Localization

> ***NOTE**: THIS DOCUMENT IS OUTDATED. Follow issue 15243 for updates.*

Table of Contents

Localization on the pipeline (CDPX)

The localization step is run on the pipeline before the solution is built. This step runs the build-localization script, which generates resx files for all the projects with localization enabled using the `Localization.XLoc` package.

The `Localization.XLoc` tool is run on the repo root, and it checks for all occurrences of `LocProject.json`. Each localized project has a `LocProject.json` file in the project root, which contains the location of the English resx file, list of languages for localization, and the output path where the localized resx files are to be copied to. In addition to this, some other parameters can be set, such as whether the language ID should be added as a folder in the file path or in the file name. When the CDPX pipeline is run, the localization team is notified of changes in the English resx files. For each project with localization enabled, a `loc` folder (see this for example) is created in the same directory as the `LocProject.json` file. The folder contains language specific folders which in turn have a nested folder path equivalent to `OutputPath` in the `LocProject.json`. Each of these folders contain one `lcl` file. The `lcl` files contain the English resources along with their translation for that language. These are described in more detail here. Once the `.resx` files are generated, they will be used during the `Build PowerToys` step for localized versions of the modules.

Since the localization script requires certain nuget packages, the `restore-localization` script is run before running `build-localization` to install all the required packages. This script must run in the `restore` step of pipeline because the host is network isolated at the `build` step. The Toolset package source is used for this.

The process and variables that can be tweaked on the pipeline are described in more detail here.

The localized resource dlls for C# projects are added to the MSI only for build on the pipeline. This is done by checking if the `IsPipeline` variable is defined, which gets defined before building the installer on the pipeline here. This is done because the localized resx files are only present on the pipeline, and not having this check would result in the installer project failing to build locally.

UWP Special case

C# projects normally expect localized resource files with the language id in the file name as Resources.`langId`.resx, where `langId` is generally a two character code except for language with specific variants (like zh-Hans or pt-BR):

For example, `path\Resources.resx` for English and `path\Resources.fr.resx` for French.

UWP differs from this as it expects the resources to have the same Resources.resw file name, but they should be present in language specific folders, with the full language ID (such as fr-fr, zh-hans, pt-br, etc.)

For example, `path\en-us\Resources.resw` for English and `path\fr-fr\Resources.resw` for French.

Since the pipeline generates it in this format, a script is run to move these resw files to the correct format expected by all UWP projects. Currently the only UWP project is Settings.UI. The script used for moving the resources can be found here. The equivalent full language IDs for each shortened language ID obtained from the pipeline has been hardcoded in the script.

Enabling localization on a new project

To enable localization on a new project, the first step is to create a file `LocProject.json` in the project root.

For example, for a project in the folder `src\path` where the resx file is present in `resources\Resources.resx`, the LocProject.json file will contain the following:

```
{
    "Projects": [
        {
            "LanguageSet": "Azure_Languages",
            "LocItems": [
                {
                    "SourceFile": "src\\path\\resources\\Resources.resx",
                    "CopyOption": "LangIDOnName",
                    "OutputPath": "src\\path\\resources"
                }
            ]
        }
    ]
}
```

The rest of the steps depend on the project type and are covered in the sections below. The steps to add the localized files to the MSI can be found here.

C++

C++ projects do not support `resx` files, and instead use `rc` files along with `resource.h` files. The CDPX pipeline however doesn't support localizing `rc` files and the other alternative they support is directly translating the resources from the binary which makes it harder to maintain resources. To avoid this, a custom script has been added which expects a resx file and converts the entries to an rc file with a string table and adds resource declarations to a resource.h file so that the resources can be compiled with the C++ project.

If you already have a .rc file, copy the string table to a separate txt file and run the convert-stringtable-to-resx.ps1 script on it. This script is not very robust to input, and requires the data in a specific format, where `IDS_ResName L"ResourceValue"` and any number of spaces can be present in between. The script converts this file to the format expected by `resgen`, which will convert it to resx. The resource names are changed from all uppercase to title case, and the `IDS_` prefix is removed. Escape characters might have to be manually replaced, for example .rc files would have escaped double quotes as `""`, so this should be replaced with just `"` before converting to the resx files.

