Stored Tagging File Utility (Tag)
Cycle 1 Report

Andrew Kant
Jonathan “JD” Davis
Corey Sullivan
David Rehfeldt

Dr. Henry Welch
01/13/2006

Table of Contents

Table of Figures

Proposal

The purpose of our project is to provide a computer user an easy way to add meta-information to individual files on his or her computer. In today's world, computer users can become inundated with files on their hard drive and lose track of things over time. There are some solutions to this problem, but most do not scale to the full set of files on a computer and are limited to a subset of files. For example:

- Most compressed audio formats include support for a tagging scheme called ID, which audio players can use to sort and keep track of files.
- Image files have their own standard called EXIF for keeping meta data on photographs.
- In addition, applications such as iPhoto and Picasa allow users to store information about photos.

The goal of our project is to bring one solution for saving meta information to all file types, regardless of content. We plan on doing so using the schema of "tagging" or attaching user-defined keywords to files. Our application will provide an easy method to adding or removing tags from files, and searching by tags for files. It will be operating system independent, and feature an open architecture such that plug-ins for other applications (music players, video players, desktop search tools) can be written to utilize the tags.

As an example case for tagging a file say you download ten audio files of speeches given at a certain conference but do not have time to listen to all of them at once. You could tag all ten of them as "new" or "have not listened to" and then later as you listen to them one at a time you could update them the tags to say "old" or "listened to". This way you can keep track of which ones you have and have not listened to easily. Also, you could tag each of the speeches by the presenter and later find all speeches by the same person across multiple conventions.

Requirements

Tier 1

Backend System

- **Add tags to one file –** The user chooses the file and adds the tags to the file.
- **Add tags to multiple files –** The user selects multiple files and adds tags to the files.
- **Edit the tags of multiple files –** The user chooses multiple files and edits tags common to all of the files.
- **Edit the tags of a file –** The user chooses a file and edits tags associated with that file.
- **File searching –** The user can search for multiple tags using all or any of the tags and support Boolean search logic.
- **File searching –** The user searches for files with inputted tags. The search returns all of the files with the matching tags.
- **Search and replace tags –** Search for a tag in all tagged files and replace with the new given tag. Show the user results and get confirmation before replacing. (Example: replace *SDL portfolio* with *Old SDL portfolio*).
- **Remove tags from multiple files –** The user chooses multiple files and removes tags common to all of the files.
- **Remove tags from one file –** The user chooses a file and removes tags associated with that file.
- **Deleting tags –** The user can delete all tags of multiple files or certain tags of the user's choosing.
- **Unicode support–** The program supports Unicode. The tags must be Unicode safe.
- **Networked drive –** The application should be able to tag files on remote/networked drives so long as they have a local path (not http:// or ftp://)

Tier 2

OS Integration

- **Deleting multiple files –** The user deletes multiple files. The application removes all tags associated with the deleted files.
- **Deleting one file –** The user deletes a file. The application removes the associate tags.
- **Ghost file cleanup –** Go through and remove any tags for files which can no longer be found.
- **Copying multiple files –** The user copies multiple files. The tagging services copies all tags associated with those files.
- **Copying one file –** The user copies a file. The application also copies the file's tags.

- **Software installation –** Upon installation, the installer also installs and sets up the database.
- **Software installation –** The user installs the service (if there is one) and tag utilities.
- **Moving multiple files –** The user moves multiple files. The tagging application remembers all of the files' tags. *(The file index is updated with the new path/filename)*
- **Moving one file –** The user moves a file. The application still knows what tags are assigned to the file.
- **Software uninstall –** The user uninstalls the software and that removes both the software and the stored tags.
- **Searching front-end –** The searching application should be able to do everything the integrated features can do.
- **Help file/dialog –** The user should be able to list all command line options with descriptions.
- **Linux man page –** A man page should be available on Linux/UNIX systems.

Tier 3

Backend

- **Associated tags –** The user should be able to see other tags that files with similar tags use.
- **Mass export –** The user wishes to back up the tags. The service can save the database as an XML file (possibly compressed).
- **Importing tags –** The user imports tags and the service merges the imported tags with the database.
- **Exporting tags –** The user selects files to export. The service stores the tags and outputs the XML.
- **Associated files –** The user should be able to see other files with the same tags as any given file.

