



## DesignedNet Expedition File System Overlay Document

Expedition is a SQL server database model to duplicate and expand upon the functionality offered by traditional file systems such as FAT and NTFS. The database is designed to maintain a record for each file contained on a disk. A folder structure allows for a hierarchal organization of items.

The primary goal for augmenting the existing file system is to expand the functionality to allow for numerous files to be grouped together and allow that group to exist in multiple folder locations. The only way this is accomplishable with current technology is to place grouped files in a folder and create short cuts to that folder in other locations. Expedition will also allow for historical and archival information to be maintained for files present on offline media.

In order to implement the required file system functionality four primary entities have been identified: Path, File, Item and Folder. Each entity is described from a functional level in the following section.

The Path entity is responsible for keeping track of root location or starting point for a file system overlay. Paths reference a directory located on a local drive, network share or removable disc. A username and password may be persisted for each path if access security is enabled. Each path maintains a root universal resource locator (url), label and serial number.

The File entity represents a physical file located in a sub-directory of a specified path. Each file's location is stored as a url relative to the root url of the path. Files are typed as one of the currently support primary file types: Image, Audio, Video.

The Item entity allows for the grouping of files together for folder assignment and overlay profile populate. Each item record loosely references another record in any database table. The item type determines which table to locate the record in by primary key. The current implementation abstracts the Path entity to allow grouping.

The Folder entity maintains a hierarchal tree structure with role based security. Each folder or sub-folder has an identifying name and one of four role security levels: Private, Protected, Public, Published. All items assigned to the folder assume that folders security level.

Two additional child entities are required to fully implement the requirements. The ItemFile and ItemFolder entities relate an item record to either a file or folder, respectively. This allows for items to contain multiple files and each file to be referenced by multiple items. Additionally, items may exist in multiple folders and each folder may have multiple items.

## Path Import Functional Requirements

The import process begins with selecting a physical directory path to begin to overlay. The path may be selected from the list of maintained paths which have been previously imported or by browsing for a folder or drive accessible by the local machine. After a path is selected the application will verify access to the selected path and create a record in the database, if necessary.

The following options are available to control the import behavior to suit the overlay requirements. The user may limit the path and file search to files created or modified after a specified date. File filters are also available to limit the types of files imported into the database. The user may select one or more of the following file types: Image, Audio, Video. If no file types are selected only the directory structure is imported and file records are not created.

There are two distinct ways in which the directory structure is created in the database. The import from root option creates one root path and stores the file path as a path relative to the root import path. The import paths option creates a path record for each directory encountered and stores only the filename in the relative path column for the file record. This second option is intended to be used in junction with the file discovery feature which stores only the directory structure as paths and dynamically queries the file system during browsing. The feature is intended to significantly reduce the amount of stored data when files are already group in directories and will not be referred to individually or have file profiles created.

The folder structure maintained by Expedition is used to sort, organize and categorize the items imported. The option to auto-create a folder structure, which parallels the imported directory structure, under a selectable root path is provided to preserve the inherent data provided by the logical directory structure arrangement. If at least one file type is selected for import, each identified file is added as an item to the created folder, regardless of the path import mode. If no file types are specified and the option to create multiple import paths is selected, each path is added to its respective folder as an item. If no file types are selected and only a single path is created during import then no folders are auto created and the path is assigned to a default import folder as an item. If the folder structure creation option is disabled then all imported items, files and paths, are assigned to a single default import folder.

During the import process, a status window will display the estimated progress of path and file import operations. An estimate of the total file size (KB) of data to be imported will be obtained prior to path transversal and used to compare to the sum of the imported file sizes to determine an approximate percentage complete. The status window will be updated after processing all the files in each folder for a balance of speed, simplicity and responsiveness. A timer will display the time elapsed and indicated the time remaining based on the calculated percentage complete.



# Application Architectural Implementation

Expedition is intended to demonstrate the technological capabilities of the DesignedNet application framework and therefore makes significant use of the base objects. Following the best practice guidelines for DesignedNet, the application will use a SQL Server 2000 database and three application layers, Data Access, Business Logic and User Interface.

The Data Access layer (namespace: Dal) includes both stored procedures and the code required to communicate with the SQL database. Each primary entity identified will implement all standard data access verbs (List, