Goals

Add test coverage for the Maestro.Task project with particular focus on the code that changed as part of the post-build signing changes. Each included method will have test coverage for the golden path(s) as well as interesting variations and missing/invalid input. I plan to start with the golden path and then use a combination of the VS code coverage tool and logic to identify the rest of the test cases for each method.

Note that the specific test cases listed are an initial guess based on reading code that I am new to and are likely to change as my understanding of the code changes.

Methods to be covered (ones that were changed for post-sign build have a * and will be the first to get coverage): **-** [ ] *PushMetadataAsync (this will get tests late even enough it was changed because most of its logic is calling the other methods listed below, so I'm going to use Code Coverage to guide the test case creation) - [ ] GetBuildDefaultChannelAsync - [ ] Positive cases, validate that the correct subset of channels is returned - [ ] Given all valid data (known good data set from a repo) - [ ] Given GitHub source data - [ ] Given AzDo source data - [ ] Given both GitHub & AzDo source data - [ ] Negative cases, validate that the returned data is empty or a meaningful exception is thrown - [ ] Given that both GitHub & AzDo are empty - [ ] Given empty/null values in other properties that are logged/checked - [ ] GetBuildDependenciesAsync - [ ] Positive cases, validate that the correct subset of BuildRefs is returned - [ ] Given golden path (build is found and it has a set of dependencies that have all relevant values filled in) - [ ] Given build that has product dependencies - [ ] Given build that has tool dependencies - [ ] Given a build with both product & tool dependencies - [ ] Given a build with non-required fields empty (should have no impact on the returned list) - [ ] Given one dependency with no builds found (GetBuildId returns null & this method logs then continues, return an empty list) - [ ] Given a build with no dependencies (return an empty list) - [ ] Negative cases, expect a meaningful exception is thrown - [ ] Given an empty RepoRoot - [ ] Given that the RepoRoot is not defined in the input file - [ ] Given an incorrect (invalid) RepoRoot - [ ] GetBuildId - [ ] Positive cases, validate that the correct buildId is returned and that the correct assets have been added to the assetCache - [ ] Golden path (given assets have a matching commit, buildId has a value, and build has assets in it aren't in the given list) - [ ] Given assets are the only assets for build - [ ] Negative, expect a meaningful expection is thrown - [ ] Given assets don't have a matching commit so no buildId is found (no exception, return null) - [ ] Given assets are missing field values that are expected/used in logic - [ ] Given null/empty arguments (if possible from caller) - [ ]* **GetBuildManifestsMetadata - [ ] Positive cases, validate contents of returned values (buildsManifestMetadata, signingInfo, manifestBuildData) - [ ] Given a single manifest - [ ] Given multiple manifests - [ ] Given no manifests - [ ] Negative cases, expect a meaningful exception is thrown - [ ] Given a file that isn't a manifest - [ ] Given badly formatted XML**

