Goliath Framework Functional Specification

Document Status: Draft
Document Owner: Ken McHugh

Change Log

Date	Change Author	Description
29/12/08	Ken McHugh	Started Document

Table of Contents

What is this document anyway?

This document is meant to be a hybrid of both technical spec and functional spec. The idea being that as a feature is planned, it gets included in the document. The functional bits get added first where the eye in the sky view is taken and the feature is defined in terms general to the user. As in “Click the backup button” rather than “Call the onclick eventhandler on cmdBackupTheWorld and iterate through all of the blah blah blah blah”. When a developer is assigned to the feature, they will read the document, or at the very least, the section pertaining to what they are to do. Yes please, make sure you know what you are doing before you do it. After reading, they will add a technical section to the feature which will allow them to add in all of the technical notes and jargon that can easily be skipped over by those who can't be bothered reading it and only want to figure out what the program is meant to do, not what the developer decided it should do in the end. Testers, that is you. Once the developer has all of their code checked in, they will check in the document, and everything will work perfectly, so the testers will scramble to report inane bugs such as “The pixel on the bottom of the 'T' looks funny”.

This document is not meant to impress all of the high level VC's that walk around with a broomstick tied to their back and wouldn't know a joke if it bit them on the nose. This document is meant for the people who are actually developing Goliath, or are developing applications built on Goliath. The idea of introducing humor is so that you will actually read the document and maybe even remember some of it. And the ultimate goal is that you actually go back to it once in a while to make sure you are doing what you are supposed to be doing instead of using it to insulate the walls to your new home.

By the way, if you are a VC reading this document, there is a good chance you have been given the wrong one, stop now and make sure.

And if you are a developer and don't know what a VC is, then you are pure and holy and all that is right with the world.

Introduction

Goliath: The big idea

The Goliath framework has been designed to be a development tool box. It is not meant to be a final product, but rather a building block to build applications on. It is also not meant to be the be all end all, using Goliath is supposed to make writing code easier, not meant to write code for you. All of the stupid things that you have to do everyday like opening files, reading database records, will hopefully be replaced by other stupid things that you have to do everyday but are a bit easier and are not quite so aggravating to do. Writing in the yaw controls to the flight simulator you have hidden in the product because you have saved a week not having to write a logger for example. Or writing technical specs and trying to introduce a bit of humor into developing.

How can it be used

Why should I use it

Bug tracking

Relase process

Automatic updates

Application Startup

Startups

Permissions

Modules

Introduction

Application Process Loop

Commands and Command Queues

Permissions on Commands

Sessions

Application Logging

Application Settings

Security

Introduction

The security module in Goliath is meant to provide a to control user and agent access to resources and application functionality. It is a requirement that any process that must interact with the Goliath framework or it's resources must log in in order to do so. This means that if you are the Administrator, a new user, or the latest spyder, you must have proper access to the system in order to do anything.

While trying to provide a top notch resource level security system, we recognise that we are not perfect, unfortunately, and we know that most organisations already have something that they use. So we are also trying to make it flexible enough to allow it to be customised. So when you have just finished developing your super secure single entry password, fingerprinting, iris scanning security database, you can write a bit of code to let Goliath access it instead of its default hashed password security database.

Resources

In Goliath, a resource is considered anything from a record in a database to a file on the system. However with this in mind, the security system can only work on processes in the Goliath environment, this means the users and agents that are logged in to the Goliath system. So yes that does mean the super virus that adds "ha ha I got you" to the end of all your code files and scrambles the rest of the code with "Stuff I promise I really didn't write" will still be able to do that if you run it on your system.

This means that access to everything in the system can be controlled. You can lock out your Release Candidates so that Anonymous users can only access the beta until they pay you what you deserve for writing software that all those Anonymous users tested for you anyway.

