**Egret Versioning Guide**

Versioning allows the Egret application to be upgraded as incremental changes are checked in to 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 GitHub, Git, Visual Studio, the Visual Studio package manager console, and the Visual Studio Publish command. Publish configuration will exist in the source code.

**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.

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 new Major Version of Egret is released, the Minor Version and Patch Version will be reset to 0. Major Versions will very rarely, if ever, be updated.

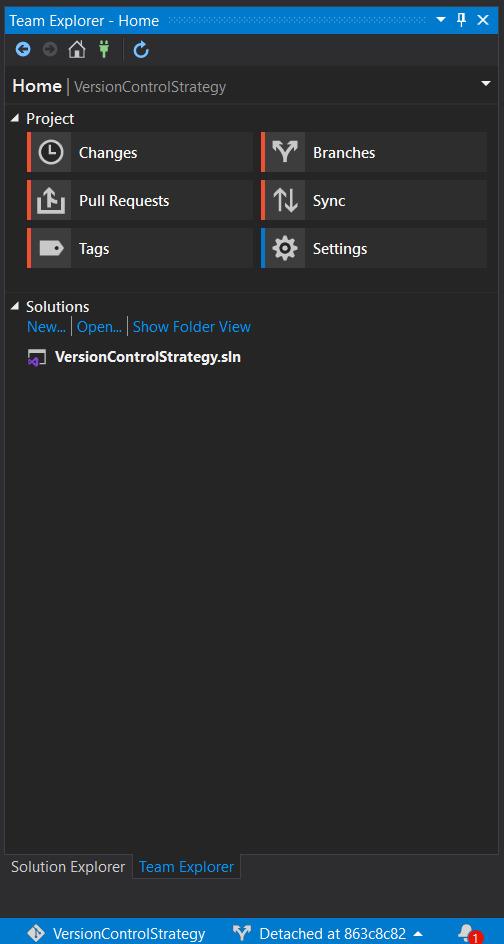
Minor Versions will be determined by database changes, as opposed to file changes alone. When a new feature in Egret requires database changes, the Minor Version will be increased and the Patch Version will be reset to 0. An example would be adding or removing a column from a table. These may introduce 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. There is nothing here that cannot be done by using Git from the command line, but using the Visual Studio Team Explorer will dramatically simplify the process.



The “Changes” menu is only used for checking code into the GitHub repository. It will not be necessary unless you planning to program changes for Egret or have forked the Egret project and wish to makes changes to your own repository.

At this time, “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” do not apply to Egret. Those 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 by telling you a Node package you don’t even use has a never version. 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. This is ideal, as multiple check-ins could have been made since the last time you download the latest files. You may want to test these changes incrementally to be sure the system is working as expected. “Pull” will allow you to download all updates and immediately “install” them at the same time.

“Tags” will also be used heavily. All updates will be recorded as Tags, which represent 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 is checked-out at a time.

“Settings” allows you to store system information. 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 Git command line or 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.

If the Patch Version is the only number in the Version that is being updated, no additional changes need to be made. You may wish to test these changes before you Publish them, and if they are not satisfactory, then a rollback can be made.

If the Minor Version is being updated, you will need to first perform a Git Pull to acquire the latest files. Then you will need to go into Visual Studio and perform a database migration. With the Egret solution open, note the most recent Version in the Migrations folder. Open the Package Manager Console and type update-database <version>, where <version> represents the Migration you wish to install. Example: update-database v1.1.0. Although the changes are incremental and you can therefore “jump” several migrations, it is recommended that you install migrations one-by-one while testing the features in-between. Please note that a successful migration immediately updates the database. If the files are not published to your IIS folder, those updates will be missing, and users may encounter errors or unexpected functionality, so remember to Publish after a reversion, too.

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**

Each Version will be given a Git Tag. By default, a Git Pull will checkout the most recent Tag. Git Tags, when used for reverted 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, you can use the Git Command Line and type: git checkout tags/v1.1.4. Note that the tag and file changes for v1.1.5 will still exist in your repository, but will not be present in your files. If v1.1.5 is faulty, it may need to be deleted from the GitHub repository. This could be involved, but at least you can revert and get back to business. Do not forget to perform a Publish or the files will not be pushed to IIS.

Reverting database changes can be very tricky. The potential to lose data is real. To revert database changes from, say, v1.2.0 to v1.1.0, you will first need to use the command git checkout tags/v1.1.0. You will also need to use the Visual Studio Package Manager Console to perform update-database v1.1.0. The Package Manager Console should warn you if data may be lost. Once this is done or you have handled potential data loss, perform a Publish.

**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.

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