**Egret Versioning Guide**

Versioning allows the Egret application to be upgraded with incremental changes as stored in the Egret GitHub repository. This allows users to pull new features into their current Egret installation, and also allows for those changes to be reverted.

The key tools used to manage the Version of your application include the following:

* Git
  + allows for source control in the first place
* GitHub.com
  + home to the Egret source control repository
* Visual Studio Team Explorer window
  + allows for easy interactions with Git
* Visual Studio Package Manager Console
  + controls which database migration is installed
* Visual Studio Publish command
  + pushes file changes to the directory that the server sends to client workstations

Keep in mind that there are really two things being updated:

* The Database
* The Application

The Database structure and content can be managed with Migrations, but the database itself is stored locally on the server, apart from the files that comprise the Egret application.

The Application is manged by files that are stored in Git.

The Migrations are also managed by Git, but a special command must be run to push these changes from their files to the database.

**Version Structure**

The versioning for Egret will loosely follow Semantic Versioning and will utilize three numbers separated by periods. The first number will represent the Major Version, the second number will represent the Minor Version, and the third number will represent the Patch Version.

Major.Minor.Patch

v1.2.0

If the Version is “v1.2.0”, this means the Major Version is 1, the Minor Version is 2, and the Patch Version is 0.

Major Versions will be determined by the overall architecture, and will likely involve breaking changes that require the installation of additional software. It may also be determined by overhauls of the user interface. When a Major Version of Egret is released, the Minor Version and Patch Version will reset to 0. The Major Version will very rarely, if ever, be updated.

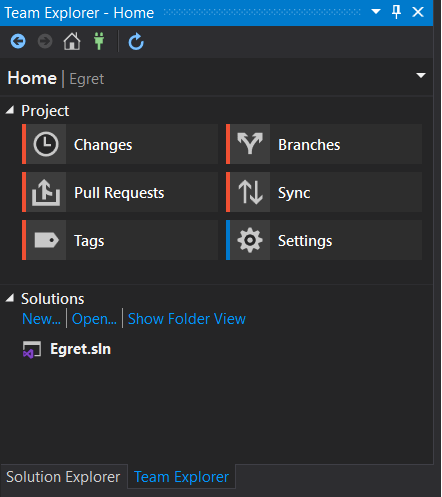
Minor Versions will be determined by database changes, as opposed to file changes alone. When a new feature in Egret requires database changes, the Minor Version will be increased and the Patch Version will reset to 0. An example would be adding or removing a column from a table. These may introduce breaking changes.

Patch Versions will be determined by updates that do not require database changes. A few examples include changing the display text on a form field, changing a color, or making additions to unit tests. If you are requesting new functionality, you can always ask whether the changes will affect the database.

The intention is that version numbers will be incremented by 1 for each update. In rare cases there may be multiple. There is no intended limit to these numbers. “v1.12.298” is a viable version.

**Change Management**

Change Management will be performed using the Visual Studio Team Explorer window. If you do not see this in Visual Studio, go to View > Team Explorer. There is nothing here that cannot be done by using Git from a command line, but using Git from a command line can be very complicated and is prone to errors. The Team Explorer window dramatically simplifies this process. (I use Visual Studio in Dark Mode, this screen may appear white for you)

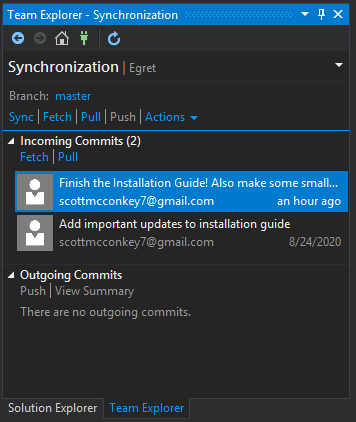


There are six key menus in the Team Explorer:

* Changes
  + Will not be used. This menu is for checking code into the GitHub repository. This will not be necessary unless you are planning to program changes for Egret or have forked the Egret project and wish to makes changes to your own repository.
* Branches
  + Will not be used. All changes and Tags will be on the master branch. There is no plan for Egret to update multiple branches at the same time.
* Pull Requests
  + Will not be used. Pull Requests are managed at the repository level. Egret is currently being maintained by one developer, and the only pull requests I ever see are from annoying Javascript bots that think they’re being really helpful telling me Node packages I don’t even use have newer versions. Piss off, bots.
* Sync
  + Will be used heavily. The “Incoming Commits” will be used to pull file updates from the GitHub repository. “Fetch” will allow you download all updates to Egret without installing those changes. “Pull” will allow you to download all updates and immediately install them at the same time.
* Tags
  + Will be used heavily. Almost all updates will have their own tag, which represents the Version. Tags can be easily switched between by right clicking the appropriate tag and selecting “Checkout”. This makes the files of that Tag “active”, so-to-speak. Other tags will still exist on your computer, but only one can be checked out at any given time.
* Settings
  + Will be used occasionally. The “Remotes” Fetch option will need to be configured to pull from the Egret GitHub repository. See the Install documentation for further details.