Another nice feature of resources is that if you don't have access to it, but you know about it, you can request access. Upon requesting access the owner will receive a notification from which they can say that you have permanent access, or only have access for the next 10 minutes. They can state that you can have read only access or allow you to change the resource. They can also tell you to bugger off, you are not getting access to my pay rates.

Ownership

Every resource that is added to the system must have one owner, even if that owner is the system user. That owner can do anything they want with that resource, even if they have been specifically denied permission. This means that even the system administrator can not stop a user from deleting their own resources. This is not quite true, if you have a malicious user, a System Administrator can take ownership of all of their resources, and deny access to all of those resources to the malicious user. Ownership of a resource can be transferred by the owner to another user. Ownership must lie with a user, a resource can not be owned by a group. We want someone to take responsibility for once. All transfers of ownership are logged. When a user decided to transfer ownership of a resource, a notice if first passed to the new owner where they must accept ownership. It is not an automatic transfer. Again this is for responsibility, if you get a notice saying someone is transferring a resource that was originally yours back to you, someone probably just viewed the resource by taking ownership and trying to give it back. As a note, only system admins have this ability.

Users

Ahh, users, the bane of our existence. If it wasn't for them we wouldn't have to write software in the first place. We could all go and stock shelves and work a mere 8 hours a day and take holidays over the Christmas time.

The words User and Agent are used interchangeably in Goliath, a user is anything that needs to access the system and a user is always logged in and recorded in the system, even anonymous users. See the Anonymous User and Anonymous group sections later for more information.

A basic user must have a name to be displayed, a password, described later, and an email address so that the system administrator does not have to get involved when a user forgets their password. And also doesn't have to let anyone know when they have forgotten theirs. Point to not here, don't make the administrator email address an address that anyone can get in to. Make it a person who is responsible for that account. That is the point. Security only works when there is known responsibility and audit trails. If anyone can change the password of the Sys admin, then it's never going to be a secure system. If you want to have multiple system administrators then add the users and give them the proper security permissions. Make them responsible for what they do.

A user may be locked, which means they have been locked out of the system, either because they forgot their password to many times, or because an administrator user has decided they should not be allowed in the system. Locking a user requires a reason, so when you lock out the user who just left you company, and they call up to say they forgot their password, you can check why they were locked out.

A user may also expire, once a user expires, they can not log in until an administrator extends their account.

The Users are also customisable, so you can write your user that stores their entire life history day by day if you want to. You can also customise where users and their information is stored, so you can write your user to access your internal company one access password system if you want. Keep in mind that if you do this, you need to also change how the users change their passwords.

Passwords

Abracadabra. Every user and agent requires a password, see anonymous user for exception, this password is defined by a regular expression which by default means the password must contain mixed case, must contain a number, and must be between 6 and 20 characters. The password can be based on a dictionary word as within the system the password is mauled and adjusted anyway. This means that the password that is actually stored, is not the same as what the user typed in. Meaning that as an administrator of the database, you couldn't go and look up the passwords and just log in as another user. Nobody knows your password, more responsibility added to the user, basically, if something bad happens and your account did it, then it is your fault, because either you gave someone your password, or you wrote it down and stuck it to the bottom of your keyboard. If password retrieval is simple then users won't mind forgetting their password and having to go through the process of retrieving it using their email address. Okay, so it is possible that the password was hacked, but you get the point, attach responsibility to accounts to try to stop users from sharing their password because the user in the next cubicle could not access the file that the original user could see no problem at all. There is a reason, and whether that reason is because the system administrator was hit by a bus on the way in and never got around to it, or because the user wasn't supposed to have access to their notice for termination until it was reviewed by their supervisor to ensure the clause about bypassing security was in it.

System User

The system user, this user is not really a user at all. The Goliath Framework runs under the context of the System user. Everything that it does, it does through the system user. It is not possible to log in as the system user, it is not even possible to extend the system user for your own applications. This is a special user and we are not sharing. Or trying not to anyway. The system user has full access to everything in the system. It can pretty much do whatever it wants. Any other type of user can ask the system user to do something for them, but they must have permission to ask, and they have to have permission for what they are asking the system to do anyway. If you want to do something that you don't have permission to do, go and ask your System Administrator to allow you to do it.