OS Integration

- **Tag directory –** The application should be able to build a browse able tree of HTML pages listing files by tag. *(Similar to JavaDoc)*
- **Folder tagging –** While right-clicking a folder, be able to tag all files in the folder.
- **Installation –** OS integration should be optional. *(Either at runtime or in the installer)*

Proposed Architecture

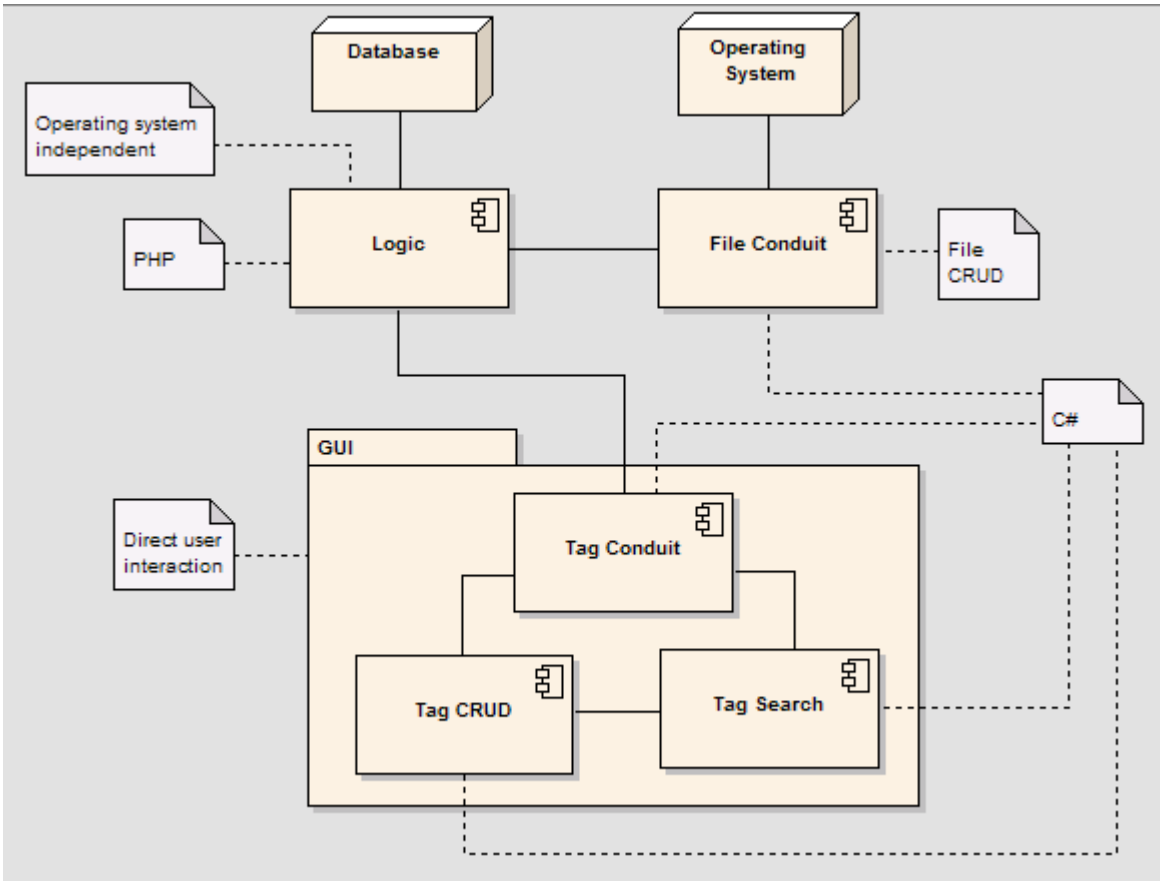

Figure 1 - Overall system architecture

The main qualities of the Tag software are the portability of the back-end logic and the usability of the program. The high portability of the back-end is achieved by abstracting the back-end logic from the front-end functionality. The sole purpose of the back-end logic is to communicate with the tag database. This functionality is independent of the executing operating system and is decoupled in the architecture. The front-end is made up of the graphical user interface that aids in the usability of the system and the conduit modules that contains the coupling to the underlying operating system and file system.

