Writing .NET Core console apps: An advanced step-by-step guide"`).
- **title**: Appears in search engine results. The title shouldn't be identical to the title in your H1 heading, and it should contain 60 characters or less.
- **description**: Summarizes the content of the article. It's usually shown in the search results page, but it isn't used for search ranking. Its length should be 115-145 characters including spaces.
- **author**: The author field should contain the **GitHub username** of the author.
- **ms.date**: The date of the last significant update. Update this on existing articles if you reviewed and updated the entire article. Small fixes, such as typos or similar do not warrant an update.

Other metadata is attached to each article, but we typically apply most metadata values at the folder level, specified in **docfx.json**.

Basic Markdown, GFM, and special characters

You can learn the basics of Markdown, GitHub Flavored Markdown (GFM), and OPS specific extensions in the general articles on [Markdown](#) and [Markdown reference](#).

Markdown uses special characters such as *, ` , and # for formatting. If you wish to include one of these characters in your content, you must do one of two things:

- Put a backslash before the special character to "escape" it (for example, `*` for a *).
- Use the [HTML entity code](#) for the character (for example, `*` for a *).

File names

File names use the following rules:

- Contain only lowercase letters, numbers, and hyphens.
- No spaces or punctuation characters. Use the hyphens to separate words and numbers in the file name.
- Use action verbs that are specific, such as develop, buy, build, troubleshoot. No -ing words.
- No small words - don't include a, and, the, in, or, etc.
- Must be in Markdown and use the .md file extension.
- Keep file names reasonably short. They are part of the URL for your articles.

Headings

Use sentence-style capitalization. Always capitalize the first word of a heading.

Text styling

Italics Use for files, folders, paths (for long items, split onto their own line), new terms.

Bold Use for UI elements.

`Code` Use for inline code, language keywords, NuGet package names, command-line commands, database table and column names, and URLs that you don't want to be clickable.

Links

See the general article on [Links](#) for information about anchors, internal links, links to other documents, code includes, and external links.

The .NET docs team uses the following conventions:

- In most cases, we use the relative links and discourage the use of `~/` in links because relative links resolve in the source on GitHub. However, whenever we link to a file in a dependent repo, we'll use the `~/` character to provide the path. Because the files in the dependent repo are in a different location in GitHub the links won't resolve correctly with relative links regardless of how they were written.
- The C# language specification and the Visual Basic language specification are included in the .NET docs by including the source from the language repositories. The markdown sources are managed in the [csharp](#) and [vb](#) repositories.

Links to the spec must point to the source directories where those specs are included. For C#, it's `~/_csharp/spec` and for VB, it's `~/_vb/spec`, as in the following example:

```
[C# Query Expressions](~/_csharp/spec/expressions.md#query-expressions)
```

Links to APIs

The build system has some extensions that allow us to link to .NET APIs without having to use external links. You use one of the following syntax:

1. Auto-link: `<xref:UID>` or `<xref:UID?displayProperty=nameWithType>`

The `displayProperty` query parameter produces a fully qualified link text. By default, link text shows only the member or type name.

2. Markdown link: `[link text](xref:UID)`

Use when you want to customize the link text displayed.

Examples:

- `<xref:System.String>` renders as [String](#)
- `<xref:System.String?displayProperty=nameWithType>` renders as [System.String](#)
- `[String class](xref:System.String)` renders as [String class](#)

For more information about using this notation, see [Using cross reference](#).

Some UIDs contain the special characters ` , # or * , the UID value needs to be HTML encoded as `%60` , `%23` and `%2A` respectively. You'll sometimes see parentheses encoded but it's not a requirement.

Examples:

- `System.Threading.Tasks.Task`1` becomes `System.Threading.Tasks.Task%601`
- `System.Exception.#ctor` becomes `System.Exception.%23ctor`
- `System.Lazy`1.#ctor(System.Threading.LazyThreadSafetyMode)` becomes `System.Lazy%601.%23ctor%28System.Threading.LazyThreadSafetyMode%29`

You can find the UIDs of types, a member overload list, or a particular overloaded member from

`https://xref.docs.microsoft.com/autocomplete` . The query string `?text=*<type-member-name>*` identifies the type or member whose UIDs you'd like to see. For example,

`https://xref.docs.microsoft.com/autocomplete?text=string.format` retrieves the [String.Format](#) overloads. The tool searches for the provided `text` query parameter in any part of the UID. For example, you can search for member name (`ToString`), partial member name (`ToStri`), type and member name (`Double.ToString`), etc.

