# Implementation Specification for eFlash

***Final Version***
**Revision 1.0**

**Authors**: Siu-Pang (David) Chu, Sean Gabriel, Daniel Honegger, Darren Lam, Ying Tat Ng, Aretha Samuel, Kenneth Wong, Tony Wu, Anthony Yee

**Date: March 20, 2006**

## PROJECT ABSTRACT

The eFlash team is designing a study-skills application designed to eliminate a student's need for paper flash cards. Our software intends to replace these vital learning tools while improving on the issues inherent in a pen-and-paper system.

## DISTRIBUTION NOTES:

**Review Deadline Date:** ______________________________
**Review Meeting Date:** ______________________________

**Comment(s):**
______________________________________________________
______________________________________________________
______________________________________________________
______________________________________________________

**Reviewer List:**

**Distribution List:**

## Document Revision History

| Revision | Date | Author(s) | Comments |
|---|---|---|---|
| 0.1 | 2006-03-18 | Chu | Initial draft, Interfaces |
| 0.2 | 2006-03-19 | Gabriel | Overview through Design Decisions |
| 0.3 | 2006-03-19 | Wu | Software Architecture |
| 0.4 | 2006-03-19 | Yee | Major Views |
| 0.5 | 2006-03-19 | Samuel, Lam, Ng | Implementation through Invariants |
| 0.6 | 2006-03-19 | Honegger | Interfaces |
| 0.7 | 2006-03-19 | Wong | Recourse Usage through Alternative Implementations |
| 0.8 | 2006-03-20 | Chu | Preliminary version with revisions |
| 1.0 | 2006-04-10 | Gabriel | All issues resolved |

# Table of Contents

# Document Overview

## Purpose

This specification describes the implementation of the project for the purpose of planning, implementing and recording the implementation. It is useful to record in a central location all relevant information about the implementation of the project. The detail is also helpful for generating schedules and for helping new employees get up to speed. Implementations that were considered but not chosen can be described in the *Alternative Implementation* section.

## Short Description

eFlash has been designed around the need of students from various academic disciplines to compile their subject knowledge into a flash card repository. Among our major design decisions, we will implement functionality for mixed media flash cards, learning quizzes, printable layouts, and PowerPoint format export.

These features, drawn from our requirements and function design specifications, will be delivered through an event-driven GUI (view) that interacts with an application layer (controller) sitting on top of a back-end database (model). Our application layer is home to the eFlash Creator and eFlash Viewer, both of which will be primarily call-and-return among various classes.

## Authors

Development will be the main author of this document. Other team members have contributed their expertise in various areas as well.

## Target Audience

The audience for this specification is fellow developers in the work group.

## Specification Flow

The Implementation Specification should be written concurrently with the Functional Design Specification.

## Review Considerations

The primary review objective is to check that the Implementation Specification contains the functionality contained in the Functional Design Specification.

# Introduction

## Introduction – Requirements

When designing eFlash, we were looking for all of the problems in students' current ways of studying, and primarily focused upon the difficulties of using flash cards as an exclusive study aid. Customer feedback pointed to several recurring ideas:

1. **Flexibility.** Flash cards are often a "one-time deal," and once created are difficult to edit and embellish without starting over again. Our customers appreciate the freedom of a blank canvas when laying out, coloring, and otherwise designing their flash cards, but recount the tedium in perfecting the cards for use. They also point to the wasted effort spent in correcting minor mistakes to a flash card.

2. **Self-testing.** The name of flash cards implies their use, i.e. a student will quickly view one side to see if he/she knows the corresponding answer on the other side. Our customers appreciate being able to control the pace of such an exercise, but also spend a great deal of time tracking their own progress and making mental notes, which detracts from their overall studies. Occasionally, there are also defects in the media (seeing through to the other side) that defeat the purpose of such quizzing.

3. **Mixed media support.** While flash cards are the preferred study tool for foreign language vocabulary and others sets of terms and definitions, they are impractical for storing visual information (art, diagrams, and some complex foreign alphabets) and impossible for storing audio information (music, pronunciation).

4. **Sharing and reusability.** Every semester, thousands of students across the country invariably end up making near-identical decks to study similar material for a common subject. The problem lies in the limitations of the medium – sharing a deck with a fellow student requires a physical meeting, and the creator ends up losing his or her copy of the deck as a result.

5. **Search.** Finding a card within a large deck is a fully manual and time-consuming process for a student. Generally, decks are built with categories in mind, but having only one piece of physical media makes it difficult to cross-reference cards across multiple decks without creating duplicates, again a time-consuming process. Changing one such card would also require a change to all of the duplicates.

## Introduction – Tasks

Here is a time estimation (given in units of weeks per programmer) of the major functionality in eFlash, used to satisfy the requirements listed above. Not listed is the database access layer, which we roughly rate as 2 units of work effort.

1. **eFlash Creator.** This subcomponent of eFlash is responsible for creating flash card decks that can be used either in the Viewer or in another application (through our export functionality). It is compromised of the following functionality:

   - Text-input forms designed to collect lists of definitions quickly (1).
   - Support for three types of objects on flashcards:
     1. Text – A single area of text that supports custom text formatting. (½).
     2. Visual – A movable resizable image (1).
     3. Sound – A sound player with control buttons (play and stop). (1½).
   - A simple interface that allows users to build/edit flashcards quickly (1½).
   - Formatting tools to allow customization of color, fonts, etc. (½).
   - Multi-sided cards for multilingual students (½).
   - Export to PowerPoint presentation format (1).
   - ~~Automatic print layouts, should physical flash cards be desired (½).~~

2. **eFlash Viewer.** This subcomponent of eFlash handles the flash card decks that have been authored by the Creator, and provides a simple viewing interface as well as a self-test mode for users desiring to quiz themselves. It is compromised of the following functionality:

   - Simple pagination through a created deck (1).
   - Quizzing mode that tracks self-imposed “correct” or “incorrect” answers when paginating through a deck, with optional timing constraints (1).
   - Randomization of deck order, or heuristic-based ordering that tests a user on his weakest cards after gathering information from previous quizzes (1½).
   - Tracking of decks and quiz statistics through modularized user profiles (½).
   - Integrated search based on card content (1).
   - Ability to cull cards from many sources into a new collection, facilitated through such search functionality (1).

3. **eFlash network connectivity.** This subcomponent of eFlash integrates with the Viewer to allow remote access to decks that have been registered with a centralized server, as well as allow the user to contribute to the network database with their own creations. It is compromised of the following functionality:

   - Browser access to a vast array of decks contributed by eFlash users (1).
   - Search functionality across the network that replicates a local search (1).
   - Ability to download other users’ decks into local user database and rank them (½).
   - Ability to upload own decks to the server (½).

## References / Contacts / Terminology

References:

*Requirements Specification*
*Design Specification*

Contacts:

| | |
|---|---|
| *General team contact* | Sean Gabriel |
| *Architecture and vision* | Kenneth Wong |
| | Tony Wu |
| *Customers* | *Various; see requirements spec* |

Terminology:

Please reference the functional design specification for an appropriate list of relevant terms and definitions.

## Limitations

As a specification of our implementation, we have taken care to address each component of eFlash with exacting detail. However, in actual development we intend to make use of standard components (for example, to build the GUI) that we do not detail the inner workings of here – all standard components are assumed to perform their function as intended, and we merely explain how they will be integrated into the application as a whole.

## Software Architecture

### Focus

The main focus of eFlash is user-centric. The primary goal of the system is to create a convenient software alternative to physical flashcards. If the user finds our application difficult and frustrating, he will simply revert back to using physical flashcards. For this reason, creating an intuitive and simple user interface is one of the highest priorities.

A second focus of eFlash is data-centric. This mainly applies to the lower layers of the system that interacts with the database. eFlash must gather data input from the user and store it in the database, from where it is then pulled from the database whenever the user views the flashcards. The system must ensure and preserve the integrity of this data.

### System Metaphor

~~eFlash is, in essence, your own set of flashcards that can be customized and organized with just a few clicks.~~

eFlash is, in essence, a flashcard workstation.

A place where a user can go to work on creating new eFlash cards or simply a dedicated location to view and study a user's locally stored flashcards.  In addition, just like a workstation, the user can effortlessly "look across his/her shoulder" to other eFlash users alike, and share their digitally mastered eFlash cards.

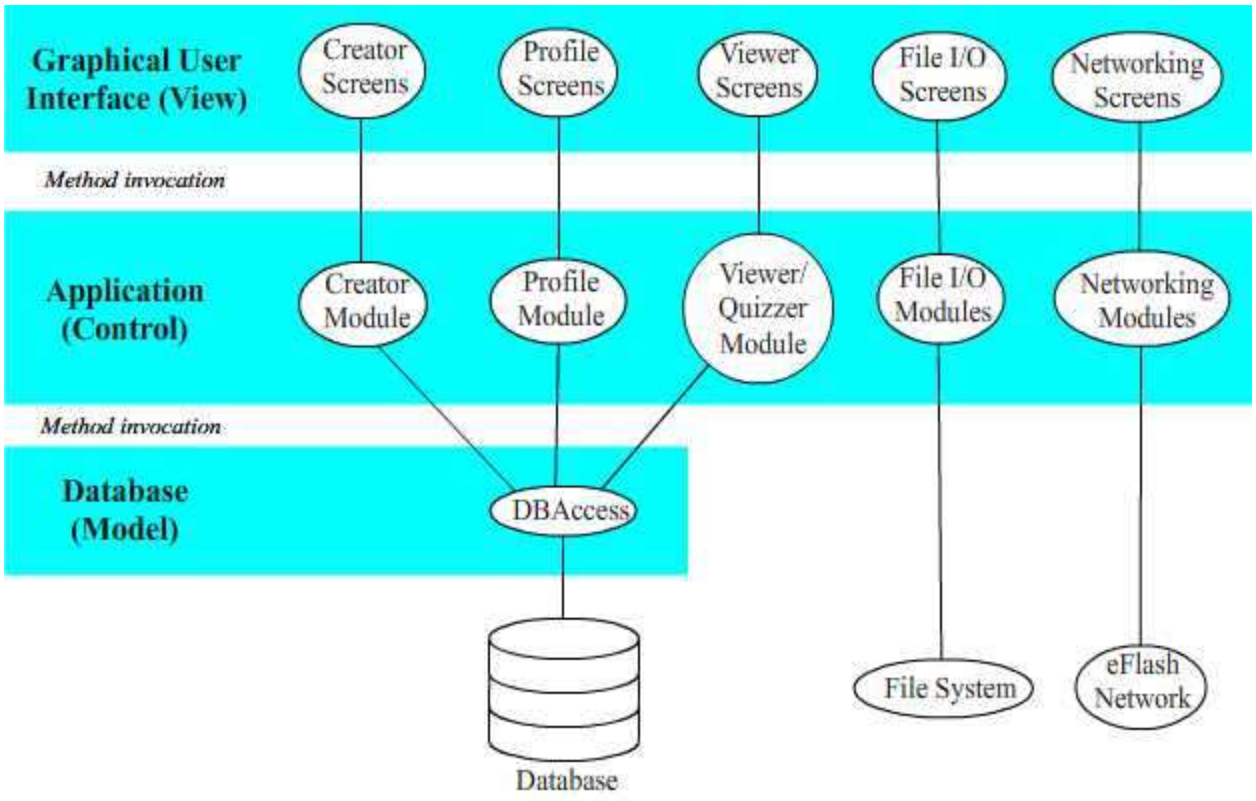
The aim of our product is to allow the user to simulate physical flashcards.  In the creator, the user is given a white canvas that resembles a flashcard, and is allowed to put words and pictures on the canvas wherever he pleases. The created flashcards are then put into decks, just like real flashcards. In the viewer, the user can browse through decks of flashcards, flipping them over to view the other side whenever needed, just as he would if he were using physical flashcards.

### Architectural Style

To maintain simplicity to the user while providing powerful functionality, eFlash uses a model-view-controller architecture. Each of the three component layers interacts with the others through method invocation. However, the view layer cannot interact directly with the model layer except through the controller. At the top, the GUI (view) layer provides the user with a fast and simple interface to the powerful features of eFlash. Below that, the application (controller) layer handles the main functionalities of our product. This layer consists of the main application code, such as the logic for the next flashcard to display, computation of user quiz statistics, etc. Finally, at the bottom is the database (model) layer, which abstracts away the implementation details of the database from the application layer. This layer provides functions, called by the application layer, that contain the database-specific details of storing, managing, and retrieving flashcard data. See the Layers of Hierarchy section below for more details about each layer.

## Layers of Hierarchy

Our system is divided into three main component layers of hierarchy: the Graphical User Interface (GUI), the Application, and Database layers.

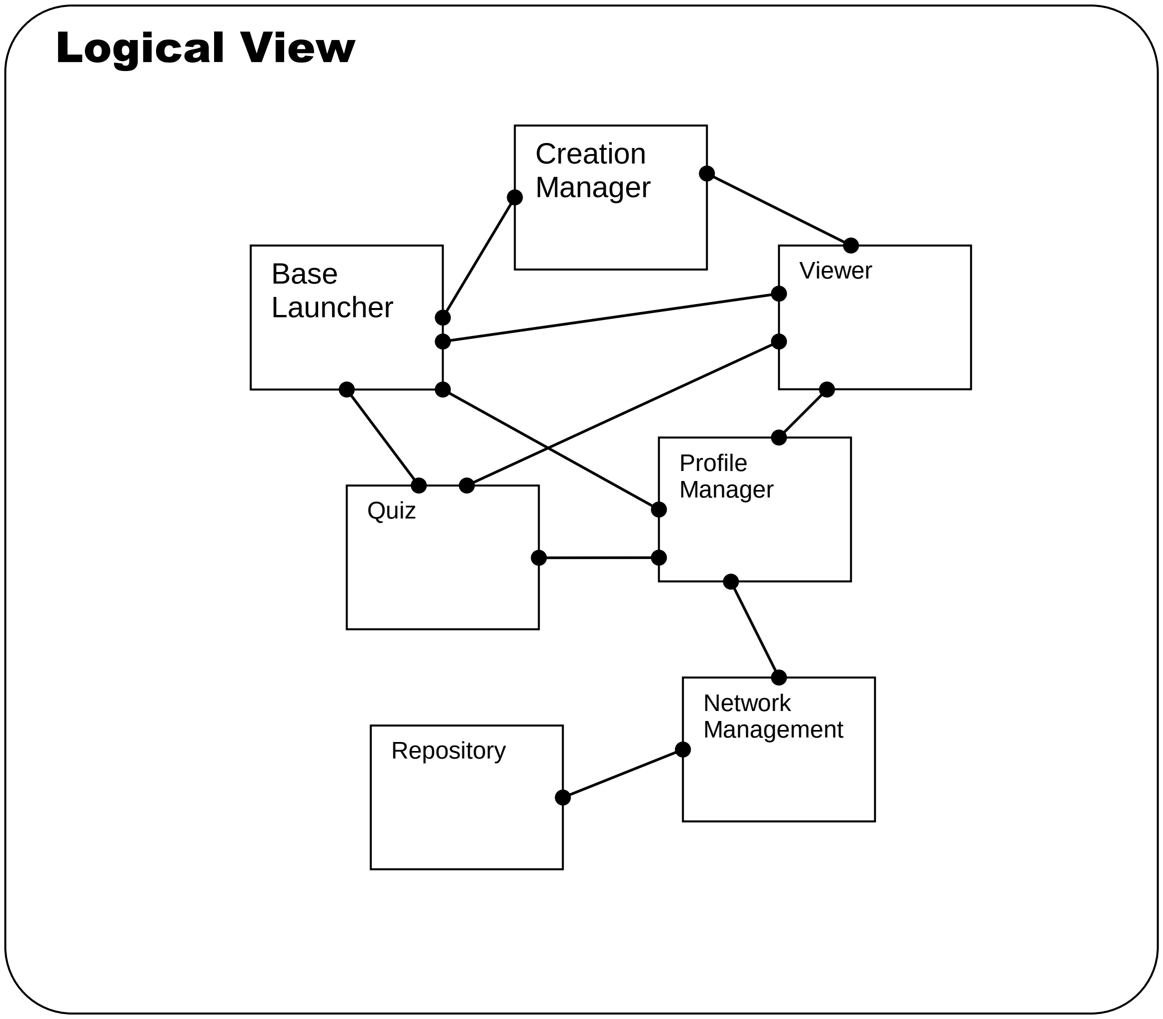


At the top, the GUI layer features an event-driven architecture. The main focus of this layer is user-centric, with the primary goal being to provide the user with a fast, intuitive way to use eFlash. The user essentially generates events through his/her clicks and other interactions, which are, through event handlers, translated into method calls to the Application layer.

