Mailman v1.0.2

Mailman is a pooled event dispatcher that is an easy-to-use yet extremely powerful tool that can be leveraged to completely decouple GameObjects from one another by acting like a middle man between them.

Table of Contents

Overview

Communication between different objects in games is a fundamental aspect of many systems. However, at larger scales, coupling objects together in rigid structures may cause issues when expanding systems, editing or refactoring. Messaging between objects ensures that objects can stay decoupled from one another whilst still providing the same functionality in a more dynamic fashion.

Mailman provides this service by allowing objects to subscribe callbacks to events based on specific message types (referred to as **mail** henceforth). Once subscribed, whenever another object sends that type of mail through the Mailman system, the subscribed function is executed and all the data stored in the sent mail is passed through to that function. This allows for data or events to be sent around the code base without the need to rigidly couple objects together.

These mail types are hard typed which allows Mailman to optimize the performance of sending a receiving mail as well as pooling them for later use, reducing the need for garbage generation. Additionally, thanks to the in-built mail editor, creating, editing, and deleting mail is extremely straight forwards and requires no coding to set them up.

Features

- Static dispatcher accessible from any part of the code base.
- Creation and sending of custom mail types with custom data through GUI and code generation.
- Pooled mail for almost completely reduced garbage generation.

- One sender to Many listeners connections.
- Many senders to Many listeners connections.
- Custom editor for creating and editing hard typed mail objects.
- Extremely high performance dispatch system.

Architecture

Core Entities

Mailman

- Mailman is a static class that stores listeners by their subscribed mail type and dispatches incoming mail to them when it is received. It is also used to fetch fresh mail from pools by the sender to reduce any garbage generation overhead.

Mail

- Mail is the custom data class that gets populated by custom data and sent through to Mailman and further dispatched to the appropriate subscribers. Once the mail as finished dispatching, it is cleared and returned to its appropriate mail pool for reuse later. The custom mail class can range from a data heavy structure (something like an event notifying that an object has been hit with information stored about the hitter), to something more simple like a event without any data (for example, an event that asks all listeners to pause their sound playbacks).

Listener

- A listener is any class including GameObjects that is able to subscribe a listener to Mailman. This is done through the following method:

```
AddListener<MailType>(Callback);
```

- Where "Mailtype" is a class reference to a custom mail and "Callback" is a function with the following signature:

```
private void Callback(MailType m) { }
```

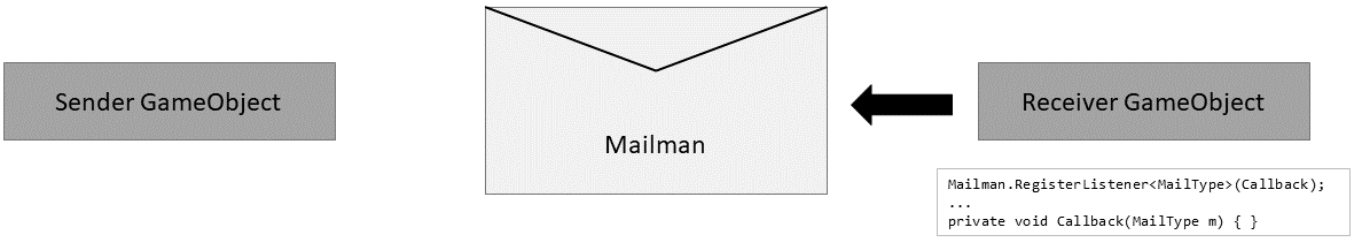
Sender

- A sender is any class including GameObjects that are able to fetch fresh pooled mail from Mailman, populate it with data, and request Mailman to dispatch it to its listeners. This is done through the following methods:

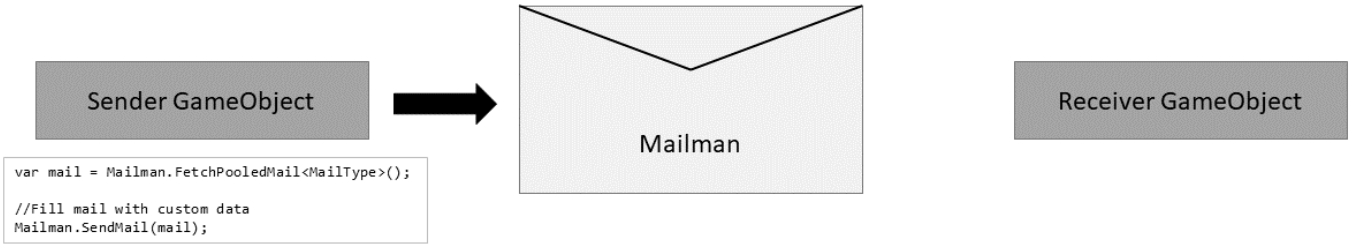
```
var mail = Mailman.FetchPooledMail<MailType>();
//Fill  mail  with  custom  data
Mailman.SendMail(mail);
```

- Where "MailType" is a class reference to a custom mail.