After generating the resx file, rename the existing rc and h files to ProjName.base.rc and resource.base.h. In the rc file remove the string table which is to be localized and in the .h file remove all `#define`s corresponding to localized resources. In the vcxproj of the C++ project, add the following build event:

```
<Target Name="GenerateResourceFiles" BeforeTargets="PrepareForBuild">
    <Exec Command="powershell -NonInteractive -executionpolicy Unrestricted
$(SolutionDir)tools\build\convert-resx-to-rc.ps1 $(MSBuildThisFileDirectory)
resource.base.h resource.h ProjName.base.rc ProjName.rc" />
</Target>
```

This event runs a script which generates a resource.h and ProjName.rc in the `Generated Files` folder using the strings in all the resx files along with the existing information in resource.base.h and ProjName.base.rc. The script can be found here. The script uses `resgen` to convert the resx file to a string table expected in the .rc file format. When the resources are added to the rc file the `IDS_` prefix is added and resource names are in upper case (as it was originally). Any occurrences of `"` in the string resource is escaped as `""` to prevent build errors. The string tables are added to the rc file in the following format:

```
#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US

STRINGTABLE
BEGIN
    strings
END

#endif
```

Since there is no API to identify the `AFX_TARG_*`, `LANG_*` or `SUBLANG_*` values from each langId from the pipeline, these are hardcoded in the script (for each language) as done here. **If any other languages are added in the future, this script will have to be updated.** In order to determine what are the language codes, you can open the rc file in Resource View, right click the string table and press `Insert Copy` and choose the corresponding language. This autogenerates the required code and can be used to figure out the language codes. The files also add the resource declarations to a resource.h file, starting from 101 by default(this can be changed by an optional argument). Since the output files will be generated in `Generated Files`, any includes in these two files will require an additional `..\` and wherever resource.h is used, it will have to be included as `Generated Files\resource.h`. While adding `resource.base.h` and `ProjName.base.rc` to the vcxproj, these should be modified to not participate in the build to avoid build errors:

```
<None Include="Resources.resx" />
```

Some rc/resource.h files might be used in multiple projects (for example, KBM). To ensure the projects build for these cases, the build event can be added to the entire directory so that the rc files are generated before any project is built. See Directory.Build.targets for an example.

Check this PR for an example for making these changes for a C++ project.

C#

Since C# projects natively support `resx` files, the only step required here is to include all the resx files in the build. For .NET Core projects this is done automatically and the .csproj does not need to be modified. For other projects, the following line needs to be added:

```
<EmbeddedResource Include="Properties\Resources.*.resx" />
```

Note: Building with localized resources may cause a build warning `Referenced assembly 'mscorlib.dll' targets a different processor` which is a VS bug. More details can be found here.

Note: If a project needs to be migrated from XAML resources to resx, the easiest way to convert the resources would be to change to format to `=` separates resources by either manually (by Ctrl+H on a text editor), or by a script, and then running `resgen` on `Developer Command Prompt for VS` to convert it to resx format.

```
<system:String x:Key="wox_plugin_calculator_plugin_name">Calculator</system:String>
<system:String x:Key="wox_plugin_calculator_plugin_description">Allows to do
mathematical calculations.(Try 5*3-2 in Wox)</system:String>
<system:String x:Key="wox_plugin_calculator_not_a_number">Not a number (NaN)
</system:String>
```

to

```
wox_plugin_calculator_plugin_name=Calculator
wox_plugin_calculator_plugin_description=Allows to do mathematical calculations.(Try
5*3-2 in Wox)
wox_plugin_calculator_not_a_number=Not a number (NaN)
```

After adding the resx file to the project along with the resource generator, references to the strings will have to be replaced with `Properties.Resources.resName` rather than the custom APIs. Check this PR for an example of the changes required.

