**Creation of Azure DevOps pipeline extension**

[PART-1]

**Objective**

* Task in Azure DevOps
* Pre-requisite/Dependencies
* Task folder/directory structure
  1. Create the folder structure
  2. Expand the structure
* Overview of files and it’s attributes
* [Quiz](https://forms.office.com/e/X0ivyqSV5q)

**Pre-requisites/Dependencies**

1. **Azure DevOps Organization with project**
2. **Node.js (ver 10.x or higher) & NPM**

* Node.js is required to **run the Azure DevOps CLI tools**. Installing Node.js will also install NPM (Node Package Manager), which is used to install other dependencies.
* Download: [Here](https://nodejs.org/) (accept the default options in the setup wizard)

1. **TFX CLI (Azure DevOps Extension Tool)**

* TFX CLI is used to create, package, and publish Azure DevOps extensions.
* Install via NPM (use any terminal):   
  ***npm install -g tfx-cli***

1. **Visual Studio Code**

* [Visual Studio Code](https://code.visualstudio.com/?WT.mc_id=AZ-MVP-5003674)
* [The VSCode PowerShell Extension](https://code.visualstudio.com/docs/languages/powershell?WT.mc_id=AZ-MVP-5003674)

**Create the folder/directory structure**

* To build a custom Azure DevOps pipeline task, we need a specific folder and file structure.   
    
  A screenshot of a computer program

  AI-generated content may be incorrect.
* This structure can be created manually, but we can also use **TFX CLI**Use the following command to generate the base structure:  
    
  *tfx build tasks create --task-name HelloWorld --friendly-name Hello World Task --description "Prints Hello World to the console." --author "Abhishek Kumar Singh"*  
  A screenshot of a computer

  AI-generated content may be incorrect.
* **Note:** Replace the values (HelloWorld, description, author, etc.). These can be modified later in the generated files.

|  |  |
| --- | --- |
| task.json | Describe the task input parameters and the main entry point. |
| Sample.ps1 | Sample PowerShell script (can be replaced).   Executed as configured in task.json. |
| Sample.js | Sample JavaScript file (optional, can be deleted). |
| icon.png | Default 32×32 icon for the task (replace with ours), that will display in the marketplace. |

**Expand the same folder structure**

A screenshot of a computer program

AI-generated content may be incorrect.

**Readme.md file (Optional)**

While not required for the functionality of our extension, a README.md file is highly recommended—especially if we plan to publish our extension publicly on the Azure DevOps Marketplace.

**vss-extension.json – The Manifest File**

It defines general metadata and controls how and where our extension integrates into Azure DevOps.

**Key Responsibilities:**

* Declares the **extension name**, **version**, **publisher**, and **icon**.
* Specifies the **target area** within Azure DevOps (e.g., Pipeline Tasks).
* Lists the **files** and **contributions** included in the extension.
* Publish our extension

In our case, we are embedding the extension into **Pipeline Tasks**, so the manifest will include a contribution of type **ms.vss-distributed-task.task.**

The syntax of the file can be found [here.](https://docs.microsoft.com/en-us/azure/devops/extend/develop/manifest?view=azure-devops&WT.mc_id=AZ-MVP-5003674)

|  |  |
| --- | --- |
| manifestVersion | Version of the manifest format.  Always set to 1. |
| id | Unique identifier for the extension.  This is a string that must be unique among extensions from the same publisher.  Contain ‘A’ through ‘Z’, ‘a’ through ‘z’, ‘0’ through ‘9’, and ‘-‘ (hyphen). |
| version | Extension version (e.g., 1.0.0)  task.json and vss-extension.json need to match and each time we want to update the package we need to update the version. |
| name | Human-readable name of the extension  A screenshot of a computer  AI-generated content may be incorrect. |
| description | Short description shown in the Marketplace  A screenshot of a computer  AI-generated content may be incorrect. |
| publisher | Ours publisher ID (set after creating a publisher profile)  A screenshot of a computer  AI-generated content may be incorrect. |
| **categories** | An array of strings representing the categories where extension belongs to.  At least 1 category must be added (e.g., Azure Repos, Azure Boards, Azure Pipelines, Azure Test Plans, and Azure Artifacts) |
| **targets** | Specifies where the extension is used (e.g., Microsoft.VisualStudio.Services)  See[installation targets](https://docs.microsoft.com/en-us/azure/devops/extend/develop/manifest#installation-targets) for more details. |
| **icons** | Icon should be 128×128 pixels in size and uploaded in one of these formats: BMP, GIF, EXIF, JPG, PNG or TIFF. |
| files | Lists folders/files to include in the extension |
| contributions | Each task **MUST** have a corresponding contribution.  Each contribution entry has the following properties:   * **id** – A unique identifier for this contribution. It must be unique within our extension. * **type** – This tells Azure DevOps what kind of contribution this is. For pipeline tasks, it should always be "ms.vss-distributed-task.task". * **targets** – Specifies where this contribution will appear. "ms.vss-distributed-task.tasks" means it will show up in the **pipeline task catalog**. * **properties** – The "name" here must match the folder name of our task (e.g., HelloWorld) and the name property in our task.json. This links the manifest to the actual task logic.   For more information, see the[contribution model overview](https://docs.microsoft.com/en-us/azure/devops/extend/develop/contributions-overview?view=azure-devops). |