- [ ] Given an empty manifest file (valid XML formatting but with nothing in it) - [ ] Given a set of manifests missing various pieces of expected data (examples below, not a complete matrix) - [ ] Manifest without any assets listed - [ ] Manifest that has an asset that contains a package with no version set - [ ] Manifest that does not contain any Blobs - [ ] Manifest with a blob that does not have a version set - [ ] Manifest with a blob that does not have any assets - [ ] Given two manifests that have different attributes (expect exception thrown in method) - [ ] AddAsset - [ ] Positive cases, validate that the asset has been added to to the asset list - [ ] Given golden path with a combo of shipping and non-shipping assets - [ ] Given empty string parameters - [ ] Negative cases, expect a meaningful exception is thrown - [ ] Given null parameters (if allowed by caller) - [ ] MergeBuildManifests - [ ] Positive cases, validate the contents of the merged manifest BuildData - [ ] Given the golden path (two BuildData objects with compatible manifests in GitHub) - [ ] Given the golden path with a mirrored repo - [ ] Given three compatible BuildDatas - [ ] Given compatible BuildDatas with null/empty assets - [ ] Given compatible BuildDatas with existent but partially empty/invalid assets - [ ] Negative cases, expect a meaningful exception is throwm - [ ] Given two incompatible BuildDatas - [ ] Given compatible BuildDatas with duplicated assets - [ ] *MergeSigningInfo - [ ] Positive cases, validate the content of the merged SigningInformation - [ ] Given two SigningInformation objects that are compatible and contain different information - [ ] Given two duplicate SigningInformation objects - [ ] Given two SigningInformation objects where one is missing some values - [ ] Negative cases, expect a meaningful exception - [ ] Given two SigningInformation objects that are not compatible (exception thrown by method) - [ ] Given null/empty arguments (if possible from caller) - [ ] LookingForMatchingGitHubRepository - [ ] Positive cases, validate that the BuildData is updated with the correct infomration - [ ] Given a BuildData that is based on GitHub (non-mirrored repo) - [ ] Given a BuildData that is based on AzDo (mirrored repo) where GitHub is the current mirror - [ ] Given a BuildData that is based on AzDo (mirrored repo) where AzDo is the current mirror - [ ] Negative cases, expect a meaningful exception - [ ] Given a BuildData with an invalid url for the AzureDevOpsRepository value - [ ] Given null/empty arguments (if possible from caller) - [ ]* **GetManifestAsAsset - [ ] Positive cases, validate the contents of the AssetData and that it has been added to blobSet - [ ] Given a list of AssetData with a location string and a manifest file that exists - [ ] Given a list of AssetData with a location string and a manifest file that does not exist - [ ] Given a golden path where the AssetVersion is set - [ ] Given a golden path where the AssetVersion is not set, but there is a non-shipping asset with a version - [ ] Given a golden path where the AssetVersion is not set and there is not a non-shipping asset with a version - [ ] Given an empty list of AssetData (expect an empty list returned) - [ ] Given a null location - [ ] Negative cases, expect a meaningful exception - [ ]**

Given other null/empty arguments (if allowed by caller) - [ ] **Given blobSet that already has a key with the same name as an asset (if possible to get into that state) -** [ ] *CreateAndPushMergedManifest - [ ] Positive cases, validate content of manifest file - [ ] Given a list of compatible assets with SigningInformation & ManifestBuildData within expected values - [ ] Given assets that are blobSet - [ ] Given assets that are not in blobSet - [ ] Given assets where some are in blobSet and some are not - [ ] Given assets that are non-shipping - [ ] Given assets that are shipping - [ ] Given other empty/null arguments (as allowed by caller) - [ ] Given a mergedManifestPath that does not exist - [ ] Given a mergedManifestPath that already contains a file with the same name - [ ] Negative cases, expect a meaningful exception - [ ] Given null SigningInformation - [ ] Given SigningInfo that has some null/empty values - [ ]* **SigningInfoToXml (this is going to be a fine line between testing our logic and testing XDocument, so these test cases are very likely to evolve as I write them) - [ ] Positive cases, validate content of the returned XElement - [ ] Given a SigningInformation with a single one of each of the types parsed and all values filled in - [ ] Given a SigningInformation with multiple of the types parsed, all of which contain the expected values - [ ] Negative cases, expect a meaningful exception - [ ] Given a SigningInformation missing various top level values that are assumed present (if allowed from caller) - [ ] Given a SigningInformation with nested values missing - [ ] Scenario Tests - [ ] Positive cases, validate that the BAR database has been updated with the correct information and that the new build exists - [ ] All values and input are valid throughout the pipeline - [ ] Negative cases, a user friendly error message is returned - [ ] RepoRoot is null or empty - [ ] AssetVersion is null or empty

Non-Goals Test coverage for any code outside of the Maestro.task project.

Methods to be excluded:

- "GetXYZ" methods where it only returns a string from the environment or calls an outside function without applying interesting logic.

Risk This is fairly low risk, since it's going to be based on the other existing test projects. There are some changes required in the product code to allow for DI and mocking attachment points, that's the biggest risk in this set of changes.

Was this helpful?