Task 2, Section C

FS InfoCat – Design Document

Leonard T. Erwine

Western Governors University

000356334

Table of Contents

[Class Design 2](#_Toc83355859)

[Abstract Design Pattern 3](#_Toc83355860)

[Hierarchical Structure Entity Types 4](#_Toc83355861)

[Correlation Entity Types 5](#_Toc83355862)

[Other Entity Types 6](#_Toc83355863)

[Concrete Design 6](#_Toc83355864)

[Application Design 6](#_Toc83355865)

[Byte-for-Byte Duplication Handling 7](#_Toc83355866)

[Alternate Correlative Listings 7](#_Toc83355867)

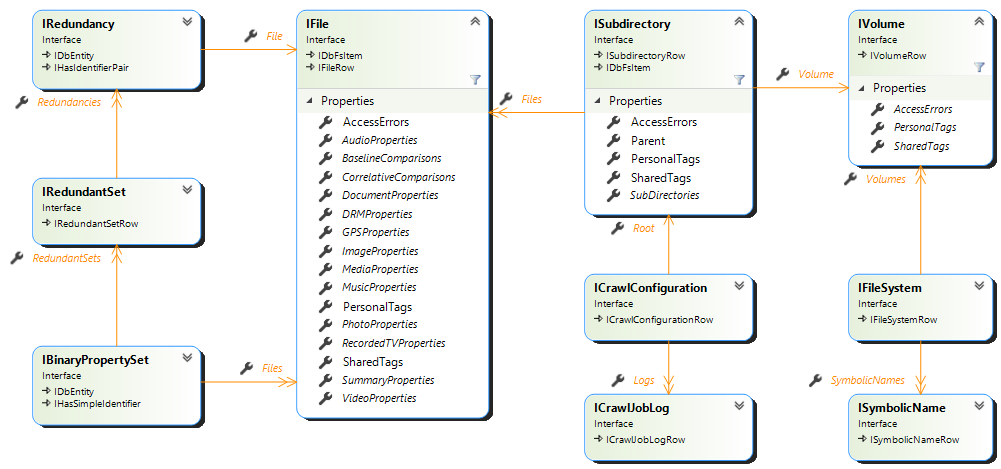
[User Interface Design 8](#_Toc83355868)

