Usage of ElectronAppInstallerCreator

Introduction

The `Battelle.TRA.Common.Utilities.ElectronAppInstallerCreator` utility is used to generate an electron application installer from a generic set of ASP NET Core projects. The utility was developed by Kevin Wegman (wegman@battelle.org) utilizing resources from the open source Electron.NET project.

Source Code Repository

Usage

To use the utility, download to a local location and and navigate to the directory with command prompt. The utility is currently stored at this share drive location.

The syntax to run the utility is:

```
path\to\Battelle.TRA.Common.Utilities.ElectronAppInstallerCreator.exe
path\to\inputFile.json
```

`path\to\Battelle.TRA.HExCAT.Utilities.ConfigUpdate.exe`:

- The path to the ElectronAppInstallerCreator utility
- ***NOTE***: if the utility is in your current directory, you can just provide the EXE name

`path\to\inputFile.json`:

- The path to the input file used to control the installation process
- If there are spaces in the path, you must use quotes around in the path
- The specification information for the input file is shown below
- ***Note***: all relative file paths specified in the json file will be relative to the directory the command is run from, ***not the directory of the input file or the directory of the utility***

Example Input File

A full input file example is provided below, with each input option section explained subsequently.

```
{
  "DeployDirectory": "Installer",
  "DatabaseFilePath": [
    "ApplicationDatabase.db"
  ],
  "PackageJson": "path\\to\\skeleton\\package.json",
  "ElectronManifest": "path\\to\\skeleton\\eletronManifest.json",
  "ElectronApplication": {
    "Name": "ApplicationName",
```

```
      "Location": "path\\to\\Application.zip",
      "FilesToCopy": [
        {
          "source": "path\\to\\additional\\resource.ext",
          "destination":
  "path\\relative\\to\\electron\\application\\root\\resource.ext"
        }
      ]
    },
    "SubServices": [
      {
        "Name": "Additional\\Service\\Name",
        "Location": "path\\to\\service\\resources\\Service.zip",
        "FilesToCopy": [
          {
            "source": "path\\to\\additional\\resource.ext",
            "destination":
  "path\\relative\\to\\electron\\application\\root\\resource.ext"
          }
        ]
      }
    ]
  }
```

`DeployDirectory`:

- The path to where the utility will create the installer
- This path can be absolute or relative to the execution directory
- ***NOTE*** - the user must have write access to the directory which is being installed to

`DatabaseFilePath`:

- A list of paths to database resources (i.e. SQLite .db files) which are copied and distributed to the application
- These files will end up in the `Data` directory in the final install
    - Explanation of the installation directory anatomy will be provided below

`PackageJson`:

- Optional parameter with path to a package.json file which is used in the installation
- If this input is not provided, the installer will default to using the package.json included in the WebApplication's SPA root folder
    - If there is no package.json, the installer will fail
- The NPM docs provide information on the layout and specification of a typical package.json file.
- Due to the way that the electron installer creator works, the installer package must include the following sections:

```
{
  "main": "main.js",
  "scripts": {
```

```
    "start": "tsc -p ."
  },
  "dependencies": {
    "electron-updater": "^4.0.6",
    "image-size": "^0.7.4",
    "portscanner": "^2.2.0",
    "socket.io": "^2.2.0"
  },
  "devDependencies": {
    "@types/node": "^10.14.4",
    "@types/socket.io": "^2.1.2",
    "tslint": "^5.12.0",
    "typescript": "^3.2.2",
    "electron": "^5.0.8"
  }
}
```

`ElectronManifest`:

- Optional parameter with path to a electron.manifest.json file which is used in the installation
- If this input is not provided, the installer will default to using the electron.manifest.json included in the ElectronApplication root folder
    - If there is no package.json, the installer will fail
- Inside, the manifest, the build section uses the common electron-builder format.
- In addition to the common electron configuration in the build section, the electron.manifest.json must have the following structure:

```
{
  "executable": "{{executable}}",
  "splashscreen": {
    "imageFile": ""
  },
  "singleInstance": false,
  "build": {
  }
}
```

`ElectronApplication`:

- A required subsection which provides information on the application which will be launched and ported to the electron window

`ElectronApplication.Name`:

- A required input which is used as the application root directory name
    - See the Anatomy of Application Created section below for more details

`ElectronApplication.Location`:

- A required subsection which provides the path to the application resources

    - This path can be either absolute or relative to the directory launched from
    - The path can point either to a zip directory or an unzipped directory

`ElectronApplication.FilesToCopy`:

- An optional input which provides a list of files to copy relative to the ElectronApplication root directory
- This is used to provide build resources to the application, such as installer icons, installer scripts or additional dependencies to install

`SubServices`:

- An optional subsection with the same subsections as the `ElectronApplication`
- The items listed in these subservices are copied to the application, but are not explicitly linked or referenced
    - To use the subservices, the `ElectronApplication` must be configured to reference and launch them

---

Anatomy of Application Created

Knowledge of the distributed application layout can be crucial if there are multiple interdependent processes or files which must be distributed appropriately. This section of the instructions will discuss the layout of a specific installer creation, the HExCAT application. The files used in the installation process for the HExCAT installation can be found here.

After the installer creator is run, there will be three files in the installation directory indicated by `DeployDirectory`:

- `win-unpacked`
    - The extracted version of the application in the format that will be deployed to client machines
- `{{ApplicationName}} Setup {{Version}}.exe`
    - The application installer - this is the file that should be distributed to users fo the application
- `{{ApplicationName}} Setup {{Version}}.exe.blockmap`
    - Blockmap file for the installer - as far as the creator of this document (Kevin Wegman) knows, this is not used or necessary

In the `win-unpacked` directory there will be a variety of files. Most of these files are dependencies or autogenerated documentation which come from the Electron-Builder. The items of interest in the installation directory are:

- `resources`
    - Resources which are utilized by the application when running
    - This directory contains the items indicated in the `input.json` configuration which are distributed to the application, such as `ElectronApplication`, `SubServices` and `DatabaseFilePath`
- `{{ApplicationName}}.exe`
    - The launcher for the application which will start the electron window and kick off the `ElectronApplication` server
    - This is the exe that is called when the users launches the application from the Windows start menu

The `resources` directory is fairly straightforward - each resource which was indicated in the installer configuration files are placed in this location. Each individual server will have its own directory with the name provided in `ElectronApplication.Name` or `SubServices[].Name`. The contents of these directories are unchanged from the contents provided to the installer scripts, unless additional files were provided in the `FilesToCopy` options. Any databases indicated by the `DatabaseFilePath` are contained within a directory named `Data`. Finally, a couple other resources which are autogenerated by the electron-builder are included in this directory.