Updater

This tool allows NaroCAD to automatically update. After installing NaroCAD, you will see two checkboxes in the options menu. When checked, the first one will activate automatically updates for stable versions and will make a second checkbox visible. The second checkbox tells NaroCAD to automatically search for newer nightly versions.

For now, the updater will start automatically and won't tell anything to the user. When an update is done, a message box will appear, telling the user that an update was successfully completed.

Detailed explanation

It is important to mention that NaroCAD is started by another process (NaroStarter) and both of them keep running until NaroCAD is closed. It's clear that one more process is needed for the updater, because files used by a process cannot be replaced when that process is running. So, we need another process in order to replace old files with the new one. Only one more thing needs to be explained. How to update the process that replaces the old files? It's clear that this must be made from another process. Since we have NaroStarter and NaroCAD, NaroCAD was chosen because most of the update logic should be found in one place. This means that the updater is done in 3 main steps:

Firstly, NaroCAD will download the new files;

Secondly, the update process will replace all new files except its running file

Thirdly, NaroCAD will download and replace the new update process's files

Changes in NaroStarter and ApplyUpdates projects:

A new project was made for this process: ApplyUpdates. The main role of this project is to replace the downloaded files with the old ones. But this process must be not started by the user; it is started from NaroStarter if certain conditions are met, conditions that ensure that the new files have been successfully downloaded: the auxiliary folder, where the new files are downloaded must exist and the file that tells ApplyUpdates that the new files were successfully downloaded must also exist. The names of the 2 folders are: auxiliary and DownloadSucceeded. When these conditions are met, NaroStarter will launch ApplyUpdater and will immediately close itself. After finishing its job, this process should launch NaroStarter again. If this were to remain like this, an infinite loop would occur. NaroStarter should somehow know if the ApplyUpdates has done its job. One solution would be to delete one of the directories named before. For example, the auxiliary directory should be deleted if the files were successfully moved. But what about the case when the ApplyUpdates process were to somehow fail in its job? The solution is to create a new folder named UpdatesFailed when NaroStarter is started. The ApplyUpdates process must only do its job and delete the auxiliary folder if successful, or do nothing if the attempt failed. This way, NaroStarter should know if the ApplyUpdates process has made an attempt to update NaroCAD, and the infinite loop is now solved. The only thing left to do is to delete the UpdatesFailed folder from NaroCAD. It cannot be deleted from NaroStarter because NaroStarter is the process that creates it. Even more, the ApplyUpdates is also unable to delete this folder, because it doesn't know how many times it has attempted to replace the old files with the new ones. The method that is responsible for the logic described above is named AttemptUpdate and it's called by NaroStarter.

The actual code:

What does ApplyUpdates actually do? This process was made as simple as possible, so that it should be changed as less as possible. This process actually does one thing: it replaces all old files with the new ones. Its minor objectives are to delete and create some helper folders, in order to communicate with the other processes. In case something goes wrong with the ApplyUpdates, the folder that tells him that the new files were successfully downloaded should be deleted. If everything works, this file should also be deleted, because there is no other use for it. That's why it's deleted either way in the ApplyUpdates process. One other minor objective of this process is to rename the old versioning file, so that it can be used by NaroCAD to compare the new versioning file with the old one. This logic could have been done here, but it's doe in the NaroCAD process because it needs more classes from other projects, and the only way to do it right were to create duplicates. In order to avoid this issue and keep this process as simple as possible, this logic is made in the NaroCAD process.

This process starts by renaming the versioning file. Afterwards, it attempts to replace all old files with the new ones. If it succeeds, some folders are created and deleted. If it fails, the versioning file is renamed, so that other attempts to update NaroCAD should be possible. Finally, the folder that tells ApplyUpdates that new files were downloaded is also deleted.

In order to rename the versioning file, a method that moves the file from a location to another is called. This method also can detect if the file is renamed, by analyzing the destination path. This method is also called when replacing the old files with the new ones by simply calling it with the correct source and destination paths. Its name is MoveWithReplaceInNaroCAD. In order to work correctly, the filepaths are verified and logging is done if something goes wrong. In oder to make sure that the detinaton path is valid, another method is called in order to create the destination path, if needed. This mehtod is called MakeSureThatDestinationDirectoriesExist and it has two parameters: the first one is the destination path, and the second one is the path to be ignored. Since most of the files have a common path this methods takes 2 parameters. In case a directory from the destination path does not exist, it is automatically created. In order to treat and log exeptions, two methods for creating and deleting directories were made: CreateDirectory and DeleteDirectory.

The second step is to replace the old files with the new ones. In order to do this the MoveAllDownloadedFiles is called. This method calls two other mehtods, one that creates the list of files that muse be moved, and the other one that actually moves the files. The CreateListOfFilesToBeMoved creates a list of strings and calls a recursive method that analyses the given directory and makes a fills the list with all existing files. The MoveFiles methods only needs to move all downloaded files to the corect locations. The MoveWithReplaceInNaroCAD is called here.

The Updater project:

The FileInformation class:

In order to keep all information about the NaroCAD files, the following information is kept:

- The source path: the path of the files needed for the instalation prcess
- The source path: the path were the file is to be copied when installing NaroCAD
- Length: the size of the file
- Hash: the has code of the file

This class is needs to be serialized for other purpses. This is why puclic methods were created. Since the members are private, puclic mehtods must exist, so that the serialization can be done automatically.

This class has two constructors , that allow creating these files with more or less information and two methods, that calculate the size and hash code of a given file.

The FileList class:

This class is a singleton class and the only imporatnt member is a list which contains the file information about every NaroCAD file that is needed by the installer. These files are the ones included in the versioning file. Because this list of files needs to be serialized, a property for this private member exists.