Back-end logic component

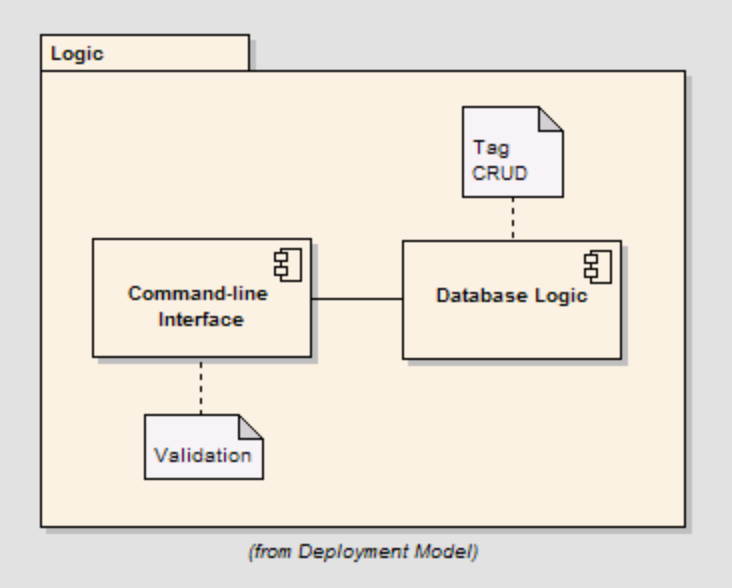

Figure 2 - Back-end logic component

The back-end logic component contains two packages: the command line user interface and the database communication logic. The command line logic is separated from the database logic to further decouple the business logic from any user interface. The command line package allows for a base user interface that is independent of the operating system due to the fact that PHP is used. PHP itself is a portable scripting language that will allow the software to have a command line that is portable between operating systems. The command line package will also contain the interface to the database logic and allow for validation of tag input. The output of the command line interface will be XML.

The database communication logic package makes use of PHP's ability to communicate with an SQL database. This package contains the business logic for tagging creating, review, updating, and deletion (CRUD). The decoupling of this package from the command line interface allows for well-defined functionality and logic isolation. The database logic will only handle the communication to the database.

GUI package

The front-end consists of t[illegible] components: the tag CRUD user interface, the tag search user interface and the tag conduit notifies the back-end logic of CRUD and search events.

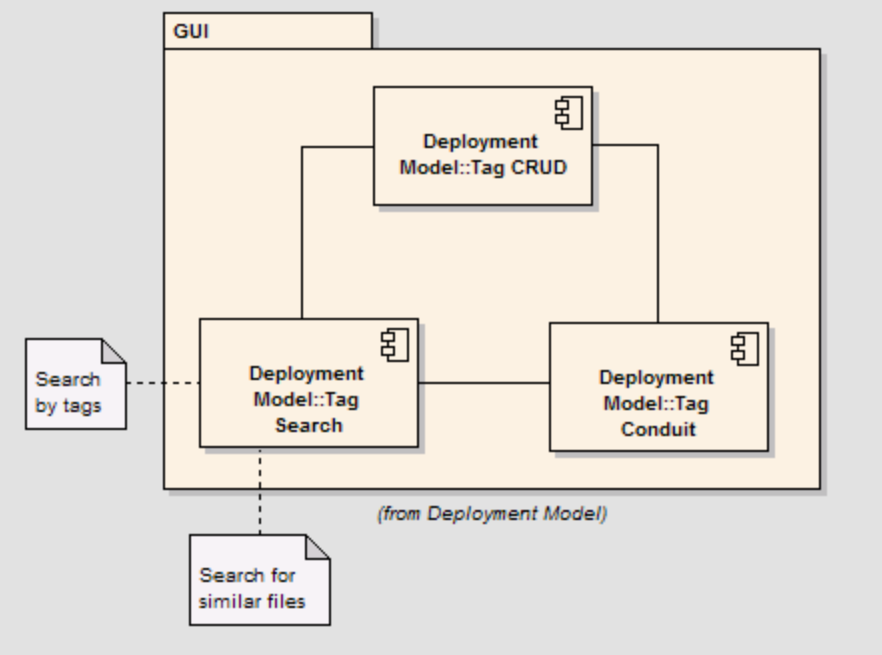

Figure 3 - GUI package

The tag CRUD user interface will make the tag CRUD functionality available within the operating system through the file manager, for example the Microsoft Windows Explorer. The tag conduit is responsible for notifying the back-end logic from actions initiated from the user. The tag conduit is the hook into the operating system's file manager.