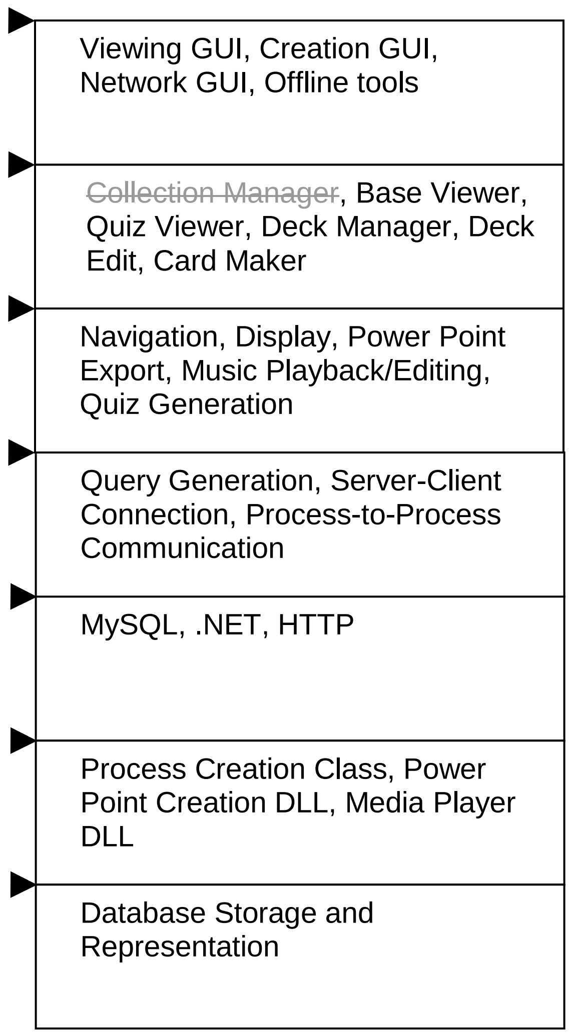
The second layer is the Application layer, which uses an object-oriented call-and-return architecture. This layer is made up of interacting modules, each of which handles some portion of eFlash functionality. In this layer are both data-centric (for example, the viewer) as well as computation-centric modules (for example, the quiz statistics calculator).

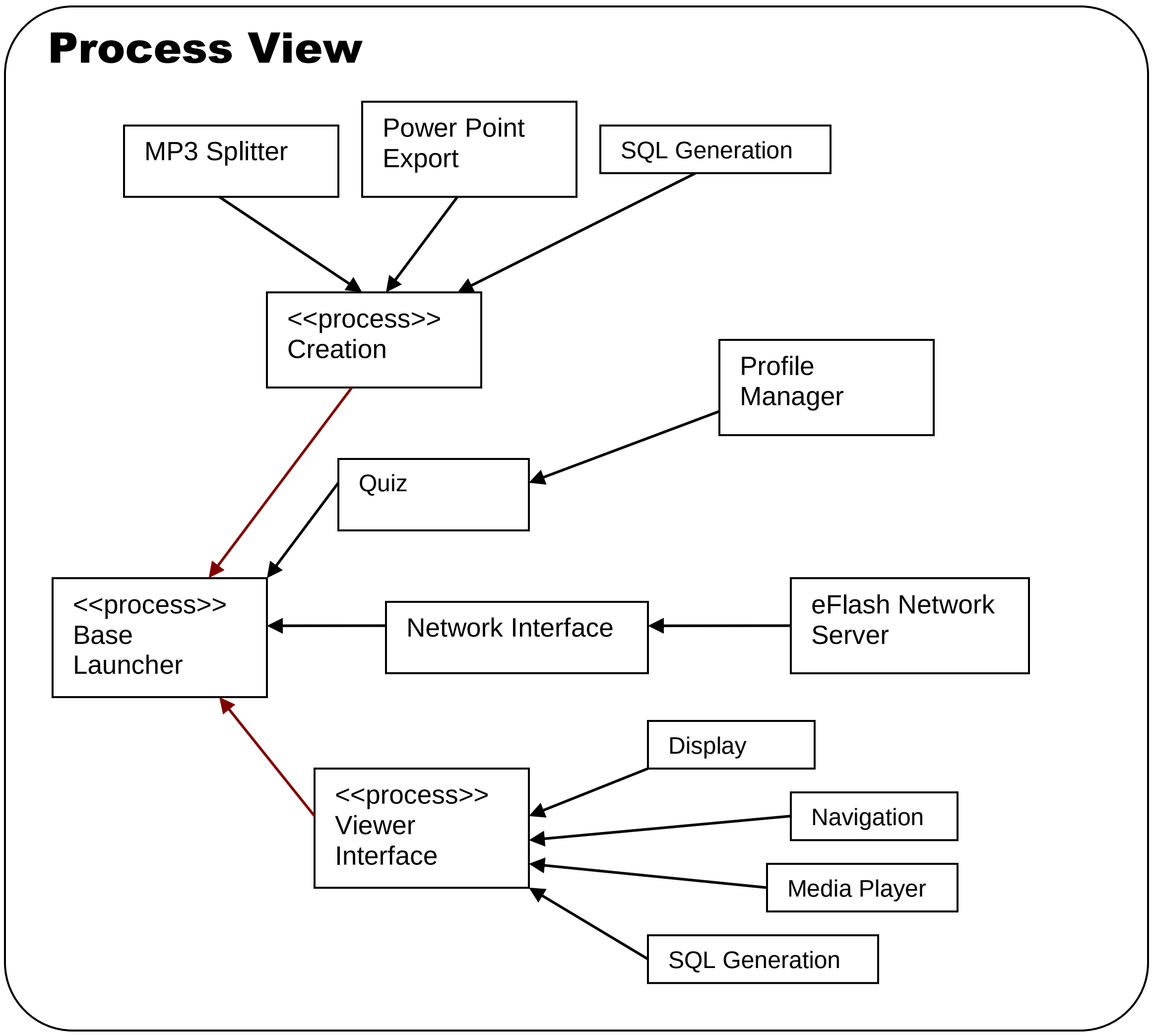
At the bottom is the Database layer, which provides the Application layer with an interface to our database. It features methods that abstract away the implementation-specific details of the database, allowing the Application layer to store and retrieve data without knowing how the database works, and providing us the flexibility to use any database we wish. The focus of this layer is data-centric, as it gathers and stores data from and to the database.

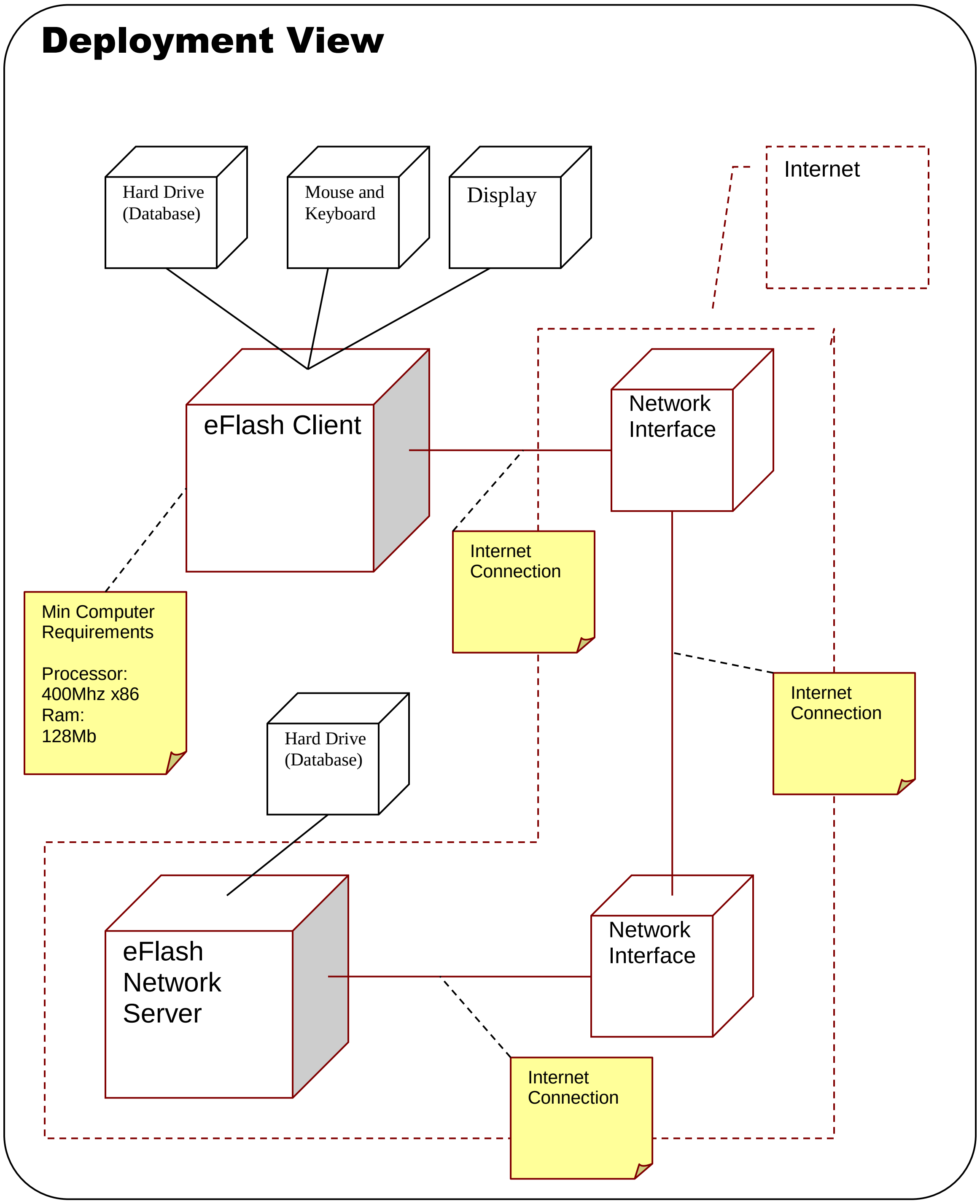
## Architecture Views



## Implementation View


Viewing GUI, Creation GUI, Network GUI, Offline tools
~~Collection Manager~~, Base Viewer, Quiz Viewer, Deck Manager, Deck Edit, Card Maker
Navigation, Display, Power Point Export, Music Playback/Editing, Quiz Generation
Query Generation, Server-Client Connection, Process-to-Process Communication
MySQL, .NET, HTTP
Process Creation Class, Power Point Creation DLL, Media Player DLL
Database Storage and Representation


Process View
MP3 Splitter
Power Point Export
SQL Generation
<<process>> Creation
Profile Manager
Quiz
<<process>> Base Launcher
Network Interface
eFlash Network Server
Display
<<process>> Viewer Interface
Navigation
Media Player
SQL Generation


Deployment View
Hard Drive (Database)
Mouse and Keyboard
Display
Internet
eFlash Client
Network Interface
Internet Connection
Min Computer Requirements
Processor: 400Mhz x86
Ram: 128Mb
Hard Drive (Database)
Internet Connection
eFlash Network Server
Network Interface
Internet Connection

# Implementation Description

## Components

### The Application Layer

This layer contains the main application code of eFlash. The modules in this layer consists of four manager modules, which deal with profiles, decks, flashcards, and collections, two modules that deal with file I/O, one to supply the logic to run quizzes, and one to handle the application's networking needs.

The following **global variables** will be used: User_name (String), User_ID (int), LocalDatabaseString (String), NetworkDatabaseString (String). Furthermore, unless otherwise specified, all components will be implemented as call and return agents that act upon the database (repository). The following screens are to be created.

- **General**
  - **Profile Manager**
    - This component provides all functions dealing with user profiles. These functions provide profile management; it is responsible for creating, deleting, and editing user profiles, login authentication, and loading profile information.
    - Receives requests from: Profile Selector Popup, New Profile Popup
    - Sends requests to: DBAccess
      - All the methods in the Profile Manager class call DBAccess to update, create, or delete records from the database.
  - **Deck Manager**
    - This module handles all requests regarding flashcard decks. Its functions include creating and deleting decks in the database, creating decks that have cards with the same Category ID, fetching a list of flashcards belonging to a given deck.
    - Receives requests from: Deck Creation/Edit Screen
    - Sends requests to: DBAccess
      - All the methods in the Deck Manager class call DBAccess to update, create, or delete records from the database.
  - ~~**Flashcard Manager**~~
    - ~~This module provides all functions dealing with flashcards in the database. These include creating, editing, deleting, and retrieving flashcards.~~
    - ~~Receives requests from: Flashcard Input Gatherer Screens, Viewer Collection Setup Screen~~
    - ~~Sends requests to: DBAccess~~

        - ~~(All the methods in the Flashcard Manager class call DBAccess to update, create, or delete records from the database)~~
    - ~~**Collection Manager**~~
        - ~~This module handles all requests relating to collections, including fetching a list of flashcards belonging to a specific collection, and adding cards to a collection, creating/deleting collections.~~
        - ~~Receives requests from: Viewer Collection Setup Screen, Viewer~~
        - ~~Sends requests to: DBAccess~~
- **Creator**
    - **(Flashcard) Creator**
        - This module contains keeps track of the internal state of the creator when the user is creating/editing decks and flashcards. It remembers the current deck, its cards, and their objects. Additionally, it provides functions for the GUI screens to call to create new cards, decks, and objects, and to save existing ones. These functions make calls to the database layer.
        - Receives requests from: Flashcard Input Gatherer Screens
        - Sends requests to: Flashcard Manager
    - **File Importer**
        - This module contains functions to read in data stored in text and then create decks and flashcards from this data.
        - Receives requests from: Import Screen
        - Sends requests to: DBAccess
- **Viewer**
    - **Viewer**
        - This module keeps track of the internal state of the viewer while the user is flipping through a deck of flashcards, such as the list of cards of the current deck as well as the current card/side being viewed. Also, it provides navigation functions, such as flipping to the previous and next cards.
        - Receives requests from: Quizzer Screen
        - Sends requests to: DBAccess, Deck Manager, Flashcard Manager, ~~Collection Manager~~
    - **Quizzer**
        - The main purpose of this module is to provide various functions needed when conducting quizzes, such as the logic to picking the next flashcard to display, ~~loading/updating current user statistics.~~
        - Receives requests from: Quizzer Screen
        - Sends requests to: DBAccess, Deck Manager, Flashcard Manager, ~~Collection Manager~~
    - **File Exporter**
        - This module has functions to output flashcard and deck information stored in

the local database to various file formats. Currently, the primary file format to export to is Powerpoint (.ppt), for users who wish to view flashcards independent of our application.
- Receives requests from: Exporter Screen

- **Networking**
  - **Networking**
    - This module handles all eFlash networking needs. This includes server login creation and authentication, retrieving lists of decks and flashcards in the server repository, and uploading and downloading decks to and from the online repository.
    - Receives requests from: Network Welcome Screen, Network Uploader/Downloader, Network Ranker
    - Sends requests to: Network Database, DBAccess (network)

### The Database Layer

This layer acts as the main data model and includes the following module.

- **DBAccess**
  - This module provides generic database access functions. The goal of this layer is to abstract away the details of the database from the application code, so that the application code does not have to deal with formulating database-specific queries.
  - Receives requests from: Profile Manager, Deck Manager, Flashcard Manager, ~~Collection Manager~~
  - Sends requests to: Local Database, Network Database

## Connectors

The two figures below, Figures 1 and 2, show the connections diagram for all the major components. The pink arrows indicate the program flow between interface screens and the black arrows indicate function calls to those components. After the diagrams, descriptions of the interface layers and how they call the components are detailed.

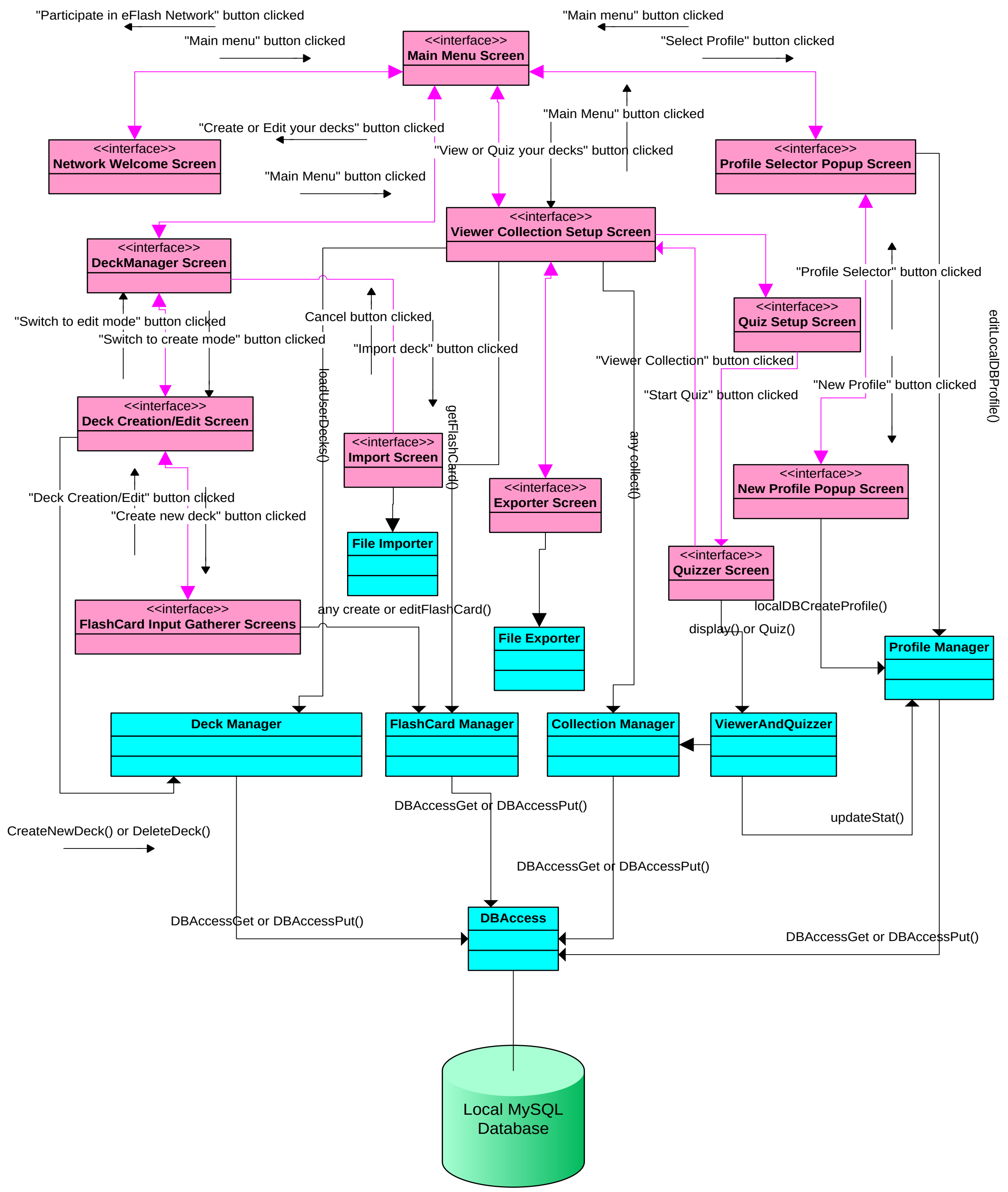

"Participate in eFlash Network" button clicked
"Main menu" button clicked
"Main menu" button clicked
"Select Profile" button clicked
<<interface>>
Main Menu Screen
"Create or Edit your decks" button clicked
"Main Menu" button clicked
"View or Quiz your decks" button clicked
"Main Menu" button clicked
<<interface>>
Network Welcome Screen
<<interface>>
Profile Selector Popup Screen
<<interface>>
Viewer Collection Setup Screen
<<interface>>
DeckManager Screen
"Profile Selector" button clicked
"Switch to edit mode" button clicked
"Switch to create mode" button clicked
Cancel button clicked
"Import deck" button clicked
<<interface>>
Quiz Setup Screen
"Viewer Collection" button clicked
"Start Quiz" button clicked
"New Profile" button clicked
editLocalDBProfile()
<<interface>>
Deck Creation/Edit Screen
loadUserDecks()
getFlashCard()
any collect()
<<interface>>
Import Screen
<<interface>>
Exporter Screen
<<interface>>
New Profile Popup Screen
"Deck Creation/Edit" button clicked
"Create new deck" button clicked
File Importer
<<interface>>
Quizzer Screen
<<interface>>
FlashCard Input Gatherer Screens
any create or editFlashCard()
File Exporter
localDBCreateProfile()
display() or Quiz()
Profile Manager
Deck Manager
FlashCard Manager
Collection Manager
ViewerAndQuizzer
DBAccessGet or DBAccessPut()
updateStat()
CreateNewDeck() or DeleteDeck()
DBAccessGet or DBAccessPut()
DBAccessGet or DBAccessPut()
DBAccess
DBAccessGet or DBAccessPut()
Local MySQL Database

Figure 1. Connections diagram for all the major components.