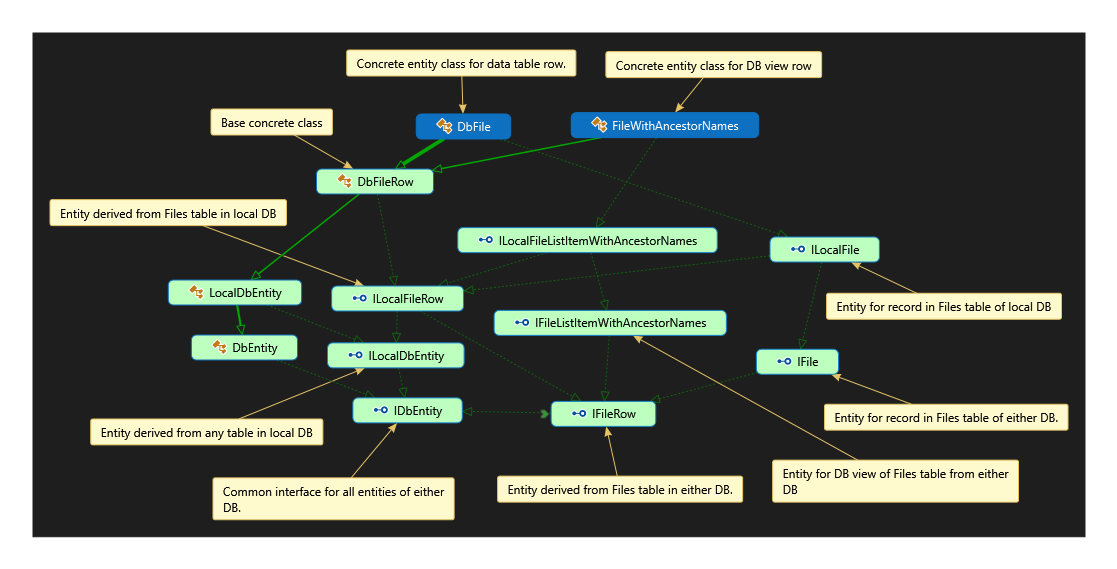
# Class Design

When FS InfoCat gathers information about files, it stores that information in a local database. The user then has the option to synchronize it with a database that everyone shares. I have chosen to use GUID values for all primary keys, making the differences between the local and remote databases more easily reconcilable. In that way, I can use the same unique identifier in both databases to refer to the same entity. Furthermore, I won’t need to worry about either key sequence being out of order as they would tend to be with auto-incrementing numerical types.

## Abstract Design Pattern

In its published form, the data will need to be presented in the same way that it is locally. To reduce the chance of inaccurate data synchronization, I designed a basic set of interface types that specifies the properties and relationships common to both the local and remote databases. Additionally, by adhering to common base types, I will afford myself opportunities of reusability by referring to common base types instead of directly referencing concrete classes.

Since the application’s primary purpose is to collect information about files, I made the File entity central to the design, with many parent tables representing different aspects from which the user discovers other correlating facets. Following is a class diagram that depicts the relationships between the most significant entity model types, with the remaining tables displayed as properties.

I have created special aggregated lookup types in the database, known as database views, which improves application performance. These views return the data from the primary table with the columns and calculated data derived from related tables. As with the primary entity types, I have also defined standard interface types to represent views on both the local and remote databases. Therefore, entities that represent table rows will share the same column types of database views based on that same table. As a result, I can be confident that I won’t forget to include any fields in the concrete classes. Following is a graphical representation depicting the polymorphistic inheritance of the interface design pattern previously described as applied to a file entity of a record from the File data table and a file entity from a database view.

## Hierarchical Structure Entity Types

The Subdirectory table is self-referencing to capture the hierarchical folder structure of the source file system. The Volume table is the parent table for top-level subdirectories, describing the logical device that contains the file. The volume table, in turn, has a File System parent entity that represents a specific file system type. Lastly, the “Symbolic Name” table maps well-known symbolic names to particular file system types listed in the File Systems table. File system type names can vary, depending upon the host system type and version. When the application encounters a file system type that has never existed before, it automatically adds a new File System record and the new symbolic name. The mappings and display names can be clarified or corrected later without affecting other database relationships.

## Correlation Entity Types

Correlation entity types have a one-to-many relationship with the File entity, with the File entity being the singular endpoint. The first correlation type is the Binary Property Set entity, which contains a file length and cryptographic hash code common to one or more files. The rest of the correlative entity types represent metadata according to various file types. I loosely based each entity type on the categories defined in the Windows Property System, which provides the metadata values.

## Other Entity Types

I included two entity types that enable users to create arbitrary tag names and associate those tags with specific files, subdirectories, and volumes. The first is the Shared Tag Definition entity which describes tags that anyone can view and use. Conversely, the Personal Tag Definition entity type which describes tags that do not get shared with others. Lastly, The Volume, Subdirectory, and File entities each have a related Access Error entity for errors that occur while accessing the file system node that it represents. The document entitled “CrawlStatusInterfaces.png” gives a more extensive graphical representation of the access error entity types and how they relate to the other entity types.

## Concrete Design

All entity types for the local database will have a corresponding entity type targeting the remote database. The remote database will have additional tables pertinent to a shared data environment, with the Host Devices entity identifying specific host machines and devices of the files in the database. Next, the Host Platforms describes types of host device platforms, such as Windows and Linux. Other tables in the remote database will be for role-based authorization and simple task tracking. Refer to the “DbEntityDiagram.xps” file for a more extensive graphical representation of all concrete database entities. Finally, the “FileMetaDataEntities.png” file accompanying this document represents all of the entity types implemented for the local database.

# Application Design

The application refers to the process of scanning through subdirectories using the term Crawling. Hence, the concept of specifying where the application looks for files is called a Crawl Configuration. Each crawl configuration tells the application where to begin looking for files, how far to go, and how long it can run. Additionally, I have provided fields that allow users to indicate whether they intend to perform the crawl regularly and how often it should occur. Finally, after the crawl finishes, the application summarizes the crawl results in the Crawl Job Log table. There are several ways to view the aggregated file information, which I will cover next.