UWP

UWP projects expect `resw` files rather than `resx` (the format is almost the same). Unlike other C# projects, the files are expected in the format `fullLangId\Resources.resw`. To include these files in the build, replace the following line in the csproj:

```
<PRIResource Include="Strings\en-us\Resources.resw" />
```

to

```
<PRIResource Include="Strings\*\Resources.resw" />
```

Lcl Files

Lcl files contain all the resources that are present in the English resx file, along with a translation if it has been added.

For example, an entry for a resource in the lcl file looks like this:

```
<Item ItemId=";EditKeyboard_WindowName" ItemType="0;.resx" PsrId="211" Leaf="true">
    <Str Cat="Text">
        <Val><![CDATA[Remap keys]]></Val>
        <Tgt Cat="Text" Stat="Loc" Orig="New">
            <Val><![CDATA[Remapper des touches]]></Val>
        </Tgt>
    </Str>
```

```
    <Disp Icon="Str" />
</Item>
```

The `<Tgt>` element would not be present in the initial commits of the lcl files, as only the English version of the string would be present.

Note: The CDPX Localization system has a fail-safe check on the lcl files, where if the English string value which is present inside `<Val><![CDATA[*]]></Val>` does not match the value present in the English Resources.resx file then the translated value will not be copied to the localized resx file. This is present so that obsolete translations would not be loaded when the English resource has changed, and the English string will be used rather than the obsolete translation.

Possible Issues in localization PRs (LEGO)

Since the LEGO PRs update some of the strings in LCL files at a time, there can be multiple PRs which modify the same files, leading to merge conflicts. In most cases this would show up on GitHub as a merge conflict, but sometimes a bad git merge may occur, and the file could end up with incorrect formatting, such as two `<Tgt>` elements for a single resource. These can be fixed by ensuring the elements follow the format described in this section. To catch such errors, the build farm should be run for every LEGO PR and if any error occurs in the localization step, we should check the corresponding resx/lcl files for conflicts.

Enabling localized MSI for a new project

For C++ and UWP projects no additional files are generated with localization that need to be added to the MSI. For C++ projects all the resources are added to the dll/exe, while for UWP projects they are added to the `resources.pri` file (which is present even for an unlocalized project). To verify if the localized resources are added to the `resources.pri` file the following steps can be done:

- Open `Developer Command Prompt for VS`
- After navigating to the folder containing the pri file, run the following command:

  ```
  makepri.exe dump /if .\resources.pri
  ```

- Check the contents of the `resources.pri.xml` file that is generated from the command. The last section of the file will contain the resources with the strings in all the languages:

```
<NamedResource name="GeneralSettings_RunningAsAdminText" uri="ms-resource://f4f787a5-
f0ae-47a9-be89-5408b1dd2b47/Resources/GeneralSettings_RunningAsAdminText">
    <Candidate qualifiers="Language-FR" type="String">
        <Value>Running as administrator</Value>
    </Candidate>
    <Candidate qualifiers="Language-EN-US" isDefault="true" type="String">
        <Value>Running as administrator</Value>
    </Candidate>
</NamedResource>
```

For C# projects, satellite dlls are generated when the project is built. For a project named `ProjName`, files are created in the format `langId\ProjName.resources.dll` where `langId` is in the same format as the lcl files. The satellite dlls need to be included with the MSI, but they must be added only if the solution is built from the build farm, as the localized resx files will not be present on local machines (and that could cause local builds of the installer

to fail). This can be done by adding the directory name of the project here and a resource component for the project can be created here in this format:

```
<Component Id="ProjName_$(var.IdSafeLanguage)_Component"
Directory="Resource$(var.IdSafeLanguage)ProjNameInstallFolder">
    <File Id="ProjName_$(var.IdSafeLanguage)_File"
Source="$(var.BinX64Dir)modules\ProjName\$(var.Language)\ProjName.resources.dll" />
</Component>
```

We should also ensure the new dlls are signed by the pipeline. Currently all dlls of the form `*.resources.dll` are signed.

Note: The resource dlls should be added to the MSI project only after the initial commit with the lcl files has been done by the Localization team. Otherwise the pipeline will fail as there wouldn't be any resx files to generate the dlls.