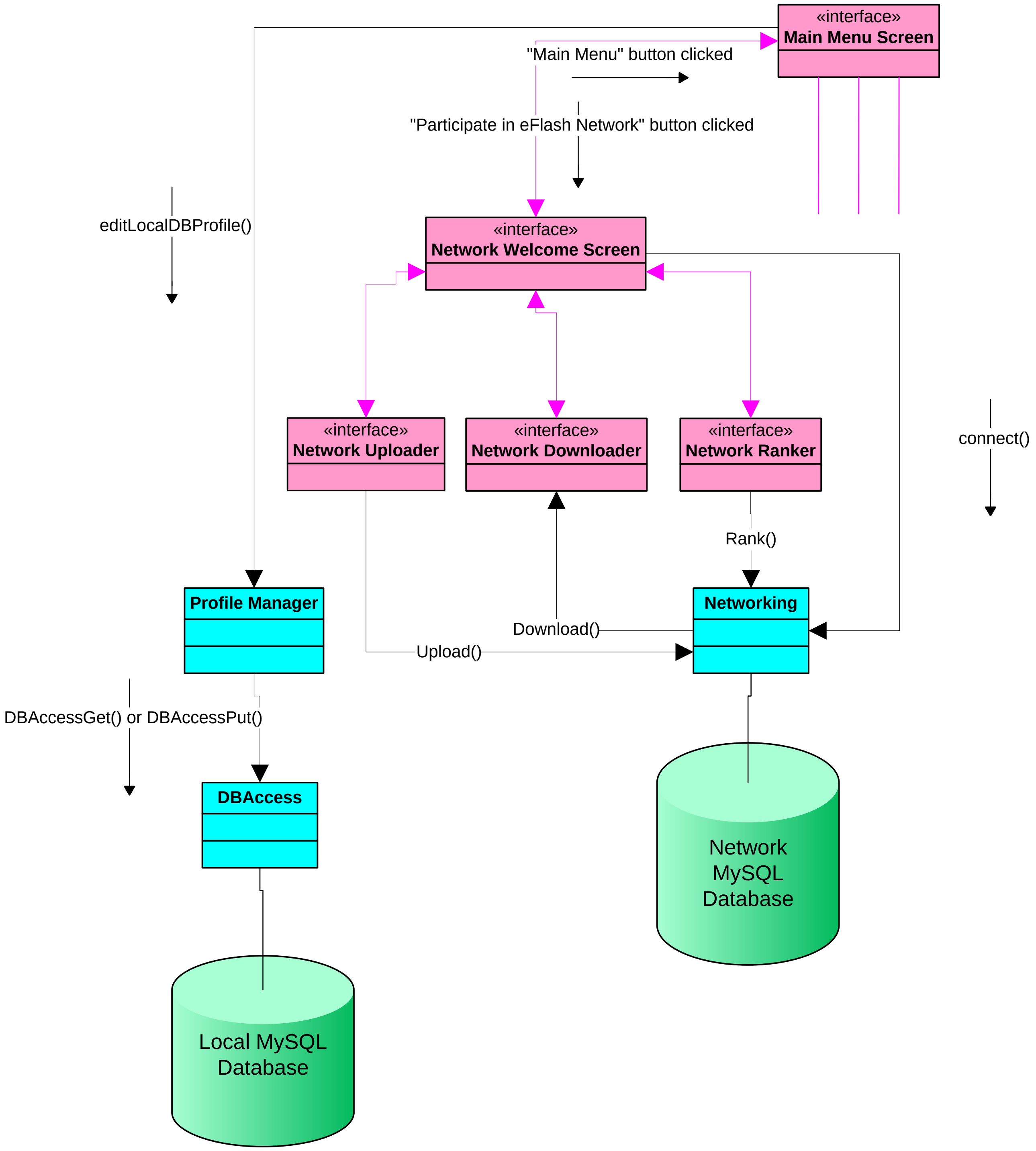


Figure 2. Connections diagram for the Networking Module.

## The Graphical User Interface Layer

This layer is the primary means by which the user interacts with eFlash. It is comprised of screen objects that follow an event-driven architecture. It interacts with the Application layer via method calls, and includes the following screens.

- **General**
  - **Profile Selector Popup**
    - Description: This Popup menu provides options to select profiles that were already created or create new profiles. This screen calls Profile Manager to access the profiles from the main database.
    - GUI connections: Main Menu, New Profile Popup
    - Application connections: Profile Manager
    - Features implemented: User profiles
  - ~~**New Profile Popup**~~
    - ~~Description: This GUI screen also calls the Profile Manager to write in the new profile created by the user to the local database.~~
    - ~~GUI connections: Profile Selector Popup~~
    - ~~Application connections: Profile Manager~~
    - ~~Features implemented: User profiles~~
  - **Main Menu / Deck Manager Screen**
    - Description: This screen serves as a portal to the other GUI screens. It displays a tree hierarchy of decks in the local database, organized by categories. The user may select any deck to modify, export, view, or quiz. Additionally, he/she may create/import new decks.
    - GUI connections: Profile Selector Popup, Layout Manager Screen, Viewer Screen, Network Welcome Screen, Import Screen, Export Screen
    - Application connections: None
    - Features implemented: None
- **Creator**
  - ~~**Deck Manager Screen**~~
    - ~~Description: This screen gives options for either creating/editing decks manually or importing deck information from a file. This screen does not directly call any components however it progresses to other screens, which do call the components.~~
    - ~~GUI connections: Main Menu, Deck Creation/Edit Screen, Import Screen~~
    - ~~Application connections: None~~
    - ~~Features implemented: Deck management~~
  - ~~**Deck Creation/Edit Screen**~~
    - ~~Description: This screen allows the user to create or edit decks by deleting or~~

~~adding flash cards. It calls the Deck Manager component so that it will carry out the functions to insert new decks or delete old ones from the database.~~

- ~~GUI connections: Deck Manager Screen, Flashcard Input Gatherer Screen~~
- ~~Application connections: Deck Manager~~
- ~~Features implemented: Deck management~~

- **Layout Editor Screen**
  - Description: This is the primary screen the user will use to create flashcards. It shows a white rectangular canvas, with a toolbar on the left to allow the user to add objects. These objects can then be repositioned and resized at will. An array of buttons at the top allows for arbitrary formatting of text.
  - GUI connections: Main Menu
  - Application connections: Deck Manager
  - Features implemented: Flashcard creating
- **Import Screen**
  - Description: This screen allows the user to import text files for the creation of decks and flashcards; therefore, this screen calls the File Importer module to get the functions completed.
  - GUI connections: Deck Manager Screen
  - Application connections: File Importer
  - Features implemented: File importing
- ~~**Flashcard Input Gatherer Screens**~~
  - ~~Description: These screens allow the user to create music, picture, or text-based flash cards. It calls the Flashcard Manager module to utilize most of these functionalities.~~
  - ~~GUI connections: Deck Creation/Edit Screen~~
  - ~~Application connections: Creator~~
  - ~~Features implemented: Flashcard creation~~

**Viewer**

- ~~**Viewer Collection Setup Screen**~~
  - ~~Description: This screen allows the user to create collections of decks and flash cards for quizzing and viewing purposes. Because this functionality requires interaction with Decks, Flashcards and Collections, it will call all three of these components: Deck Manager, Flashcard Manager and Collections Manager.~~
  - ~~GUI connections: Quiz Setup Screen, Exporter Screen, Main menu~~
  - ~~Application connections: Collection Manager, Deck Manager, Flashcard Manager~~
  - ~~Features implemented: Collection management~~
- **Viewer Setup Screen**
  - This screen allows the user to flip through decks of flashcards one card at a

time.
  - GUI connections: Quiz Setup Screen, Exporter Screen, Main menu
  - Application connections: Viewer Module
  - Features implemented: Flashcard viewing

- **Quiz Setup Screen**
  - Description:  This screen will allow the user to select what sort of quiz he/she would like to use and as well offers the option of using the Flashcard viewer to test themselves.
  - GUI connections: Quizzer Screen
  - Application connections: None
  - Features implemented: Flashcard viewing
- **Quizzer Screen**
  - Description:  This screen will make use of the ViewerAndQuizzer component to provide most of the functionalities related to producing slide shows and creating quizzes.
  - GUI connections: Viewer Collection Setup Screen
  - Application connections: ViewerAndQuizzer
  - Features implemented: Flashcard viewing
- **Exporter Screen**
  - Description:  This screen will provide the user the opportunity to export the flashcards into PowerPoint (.ppt) format. Since it only requires one function it will only call the File Exporter module to accomplish all of the needed tasks.
  - GUI connections: Viewer Collection Setup Screen
  - Application connections: File Exporter
  - Features implemented: File exporting

- **Networking**
  - **Network Welcome Screen**
    - Description: This screen serves as the portal to the other three networking screens.
    - GUI connections: Main Menu, Network Uploader/Downloader, Network Ranker
    - Application connections: Profile Manager, Networking
    - Features implemented: Networking
  - **Network Uploader/Downloader**
    - Description: This screen allows the user to browse the network decks and select which ones to upload/download. For each deck, the user may download the preview image to preview the first side of the first side of the deck.
    - GUI connections: Network Welcome Screen

  - Application connections: Networking
  - Features implemented: Networking
- **Network Ranker**
  - Description: This screen allows the user to view rankings for each deck in the online database and cast his/her won vote.
  - GUI connections: Network Welcome Screen
  - Application connections: Networking
  - Features implemented: Networking

**Control elements**

We have followed the notation from Pressman's book that dotted boxes indicate control process and dotted arrows indicate control flows (see Figure 3). Basically, most of our GUI screens, which includes Main Menu Screen, Deck Manager Screen, View Collection Setup Screen, and Profile Selector Popup Screen, are control processes because they are in event-based architectural style; when a user clicks on a button, it triggers an event and links to another GUI screen. For example, users can click on "Create or Edit your decks" button on the Main Menu Screen to go to Deck Manager Screen. For the detail about the GUI interface, please refer to your Interface section.

Since our control process only flows from one GUI screen to another GUI screen, side effects will not occur when we exit from those GUI screens. (The parts that could affect our data in the local database are those creator screens, which are considered as data elements). Since we always start at the Main Menu Screen at the beginning of our application, we do not need a separate initialization for our startup.

Moreover, we do not allow multiple threaded applications; therefore we do not need to handle communication between simultaneously executing threads. In the Networking module there are going to be multiple threads; however, the Network MySQL server will take care of these multiple transactions. Only the owner is allowed to modify his/her decks - therefore there are no concurrent issues in this regard. Users can only read multiple decks - therefore there will not be a situation where multiple users can modify the same deck.

**Control Flow**

Control flow in this program is headed by user actions, as shown in Figure 3. This program provides all sorts of features and the user is in charge of directing the flow of the program. The user would be guided through the GUI screens, and the functionalities requested would be served by the components listed above.

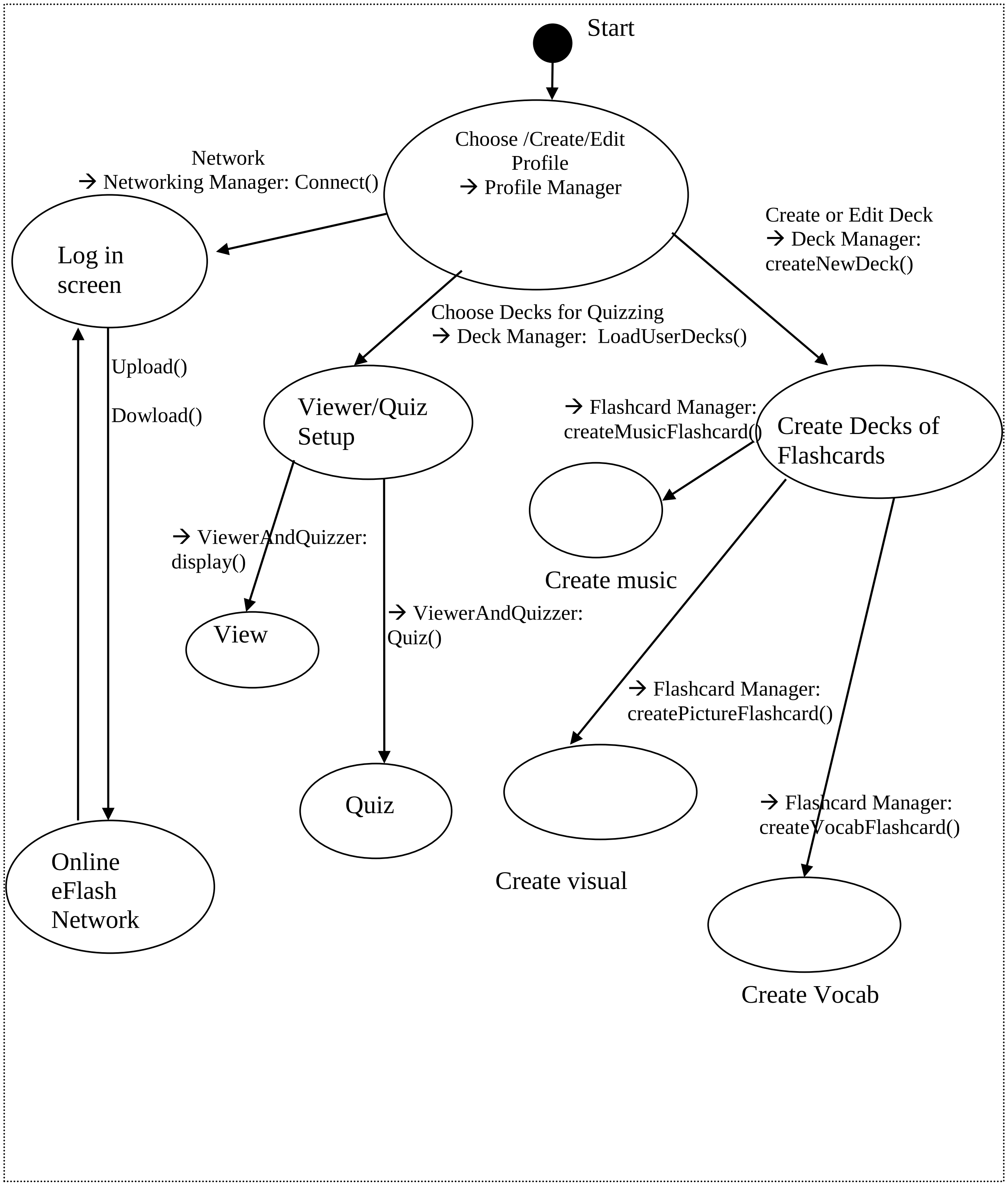

Start
Choose /Create/Edit Profile
→ Profile Manager
Network
→ Networking Manager: Connect()
Log in screen
Create or Edit Deck
→ Deck Manager: createNewDeck()
Choose Decks for Quizzing
→ Deck Manager: LoadUserDecks()
Upload()
Dowload()
Viewer/Quiz Setup
→ Flashcard Manager: createMusicFlashcard()
Create Decks of Flashcards
→ ViewerAndQuizzer: display()
Create music
View
→ ViewerAndQuizzer: Quiz()
→ Flashcard Manager: createPictureFlashcard()
Quiz
→ Flashcard Manager: createVocabFlashcard()
Online eFlash Network
Create visual
Create Vocab

Figure 3. Control Flow and Data Flow diagram.