**Installing Changes**

The first step to installing changes is to perform a Git Pull. This can be done from the Visual Studio Team Explorer if you have Git Tools installed as an extension. This will update your local Git repository with the latest files from the Egret Github location.

1. In Team Explorer, go to Sync.
2. In Sync, any new commits should appear under Incoming Commits after a few seconds of loading.
3. Selecting “Fetch” will simply download these commits to your repository but will not make them active. Selecting “Pull” will both download these and activate them. **In most cases, you will want to select “Pull”.** Unlike in my screenshot above, these will usually be labeled by Tags.
4. The versions are represented by tags in the Tags section. If you have performed a Pull, the most recent Tag will be “checked out”.
5. If only the Patch Version is being updated, you will still need to perform a Publish using Build > Publish Egret, but no additional changes need to be made and you can skip step 6.
6. If the Minor Version is being updated, you will need to go into Visual Studio and perform a database migration.
   1. Create a backup of the database. Consult the DatabaseBackupAndRecovery\_Guide document for instructions.
   2. With the Egret solution open, note the most recent Version in the Migrations folder. Open the Package Manager Console and type update-database.
   3. It is important to note that update-database is a shorthand for update-database <version>, where <version> represents the Migration you wish to install. Example: update-database v1.1.0. The update-database command defaults to the most recent Migration, but in the event of a Fetch, you can fine-tune which Migration is installed by specifying the version.
   4. Please note that a successful migration immediately updates the database.
   5. **After the database has been updated, you must build the solution and Publish it by going to Build > Publish Egret**.

Always remember to Publish new files. If the files are not published to your IIS folder, those updated files will be missing from the website, and users may encounter errors or unexpected functionality since the database and application may not be aligned.

If the Major Version is being updated, the previous steps may need to be followed. Additional steps will likely exist for installing software.

**Reverting Changes**

To revert changes, you must first know whether you are reverting to a Patch number or a Minor Version number. This will very likely depend on the most recent version you Pulled from the repository.

Reverting to a Patch number is very easy:

1. From the Team Explorer Tags section, right-click the previous tag, ensuring that both the Major Version Minor Version match the most recent, and select “Check Out”.
2. Use Build > Publish Egret to publish these files so IIS can deliver the changes.

Reverting to a Minor Version is more involved:

Reversions involve two steps: moving the database back, and moving the Git Tag back. The best way to describe the process is through an example.

Let’s say you are on Egret v1.1.0 and you do a Git Pull and install v1.2.0, but subsequently discover that you are encountering errors that are a direct result of the new version. You can perform the following:

1. Create a backup of the database.
2. In the Solution Explorer, note the most recent Migration in the Migrations folder, which for our example is v1.2.0
3. From the Package Manager Console, type update-database <version>, where <version> is the Migration directly preceding the most recent Migration, which for the example is v1.1.0
4. Once the database has been updated to the previous version, go to Tags in Team Explorer and select the tag that precedes the Tag you are trying to “undo”. This corresponds to the most recent Patch Version prior to v1.2.0, which is v1.1.0 (but could be, say, v1.1.3). Right click it and select “Check Out”.
5. Use Build > Publish Egret to publish these files so IIS can deliver the changes.

Each Version will be given a Git Tag. By default, a Git Pull will checkout the most recent Tag. Git Tags, when used for reverting changes, allow you to checkout only a specific version of Egret at one time. If you wish to move from v1.1.5 to v1.1.4, go into Team Explorer and go into Tags. Now, Tags does not tell you which Tag is currently active. However, the current version is also displayed on the Egret application itself, so you can always visit the Login page to check (as long as I remembered to update the correct file!).

Note that if you Check Out a particular Tag, the other Tags will still exist within your repository, but as mentioned, you can tell which one is checked out by the version on the Application.

When any Migration is run, there may be potential for data loss. This is why backing up the database on a regular basis is critical. Consult the DatabaseBackupAndRecovery\_Guide document for more information. If the migrations are written correctly, no data will be lost, so while I fully intend to write these correctly, I cannot guarantee they will translate perfectly. Keep data backups on hand!

**Final Notes**

I am doing my best to create a viable, safe, and reliable versioning strategy. However, please remember Egret is published under the MIT License, which is worth reading and re-reading. I cannot guarantee the quality or accuracy of any of this information or functionality. A goal of this project is to limit how many changes are made in each Version such that any bugs can safely be scoped to that particular version and quickly resolved. In a large organization there would be a formal methodology for managing tickets, but since I’m currently the only developer on this project and I have a day job, Egret is unlikely to be so formal.

There is no intention to port Patch Versions to previous Minor or Major versions. Egret uses progressive versioning only. Reversions exist simply to provide the ability to restore from an update gone awry. It should not be performed haphazardly.