The tag search user interface will make the tag searching functionality available to the user. This package will have the user interface that allows the user to search for files by tags or for files similar to each other based on similar tags. The tag search user interface will also make use of the tag conduit to interface with the back-end logic.

File conduit component

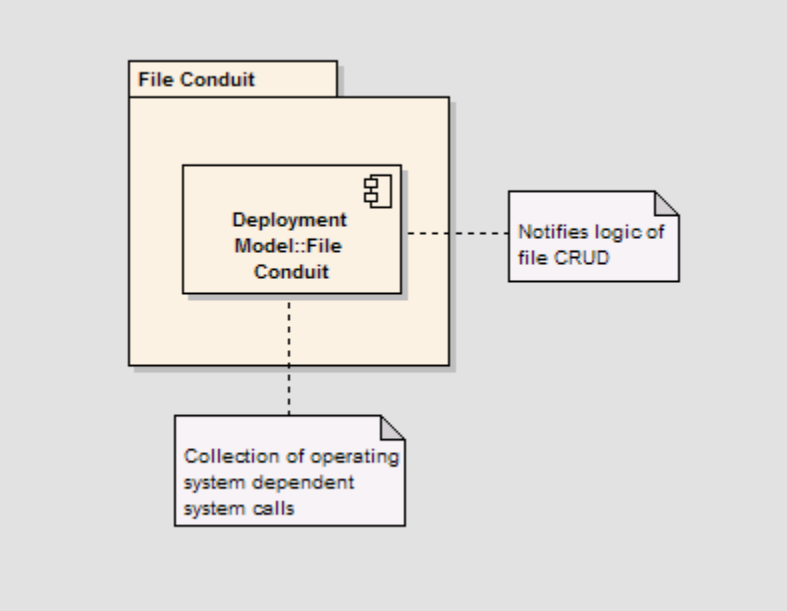

Figure 4 - File conduit package

The file conduit package is another operating system dependent package that will allow for the underlying operating system to notify the Tag software of file system changes. If the user changes the file system by copying, moving, or deleting files, then the operating system notifies the back-end logic through the file conduit to change the database based on the file system changes. This package ensures that the tags are attached to the correct files at all times.

Technologies considered for key components

Scripting languages

We decided to go with a scripting language instead of a compiled language as most scripting languages are cross-platform with no code-rewriting so long as an interpreter is available for the language on the platform of choice. In addition, by using a higher level language we would be have access to helpful features like high-level data structures, interfaces to standard libraries, and an open source license to avoid being locked into a proprietary development format. Here is a summary of the different languages we researched for our project.

PHP

Introduction

PHP was originally the initial suggestion made for this project, but it was given a fair share of research in comparison with Ruby and Python to ensure we were selecting the best technology. While originally created as a CGI alternative to solutions at the time, PHP has matured into a fully-functional scripting language with features as rich as any of its counterparts. With PHP5, object-oriented support was fully developed within the interpreter. In addition, the PHP library of extensions provides support and interfaces to numerous other systems and languages.

Pros

- Two team members (Andy and David) already are prepared to develop with this language.
- Verbose user manual with user's comments
- Object-oriented support
- GUI support through PHP-GTK
- Easy to learn, C-like syntax
- Large support for different systems through extensions (such as FTP, PostgreSQL, Win32 API, etc.)

Cons

- PHP has low security measures for poorly written code
- PHP error messages are unusually confusing
- PHP's extensions are not all thread safe
- PHP does not support Unicode characters natively
- PHP's COM support is low, which is important for Windows integration

Decision

We eventually came to a conclusion that PHP would be the best choice for our project. With the time constraints of being a two-quarter project looming over us, having two team members who can lead development through previous experience played a big

factor in choosing PHP as our language of choice. In addition, PHP5 comes with the SQLite library built-in which after having chosen SQLite was also considered, we feel confident with our choice of PHP as the implementation language for the backend system in our project.