## Data elements

The data is stored in central locations, in the local user database and the networking database. The DBAccess module is completely responsible for requesting/writing data into the database. This module is called by most of the major components and is created to present clear functionality. Situations would not arise where the modules will have different methods of communicating to the database.

One standard interface to the database will stop the practice of redundant coding. Refer to Figure 3 for data flow as well.

## Invariants

Please see the APIs section for detailed invariants given for every major function call. In addition, we have documented high level invariants with respect to the repository at the relevant sections that list the database schema.

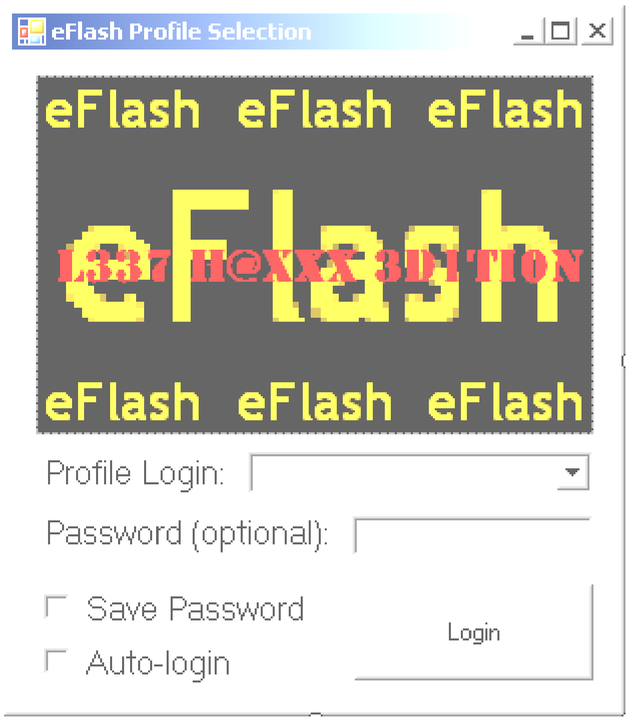
## Use Cases and Activity Diagrams

To supplement the above information and diagrams, we have created sample use cases and activity diagrams. This information is provided in our functional design specification, which is available on the project web site.

## Interfaces

This section details the individual GUI elements (and other interfaces) present in eFlash.

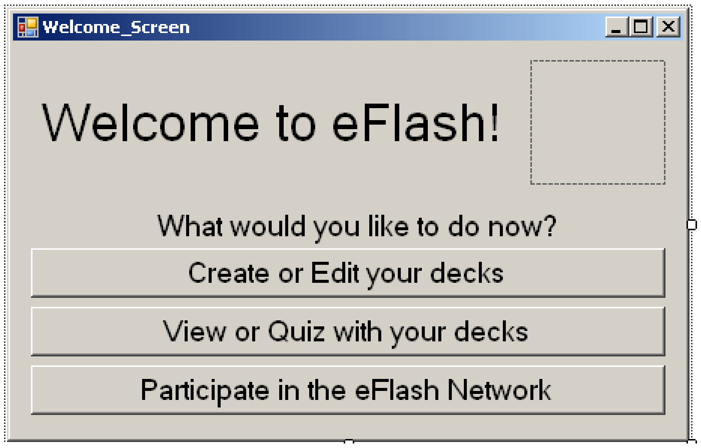
### Profile Selection Screen



**User profiles**
The interface bears a resemblance to AIM, which is familiar to our customers. Actual display image may be altered at implementation time – this is merely a mock-up.

- Choose from a list of existing profiles, with one <New User>.
- Password field is optional.
- If password exists, it can be saved by checking the box labeled "Save Password", and is then automatically filled in each time the profile is selected.
- This screen also has a checkbox to let user specify "Auto-login," which automatically logs in to the currently selected profile each time eFlash runs, bypassing the login popup.
  - Only one profile may be designated "auto-login."
- Internally, local profiles will also be associated with **nuid**, determined on first connect to the network, to be used whenever connected to the network.
- GUI Connections: Welcome Screen

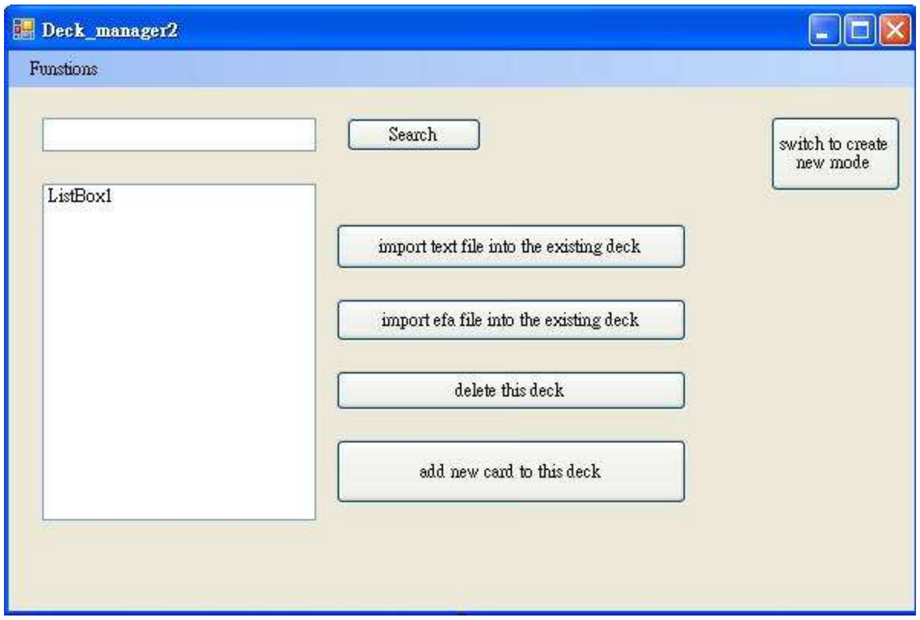
## Welcome Screen


Welcome_Screen
Welcome to eFlash!
What would you like to do now?
Create or Edit your decks
View or Quiz with your decks
Participate in the eFlash Network

**Program Navigator**

- Move to deck manager to create new deck or edit existing
- Move to view / quiz setup screen to select the deck that the user wants to quiz or view on.
- Move to the network manager to upload, download, and rate decks.
- GUI Connections: Deck Manager, View/Quiz Setup screen, Network Manager

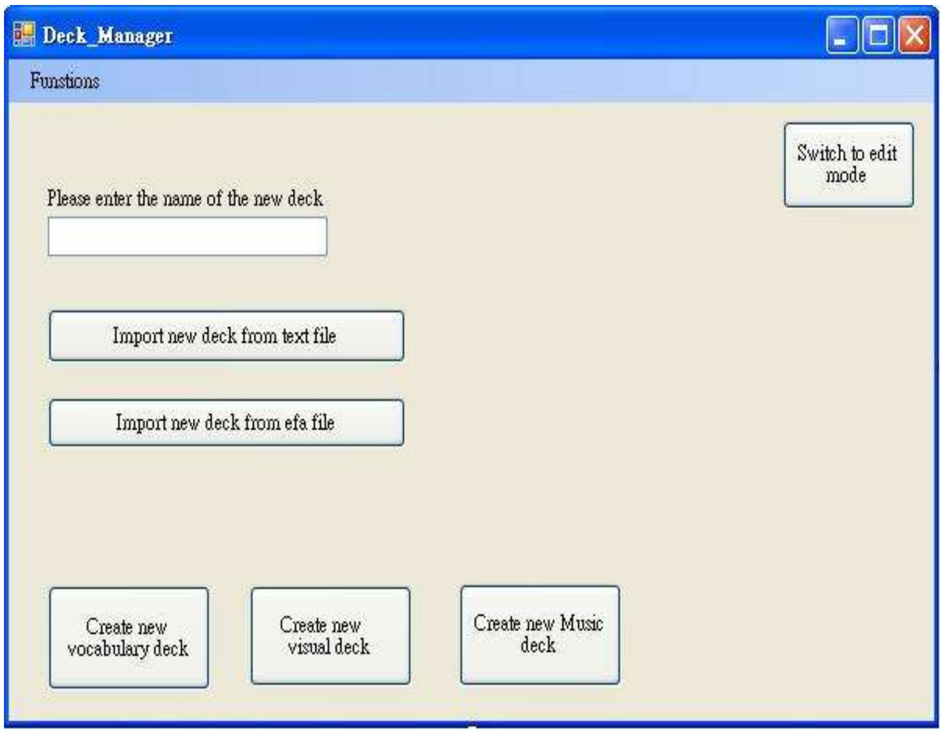
**Deck Manager** (Edit Existing Deck) (*default*)


Deck_manager2
Funstions
Search
switch to create new mode
ListBox1
import text file into the existing deck
import efa file into the existing deck
delete this deck
add new card to this deck

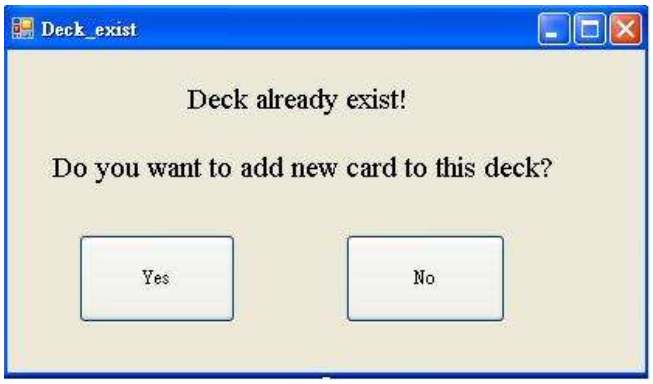
Before creating the flash card, the user must to choose which deck the new card belongs to. Once the user enters this screen, a list of the existing decks will show up in the listBox automatically. **Note that there is no longer a .efa file format.**

- **Search** - The user can search for the existing deck.
- **Import** - Highlighting an existing deck then clicking either import button, the user can import a text file to the chosen deck.
- **Delete this deck** - this function will delete the existing deck that has been chosen by the user.
- **Add new card to this deck** (*default action*) - the user can choose a deck and click this button, then the screen will jump to the create screen so the user can continue adding new flash cards to this deck.
- **Switch to "Create New Deck" mode -** switch to another mode of the deck manager.

**Deck Manager** (Create New Deck)


Deck_Manager
Funstions
Switch to edit mode
Please enter the name of the new deck
Import new deck from text file
Import new deck from efa file
Create new vocabulary deck
Create new visual deck
Create new Music deck

From this screen, the user can create a new deck with a given name and type. After the user clicked one of the three "create" button (at the bottom), the screen will jump to the create screen. Then the user can create flash cards. However, if the name of deck already exists in the local database, this window will pop-up:

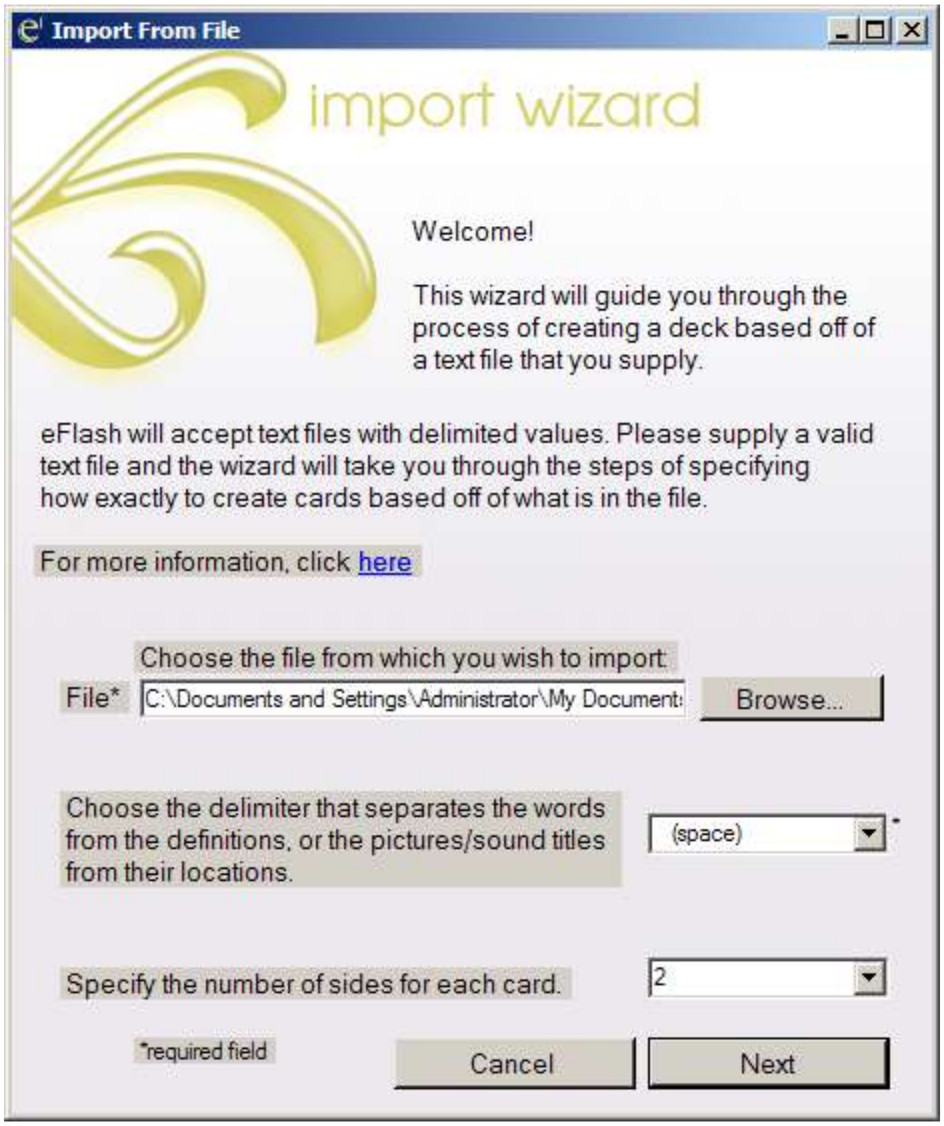

Deck_exist

Deck already exist!

Do you want to add new card to this deck?

- Selecting **Yes** will continue adding to the existing deck
- Selecting **No** will go back to deck manager 2 screen, then the user can re-enter the name.

**File Import**
GUI – text file import wizard screen



Welcome!

This wizard will guide you through the process of creating a deck based off of a text file that you supply.

eFlash will accept text files with delimited values. Please supply a valid text file and the wizard will take you through the steps of specifying how exactly to create cards based off of what is in the file.

For more information, click here

Choose the file from which you wish to import:

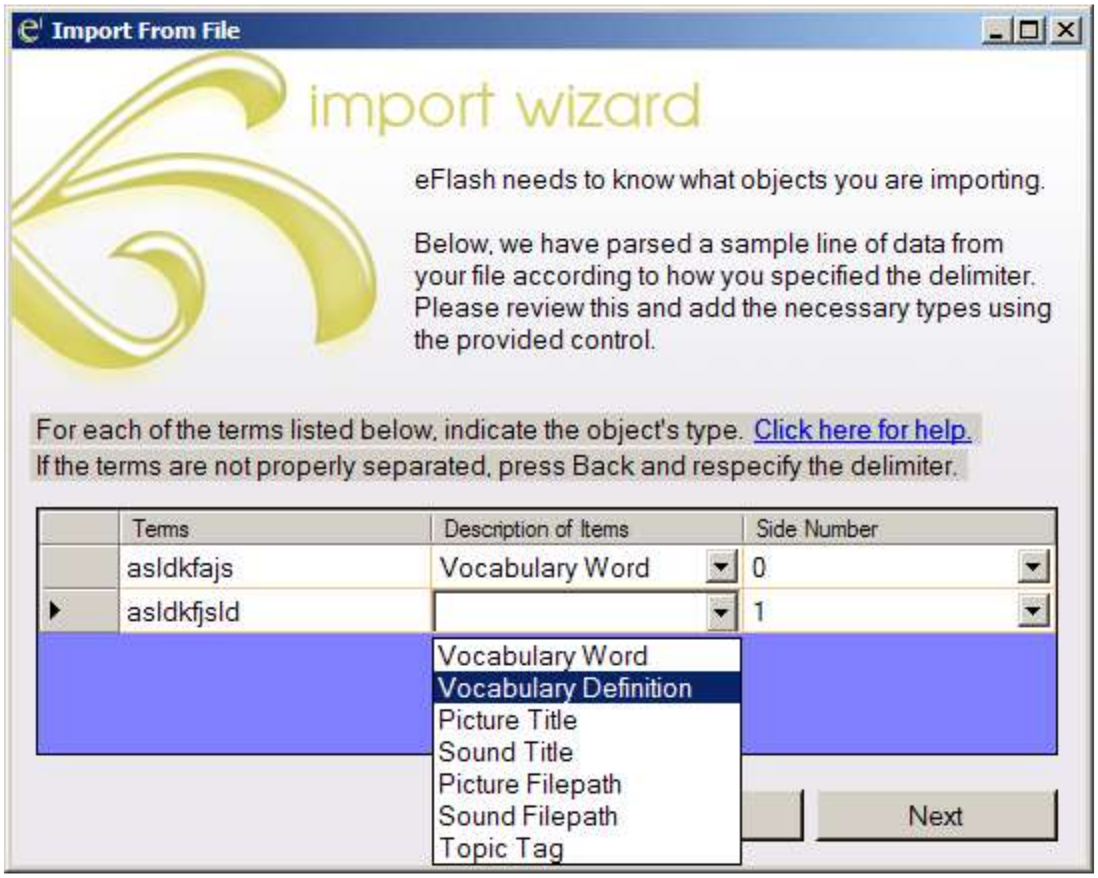
File* C:\Documents and Settings\Administrator\My Document:

Choose the delimiter that separates the words from the definitions, or the pictures/sound titles from their locations. (space) *

Specify the number of sides for each card. 2

*required field

The purpose of the import wizard is to retrieve a user-defined list of words and automatically create flashcards from that list of words.  In this screen, the user is asked to supply the location of the file, the delimiter that separates each word from its definition, and the number of sides of each card.