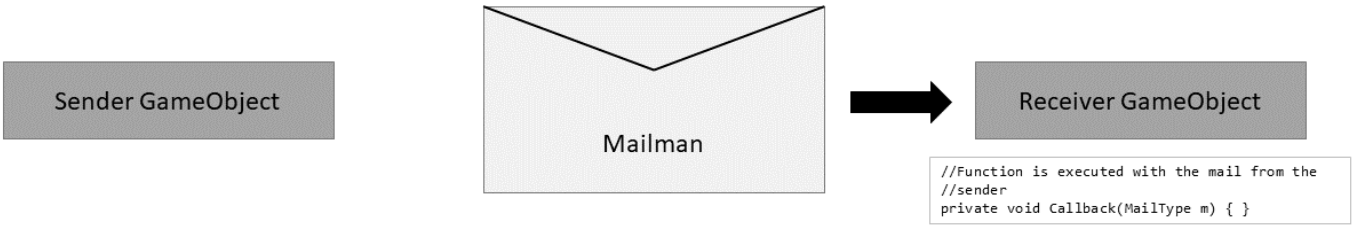
Work Flow



1. An object registers a callback function to Mailman for a certain type of mail.



2. A sender fetches pooled fresh mail from Mailman, populates it with data and then requests Mailman to dispatch it to all listeners of that mail type.



3. Mailman then runs through all listeners for that specific mail type and executes their subscribed function, passing in the mail that was populated by the sender.

Getting Started

Here is a quick rundown on how to use Mailman.

1. Create a custom mail class through the Mail Editor (You can find it under "Tools -> Big Bench Games -> Mail Editor")

2. Have listener classes subscribe to your newly created mail class through mailman with the following: `AddListener<MailType>(Callback);` Where "**MailType**" is the class type of the custom mail class you just created and "**Callback**" is a function in your listener class with the following function signature: `private void Callback(MailType m)`

3. Have a sender class fetch a pooled instance of your newly created mail class. Populate it with what ever data you specified and request Mailman to dispatch it for you.

```
var mail = Mailman.FetchPooledMail<MailType>();
//Fill  mail  with  custom  data
```

```
Mailman.SendMail(mail);
```

4. Fill out the listener's callback function to handle the incoming mail.

```
private void Callback(MailType m)
{
    //Logic
}
```