Anonymous User

Another special type of user, this time we will share. By default when ever a user or process connects to the Goliath Framework, a session is created and an anonymous user is created. This user is uniquely identified by assigning them a random identifier which allows the system to track individual anonymous users, rather than having 800 anonymous users in the system and trying to figure out how the user managed to jump from the download application window to the report bug window in a single click. We feel it is better this way, track each user as an individual, just give them anonymous access. So even though the user did not type a user name and password, they were still logged in to the system. This user is automatically added to the Anonymous group, see that later. When a user then decides to log in, they provide a correct user name and password combination, and the anonymous user is logged out and the user is logged in correctly.

Sysadmin User

The sysadmin user is a user that is always created for you if they don't exist. This is a regular user who is a member of the Admin group. It allows you to access the system when you have first installed it. By default the password is Sysadmin1, it is recommended that the first thing you do after an install is log in and change the password.

Groups

Groups allow you to assign permissions to them and then add users as members. It is preferential to do things this way. While it is possible to allow user Jane to access pail of water directly by assigning permissions directly to Jane, it is better to create a group NurseryRhyme, allow NurseryRhyme access to pail of water, then add Jane as a member of NurseryRhyme. That way, when health and safety come in and say that pail of water is too heavy for one, you can simply add Jack as a member of NurseryRhyme. Finally when Jack can't quite make it up the hill, you can delete him as a member of NurseryRhyme and add a real worker.

There is another way of doing this, where you create “resource groups”, these groups are no different from regular groups except that they generally do not have users as members, only other groups. If you create a group called hill, and allow hill access to resources pail of water and ambulance. Then create a group called NurseryRhymeWorkers and add NurseryRhymeWorkers as a member of hill. This will give NurseryRhymeWorkers access to the resources in hill. Now you just have to add users to NurseryRhymeWorkers to give them access to pail of water and when the inevitable happens, ambulance as well. The benefit here is that after Jack falls down the hill and you decide you want to have TempWorkers do Jacks job in addition to their own, you just have to add the TempWorkers group as a member of hill.

However you decide to do it, if you don't have a really good reason, and you are assigning permissions to users directly, there is a chance you are making a lot more work for yourself. Just imaging what will happen when you have assigned 1000 permissions directly to Jack, who has just decided that he has had enough falling down the hill and is getting himself a job as a stunt driver because it is safer. Now you have to create a new user and manually copy all of those permissions, had you listened you could have just added the new user as a member of the same group that Jack was in.

Groups can be members of groups, they can contain other groups as members, they can contain users as members, in fact they can even contain circular references of groups, but the Framework will just laugh at you and completely ignore the fact that you just ask it to chase it's tail trying to figure out the dependency tree for the groups a user is a member of, and it will do what it assumes you meant to tell it to do.

Admin Group

More specialised stuff. Admin groups, this group is created for you, and each time the application starts the admin group is given all of the available permissions. This is not resource permissions, just regular permissions, for the difference see the section later in the document. This means that any member of the administrator group is able to perform any of the actions defined in the system. You can still go and deny actions to the Admin group, but that doesn't really make much in the way of sense. If you have administrators, trust them, they know what they are doing. If they didn't, they wouldn't be administrators. Everyone in the Administrator group is equal, there is no concept of super administrator or super user. The idea is that the Administrator group should really only contain 2 – 3 members, and these should be people that are trusted to run the system. If you have sensitive information that you want to hide from someone, they should not be an administrator. You should have multiple admins for redundancy sake.

Administrators are allowed to take ownership of resources, they are the only users that are allowed to do so. This is to ensure that when a user leaves the system, it is still possible to control all the resources that the user owned. When taking ownership of a resource, the action is logged. This is why it was stated earlier that if you don't trust your admins, don't make them admins. It is possible for an administrator to take ownership of a resource in order to view it, then transfer ownership back to the original user, that user will get a notification and there the transfers will be logged in the system, but depending on the sensitivity of the information, the damage may already be done.