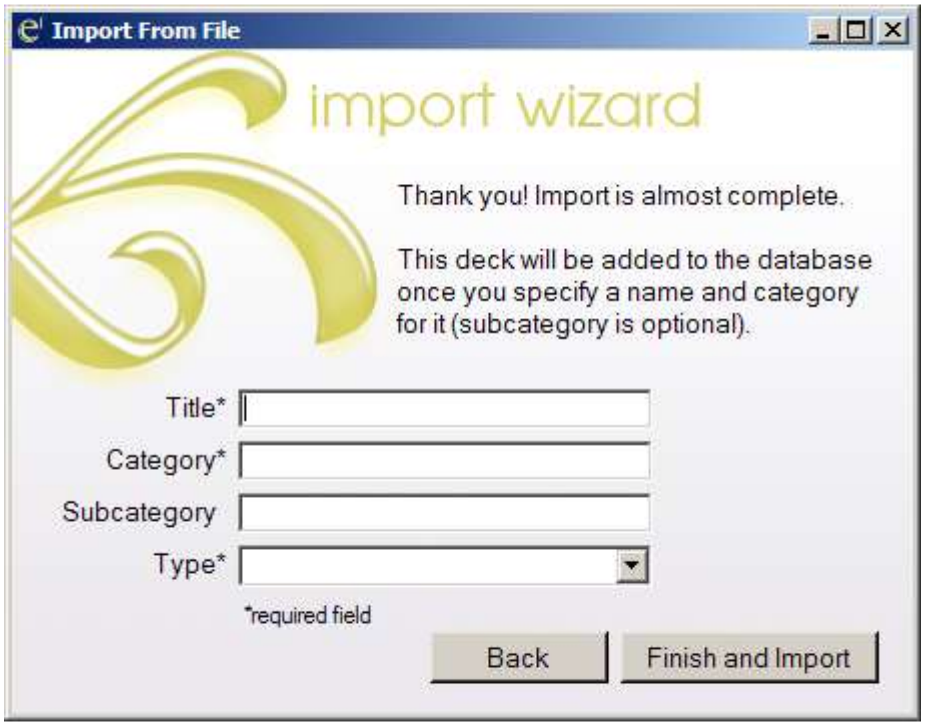
eFlash needs to know what objects you are importing.

Below, we have parsed a sample line of data from your file according to how you specified the delimiter. Please review this and add the necessary types using the provided control.

For each of the terms listed below, indicate the object's type. Click here for help.
If the terms are not properly separated, press Back and respecify the delimiter.

| Terms | Description of Items | Side Number |
|---|---|---|
| asldkfajs | Vocabulary Word | 0 |
| asldkfjsld | | 1 |

In this next screen, the first line of the import text file is being viewed on the screen. This import screen retrieves the line and separates the text into the words based upon the given delimiter. Here the user can indicate the type of word and its corresponding side on the flashcard. Then, if the user is creating a deck of flashcards for the first time, he or she is guided to this next screen below.



Thank you! Import is almost complete.

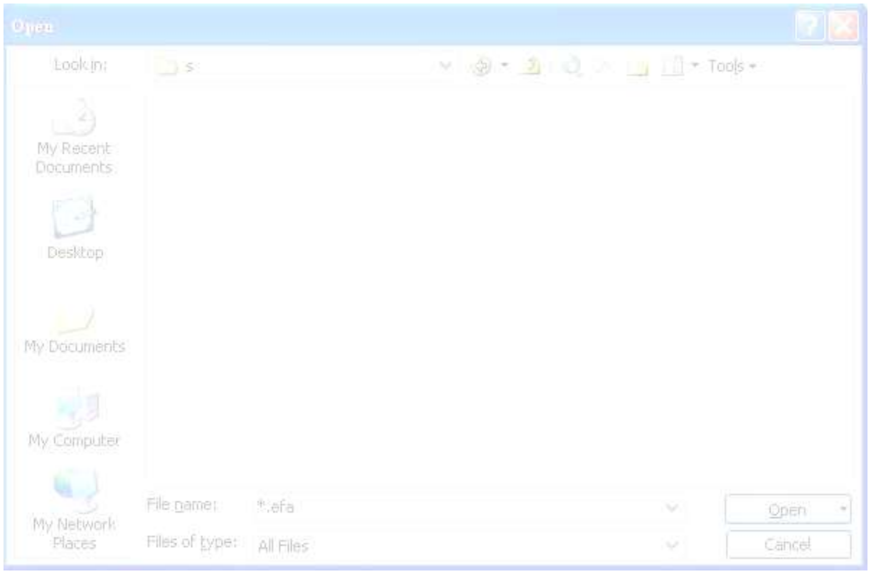
This deck will be added to the database once you specify a name and category for it (subcategory is optional).

|  |  |
|---|---|
| Title* | |
| Category* | |
| Subcategory | |
| Type* | |

*required field

Here the user can supply a title, category and type field to create the deck that will contain these newly imported flashcards.

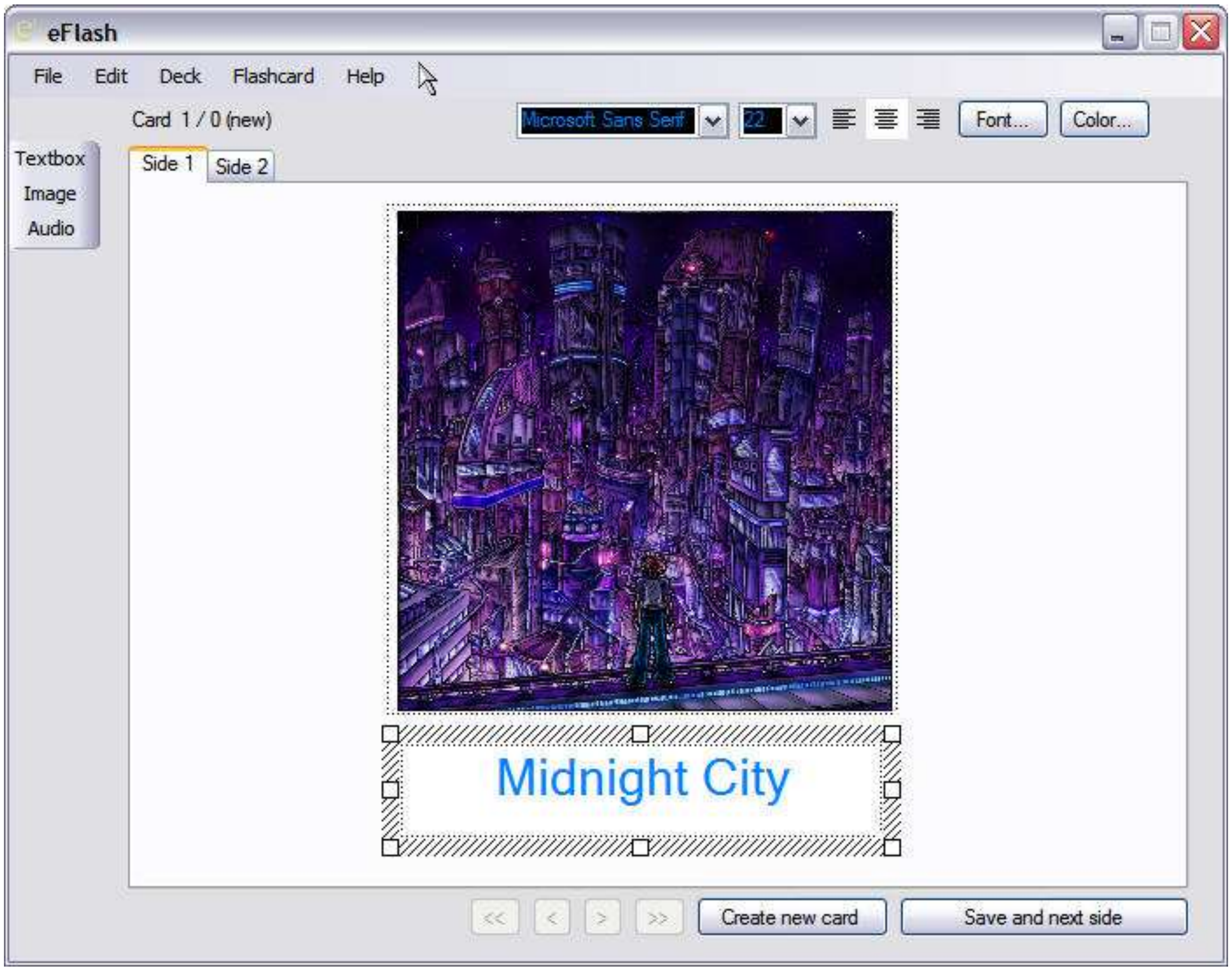
~~GUI – .efa file import wizard screen~~



File name: *.efa
Files of type: All Files

~~Browse the hard disk to find the .efa file that the user wants to import.~~

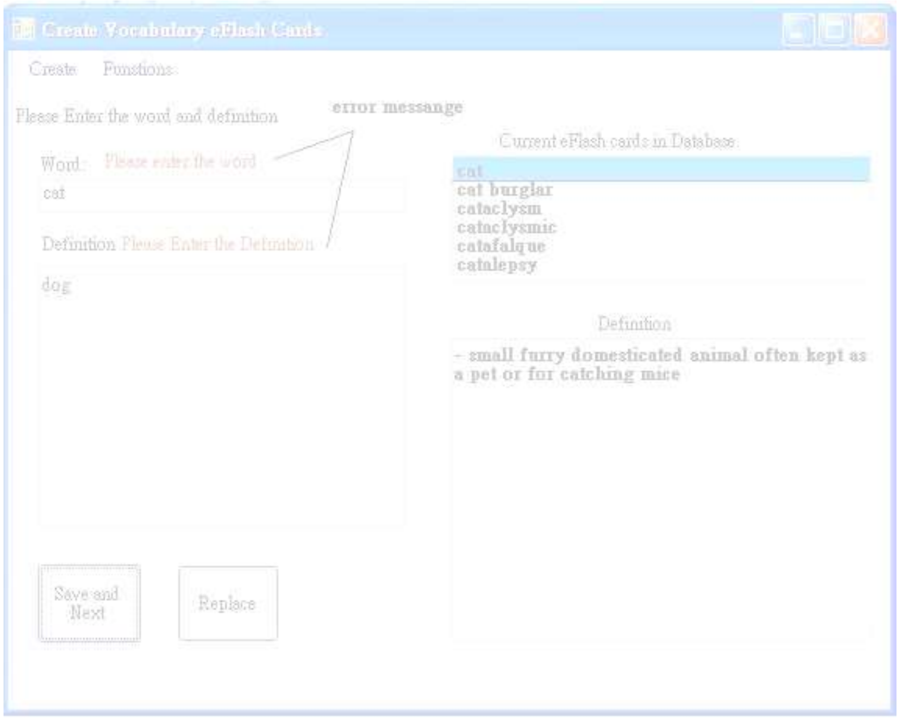
**Creator**



- **Layout Editor**
  - User can place one of three types of objects onto the flashcard:
    - Textbox – A rectangular area of text that supports arbitrary text formatting and coloring, using the formatting controls displayed above the flashcard.
    - Image box – A rectangular area that can display an image of the user's choosing.
    - Audio player – Provides controls for opening and playing sound files (not shown above).
  - All objects can be repositioned and resized at will by dragging their borders.
  - Each flashcard can have between one to five sides, with up to five objects per side.
  - Navigation buttons below the flashcard allow the user to navigate through the deck.
  - A smart "Save and next side/card" button automatically flips to the next side of the current flashcard, or, if the currently displayed side is the last side, the

first side of the next flashcard. This feature is designed to save the user time when creating large decks of flashcards.

Note: Because we have decided to use an object-based system to drag-and-drop the different types of media onto a flash card, the need for separate creators is no longer present. The following sections (showing the depreciated method of splitting flashcards into three different types) still remain useful as a reference for how to collect user input for each of the types of media, but the screenshots are no longer valid.


Create Vocabulary eFlash Cards
Create
Functions
error message
Current eFlash cards in Database
Definition
Save and Next
Replace

Please Enter the word and definition

Word: Please enter the word

cat

Definition Please Enter the Definition

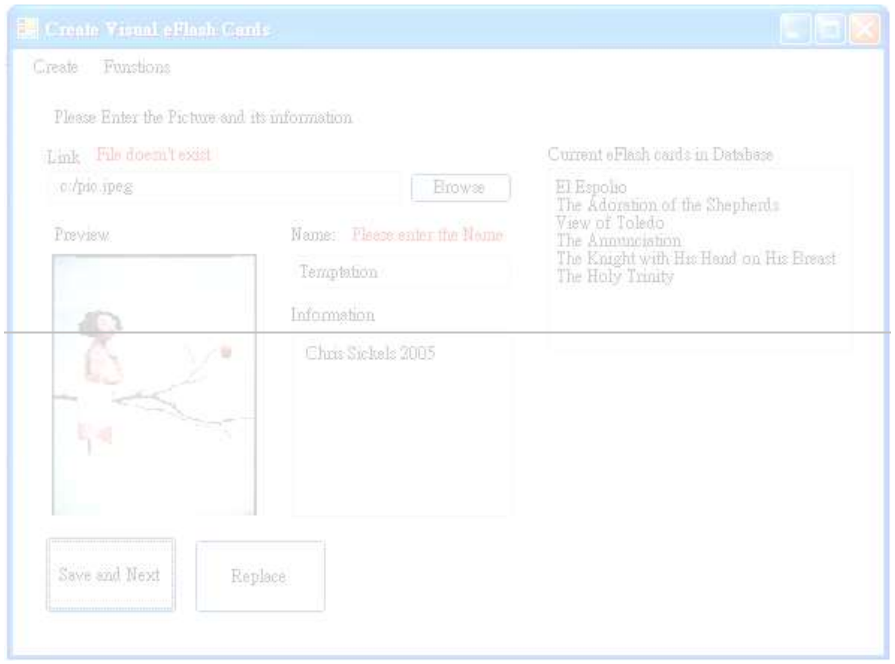
dog

cat
cat burglar
cataclysm
cataclysmic
catafalque
catalepsy

- small furry domesticated animal often kept as a pet or for catching mice

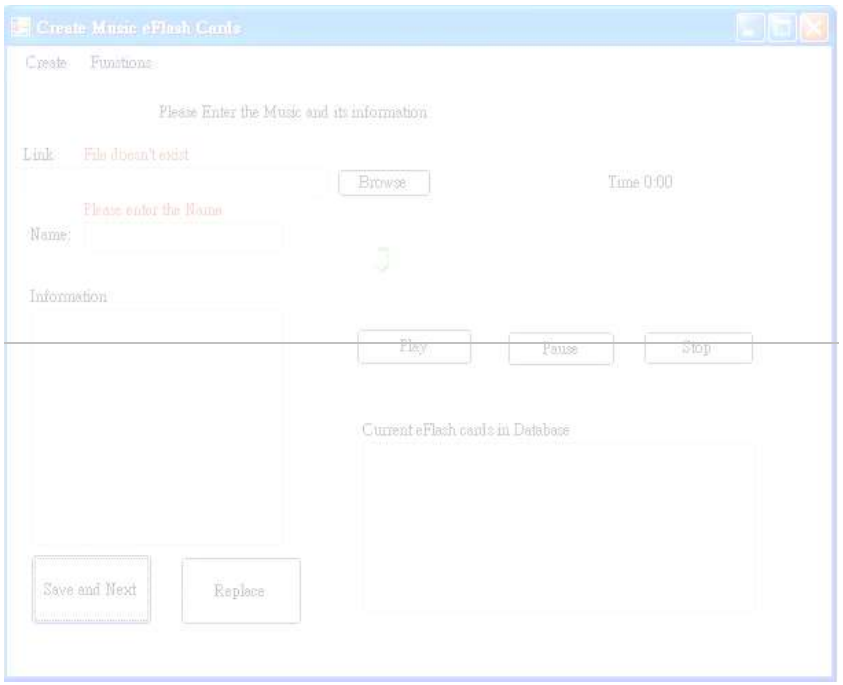
~~**Vocabulary**~~

- ~~User must input both “word” and “definition”.~~
- ~~A textbox displays the definition of the currently selected word in the currently existing list.~~
- ~~Error 1: Empty Word – Give warning: Please Enter the Word on the screen.~~
- ~~Error 2: Empty Definition – Give warning: Please Enter the definition on the screen.~~
- ~~Error 3: The Word already exists – The “Save and Next” button is disabled, and “Replace” is enabled.~~