The files needed for the download can be found in the full.iss file. In order to avoid duplicate code, this class has the responsibility to read the data from there. Because the fact that the parameters needed from the .NET framework classes are different from the ones used in the full.iss file, special methods were creatd, methods that allow adding files just as like it happens in the full.iss file. This way, some mehtods were crated, methods that allow a file to be added to the list, a list of files with a cretain extension and all files from a directory. These methods also accept more or less parameters, dependig on how the code is found in the full.iss file. Some extra conditions had to be verified, because the GetFileName mehtods is not able to tell if a file without extension is a file or a folder. One extra condition was added because the versioning file must not contain inforation about itself and information about a process that creates the versioning data, because it is not for the users.

The only thing left to do is to read the information from the full.iss file. The CreateListOfFiles method is responsible for creating the needed file list and fillling it. In order to fill it, the AddFiles method is called. One of its parameters is the result given by the FindStartString method, which return the line from which useful information can be found. The AddFiles method interprets each line, until the end sctring is found. The lines are interprated with the help of the Interpret method. Since the useful information of a line is the source path, destination path and new name of a file, the Interpret method tries to find this information. Be aware that only the source path must exist. The other two are optional. So, the first task of the interpret method is to correcly find this information. The most easy case is that in which the destination name is not empty, because this information can only refer to a specific file, and contains all the information needed for adding a file to the list of files of this class. The other two cases are a bit more complicated, because the source path can refer to a single file, a type of files from a directory or all files from a directory. Because of this, another method was called, a method that is able to interpret the source path. The InterpretSource method can be called with or without the destintaion path. It's role is to call the correct method that adds the files to the list of files of this class and find the correct parameters for it. There are three methods responsible for finding the source path, the destination path and the destination name path. Each of these methods can identify the correct path by calling two methods that find the start and end index of the desired path. The only difference is that these methods are called with different parameters.

The VersionObject class:

This class is used to keep all the versioning data. An object of this class contains a VersionSpecificData object and the list of files from the FileList class. In order to easily serialize and deserialize this class, public properties were made for the private members.

The VersionSpecificData class:

This class contains information about the current version: the name of the new versions, a list of strings with the new features and a description. The members have public methods in order for the serialization and deserialization to work properly.

The UpdateLogic class:

This is the main class that takes care of the update process. The Setup method is called before starting the update process, in order to determine if the user wants to search for newer stable or nightly versions. The Main method of this class deletes the UpdatesFailed directories, as described above, and calculates the lengts of some NaroCAD strings. Afterwards, the updater tries to extract the version object by deserializing the version file. If this attempt fails, the process will end with a messagebox that tells the user that the version file is corupt. Without this information, no other updates can be made. The next step is to keep in memory the current version and the current list of NaroCAD files. The final step is to finalize or start the update process.

In order to finalize the update process, a file that tells NaroCAD that all old files were successfully replaced must exist. The last important thing to do is to download the correct ApplyUpdates files. To do this, the version number of the current version analysed, in oder to determine if a nightly or a stable update was done. This way, the ApplyUpdated file will be downloaded from the correct location. Notice that the DownloadNewApplyUpdates method downloads a zip file and makes calls the UnpackFile method, if the download was successful. Working with archiver files is very useful because less information is downloaded, and even more, if the unpack is successful, then the file was downloaded correctly. This is a very good method to check if a download succeded or not. The FinalizeUpdates method has two parameters: the old and new list of files. After succesfully downloading the last file, all auxiliary folders and files are deleted, and a method that deletes unused files from previous is called. The DeleteUslessData method deletes a list of files that is given as a parameter. This list of files is calculated by the CreateListOfFilesToBeDeleted method. This method actually analyses the two lists of files, and determines what old files are no longer in use. With this final step, the update process is completed.

The update process starts with an attempt to download the new NaroCAD files. The method that is repsonsible for this is called AttemptToDownloadUpdateFiles. The first thing to do is to check if the user still wants to update NaroCAD. It is possible that the updater was canceled and earlyer auxiliary files might be on disk. So, if no new version should be downloaded, the auxiliary folder will be deleted, if it exists on disk. If the user wants to update NaroCAD , the new version file is downloaded in the auxiliary folder, and the two version files are compared. If the downloaded version is newer than the current version, the update process will continue. In any other case, it will end for the current NaroCAD session. The DownloadFiles method will be called if the update process continues. This method calls two other important methods: ListOfFilesToBeAdded and DowloadUpdateFiles. The download update files simply downloads the files given as a parameter to the correct location. The files that are downloaded are unpacked right after the download. If this process fails, the update process will come to an end and will continue the next time NaroCAD is started. The ListOfFilesToBeAdded method analyses which files should be downloaded. Even the files that should exist on disk are checked and if these files are corupt or missing, the ones from the newer version will be downloaded. The only file that is in the lists and should not be downloaded, is the ApplyUpdates files. This file was treated as a special download above. There are two kind of verifications: the first one searches for all differences with the help of the versioning files(including the hash code) and the second one checks for the actual lenghts of the files.

The updater code described above is started on another thread, and even if something occurs with the internet connection, only this thread will be stopped, and NaroCAD will continue its work.

The CreateNewVersionFiles project:

This project contains a single class, which is responsible for creating the versiion file and compressing all files needed for the instalation. The main method can be called with one or no parameter. The default is with no parameter. In this case, nightly version files will be created. If the parameter is called “stable”, stable versioning data will be created. Else, if the parameter is called “nightly”, versioning data for a nightly build will be created. The main methods are: CreateVersiionData and CreateCompressedFiles. The first one creates the versioning file by Initializing and serializing the VersionObject class with the corect VersionName parameter. The methods used in this class are described above and are not definded in this project. The CreatecompressedFiles method must only iterate trough the list of files creted by the VersionObject and compress them at the correct location.