**Task folders**

This modular structure allows us to manage multiple tasks (e.g., Build, Test, Publish) within a single extension.  
 **ps\_modules folders - Secondary Modules**

The ps\_modules folder is used to store **secondary PowerShell modules** that our task depends on.

**Add the VstsTaskSdk module**

* For most PowerShell-based Azure DevOps tasks, we’ll need the **VstsTaskSdk** module. This SDK allows our script to:
  + Interact with **UI input parameters** defined in task.json
  + Use helper functions like Get-VstsInput, Write-VstsSetResult, and more…
* **Create the ps\_modules folder** inside our task directory.
* **Download the module** using PowerShell:  
    
  Save-Module –Name VstsTaskSdk –Path .\<TaskFolder>\ps\_modules –Force  
    
  E.g.  
  Save-Module –Name VstsTaskSdk –Path .\HelloWorld\ps\_modules –Force  
  A screenshot of a computer

  AI-generated content may be incorrect.
* **Flatten the module structure**:
  1. The module will be saved in a versioned subfolder (e.g., 0.x).
  2. Move the contents of that subfolder directly into ps\_modules\VstsTaskSdk.
  3. Delete the versioned folder.

The result should look like this:  
  
A screenshot of a computer program

AI-generated content may be incorrect.

[PART-2]

**Objective**

* Create PS script (our task core logic)
* Configure task.json, vss-extension.json
* Create a Publisher profile
* Create the Extension package
* Publish the extension
* Install and use the extension
* Updating the Extension with newer versions

**Create the PowerShell script**

* The PowerShell script is the **core logic** of yours Azure DevOps task. It performs the actual operations defined by the user through the task UI.
* To retrieve values entered by the user in the Azure DevOps pipeline UI, use the Get-VstsInput command provided by the VstsTaskSdk module.

**Configure task.json**

It defines:

* The task’s identity and metadata
* UI controls (inputs)
* The script to execute
* Versioning and categorization

This file must be named exactly task.json.

The full syntax can be found [on GitHub](https://github.com/Microsoft/azure-pipelines-task-lib/blob/master/tasks.schema.json).

|  |  |
| --- | --- |
| name | Internal task name (alphanumeric only) |
| **friendlyName** | Display name in the Azure DevOps UI |
| description | Display description of the task. |
| instanceNameFormat | Default name shown when the task is added to a pipeline |
| **helpMarkDown** | Tooltip/help text shown when hovering over the question mark icon   Or, |
| **category** | Task category (e.g., Build, Utility, Deploy, Package, Tool) A screenshot of a computer  AI-generated content may be incorrect. |
| version | Task version in major.minor.patch format  Must be unique every time we upload.  Always increment the version number when updating the task. So failing to update it may result in the old version being executed. |
| **inputs** | Set the UI controls |
| **groups** | Allows us to create custom groups for yours UI bucket inputs. This is just a group box for our controls. |
| **execution** | The scripts the task will execute. |

**Vss-extension.json - Manifest file**To publish yours Azure DevOps task as an extension, we need to create a manifest file named **vss-extension.json**.   
  
This file defines the extension’s **identity** (name, version, publisher), specifies where the extension will appear in Azure DevOps (e.g., Pipeline Tasks), lists files and contributions included in the extension

The syntax of the file can be found [here.](https://docs.microsoft.com/en-us/azure/devops/extend/develop/manifest?view=azure-devops&WT.mc_id=AZ-MVP-5003674)