If you add a `*` (or `%2A`) after the UID, the link then represents the overload page and not a specific API. For example, you can use that when you want to link to the [List<T>.BinarySearch Method](#) page in a generic way instead of a specific overload such as [List<T>.BinarySearch\(T, IComparer<T>\)](#). You can also use `*` to link to a member page when the member is not overloaded; this saves you from having to include the parameter list in the UID.

To link to a specific method overload, you must include the fully qualified type name of each of the method's parameters. For example, `<xref:System.DateTime.ToString>` links to the parameterless [DateTime.ToString](#) method, while `<xref:System.DateTime.ToString(System.String,System.IFormatProvider)>` links to the [DateTime.ToString\(String,IFormatProvider\)](#) method.

To link to a generic type, such as [System.Collections.Generic.List<T>](#), you use the ``` (`%60`) character followed by the number of generic type parameters. For example, `<xref:System.Nullable%601>` links to the [System.Nullable<T>](#) type, while `<xref:System.Func%602>` links to the [System.Func<T,TResult>](#) delegate.

Code

The best way to include code is to include snippets from a working sample. Create your sample following the instructions in the [contributing to .NET](#) article. The basic rules for including code are located in the general guidance on [code](#).

You can include the code using the following syntax:

```
[!code-<language>[<name>](<pathToFile><queryoption><queryoptionvalue>)]
```

- `-<language>` (*optional but recommended*)
 - Language of the code snippet being referenced.
- `<name>` (*optional*)
 - Name for the code snippet. It doesn't have any impact on the output HTML, but you can use it to improve the readability of your Markdown source.

- `<pathToFile>` (*mandatory*)
 - Relative path in the file system that indicates the code snippet file to reference. This can be complicated by the different repos that make up the .NET doc set. The .NET samples are in the dotnet/samples repo. All snippet paths would start with `~/samples`, the rest of the path being the path to the source from the root of that repo.
- `<queryoption>` (*optional*)
 - Used to specify how the code should be retrieved from the file:
 - `# : #{tagname}` (tag name) or `#L{startlinenumber}-L{endlinenumber}` (line range). We discourage the use of line numbers because they are very brittle. Tag name is the preferred way of referencing code snippets. Use meaningful tag names. (Many snippets were migrated from a previous platform and the tags have names such as `Snippet1`, `Snippet2` etc. That practice is much harder to maintain.)
 - `range : ?range=1,3-5` A range of lines. This example includes lines 1, 3, 4, and 5.

We recommend using the tag name option whenever possible. The tag name is the name of a region or of a code comment in the format of `Snippettagname` present in the source code. The following example shows how to refer to the tag name `BasicThrow`:

```
[!code-csharp[csrefKeyword#1](~/samples/snippets/snippets/csharp/language-reference/operators/ConditionalExamples.csConditionalRef)]
```

The relative path to the source in the **dotnet/samples** repo follows the `~/samples` path.

And you can see how the snippet tags are structured in [this source file](#). For details about tag name representation in code snippet source files by language, see the [DocFX guidelines](#).

The following example shows code included in all three .NET languages:

```
[!code-fsharp[ToPigLatin](../../samples/snippets/fsharp/getting-started/pig-latin.fs#L1-L14)]
[!code-csharp[ADCreateDomain#2]
(../../samples/snippets/csharp/Vs_Snippets_CLR/ADCreateDomain/CS/source2.cs#2)]
[!code-vb[ADCreateDomain#2]
(../../samples/snippets/visualbasic/Vs_Snippets_CLR/ADCreateDomain/VB/source2.vb#2)]
```

Including snippets from full programs ensures that all code runs through our Continuous Integration (CI) system. However, if you need to show something that causes compile time or runtime errors, you can use inline code blocks.

Images

Static image or animated GIF

```
![this is the alt text](../images/Logo_DotNet.png)
```

Linked image

```
![alt text for linked image](../images/Logo_DotNet.png)(https://dot.net)
```

Videos

Currently, you can embed both Channel 9 and YouTube videos with the following syntax:

Channel 9

```
> [!VIDEO <channel9_video_link>]
```

To get the video's correct URL, select the **Embed** tab below the video frame, and copy the URL from the `<iframe>` element. For example:

```
> [!VIDEO https://channel9.msdn.com/Blogs/dotnet/NET-Core-20-Released/player]
```

YouTube

To get the video's correct URL, right-click on the video, select **Copy Embed Code**, and copy the URL from the `<iframe>` element.

```
> [!VIDEO <youtube_video_link>]
```

For example:

```
> [!VIDEO https://www.youtube.com/embed/Q2mMbjw6cLA]
```

docs.microsoft extensions

docs.microsoft provides a few additional extensions to GitHub Flavored Markdown.