Ruby

Introduction

While becoming popular recently, especially with buzz about the Ruby on Rails framework, Ruby has been around for over 10 years. It was created to be an easy-to-use genuinely object-oriented scripting language unlike Python and with the features of Perl without the complex syntax.

Pros

- Pure object oriented language, but can act procedural
- Highly portable, supports most major OS's
- Great database support
- Tight integration with the OS
- Can be used to write GUI's and server processes
- Can serve web pages
- Consumes modest system resources
- Can make native API calls in Windows and has COM integration

Cons

- While it's easy to learn, the syntax is a bit different than most object oriented languages
- Will need additional libraries for GUI support

Decision

We decided against Ruby, despite being fully-functional and capable of everything we would need for the backend of our system. This decision was made as a group. It was based mostly on our in-experience with the language, which we felt may be a risk to completing the project within the time available to us. Andy suggested that it may be too complex to learn to the point that we could develop a tool to the requirements set for our project.

Python

Introduction

Python is a powerful scripting language with many features that make it appeal to developers needing to throw together a quick script or a full-featured application. It comes with the "batteries included" idea that support for common tasks (socket connections, threaded applications, regular expressions) should be included with the language and easy to use. Python's interpreter is very functional and allows developers to

easily test snippets of code for syntax and logic mistakes. The language is easily extensible through C and C++ modules, and a large number of extensions to the language have been written to standard libraries.

Pros

- Cross-platform support for all major platforms
- Support for very high level dynamic data types
- Very large standard library
- Extended very easily through C/C++ Libraries
- Full native object-oriented support
- Exception handling
- Automatic garbage collection
- Introspection capabilities (Profiler comes bundled with the interpreter)
- Support for operator overloading and multiple inheritance
- Support for both procedural programs (Scripts) and full OO applications

Cons

- Syntax deviates from C and can be confusing at times
- Windows packaged interpreter is large (9.2 MB)
- Decision
- We also decided against Python, due to its complex syntax and deviations from languages which we have experience with.

Storage Solutions

XML based storage

XML is good tool for transporting data between different software systems. Along with portability, XML has good international language support through Unicode. XML is multiplatform and it is system independent. XML documents require parsing through either SAX or DOM to extract the data from the documents and each method has its own implications on performance. The documents themselves have their own physical size requires on their storage medium. The best uses for XML are transporting data from a database between different software systems, facilitating communication between software systems, and storing configuration setting for applications. The rationale for persistent static XML document must be clearly defined, if it is used for that purpose.

The best use of XML for the Tag project is as a medium to transport the tag data from our backend to the desktop application that will search for the tags. The backend's and the search tool's language will require XML support, but it is the most logical application for the language. Also, XML should be used to store personal settings of the application, because it will decrease the dependency on the executing system and is a valid use of XML. XML should not be used as the storage medium for the tags because of the physical size implications, in addition to speed considerations when searching for XML across a hard drive.

SQLite

SQLite is a C library which simulates a full database server by operating on a single file. The library will parse SQL queries up to an almost complete implementation of ANSI SQL. It is intended to be a small, fast alternative to using a full database server. We decided upon using SQLite for our project based on our needs. We felt that using a full server would consume too many resources and would be overblown for what we require of a storage medium. The features missing from SQLite's implementation of SQL should not greatly affect our needs.

MySQL

MySQL is the world's most popular open-source database solution. Supporting features for transactions, database clusters, replication, and an almost complete implementation of SQL, MySQL is the choice database for some of the internet's biggest user-based sites. Detractors say that MySQL is not ready for prime time yet, without support for some advanced features of SQL and an unproven record for stability. The installer download for Windows is quite large at 16.1 MB, with an unpacked size of 94 MB. This was one of the factors in deciding against MySQL. In addition, even scaled the server needed to be run as a system service (or daemon) in the background at all times consuming some resources.

PostgreSQL

PostgreSQL is the most feature-rich and advance open-source database available. PostgreSQL is often seen as a more favorable choice to MySQL as it offers a complete implementation of SQL with proven reliability. The problem we ran into was that the Windows version of PostgreSQL requires an NTFS partition for data storage. As the goal of our project is to create a tool independent of file system or operating system, requiring the user to have an NTFS partition goes against this idea. We did not choose PostgreSQL for this reason.