~~**Visual**~~

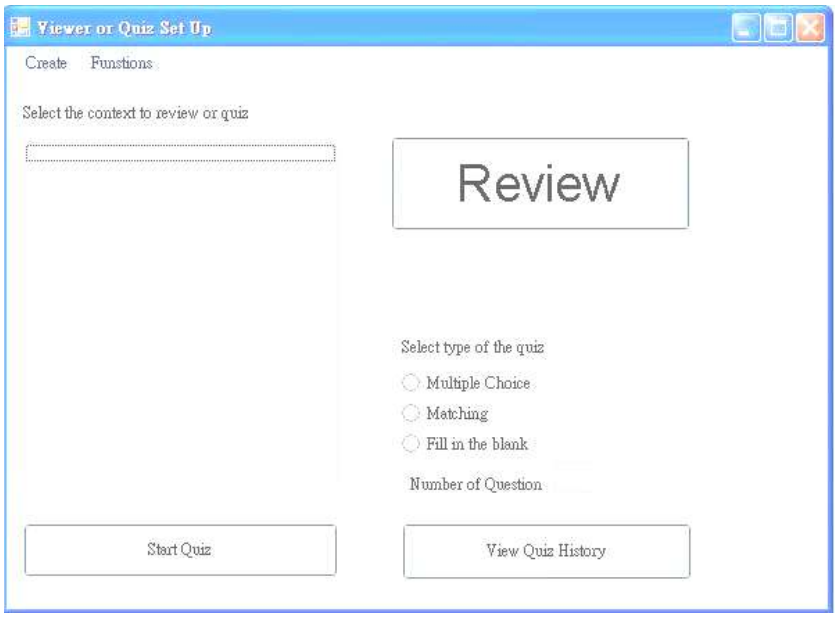
- ~~User can either directly input image path or browse for it in the local file system.~~
- ~~User can input optional Name and Information fields.~~
- ~~Displays a preview of the image specified by Link.~~
- ~~Error 1: Empty Link – Give warning: Please Enter the link/ File doesn't exist on the screen.~~
- ~~Error 2: There already exists a flashcard with the given image. – "Save and Next" disabled, "Replace" enabled.~~



~~**Music**~~

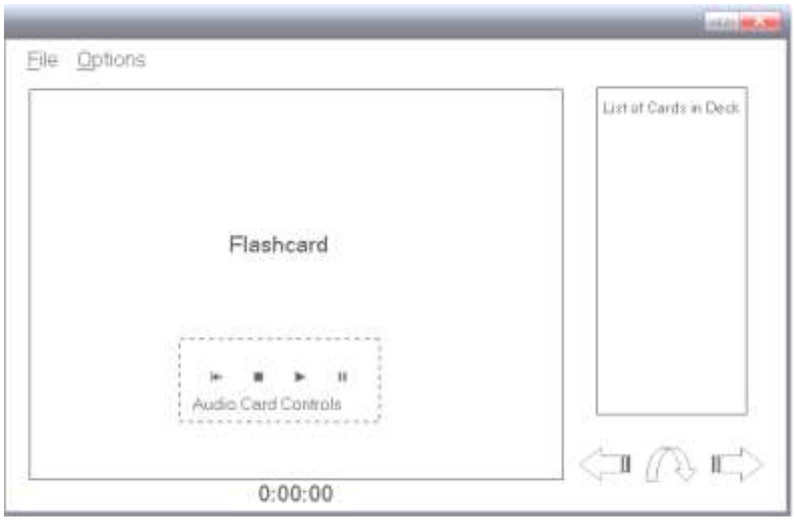
- ~~User can either directly input sound file path or browse for it in the local file system.~~
- ~~User can input optional Name and Information fields.~~
- ~~Displays a slider bar and sound control buttons to let user preview the audio file specified by Link at any position.~~
  - ~~Play – Play the sound from the current position.~~
  - ~~Pause – Stop playing the sound.~~
  - ~~Stop – Stop playing the sound and set current position to beginning of file.~~
- ~~User can use the slider bar to indicate a segment of the audio file to use for the flashcard (instead of using the entire length of the file).~~
- ~~Error 1: Empty Link – Give warning: Please Enter the link / File doesn't exist on the screen.~~
- ~~Error 2: There already exists a flashcard with the given audio file. – "Save and Next" disabled, "Replace" enabled.~~

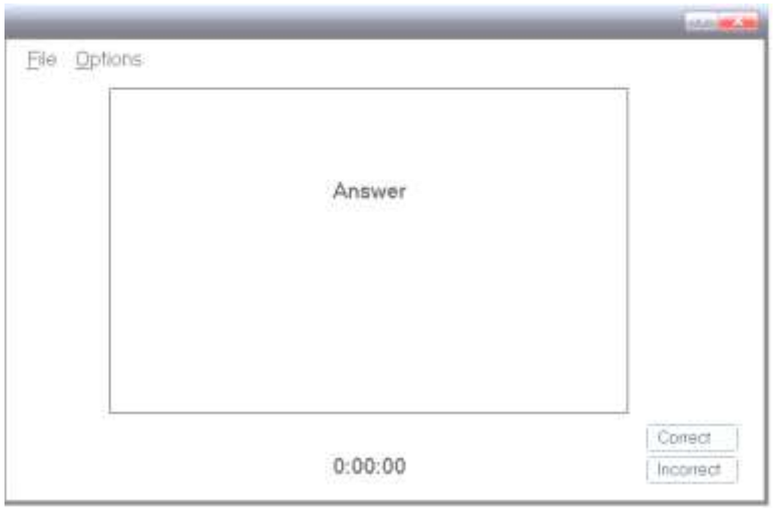
## View / Quiz Setup screen



- User can choose deck(s) as the context of the Quiz or Review
- User can choose the method of the Quiz, number of questions in the Quiz (default is 10), or select View History to view a record of previous quiz results.

## Viewer



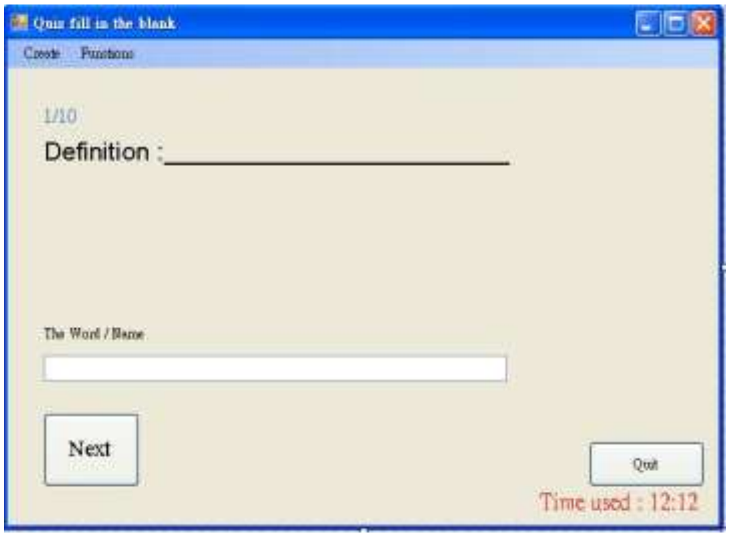


**Flashcard Viewing**

- Displays front and back of flashcard.
- In the case of audio flashcards, display sound control buttons Play, Pause, and Stop.
- Show timer to keep track of time elapsed.
- Show list of flashcards in current collection.
  - Clicking on a flashcard in the list jumps to that flashcard.
- Options (each option has a corresponding keyboard key):
  - Previous flashcard.
  - Skip to next flashcard (without flipping).
  - When viewing front side: flip to next side of flashcard
  - When viewing back side: indicate whether the user got it correct or not.
  - Exit.

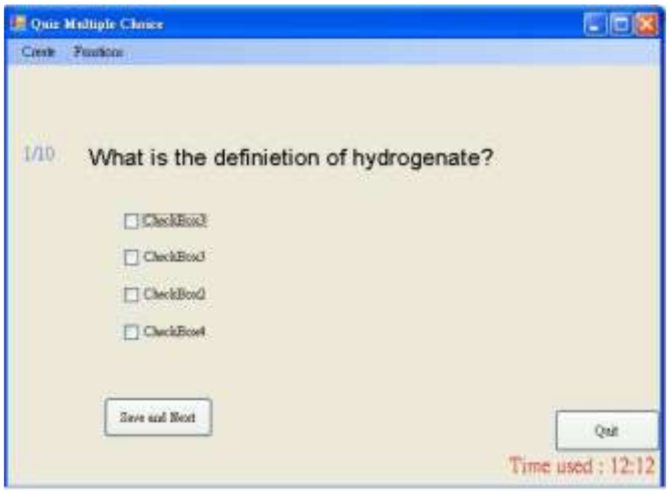
**Quiz Types**

- **Fill in the blank**
  Given an image / music / definition, the user have to type in the word that has this definition / the name of this picture / the name of this music.



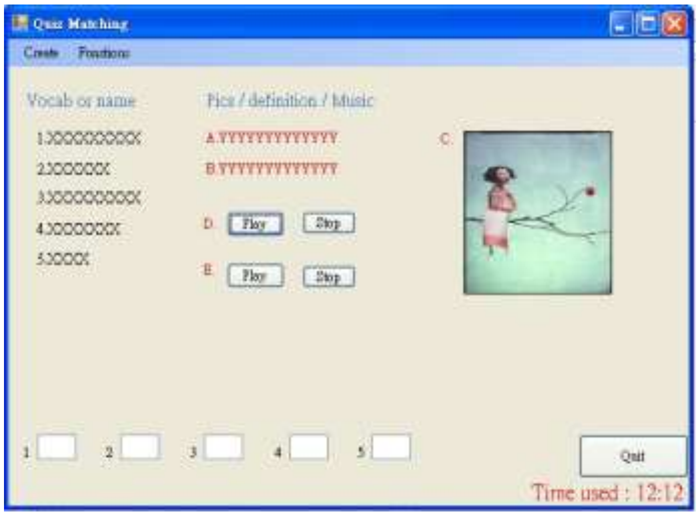
**Multiple choice**

Given an image / music / vocabulary, the user will be asked to choose its name / definition out of the four given choices. The choices will be randomly selected from the same deck of the right answer.

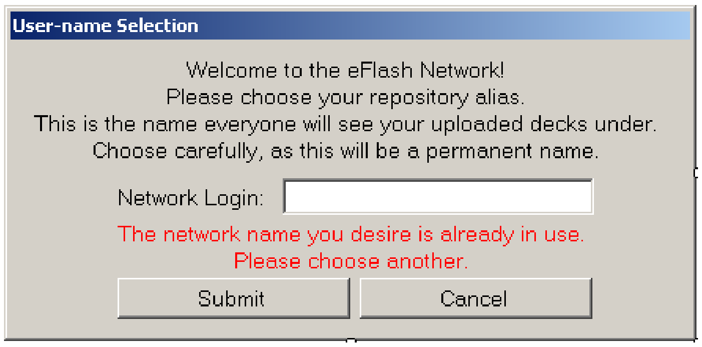


**Matching**

There are two columns in the screen. On the left hand side is the front of the card, and the definition or music or picture will be displayed on the right column. Users match each item on the left column with a picture/music/definition from the right column and put the answer in the bottom answer box.



## Network Opening Screen



Welcome to the eFlash Network!
Please choose your repository alias.
This is the name everyone will see your uploaded decks under.
Choose carefully, as this will be a permanent name.

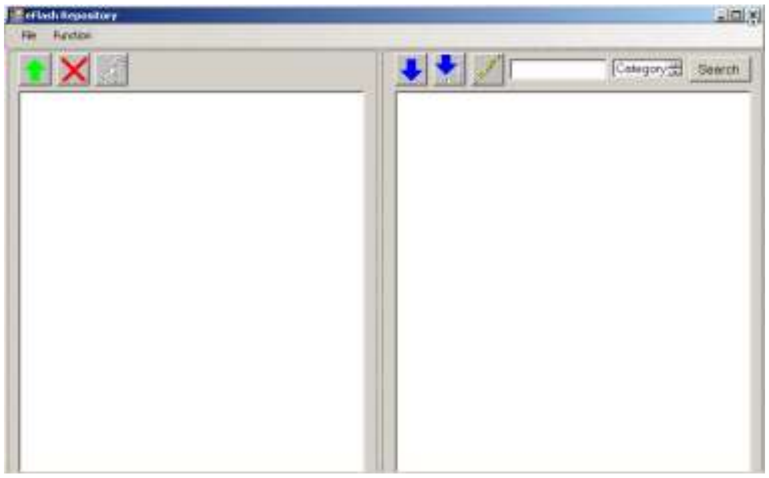
Network Login:

The network name you desire is already in use.
Please choose another.

(This screen has been revamped)

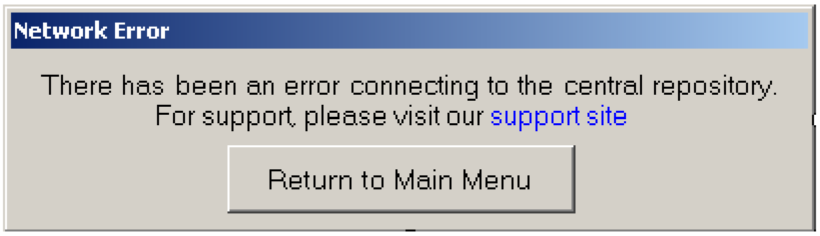
- Displays welcome to eFlash networking options
- Only shown if the current user profile does not have an associated network login name.
- ~~Prompts user for a network login name.~~
- ~~Submit~~
  - ~~Contact server database. Make sure requested login name is unique.~~
    - ~~If unique: proceed to network browser.~~
    - ~~If not unique: display error message and prompt user again.~~
- Exit
  - Return to Main Menu Screen

## eFlash Network Uploader/Downloader



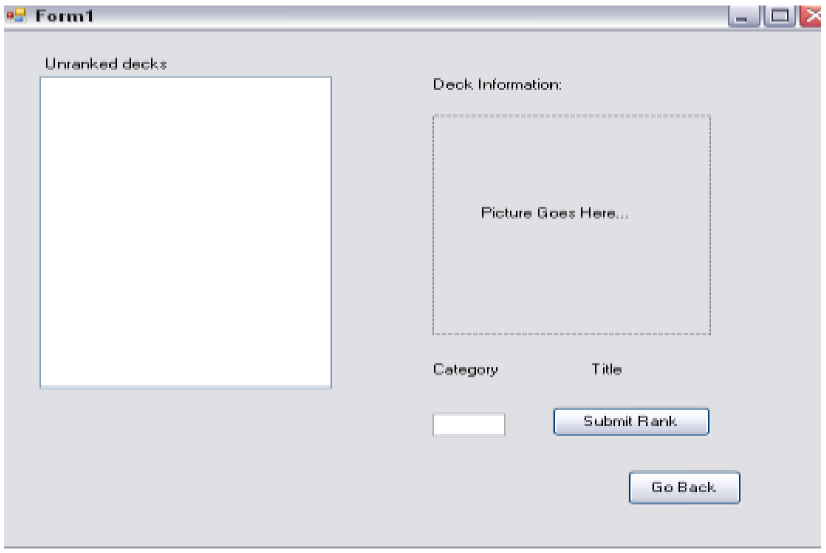
(This screen has been revamped)

- Displays two file-hierarchy panels
- Left panel shows local flashcards and upload-related options.
  - Upload
    - Upload selected file if new file.
    - Upload selected file (overwrites) if previously uploaded.
    - Error: Cannot connect to server.
  - Delete
    - Remove uploaded file from server.
    - Error: Cannot connect to server.
  - View ratings
    - Expand description window below file name in hierarchy listing – shows number of downloads, number of raters, and average rating
    - Error: Cannot connect to server.
- Right panel shows server flashcards and download-related options.
  - Download
    - Downloads selected file, updates view (appears in Downloaded and not yet rated category).
    - Error: Cannot connect to server.
  - Download and view
    - Downloads selected file, transitions to deck viewer.
    - Error: Cannot connect to server.
  - Search
    - Searches server for keyword, displays all results in categorical hierarchy in browse hierarchy.
    - Error: Cannot connect to server.
  - Return to browsing
    - Reverts right panel back to browse deck hierarchy.
    - Error: Cannot connect to server.
- Shows error message popup if encounters connection problems.