Checked lists

A custom style is available for lists. You can render lists with green check marks.

```
> [!div class="checklist"]
> * How to create a .NET Core app
> * How to add a reference to the Microsoft.XmlSerializer.Generator package
> * How to edit your MyApp.csproj to add dependencies
> * How to add a class and an XmlSerializer
> * How to build and run the application
```

This renders as:

- How to create a .NET Core app
- How to add a reference to the Microsoft.XmlSerializer.Generator package
- How to edit your MyApp.csproj to add dependencies
- How to add a class and an XmlSerializer
- How to build and run the application

You can see an example of checked lists in action in the [.NET Core docs](#).

Buttons

Button links:

```
> [!div class="button"]
> [button links](dotnet-contribute.md)
```

This renders as:

You can see an example of buttons in action in the [Visual Studio docs](#).

Step-by-steps

```
>[!div class="step-by-step"]  
> [Pre](../docs/csharp/expression-trees-interpreting.md)  
> [Next](../docs/csharp/expression-trees-translating.md)
```

You can see an example of step-by-steps in action at the [C# Guide](#).

Voice and tone guidelines

7/30/2019 • 3 minutes to read • [Edit Online](#)

A wide variety of people including IT Pros and developers read your documents both to learn .NET and to use it in their regular work. Your goal is to create great documentation that helps the reader on their journey. Our guidelines help with that. Our style guide contains the following recommendations:

You can see examples of each of these as you read this style guide. We've written this guide following our guidelines to provide examples for you to follow as you build documentation for .NET. We also provide contrasting samples so you can see what articles look like when you don't follow our guidelines.

Use a conversational tone

Appropriate style

We want our documentation to have a conversational tone. You should feel as though you are having a conversation with the author as you read any of our tutorials or explanations. For you, the experience should be informal, conversational, and informative. Readers should feel as though they are listening to the author explain the concepts to them.

Inappropriate style

One might see the contrast between a conversational style and the style one finds with more academic treatments of technical topics. Those resources are very useful, but the authors have written those articles in a very different style than our documentation. When one reads an academic journal, one finds a very different tone and a very different style of writing. One feels that they are reading a dry account of a very dry topic.

The first paragraph above follows our recommendation conversational style. The second is a more academic style. You see the difference immediately.

Write in second person

Appropriate style

You should write your articles as though you are speaking directly to the reader. You should use second person often (as I just have in these two sentences). 2nd person doesn't always mean using the word 'you'. Write directly to the reader. Write imperative sentences. Tell your reader what you want them to learn.

Inappropriate style

An author could also choose to write in third person. In that model, an author must find some pronoun or noun to use when referring to the reader. A reader might often find this third person style less engaging and less enjoyable to read.

The first paragraph follows our recommended style. The second uses third person. Please write in second person. You probably found that much easier to read.

Use active voice

Write your articles in active voice. Active voice means that the subject of the sentence performs the action (verb) of that sentence. It contrasts with passive voice, where the sentence is arranged such that the subject of the sentence is acted upon. Contrast these two examples:

The compiler transformed source code into an executable.

The source code was transformed into an executable by the compiler.

The first sentence uses active voice. The second sentence was written in passive voice. (Those two sentences provide another example of each style).

We recommend active voice because it is more readable. Passive voice can be more difficult to read.

Target a fifth grade reading level

We provide this final guideline because many of our readers are not native English speakers. You are reaching an international audience with your articles. Please take into account that they may not have the English vocabulary you have.

However, you are still writing for technical professionals. You can assume that your readers have programming knowledge and the specific vocabulary for programming terms. Object Oriented Programming, Class, and Object, Function and Method are familiar terms. However, not everyone reading your articles has a formal computer science degree. Terms like 'idempotent' probably need to be defined if you use it:

The `close()` method is idempotent, meaning that you can call it multiple times and the effect is the same as if you called it once.

Avoid future tense

In some non-English languages the concept of future tense is not the same as English. Using future tense can make your documents harder to read. Additionally, when using the future tense, the obvious question is when. So if you say 'Learning PowerShell will be good for you' - the obvious question for the reader is when will it be good? Instead, just say "Learning PowerShell is good for you".

What is it - so what?

Whenever you introduce a new concept to the reader, define the concept and only then explain why it's useful. It's important for reader to understand what something is before they can understand the benefits (or otherwise).