## Byte-for-Byte Duplication Handling

When the application encounters a file with the same length as another file, it will calculate an MD5 checksum of the binary file contents. If it shares the same checksum value as another file of the same length, the application treats it as potential duplication. The Binary Duplications listing shows the potential byte-for-byte redundancies. Each Binary Properties entity contains a unique file size and cryptographic hash associated with one or more File entities. Additionally, the Binary Properties entity can have zero or more related Redundancy Set entities. A Redundancy Set is a sub-grouping of potentially duplicate files that share the same mitigation status. The mitigation status value has the following options: Unconfirmed; Not Redundant, Pending Validation, Deferred, Justified, Insupportable, Violation, Attrition, and Deleted. Files with the status of Insupportable are those that do not belong in their current location. The violation status indicates that the file in its current location presents a risk or liability, such as a breach of confidentiality. Attrition flags a file for deletion. I used the term “Attrition” so it won’t be confused with the “Deleted” status, which is for files already deleted.

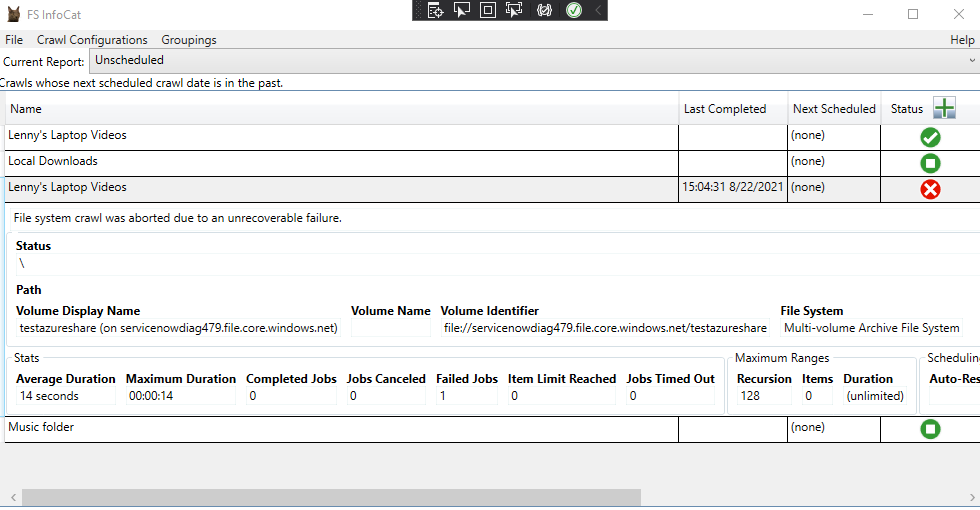
## Alternate Correlative Listings

The Summary Properties list includes extended file property values that are agnostic to any file type. Other correlative listings show file properties according to specific file types, such as video and audio files. The user can sort and review these listings to look for other kinds of duplications.

# User Interface Design

FS InfoCat uses a page-based navigational system to navigate the application’s list views, detailed views, and editing forms. The main application window has a static menu bar at the top and a status bar at the bottom, with the dynamic page content in the middle region. I configured the application to load the listing of recent and upcoming scheduled crawls when the application starts up. I chose this as the initial page because it gives practical situational awareness of the recently completed crawls and those whose due dates are soon or overdue. From there, they can open one of the crawl configurations listed or navigate to other listings using the menu bar at the top. The following illustration is a low-fidelity mock-up of the main application window at startup.

|  |
| --- |
| File | Crawl Configurations | Grouped File Listings |
| |  |  |  |  | | --- | --- | --- | --- | | Description | Last Completed | Next Scheduled | Status | | Local Downloads | 9/10/2021 | 10/10/2021 | Completed | | Web Root Folder | 9/30/2021 |  | Access Denied | | Pastor’s Podcasts | 5/5/2021 | 9/5/2021 | Not Started | |  |  |  |  | |  |  |  |  | |
| User\_name – not logged in |



References