**Chronos Project: *Quest Master* and *Adventurer***

**Database Sharing Architecture**

This document explains how the various database files are used and shared by the *Quest Master* and *Adventurer* programs. It explains the data initialization and singleton properties of the database registries and other database files. This document also explains the special significance of how Registry.getInstance(boolean) works.

**Database Requirements**

1. *Adventurer* requires a repository of data from which to draw on when it plays. This data is updated during play, so the database must have read/write access.
2. *Adventurer* cannot corrupt the original repository during play because another copy of the default (original) data must be available for playing the same adventure again later. Therefore, Adventurer must use a copy of the original files.
3. *Quest Master* must be able initialize the repositories so that a minimum set of default data is available before a user defines any new data for those repositories. This requires write only access to the files. Therefore, the initial set of data must be hard-coded to meet the minimum playing requirements for *Adventurer*.
4. *Quest Master* users must be able to define new data and update the initial repositories to expand the data available to *Adventurer*. Therefore, through the *Quest Master* GUI, new data must be added through read/write access.

**Registry Implementation**

The database will be managed by *db4o*, an open source, object-oriented database made by Versant. All data repositories are persistence versions of the Registry class. The following explains the various registry files and its file name to be shared

1. Items.reg: All “things” in the game, such as weapons, armor, provisions, etc.
2. NPCs.reg: Non-player characters with which the player Heroes interact. Each building has a “building master”, that is, an owner, manager, or guild master, that is an NPC. Sometimes NPCs are met in the Arena as friends or opponents.
3. Buildings.reg: All buildings that can be in a particular town. Initial buildings are the four Guilds, one each for Fighters, Wizards, Clerics, and Rogues; also an Inn, General Store, Bank, and Jail. Buildings requires NPC and Items data.
4. Skills.reg: All the Skills that a Hero may have, either by having a particular occupation, or because of the Hero’s race and/or Guild membership.
5. Occupations.reg: All the occupations that a Hero may have when they start as a peasant. Each occupation has an associated skill that the Hero attains with that occupation.
6. Towns.twn: A collection of the towns, each containing a list of building names and the arena name. During *Adventurer* initialization, the names are converted to the objects themselves for play. A town must have at least the Inn to serve as a home base (for food, information, and lodging) and an Arena, the questing exploration area that defines the adventure. The initial town is named BiljurBaz.twn, has all nine buildings and the arena *Quasqueton*.
7. Arena.dgn: One arena per .dgn file. These are large files—a collection of Rooms that comprise the map of the adventure through which the Hero explores. Rooms contains doors, NPC, and Items. These items are treasures, traps, puzzles, and other parts of the arena. An Arena requires NPC and Items data. The initial Arena is held in Quasqueton.dgn

The Quest Master user can define or update new elements of each of these registries, defines the Town and Arena objects. Although the user can update a hard-coded Registry element, it will return to its original status if the Registies are ever re-initialized again.

**Registry,getInstance(bookean)**

This method must be written in the derived classes of the abstract Registry class:

static public <Registry subclass> getInstance(boolean createFlag)

to meet the following requirements:

* If the Registry does not exist and createFlag is true, then the subclass of Registry is created.
* If the Registry does not exist and createFlag is false, then the subclass of Registry is not intended expected to be created, so the method returns null.
* If the Registry exists and is open, the createFlag is irrelevant, then the reference to the Registry is returned to the caller.
* If the Registry exists and is closed, the createFlag is irrelevant, then the Registry subclass is opened, and the reference is returned to the caller.

These rules are summarized in the table below.

**Table 1. Registry state decision table**

|  |  |  |  |
| --- | --- | --- | --- |
| **EXISTS** | **OPEN** | **createFlag** | **ACTION** |
| T | T | n/a | Return reference |
| T | F | n/a | Open database |
| F | n/a | T | Create database |
| F | n/a | F | Return null error |

Figure 1 shows how these relationships are initialized and maintained.

1. Quest Master creates the five original registries needed through hard-coding needed for the first adventure: Bil’jurBaz containing Quasqueton and nine buildings (read-only access).
2. During play, the Quest Master user defines, updates, or deletes elements of the registries through the GUI (read/write access). If a registry file is deleted from some reason, all data saved from the GUI will be lost. Until all the registries needed to support the default adventure is complete, it is recommended that all data is hard-coded. In case the registries are lost, then re-initializing Quest Master will re-create the entire default. adventure.
3. When the Adventurer player selects a town for the first time, the town is initialized. This means all building names are instantiated from the building registries, and their constituent Items and NPCs (read-only access). The Arena is copied from the Arena file and its constituent NPCs and Items (read-only access). The town is now a hierarchy of objects saved in an .adv file.
4. During Adventurer play, any changes to any of the components of the Town, Arena, Buildings, Items, NPC, etc will be modified in the .adv file (read/write access), and not the original registry files.

A similar flow is followed when a Hero is created by Adventurer.

1. When an Adventurer player creates a new Hero, it pulls original Occupation and Skill registry data (read-only access) that was created by Quest Master. The Hero is saved in the “Hall of Heroes”, a registry file called heroes.drm.
2. During Adventurer play, the Hero’s state is changed (hunger, hit points, items in his/her inventory, etc.) and that state is saved when the Hero is saved back into the .drm file. The only difference is: if the Hero is killed, the player can “resurrect” the Hero from the Hall of Heroes as if that Hero was not awakened in the first place.