Key project/technology issues

Costs

Since this is completely a software project, there are no hardware costs. The only costs that would be incurred would be due to software. However, all of the software chosen for this project is either open-source and free or already provided on our MSOE laptops. There are no projected monetary costs for this project.

Lifecycle

This project officially began at the beginning of Cycle 1, which was December 28th, 2005, when research for the project was started. The development will start in the middle of Cycle 1 on December 18th, 2005 when we began our requirements process. The projected completion date for the project is May 14th, 2006. The lifecycle of our product is dependant on the operating system it is ran on. It could have an unlimited lifecycle if the operating systems the project is created for are always used. Realistically, however, the operating systems will eventually become obsolete, rendering the version we deploy of our program unusable. Assuming that current operating systems are no longer in common use within five years, that is also the estimated lifecycle of our project. However, modifying the project with new and upcoming operating systems will increase its lifecycle.

Risk factors

There are a few risks involved with our software project. Since our project is completely software, all of the risks are due to software issues that we cannot foresee at this time. These risks pertain to the operating systems, technologies, and time constraints. The risks involved with the operating systems are that there could be possible software constraints in the OS and that we will be attaching our code to commonly used OS commands. The first risk is that the operating systems we chose to implement our software on may in fact have constraints or security features that will not allow us to implement our intended project. An example would be if there is no way for us "hook" on to move or delete commands for files, meaning we cannot update our database, due to the OS security setup. Our research has shown that we should have the ability to accomplish everything we need, so this is considered a low-probability risk. Our contingency if this does happen is that we do have two operating systems to choose from. If one fails, we can go to the other. The other risk is that our software will be "hooking" on to common operating system command calls, so if our code is not properly tested and verified, we could cause serious system errors. We plan on thoroughly testing our software throughout our entire process as well as making sure our software does not impede on any operating system calls.

The risks involved with our technologies chosen pertain to size constraints and compatibility. If any of the technologies we use are incapable of creating the software and the program with the intended features, it would be a massive set-back for the project.

Also, if our way of storing our tags ends up taking massive amounts of space, it would be considered a major problem as well. We have researched our technologies and are confident that they are capable of implementing all of the features we desire and can store our data to our satisfaction. We consider these risks to also be minimal, but if they do occur, we have researched back-up technologies we can replace our current choices with. The final risk is time constraints. Since the team is using technologies new to some members, and also implementing software that is new to all of us, our estimates on how much time it will take to implement the software could be wrong. It is because of this that we decided to use a modified development process of Extreme Programming to develop our project, because it is a dynamic process that allows for constant changes in the plan, which will be necessary to keep us as productive as possible.

Other

One issue we'd like to bring up is our decision to use Extreme Programming (XP) for developing this project. We decided as a team to use this method of programming for a few reasons. One of the reasons is simply to have experience in a new method, which we thought would be helpful to us in our future careers. We also chose it because extreme programming is a very dynamic process which is useful for projects in which conditions or requirements are changing often. We felt this was important because no one on the team has experience in creating software to give the sort of functionality to an operating system that we plan on doing. Since none of us have that experience, it is highly likely that our plan will need to be changed on a weekly basis. Another appealing thing about XP is the paired development work. Since so[illegible]echnologies that will be used will be ones some team members have not had experience, we felt developing in teams would be greatly beneficial. Another advantage of XP is that it is made to streamline productivity by keeping all of the members busy at all times, and by keeping them focused on each current task. It stresses getting functionality that needs to be done accomplished first before adding extra features. And finally, another reason we chose XP is because although it encourages faster deployment, it does not hinder on quality. Throughout the process, XP requires lots of testing so that in the final stage, little testing is done. XP has a unit test phase after each component is completed and then another "acceptance" or system test when a component is added to the main program, so when an error or defect occurs, it can be found immediately and fixed immediately. However, it should be noted that we are using a more modified version of XP. One of the downfalls of XP is that it doesn't stress very much on documentation, which we feel is important, so we have decided to create the proper documentation as expected of us by MSOE professors throughout the project.