5. Add the following function to your listener's deconstructor or OnDestory method.

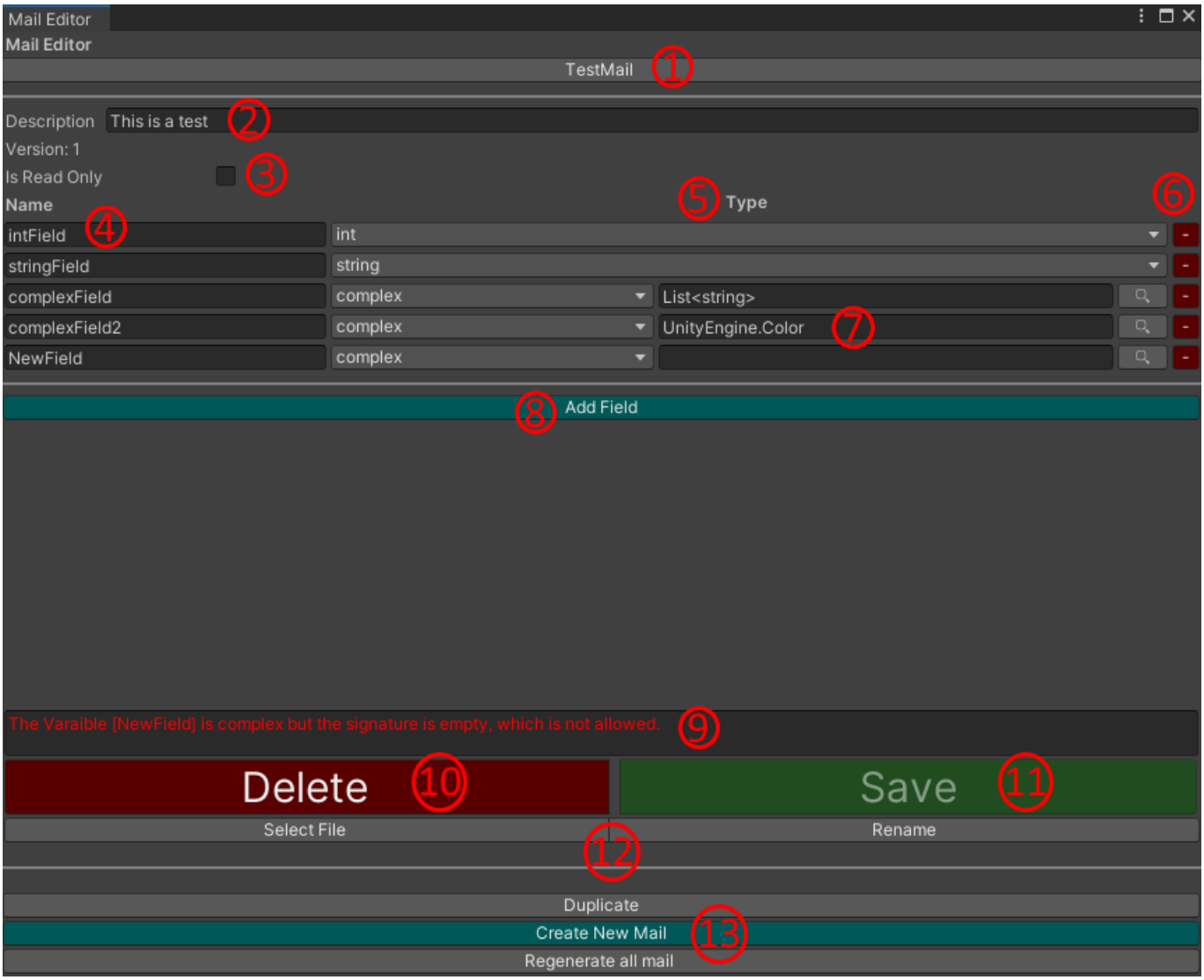
```
Mailman.RemoveListener<MailType>(Callback)
```

   This will ensure that before the class instance has been destroyed, Mailman removes the callback reference of the listener.

Usage

Mail Editor

The creation and editing of custom mail classes can be quite tedious when having to make many different types of mail classes with different internal data. As such, Mailman comes with its own custom mail editor. This editor window can be access through **Tools -> Big Bench Games -> Mail Editor**.



1. Mail selection dropdown.
   Displays the current selected mail class. Click this button to open a dropdown of all the mail classes present in the project.

2. Description.
   This is a description of the mail class. The string here will be inserted into the comment header of the mail class during code generation.

3. Is Read Only checkbox.
   To prevent the mail editor from overriding these custom written classes, please check the "Is Read Only" tick box at the top of the mail editor or if writing your own classes manually, add the class attribute `[ReadOnly(true)]` to the top of the class. This will prevent any unintended overriding from occurring to these classes during editor use.

4. Variable names.
   The user can then add and remove variables they want in the mail class and after pressing save, the automatically generates custom mail class gets regenerated and populated with the newly added variables. Note: The editor automatically detects critical issues in variable names such as using reserved c# keywords or starting a variable with numbers.

5. Variable type.
   The Mail Editor supports all built-in c# data types (bool, byte, sbyte, char, decimal, double, float, int,

uint, long, ulong, short, ushort) as well as strings natively.

6. Remove variable button.
   Click this button to remove the variable from the mail class.

7. Complex variable signature field.
   If you would like to add in a **custom data type**, add a variable and select "Complex" from the type dropdown. A input field will appear next to the type dropdown. Here you can type out the type signature you would like to be generate in the custom mail class. For example, if you would like to generate a list of strings in custom mail class, you would type out `List<string>` in the custom signature area. Please note that no clean up code will be generated for complex types. Additionally, the entire assembly path must be provided here, for example `UnityEngine.Color`, if you cannot remember the assembly path, click on the **looking glass button** next to the text box to search for the variable you need. (This is recommended to make sure that the class generated does not contain any errors).

8. Add variable button.
   Click the "Add variable" button to add a new variable to the class.

9. Validation Error display.
   If there are any issues with the variable names or complex field type names, they will be displayed in this text box.

10. Delete button
    Click this button to delete the mail class. Note: This permanently deletes the mail class, so be careful.

11. Save Button
    Click this button to save the mail class. Note: This will regenerate the mail class using the MailTemplate.cs class and override everything in the class.