Network Error

There has been an error connecting to the central repository.
For support, please visit our support site

**eFlash Network Ranker**

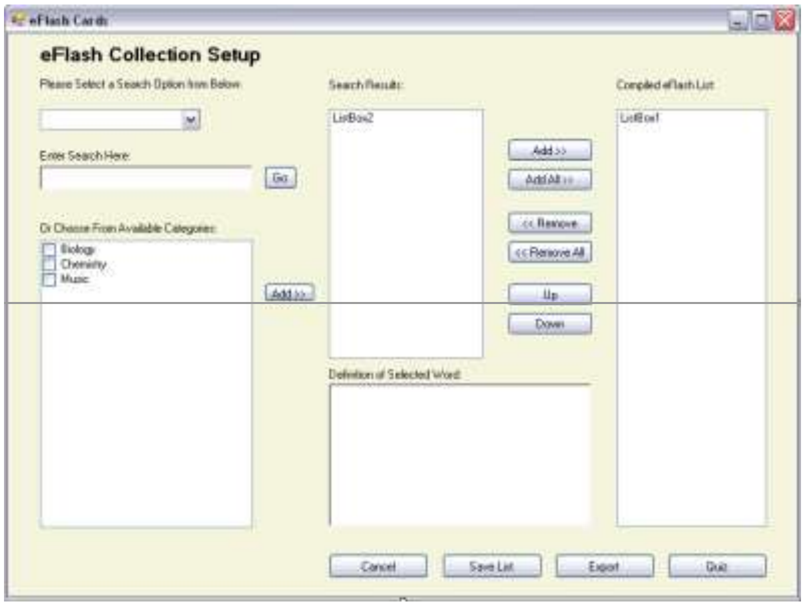


(This screen has been revamped)

- Divided up into a left and right section.
- Left panel shows local decks that were downloaded and unranked.
  - Preview
    - Download preview and display information about deck.
    - Error: Cannot connect to server.
  - Submit Rank
    - Sends insertion rank query to remote database
      Calculates network user's new average ranking

~~**Collection Manager**~~

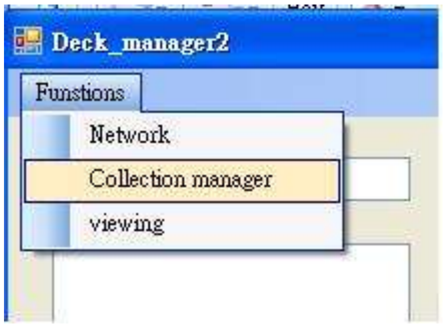
~~The Collection Manager will be used to manage the collection of the decks. Collection can not be directly accessed though the start menu, but it can be accessed thought the drop down menu "functions".~~



~~Collection management~~

- ~~Display:~~
  - ~~List of existing collections.~~
  - ~~Hierarchical list of categories and decks.~~
  - ~~Hierarchical list of search result decks and their flashcards.~~
  - ~~Definition/image/sound of the selected flashcard.~~
  - ~~List of flashcards in the current collection.~~
- ~~Deck search~~
  - ~~Searches all decks and displays the results in the appropriate list.~~
- ~~Options:~~
  - ~~Run search.~~
  - ~~Select flashcard to display its contents in the preview box.~~
  - ~~Add selected flashcard to collection.~~
  - ~~Remove selected flashcard from collection.~~
  - ~~Save the current collection.~~
  - ~~Begin quiz on the current collection.~~

With the removal of the Collection Manager, all relevant functionality will be absorbed into the Deck Manager. See the APIs section for more information.



**The drop menu**

The drop down menu exists for most of screens; it allows the user to switch from one process to another process easily and quickly whenever they'd like (barring error pop-ups or other dialogue / prompt pop-ups).

## Data Input and Output Interface

The five areas of data transfers into and out of the application layer are importing from file, exporting to file, printing, communication with the local database, and communication with the online database.

### *Importing from file*

The importer function in eFlash can import text based word-definition pairs and images and sounds. For images and sounds, the importer function expects that a valid filename of a picture or sound is given. It then creates a copy of that file in the eFlash program directory and saves the reference to that file in the database. The importer function can only work properly if the user correctly defines the input file. See the GUI section for more details on how the importer wizard works.

### *Exporting to file*

We support exporting to Power Point presentation format (*.ppt). This user-driven requirement is accomplished through use of the Power Point API to construct card slides one by one then save the generated presentation file. The details of the Power Point API are discussed in the section on other interfaces.

### *Printing*

Because interfacing directly with the printer is too time consuming to implement directly, and since this feature is only requested by a small niche of our customer base, we will utilize another standard issue program for Windows operating systems. This decision is detailed in the design decision section below.

### *Local Database Communication*

All local database communication is performed through SQL queries generated in the application layer. The following is a list and description of the tables in the local database. Note that each ID is a non-null, auto-incrementing sequence beginning at 1.

| **Users** | | | |
|---|---|---|---|
| uid | sequence | primary key, auto-increment | User ID |
| name | varchar(20) | not null | User Name |
| pw | varchar(15) | - | User Password |
| login_setting | enum | ‘no_save’,’save_pw’,’auto_log’ not null | Login Settings |
| nuid | integer | references NetworkUsers | associated Network User ID |

Notes:

- nuid automatically assigned if internet connection present, automatically sets nname to name if name is unique in NetworkUsers
- auto_login implies save_pw state is also checked

| **Cards** | | | |
|---|---|---|---|
| cid | sequence | primary key, auto-increment | Card ID |
| tag | tinytext | - | topic tags (fruit, nouns, rock) comma delimited topics |
| uid | integer | references Users, not null | owning User ID |

Notes:

- cards from online repository have downloader set as creator
- cards shared between users on the local machine are copied only when changes are made (edited cards, but edited by non-owner)
    - this is a deep copy (references the Objects table as well)

| **Decks** | | | |
|---|---|---|---|
| did | sequence | primary key, auto-increment | Deck ID |
| type | enum | ‘noQuiz’,’text’,’image’,’sound’ not null | Deck Template |
| cat | tinytext | not null | category name |
| subcat | tinytext | - | sub-category name |
| title | tinytext | not null | deck title |
| uid | integer | references Users, not null | owning User ID |
| nuid | integer | references NetworkUsers | Network User ID of uploader |

Notes:

- uid may be owner of view only, not actual cards in view
- nuid allows deck ranking
- type means that every card in that deck will follow a certain deck template
- cascading delete (remove all relevant cards/objects when a deck is deleted)

| **CDRelations** | | | |
|---|---|---|---|
| did | integer | references Decks, not null | Deck ID |
| cid | integer | references Cards, not null | Card ID |
| superkey on (did, cid) | | | |

Notes:

- organizes cards into decks (or views, i.e. collections of cards)

| **Objects** | | | |
|---|---|---|---|
| cid | integer | references Cards, not null | Card ID |
| side | integer | constrained >= 1, not null | side # (for N-sided cards) |
| type | enum | { text, image, sound } | type of object |
| x1 | integer | >= 0, <= 100, not null | x-position of object start |

| | | | |
|---|---|---|---|
| x2 | integer | >= 0, <= 100, not null | x-position of object end |
| y1 | integer | >= 0, <= 100, not null | y-position of object start |
| y2 | integer | >= 0, <= 100, not null | y-position of object end |
| data | text | not null | see notes |
| superkey on (did, cid, x1, y1, x2, y2) | | | |

Notes:

- x, y placement specified in terms of percentage of card space to avoid conflicts from differing resolutions (specified in terms of a percentage of screen space, from 0-100)
- all font formatting entries are stored within data in a rich text format, if the object type is text..
- data will be the word, if type is text – else data will be a path (C:\...) to where the image/sound of interest is stored
    - used to be BLOBs, but nixed based on performance issues
    - the path will be within the eFlash working directory (to prevent users from tampering with it)

| **QDRelations** | | | |
|---|---|---|---|
| qid | integer | references Quizzes | Quiz ID |
| did | integer | references Decks, not null | Deck ID |
| superkey on (qid,did) | | | |

Notes:

- holds relations for which quiz uses which deck

| **Unranked** | | | |
|---|---|---|---|
| ldid | Integer | Not null, references Decks.did | Local Deck ID |
| rdid | Integer | Not null | Remote Deck ID |
| lnuid | Integer | Not null, references Users.nuid | Local Owner Network User ID |
| rnuid | integer | Not null, | Remote Owner Network User ID |
| Primary key on did, btree on lnuid | | | |

Notes:

- keeps track of decks that have not been ranked
- to keep updated, must add entry into this table whenever a deck is downloaded
- when a remote deck is ranked, the corresponding entry in this table is deleted
- if trying to rank a deleted remote deck, results in error

| **Statistics** | | | |
|---|---|---|---|
| qid | sequence | primary key, auto-increment | Quiz ID |
| uid | integer | references Users, not null | owning User ID |

| | | | |
|---|---|---|---|
| type | enum | { matching, fill-in, multiple-choice } | quiz type |
| score | integer | - | Score on quiz out of 100% |

Notes:

- keeps track of general statistics for every quiz per user, as well as the specific quiz type

*Online Database Communication*

All online repository communication is performed through remote server calls via remote server calls. The base unit of transfer for repository data transfer is the deck. The following is a list of the repository database tables, data types, and description of properties. Note that each ID is a non-null, auto-incrementing sequence beginning at 1.

| **NetworkUsers** | | | |
|---|---|---|---|
| nuid | sequence | primary key, auto-increment | Network User ID |
| nname | varchar(20) | not null | Network Alias |
| join_d | date | not null | date nname created |
| num_up | integer | default 0 | number uploaded |
| num_down | integer | default 0 | number downloaded |

Notes:

- nname default set to profile name if unique; prompted for name if non-unique upon first connecting to network
- no password necessary – required association with local profile, which has optional password

| **NetworkDecks** | | | |
|---|---|---|---|
| ndid | sequence | primary key, auto-increment | Deck ID |
| cat | tinytext | not null | category name |
| subcat | tinytext | - | sub-category name |
| title | tinytext | not null | deck title |
| date | date | not null | date deck last uploaded or modified |
| rat | float | 0 <= rat <= 5 | average user ranking |
| num_v | integer | default 0 | number of votes for this deck |
| nuid | integer | references NetworkUsers, not null | owning User ID |
| deck | BLOB | not null | list of commands for creating deck in local DB, as well as any binary attachments |
| size | Integer | not null | size of deck used for parsing |
| preview | BLOB | not null | binary image of 1st card |

| | | | |
|---|---|---|---|
| | | | in deck |
| psize | Integer | not null | size of preview used for parsing |

Notes:

- the base unit for network transfer is deck to avoid incredibly large object tables

| **NetworkRatings** | | | |
|---|---|---|---|
| ndid | Integer | not null, references NetworkUsers.ndid | Quiz ID |
| nuid | Integer | not null, references NetworkUsers.nuid | Network User ID |
| rank | Integer | 1,2,3,4,5, not null | Score given by Network User ID |
| Primary key (ndid,nuid), btree on ndid | | | |

Notes:

- stores ranks from network users.  Take average for a deck to compute deck rank.

## APIs

### Profile Manager --- manages local profile information

**Method**: localDBCreateProfile()
**Input**: *user_name* (String), *password* (String)
**Output**: Boolean. Returns true if the method is successful, false otherwise.
**Description**: This method creates a new profile record in the **Users** database (refer to Users Table Schema in the *Local Database Communication* section of the document). The profile record's fields include the User ID (**uid**), User Name (**name**), User Password (**pw**), the associated Network User ID (**nuid**), and saved profile information (an enumerated type keeping track of saved password and auto-login). The **name** field, which must contain unique values, will include the *user_name* input, and the **pw** field will include the *password* input. The password input may be an empty string since the user is not required to give a password. The **nuid** field does not get set at this stage. It will be assigned a value when the user chooses to log on to the internet. The **Networking** Component will be responsible for setting this field.
**Invariants**: Inputs: *user_name* and *password* must both be String objects. The *user_name* input must not be an empty string; however the *password* input can be an empty string. Outputs: The function will return true if the method created the profile properly, otherwise it will return false.

**Method:** deleteLocalDBProfile()
**Inputs:** *user_name* (String)
**Outputs:** Boolean, output true of false depending on whether the method was successful
**Description:** This function will delete the record based on the input given. It will delete all records from the other databases that have the same unique **uid.** These databases include Users, Cards, Decks, Statistics, Quizzes and the Online Databases.
**Invariants:** All profile names in the Users database will be unique. The input must be a String object and the output returned must be a Boolean.

**Method:** editLocalDBProfile()
**Inputs:** *user_name* (String), *new_user_name* (String), *new_password* (String)
**Outputs:** Boolean
**Description:** Retrieve the original record from the Users database, and change either the password or the user name or both. This depends on whether the *new_user_name* and *new_password* inputs are empty strings or not. If they are not empty strings then update the profile record. Return true if no error occurs, otherwise return false.
**Invariants:** The *user_name* input must be a non-empty String object, and the other two inputs as well must be String objects; however they can be empty. Boolean values are returned.

**Method**: loginUser()
**Inputs:** *user_name* (String), *password* (String)
**Outputs:** Boolean
**Description:** If the *password* input is a non-empty string, then authenticate the user. If the

passwords match, then set the global variable User_name to *user_name*, set User_ID to the corresponding **uid**, and return true; however if there is no such user name, or the passwords do not match then return false. If the *password* input is an empty string then set User_name to *user_name*, set User_ID to the corresponding **uid**, and return true, since there is no authentication involved.
**Invariants:** Both inputs are string objects and Boolean values are returned as outputs.

## Deck Manager ---manages the decks

**Method:** loadUserDecks()
**Inputs:** none
**Outputs:** database recordset
**Description:** This function will access the global variable User_ID and grab all deck and card records belonging to that User_ID from the Decks and Cards Databases. It will output all of the user's decks and cards onto the display screen of the eFlash Collection Setup.
**Invariants:** The User_ID must be the correct one and all names of the decks and cards must be in the proper format.

**Method:** createNewDeck()
**Inputs:** *deck_name* (String), *deck_category_name* (String), *deck_subcategory_name* (String)
**Outputs:** Boolean
**Description:** This method creates a new deck record in the **Decks** database (refer to Decks Table Schema in the *Local Database Communication* section of the document). First it will check to see if the deck name already exists in the user profile's portion of the database. If it does, this function will return false; if not, then the function will proceed to add the appropriate record into the database. The *deck_name* and *deck_category_name* must be non-empty strings since the **title** and **cat** fields are required to be non-null. If the *deck_subcategory_name* is non-empty then the **subcat** field is set, and the **uid** field is set by the global variable User_ID. Once all the fields are specified and the record is added to the database, then the function will return true. If there are any errors throughout the process, it will return false with an exception.
**Invariants:** All deck titles will be unique within the user's portion of the database, and the *deck_name* and *deck_category_name* inputs are required to be non-empty strings.

**Method:** deleteDeck()
**Inputs:** *deck_name* (String)
**Outputs:** Boolean
**Description:** This function will delete an existing deck record from the Decks database. If the deck does not exist or if any other error occurs, this function will return false. If the deck exists, its records will be deleted in the Decks and CDRelations Tables, and true will be returned.
**Invariants:** The input is a string object and the output is a Boolean value.

**Method:** createDecksFromExistingCardsWithSameTag()
**Inputs:** *new_deck_name* (String), *deck_category_name* (String), *card_TagName* (String)

**Outputs:** Boolean
**Description:** This function will create a new deck of cards that have the same Tag. For example, it might create a deck of vocabulary cards that have Food as their Tag name. It will first create the new deck by calling the createNewDeck() function, then, it will grab all the **cid**s from the Cards database that have the same Tag name and add new CDRelation records into the CDRelation database. If no errors occur, the function will return true, otherwise false.
**Invariants:** The *card_TagName* must be in the user's portion of the Cards database, and the *new_deck_name* input must contain a new, unique deck title.

### File Importer --- manages text file importing

**Method:** importFromTextFile()
**Inputs:** *text_file_location* (String), *text_file_format* (String), *char_delimiter_enum* (enum)
**Outputs:** Boolean
**Description:** This function will extract data from the text file and call the functions in the Flashcard and Deck Manager components to create decks and flashcards. The *char_delimiter_enum* input indicates what type of delimiter separates the number of fields in the flashcard. The *text_file_format* input gives information on how to build each card, based on the ordering of the now delimited sections of each line. For example, a definition from the web is "acumen n. Quickness of intellectual insight, or discernment; keenness of discrimination."[1]. The *text_file_format* input could be {front-side: word name} {back-side: part of speech} {back-side: definition}, and the *char_delimiter_enum* would indicate that it is a space that delimits the different sections (refer to the File Importer section in the Interfaces section). Based on these formats, the function will go line by line to extract the appropriate data and call the createFlashcard() function to place them in the databases. If everything is done properly, then the function will return true, otherwise it will return an exception and the value false.
**Invariants:** The most important invariant is that the text-file entries match the format provided by the user, and the text file location must be properly specified.

### File Exporter --- manages file exporting

**Method:** exportToNetworkFlashcardFormat()
**Inputs:** *deck_name* (String)
**Outputs:** BLOB object
**Description:** This function will create a BLOB object of the specified deck. The function will gather the deck_id based on the *deck_name* input given, and then gather all the cards that belong to that deck. Then it will first create the SQL query command for deck creation, and then build more SQL commands for flashcards' creation. These queries along with the BLOB data from the flashcard will be composed into another BLOB object, which the function will return if everything works out properly, otherwise an exception is thrown and false is returned.
**Invariants:** The SQL queries must be correctly formed, and the flashcards' BLOB data

[1] http://www.freevocabulary.com/

objects must not be corrupted.

**Method:** exportToPPTfile()
**Inputs:** *deck_name* (String), *ppt_file_location* (String)
**Outputs:** Boolean
**Description:** This function will create and save a powerpoint presentation based on the given deck. It will gather the deck_id based on the *deck_name* input given, and then gather all the cards that belong to that deck. Then it will create ppt slides that correspond to the data from the deck's flashcards, and save the finished ppt presentation to the specified file location. Refer to the PowerPoint presentation module under the **Other object/module Interfaces** section for code help. If everything works out properly, then the Boolean true is returned, otherwise an exception is thrown and false is returned.
**Invariants**: Proper inputs must be specified, and the correct presentation is formed.

- **FlashCard Manager ---manages flash cards**

**Method:** createVocabFlashcard()
**Inputs:** *word* (String), *definition* [] (StringArray), *deck_name* (String), *card_tag_name* (String), *Obj_X_coord* (integer), *Obj_Y_coord* (integer),
**Outputs:** enum (1 = error, 2 = card already exists, 3 = successful)
**Description:** This function creates a many-sided vocabulary card that contains as many definitions for the word as possible. This method creates a new card record in the **Cards** database (refer to Cards Table Schema in the *Local Database Communication* section of the document), as well as a new **CDRelations** record, and new **Objects** record. The function will take the information specified by the inputs, and create these appropriate records. If any of the inputs are empty objects then an error message is thrown and the number 1 is returned. If the *word* name already exists in the Cards database, then an error message is thrown detailing this problem and the number 2 is returned. If there are no errors, then the number 3 is returned.
**Invariants:** All flashcards belong to a deck; therefore *deck_name* input must be specified, and its name should be one of the existing decks in the Decks database. All inputs must be non-empty objects.