|  |  |
| --- | --- |
| manifestVersion | Version of the manifest format.  Always set to 1. |
| id | Unique identifier for the extension.  This is a string that must be unique among extensions from the same publisher.  Contain ‘A’ through ‘Z’, ‘a’ through ‘z’, ‘0’ through ‘9’, and ‘-‘ (hyphen). |
| version | Extension version (e.g., 1.0.0)  task.json and vss-extension.json need to match and each time we want to update the package we need to update the version. |
| name | Human-readable name of the extension |
| description | Short description shown in the Marketplace |
| publisher | Yours publisher ID (set after creating a publisher profile) |
| **categories** | An array of strings representing the categories where extension belongs to.  At least 1 category must be added (e.g., Azure Repos, Azure Boards, Azure Pipelines, Azure Test Plans, and Azure Artifacts) |
| **targets** | Specifies where the extension is used (e.g., Microsoft.VisualStudio.Services)  See[installation targets](https://docs.microsoft.com/en-us/azure/devops/extend/develop/manifest#installation-targets) for more details. |
| **icons** | Icon should be 128×128 pixels in size and uploaded in one of these formats: BMP, GIF, EXIF, JPG, PNG or TIFF. |
| files | Lists folders/files to include in the extension |
| contributions | Each task **MUST** have a corresponding contribution.  Each contribution entry has the following properties:   * **id** – A unique identifier for this contribution. It must be unique within yours extension. * **type** – This tells Azure DevOps what kind of contribution this is. For pipeline tasks, it should always be "ms.vss-distributed-task.task". * **targets** – Specifies where this contribution will appear. "ms.vss-distributed-task.tasks" means it will show up in the **pipeline task catalog**. * **properties** – The "name" here must match the folder name of yours task (e.g., HelloWorld) and the name property in yours task.json. This links the manifest to the actual task logic.   For more information, see the[contribution model overview](https://docs.microsoft.com/en-us/azure/devops/extend/develop/contributions-overview?view=azure-devops). |

**Create a Publisher profile**Before we can publish our extension to the Azure DevOps Marketplace, we need to create a Publisher Profile.

A publisher is our identity on the Marketplace. It allows we to:

* Manage our extensions
* Track downloads and usage
* Share or publish extensions to organizations or the public

This page in the [Microsoft Docs](https://docs.microsoft.com/en-us/azure/devops/extend/publish/overview?view=azure-devops&WT.mc_id=AZ-MVP-5003674#create-a-publisher)  shows how to do this step by step.

Once created, we’ll receive a **Publisher ID**. Use this ID to fill in the "publisher" field in yours vss-extension.json.

**Create the Extension Package**

* Once our extension is ready and our manifest files are configured, package it using the TFX CLI.
* In the root directory of our extension, run:  
    
  ***tfx extension create --manifest-globs vss-extension.json  
  Or  
  npx tfx-cli extension create  
    
  A screen shot of a computer

  AI-generated content may be incorrect.***
* This will generate a .vsix file in the same directory. This .vsix file is our deployable extension package.  
    
  A screenshot of a computer

  AI-generated content may be incorrect.

**Updating the Extension**

If we make changes to our task:

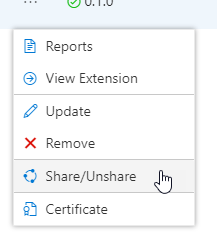
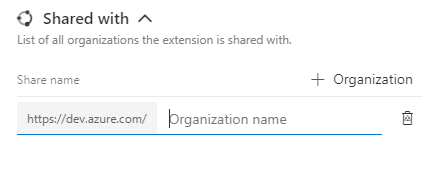
1. Update the version in both task.json and vss-extension.json.
2. Re-run the packaging command to generate a new .vsix file.

**Public or private**  
Private (default): Only visible to organizations we explicitly share it with.

Public: Visible to everyone on the Marketplace. There are some prerequisites to be verified for that, which we can find [here.](https://docs.microsoft.com/en-us/azure/devops/extend/publish/publicize?view=azure-devops&WT.mc_id=AZ-MVP-5003674)

**Publish the task**

* Go to the Azure DevOps Marketplace.
* Log in with our publisher account.
* Click New Extension > Azure DevOps.
* Upload yours .vsix file.  
    
  A screenshot of a computer

  AI-generated content may be incorrect.
* After verification, share it with specific organizations.
* Click the three-dot menu (⋮) next to our extension.
* Select Share/Unshare from the dropdown.  
    
  
* Type the name of the Azure DevOps organization we want to share the extension with.
* Once we type the name and click outside the field (or press Enter), the organization will be added.
* That organization will now be able to install and use our extension in their pipelines.  
    
  

**Install and use the extension**

* Go to the Azure DevOps Marketplace.
* Open the three-dot menu (⋮) next to our extension.   
  A screenshot of a computer

  AI-generated content may be incorrect.
* Click View Extension to access the extension’s detail page.
* Log in with an administrative account of the organization we shared the extension with.
* On the extension page, click Install.
* Select the organization where we want to install the extension.

**Testing extension**

* Package the Extension (Without Publishing)  
  ***tfx extension create --manifest-globs vss-extension.json***
* Upload to a Private Organization for Testing
  + Authenticate (if not already):  
    ***tfx login --auth-type pat --token <your-pat>***
  + ***tfx extension publish --vsix <your-extension-name>.vsix --publisher <your-publisher-id> --share-with <your-org-name> --auth-type pat --token <your-pat>***

A screenshot of a computer screen

AI-generated content may be incorrect.