12. Select file and Rename class button
    Click the "Select File" button to select the current mail class file.

13. Duplicate mail, Create new mail and Regenerate all mail button.
    The duplicate button will duplicate the currently selected and displayed mail, a popup will appear that allows users to name the duplicated mail and clone the mail multiple times. Click the Create New Mail button to create a new mail class. When creating a new mail class, after passing in a valid name and location, the editor uses a code template (Assets/BigBenchGames/Mailman/Mail/MailTemplate.cs) to generate a custom mail class at the specified location.
    Click the Regenerate all mail button to regenerate all mail classes that do not have the "Is Read Only" checkbox set.

Users are free to implement their own custom mail classes directly in editor so long as they implement all necessary helper functions, which can be found in the aforementioned MailTemplate.cs class. Thus it is recommended that users first generate a blank mail class using the mail editor and then editing the code from there.

Mailman.cs

The core and fundamental feature of Mailman is a dictionary that stores listener delegates keyed by the type of the mail class these delegates listen for. This way, accessing these stored delegates is extremely fast and requires no boxing or unboxing, which would be the case in hash tables or lists.

Here are the core functions of Mailman:

```
public static void AddListener<T>(CallbackHandler<T> l, int priority = 0, int
defaultPoolSize = INITIAL_POOL_SIZE, bool createPool = true)
```

- This function registers a listener callback "l" to the mail dispatch "T". If Mailman does not have any data for this mail type, it will take "defaultPoolSize" and "createPool" and if "createPool" is true, it will generate a pool of size "defaultPoolSize". Note: "defaultPoolSize" will default to 5 if not specified. Additionally, a priority can be specified for the callback. The higher the priority, the earlier it will be executed.

```
public static void RemoveListener<T>(CallbackHandler<T> l)
```

- This function removes a callback "l" from the mail dispatch data structure. This function should be called either in code logic or in the deconstructure/OnDestory method to prevent undesired behavior and potential memory issues.

```
public static void SendMail<T>(T letter, bool createPoolIfMissing = true, int
defaultPoolSize = INITIAL_POOL_SIZE)
```

- This function sends a letter of mail type "T" to all registered listeners in the dispatch. If a mail pool is missing, the function by default will make a pool of size "defaultPoolSize". This can be prevented by passing false to "createPoolIfMissing".

```
public static T FetchPooledMail<T>(int defaultPoolSize = INITIAL_POOL_SIZE, bool
createPoolIfMissing = true)
```

- In order to prevent unwanted garbage collection, this function should be called to fetch a fresh mail object of type "T". If the pool is empty, the function will create a new instance of the mail type. If the pool is missing, the function creates a new pool of size "defaultPoolSize". This can be prevented by passing false to "createPoolIfMissing", however is not recommend for performance reasons.

Here are the helper functions of Mailman:

```
public static int GetPoolSizeForPooledType<T>()
```

- This helper function returns the available pool size of the mail type "T". If the pool does not exist, the function will return -1.

```
public static int GetSubscriberCountForPooledType<T>()
```

- This helper function returns the number of subscribers for the mail type "T". If Mailman has not created data for this mail type, the function will return -1.

Planned Features

- Editor window displaying current mail subscribers.
- Delayed mail sending.
- Editor component that lets you trigger a mail send with serialized fields
- Add support for Unity's new visual scripting system

Changelog

v1.0.2

- Features:
    - Added the ability to use Abstract classes as complex types in mail editor GUI.
- QOL Improvements:
    - Mail file path is now stored in editor prefs and will be cached to prevent having to reinput.
    - Complex type auto-complete now has a cached "Display Amount" field allowing users to increase or decrease the amount of results shown
        - Note: The larger the display amount the greater impact this will have on the GUI performance. (5 as default should be fine)
- Bugs:
    - Fixed bug where renaming a mail class through the GUI would place the renamed mail back into the default folder instead of previous location.

v1.0.1

- QOL Improvements:
    - Added the ability to make multiple duplications of mail types by selecting the mail type in the editor and clicking duplicate and filling out the popup.
- Bugs:
    - Fixed a niche bug where if the dispatch of mail A resulted in the calling of a dispatch of mail B which in turn added a listener to mail type A, an infinite loop would occur resulting in a stack overflow crash.

v1.0.0

- Initial release

Support

If you have any problems, questions or suggestions, feel free to send us an email at bigbenchgames@gmail.com! We'll try and help out the best we can.

Check out our website: https://www.bigbench.games/

Thanks!
Alex