**Method:** editMusicFile() → refer to the section on the module Mp3splt in the **Other object/module Interfaces** section for implementation details.
**Inputs:**
**Outputs:**
**Description:** This function, which will be called by createMusicFlashcard(), will create the user-edited music file.
**Invariants:**

**Method:** createMusicFlashcard()
**Inputs:** *sound_location* (String), *sound_title* (String), *description* (String), *deck_name* (String), *card_tag_name* (String),
**Outputs:** enum (1 = error, 2 = card already exists, 3 = successful)
**Description:** This function, along with editMusicFile(), will create the music flashcard.

Just as a note, the *sound_location* is the file-path of the user-created sound file, which was created through the mp3splt module. This file-path name is an eFlash-program specified file location. It will take the inputs and create a new **Cards** record, **CDRelations** record, and new **Objects** record. If the sound file's location is not a valid link, or any other error occurs, an error message is thrown and the number 1 is returned. If the *sound_title* already exists in the Cards database, then an error message is thrown detailing this problem and the number 2 is returned. If there are no errors, then the number 3 is returned.
**Invariants:** The *sound_location* must be a valid path and non-empty. The *sound_title* must also be non-empty; however the other inputs are optional.

**Method:** createPictureFlashcard()
**Inputs:** *pix_location* (String), *pix_title* (String), *description* (String), *deck_name* (String), *card_tag_name* (String), *Obj_X_coord* (integer), *Obj_Y_coord* (integer),
**Outputs:** enum (1 = error, 2 = card already exists, 3 = successful)
**Description:** This function creates a picture flashcard that contains the picture on the front side and its description on the back side. It will take the inputs and create a new **Cards** record, **CDRelations** record, and new **Objects** record. If the picture's location is not a valid link, or any other error occurs, an error message is thrown and the number 1 is returned. If the *pix_title* already exists in the Cards database, then an error message is thrown detailing this problem and the number 2 is returned. If there are no errors, then the number 3 is returned.
**Invariants:** All inputs must be non-empty objects. The picture's location must be valid.

**Method:** editFlashCard()
**Inputs:** *card_name* (String), *deck_name* (String), *data_filepath* (String), *word* (String), *definition* (String Array), *card_tag_name* (String), *Obj_X_coord* (integer), *Obj_Y_coord* (integer),
**Outputs:** Boolean
**Description:** Except for the *card_name* input, if any of the resulting fields are non-empty, then this function will take the new (non-empty) fields and substitute them for the older attributes of the card. If no errors occur, this function returns true, otherwise it returns false.
**Invariants:** The *card_name* input must be non-empty; however all the other inputs are not required to be non-empty. However, the function must not substitute non-empty fields for empty attributes.

**Method:** deleteFlashCard()
**Inputs:** *card_name* (String)
**Outputs:** Boolean
**Description:** This function deletes all entries containing references to this card that are within the user's portion of the database. If no errors occur, this function returns true, otherwise it returns false.
**Invariants:** The *card_name* must be a non-empty field, and must contain a valid name. The function only deletes the user's copies of the flashcard, not the other user's copies.

**Method:** getFlashCard()
**Inputs:** *card_name* (String)

**Outputs:** database recordset
**Description:** This function retrieves the card's object info from the user's portion of the database.
**Invariants:** The *card_name* must be a non-empty field, and must contain a valid name.

**Method:** addCardToDeck()
**Inputs:** *card_name* (String), *deck_name* (String)
**Outputs:** Boolean
**Description:** This function adds an already existing card to the specified deck by adding a **CDRelations** record to the database. If no errors occur, this function returns true, otherwise it returns false.
**Invariants:** Both input fields must be non-empty.

**Method:** removeCardFromDeck()
**Inputs:** *card_name* (String), *deck_name* (String)
**Outputs:** Boolean
**Description:** This function deletes the card from the specified deck by deleting the **CDRelations** record from the database. If no errors occur, this function returns true, otherwise it returns false.
**Invariants:** Both input fields must be non-empty.

**DBAccess --- abstraction of SQL queries.**

**Method:** DBAccessGet()
**Inputs:** *search_string* (String), *Table_Name* (String), *LocalOrOnlineDB* (Boolean)
**Outputs:** database recordset
**Description:** This function will retrieve the appropriate set of records requested from the specified database.

**Method:** DBAccessPut()
**Inputs:** database recordset, *Table_Name* (String), *LocalOrOnlineDB* (Boolean)
**Outputs:** Boolean
**Description:** This function will correctly put the appropriate records into the specified database. If no errors occur, this function returns true, otherwise it returns false.

**ViewerAndQuizzer --- sets up viewing and quizzing on decks**

| | |
|---|---|
| Private correct: int | |
| Private current_CID: int | |
| Private total_time: time | |
| Private type: int | |
| Private no_ofQuestion: int | |
| **Method** | **Description** |
| loadCard(cid:int []): void | Retrieves all card information |
| next(): void | Load next flash card |
| prev(): void | Load the previous flash card |

| | |
|---|---|
| flip(): void | Flip the flash card |
| shuffle(did: int): int[] | Returns shuffled cid ordering for deck |
| quiz(noOfQuestion: int, type: int, did: int): void | Create the quiz with given deck or compilation, number of questions and type of the quiz. Data of result will save directly into the statistics table of the database. |

**Network --- handle all the functions about networks.**

| Method | Description |
|---|---|
| connect(login: String, password: String): boolean | Establish Connection to the server |
| upload(arrayOfDeck: int []): void | Upload decks to the server |
| download(arrayOfDeck: int []): void | Download decks to the local disk |
| search(deck: search) | Search for a deck on the server |
| rateDeck(deck: int) | Rate selected deck |

**Other Object/Module Interfaces**

There are two external interfaces in this project.

**Mp3Splt (support for music editing)**

Mp3Splt is a command line utility to split mp3 and ogg files selecting a begin and an end time position, without decoding. It's very useful to split large mp3/ogg to make smaller files or to split entire albums to obtain original tracks.

Splitting the Mp3 will be done via an external application called mp3splt. Mp3splt can run from the command line, using the format "mp3splt FILE... [BEGIN_TIME] [END_TIME...]"
where the begin time and time are in the format "minutes.seconds." In C#, applications can be run with command line arguments through the ProcessStartInfo Class. For instance,

```
    using System;
    using System.Diagnostics;
    using System.ComponentModel;
```

mp3splt only provides command line options for splitting file. Since C# does not natviely support mp3 playback commands, we will have to use MCI from the winmm.dll library, which looks something like the following:

```
public void Open(string sFileName)
        {       Pcommand = "open \"" + sFileName + "\" type mp3 alias
MediaFile";
                mciSendString(Pcommand, null, 0, IntPtr.Zero);
                isOpen = true;}
```

```
public void Play()
        {           PCommand = "play MediaFile";
                    mciSendString(Pcommand, null, 0, IntPtr.Zero);
        }
```

Users input start and end times in the GUI, which are passed as arguments to Player.Play(start,end), or if the user chooses to cut the file, it invokes mp3splt with the approprite information.

**Office PIA (support for PowerPoint export)**

In order to automate a PowerPoint presentation through Microsoft Visual Studio, we have to first install a package called Primary Interop Assemblies (PIA). Also, we have to include appropriate library like:

```
using Microsoft.Office.Core;
using PowerPoint = Microsoft.Office.Interop.PowerPoint;
using Graph = Microsoft.Office.Interop.Graph;
using System.Runtime.InteropServices;
```

into our source code. Then, we can program simply calling the Objects from the libraries like:

```
Set newPres = Presentations.Add(True)
newPres.Slides.Add 1, 1
newPres.SaveAs "Sample"
```

(for creating a new presentation, adds a slide to the presentation, and then saves the presentation.)

Here are some references from MSDN:

The library reference for PowerPoint -
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbapp10/html/pptocObjectModelApplication.asp

The URL to download PIA -
http://www.microsoft.com/downloads/details.aspx?familyid=3c9a983a-ac14-4125-8ba0-d36d67e0f4ad&displaylang=en

The URL showing how to create PowerPoint automation through an example –
http://support.microsoft.com/default.aspx?scid=kb;EN-US;303718

## Design Decisions

In this section, we detail the major design decisions that were made in choosing an implementation.

## Approaches

- **Database tables** were an integral part of developing eFlash, and we have listed the schema earlier in the document. The network access method is essentially SQL through RPC, where the database tables on the eFlash server will closely mimic the local data store. These details, again, are listed in previous sections.
- In adding support for **audio flash** cards, we will allow the user to choose snippets of a sound file, rather than the whole sound itself. Splitting mp3s will be done via an external application called mp3splt.  Mp3split can run from the command line, and given arguments such as the begin time and end time, and run in C# through the ProcessStartInfo Class.
- **Installation** is easily accomplished in Visual Studio by creating a Setup and Development project, which installs other projects. To install MySQL, which will provide the infrastructure for our database, we intend to make a custom installer class and have it run the MySQL setup file, which we will include in our installation package.
- To allow users to create **printable flash cards**, we will save development time by not writing our own PostScript interface, and instead piggyback over another application that is capable of printing on its own (for instance, executing a DDE call to the system web browser to print an image).
- In order to automate the creation of a **PowerPoint presentation**, we will compile the Microsoft Office Primary Interop Assemblies (PIA) into our project, which provides an interface in C# for easily laying out a slide deck to the user's specifications.
- The types of **learning quizzes** we will support in eFlash are matching, multiple choice, and fill in the blank. Answers to these quizzes will be compiled into statistics that can compile future quizzes to test the user on their weakest areas, allowing them to excel on the subject as a whole. Note that the self-test quiz mode is not included with this list.

## Technology

We will be using Visual C# to develop our application code. In particular, C# affords us many of the advantages of a modern programming language, flexibility for team members used to programming in iterative styles, and a wealth of reusable components (the Visual part) to speed up application development. We are excited to use Visual Studio 2005 as our primary IDE, whose award-winnings tools have been shown to improve the success of projects in the past.

**Resource Usage**

*Local:*

Storage:

- -Installation: 50 MB Database, 10 MB Application = 60 MB Total
- -Additional Storage Dependent on # and Type of Flash Cards uploaded per user
  - Text Flash Card = 5 KB per card
  - Picture Flash Card = 1 MB per card
  - Music Flash Card = 2 MB per card
- -Average additional storage:
  - -Assumptions:
    - -3 users on average per computer
    - -250 flash cards on average created per user
    - -Text based flash card most popular
    - -Average size flash card: 800 KB
  - -Total: 750* 800KB = 600000 = 600 MB

Memory:

- -Database: 10 MB real memory, 215 MB virtual memory
- -Application: 5 MB real memory

Computation Power on 750 MHz or greater CPU:

- -Inserts and modifications to tables within databases very infrequent, thus not counted
- -Textual input: Less than 2% of CPU Power.
- -Changing between Graphical Interfaces: Less than 13% of CPU Power, but only a quick spike in CPU power noticed

*Network:*

Storage:

- -Installation: 50 MB Database
- -Additional Storage Dependent on # and Type of Flash Cards uploaded per user
- -Average additional storage:
  - -Assumptions:
    - -10,000 users total
    - -100 flash cards uploaded on average per user
    - -Text based flash card most popular
    - -Average size flash card: 800KB
  - -Total: 1,000,000 * 800KB = 800,000,000 KB = 800 GB

Memory:

- Database: 10 MB
- -Dependent on # of users browsing or altering the database.
- -At least 2GB of memory should be dedicated for quick returns of previously stored queries, due to large deck tables

Computation Power:

- -Depends on the number of users connected.
- -Possible up to 100% CPU usage if all 10,000 users connected at the same time

## Error Handling and Error Recovery

- If a user attempts to create a flashcard with a name that is already pre-existing in their profile of owned cards, to prevent errors, cards are handled by their card id.
- If a user opens a pre-existing flash card owned by somebody else and attempts to rewrite or edit the flashcard in the eFlash Creator, to create protection between users and limiting error messages, a duplicate of the card will automatically be made and stored under the current user's profile.
- To prevent multiple users from having the same username on the local client, an error message will appear. Then the user must choose a new user name.
- To keep multiple users unique in the eFlash network, a unique network id will also be assigned from the eFlash network when a username is created on the local client, given that there is an Internet connection. If there is no Internet connection, a unique network id will not be assigned until one attempts to connect to the eFlash network through the local application with an Internet connection. Note: Authentication to the eFlash network does not require that the user name be unique from other eFlash members, because usage of the unique network id in combination to the user name solves this problem.
- If a user attempts to load a deck to the eFlash network and the deck name already exists, to avoid an error, there will not be an error because the deck is differentiated by the deck id. ~~the user will be prompted with a pop-up error message "Deck name already in use on eFlash network." The user can then either "Replace current deck with new deck" or "Change name of conflicting deck on network" or "Change name of conflicting deck on local client" or "Cancel transaction".~~
- To prevent a user on the local client from importing decks into another local users profile, a user is only allowed to import to his/her own profile.
- If a user attempts to import a deck from the eFlash network to the local client, and a deck with that name already exists, to avoid an error, decks are differentiated by deck id, and assigned a new one during import. ~~, the user will be prompted with a pop-up error message "Deck name already in use on local client." The user can then either "Replace current deck with new deck" or "Change name of conflicting deck to be imported" or "Change name of conflicting deck on local client" or "Cancel transaction".~~

## Coding Standards

.NET is the environment framework of choice due to its ease of deployment on Windows based systems. It is to be used to create the Graphical User Interface portion of the project. The design decisions of the framework that will be used, is as outlined in the following Microsoft Corporation document.

http://msdn.microsoft.com/archive/default.asp?url=/archive/en-us/passport25/serviceguide/designui/designui.asp

The Visual C# programming language will be used to create functions that interact with buttons on the interface. While there are many styles for programming, we will adopt the programming standard for VC# as specified by Microsoft Corporation. Below is a sample reference for VB, to which many of the same rules apply to VC#:

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/csref/html/vcoriCProgrammersReference.asp

MySQL is the database language that will be used to query both the local and network databases. Information on the guidelines for syntax and standards can be found at the MySQL link below:

http://dev.mysql.com/doc/refman/5.0/en/

In addition to following the programming and design conventions mentioned above, each programmer will also be responsible for documentation of all functions, and comments for complex conditional statements. Modularity will be a main focus for our design goal.

**Issues**

*Open Issues*

- None.

*Resolved Issues*

- Major design decisions – see design decisions section above.
- Screens and tables updated to reflect latest design.
- References to .efa file and collection manager have been removed.
- No (function-based) communication between Creator and Viewer.
- Explicitly note that the client is to be as thin as possible.

Justification and additional changes are available in meeting transcripts on our team site.

**Alternative Implementations**

- Rather than using a database for local storage, a local file system could have been used with pointers held in text files. However, due to the complex tasks required, ie statistical analysis, “sharing” of flashcards, multiple users, searching, etc. we decided to use a database to simplify those tasks.
- In order to export and import cards to and from the eFlash network, creating a special file type was considered. However this ended up complicated the task in that it would require learning how to register a file type with different operating systems, and storing and extracting information from the file type. The easier solution and what is going to be implemented is sending queries across the network to populate the database.
- Rather than having the eFlash network connector built into the eFlash application, we considered having a website with a built in eFlash network connector that allowed a user to browse the eFlash network from any computer. However this also complicated measures in that we would have to return back to using a file type for importing and exporting. Also, this would make it more difficult for the user in that he/she would have to go to a separate location in order to browse the eFlash network.

## Checklist

The following checklist is provided to help the reviewers (and author) prepare for the review by providing a set of questions for the reviewer to answer about the specification document. If the answer to any question is "no", that item should be identified as an issue at the review. The checklist is only a guideline; it should not be solely relied upon for a complete review. Reviewers may want to add their own questions to the checklist.

| Y | N | CONTENT |
|---|---|---|
| ___ | ___ | Has the Functional Specification been reviewed and accepted by the appropriate reviewers? (This is a prerequisite). |
| ___ | ___ | Will the specified implementation correctly and efficiently support all of the features defined in the Functional Specification? |
| ___ | ___ | Is the specified implementation maintainable? |
| ___ | ___ | Is the implementation set up so that it can be easily enhanced for projected future requirements? |
| ___ | ___ | If there is risk associated with adding this product / feature, does the specification describe how the product / feature can be disabled? |
| ___ | ___ | Are you satisfied with all parts of the document? |
| ___ | ___ | Do you believe all parts are possible to implement? |
| ___ | ___ | Are all parts of the document in agreement with the product requirements? |
| ___ | ___ | Is each part of the document in agreement with all other parts? |
| | | COMPLETENESS |
| ___ | ___ | Have alternative implementations been considered (and documented)? |
| ___ | ___ | Has the impact of the specified implementation on existing code / tools been considered? |
| ___ | ___ | Are the limitations of the specified implementation sufficiently documented? |
| ___ | ___ | Are dependencies and assumptions thoroughly documented? do these section appear to be complete? |
| ___ | ___ | Where information isn't available before review, are the areas of incompleteness specified? |
| ___ | ___ | Are all potential changes to the Implementation Specification specified, including the likelihood of each change? |
| ___ | ___ | Are all sections from the document template included? |
| ___ | ___ | Are you willing to accept the specification with the items in the open issues section unresolved? |
| | | CLARITY |
| ___ | ___ | Is the control flow and the data flow clear? |
| ___ | ___ | Are all items clear and not ambiguous? (Minor document readability issues should be handled off-line, not in the review, e.g. spelling, grammar, and organization). |