Pull request review process for the .NET docs repositories

7/30/2019 • 3 minutes to read • [Edit Online](#)

Many repositories don't have PR Merger webhooks enabled, which automatically merges minor PRs. This article describes the PR review process for those repositories. The PR review process was designed with these goals:

1. Publish high-quality content from our team, product team members, and community members.
2. Provide timely, actionable feedback to authors in a consistent manner.
3. Facilitate discussion between authors and reviewers.

The processes continue to evolve as teams innovate, and as the platform matures.

Reviewers

One of the content team members reviews every PR. Content team members may request a review from the specific product team members to verify technical accuracy. The content team uses GitHub's Code Ownership feature to automatically request reviews from content team members. As part of this process, a reviewer may tag other team members to review internal PRs. Team members see requested reviews from team members and community members in the same queue.

Community members can review PRs and provide feedback as well. However, at least one member of the core content team must approve any PR before it is merged.

Review process

Reviews follow one of the three paths based on the PR:

- **Small PRs** - Small PRs are single bug fix: typos, broken links, small code changes, or similar changes.
- **Major contributions** - Major contributions are significant edits to an existing article, new articles, or edits to a number of articles.
- **Work in progress** - Authors can request an in-progress review by opening a PR with the "[WIP]" tag in the title. The "WIP" tag stands for "Work in Progress."

The processing used by the Contributor License Agreement (CLA) bot is a good guideline for the distinction between "small" and "large" contributions. PRs that do not require the CLA to be signed are likely "small." PRs that do require the CLA are likely "large."

Small PRs

The changes in small PRs are reviewed for accuracy, and checked using the build on the review site. Because they are small, these PRs do not trigger a review of the entire article.

Reviewers may notice other small changes that would improve an article. If those changes are also small, suggest them as review comments. If the suggested changes would trigger a larger review, open an issue and address those later.

Larger changes

Larger PRs undergo a more thorough review. The following are all checked:

- Sample code must be included in the PR, in source and as a downloadable zip file.
- Sample code compiles and runs correctly.

- The article clearly describes the goals for the reader, and those goals are met.
- The article meets [style and grammar guidelines](#) and follows our [voice and tone principles](#).
- All links should resolve correctly.

Content team members will review the PR, and submit the review with comments. If only minor changes are requested, team members may "approve" the PR with the feedback. The author can then address the feedback and merge the PR. Most reviews will request changes and when those changes are made, the reviewer will review again.

If the edits require a technical review, the content team reviewer will request a review from the appropriate product team member.

Review "WIP" pull requests

New authors may want feedback earlier in the process. They can open a PR adding the "[WIP]" label to the title. In a comment, they can request an early review.

These early reviews are not as thorough as a full PR review. The content team will comment, but will not "approve" or "request changes" using GitHub's review feature. These early reviews focus on the structure of the article: the outline, the overall content, and the samples. These reviews do not include a thorough check for grammar and correct links.

Respond to reviews

The author updates the PR to respond to comments marking each addressed comment with the "+1" reaction to indicate it was addressed. If the author disagrees with the comment, or addresses the comment in a different, the author adds a comment explaining their change.

The author @-mentions the original reviewer in a comment to request a new review.

Response time expectations

Content team members will typically review new PRs in under two business days. If it takes longer to review, one of the team members will add a comment to acknowledge the PR and set expectations.

Once a review has been submitted, authors should try to respond to comments in a week or less. Volunteers may not be able to meet these expectations because of other commitments. In those cases, team members ask if the community author will update the PR. If not, the team member updates and merges the PR for them.

Additional Git and GitHub resources

7/30/2019 • 2 minutes to read • [Edit Online](#)

If you are unfamiliar with Git or GitHub, these resources can help you learn, be productive, or answer questions.

Git source control resources

- [Git basics](#): This has a basic overview of how git works.
- [Pro Git e-book \(web\)](#): This is a thorough git reference, in HTML format.
- [Pro Git e-book \(PDF\)](#): Same as the preceding link, in PDF form.
- [Learn Git course from Codecademy](#): Git tutorial from Codecademy.
- [Try Git course from Code School on Pluralsight](#): Git tutorial from Code School on Pluralsight.
- [Git and Github course from Udacity](#): Git and Github tutorial from Udacity

GitHub resources

- [15-minute interactive primer](#): This is an online git tutorial. It exposes you to the basics of git.
- [Cheat sheets](#): A quick reference to common GitHub commands and workflows.
- [GitHub Guides](#): The home of GitHub documentation.
- [GitHub learning resources](#): Other useful GitHub resources.
- [GitHub training services](#): A listing of tutorials and training offerings from GitHub.
- [Glossary](#): A handy glossary of git and GitHub terms.
- [GitHub Student developer pack](#): Give students free access to the best developer tools.