Everyone Group

The everyone group is another group that is automatically created for you. Every user in the system is a member of the everyone group. This is for assigning access to things everyone should have access too. Everyone should be able to log in, log out, and exit the application for example.

You may want everyone to have access to all of the look up records in the system as well. You have to be careful here as even anonymous users are members of the Everyone group. If you want to give all logged in users access to the New Free Products area, but not to anonymous users, you have to add Permission to the Everyone Group allowing access, then add Permission to the Anonymous Group denying access. It is simple, just worth mentioning. The everyone group is just like any group. You can remove members from the everyone group, but it is not really recommended.

Anonymous Group

The anonymous group is the group that all the anonymous users are automatically a member of when they are created. You should use this group to give anonymous members specifically deny access to resources and actions. See the Permissions section for a description of how permissions work.

Permissions

Permissions are the mechanism for controlling access to resources and actions. In order to grant access, you must add the permission and say that access for read, write, or execute, is allowed. You can also specifically say that access is denied.

Deny overrides allow, so if a user is a member of a group that allows access to a resource, but they are also a member of a group that denies access, that user is denied access.

If the permission is not applied, then the user is also denied access.

So for example, exaggerated of course, if there is a resource in the system, say the printer. You have created a resource group called 3dFoamPrinter for the printer in the new high tech 3d printing lab. Then you create the group 3dPrinters which you add as a member of 3dFoamPrinter. You then add Jane as a member of 3dPrinters because she has also become tired of falling down hills and you have just managed to convince her to stay around. Unfortunately you forgot to actually give 3dFoamPrinter access to the Printer resource. So Jane trips over the water cooler in her mad rush to get to the printer to pick up her nice 3d foam pail of water that wasn't actually printed because no permission was assigned.

So you grant permission to the 3dFoamPrinter and tell Jane all is well, she will be able to print now. Two weeks later you realise that costs are skyrocketing and you haven't gotten a singe pail of water because every time the printer gets around to printing the water, it leaks through the foam, and Jill keeps forgetting to use the 3d printer that you specifically bought for her to print her pails out of iron. You simply deny permission to the Foam printer to Jill, so she can use other printers that the 3dPrinters are allowed to user, she just can't use the foam printer anymore.

Permissions themselves use a simple system for defining the access type. It uses the values 0, 1, 2, and 4. 0 means no access, or access denied. 1 is read access, 2 is write access, and 4 is execute access. If you want to give both read and write access, you simple add them together. 3 is read/write, 7 would be read, write, execute.

Permissions can also expire, so you can state that a user or group will only get a permission for a certain amount of time.

Regular Permissions

Regular permissions are the permissions that allow an activity or action on the system.

Regular permissions are the permissions that allow you to see and execute the Format C drive option on the menu. Generally, if you do not have permission for an action, you will not see the action.

Resource Permissions

Resource Permissions on the other hand are the permissions that allow you to access resources in the system. It is possible to have access to every action in the system but not to be able to see a single resource, system administrators are like this to start out.

Security Managers

The security manager is where the customisation takes place. If you want to use your current security system, whether it be LDAP, WindowsNT accounts, or a simple text file. You have to write a security manager for your specific type of access. If you want the security manager to be able to create users, you have to write that. If you want the security manager to allow users to change their own passwords, you have to write that. Everything that you need to do to customise you have to do yourself. The default security manager allows all of the actions and reads and writes user and permission information from the datasource created by the application.

User Interface

Data Sources

What is a data source?

Data Object Layer

Business Object Layer

Data Source Security

Web Server

Automated Testing

Automated Crash Reporting

Bug Tracking

Automated Updates

Release Process

Automated Build Process

Source Control

Check-in Process

Sample Code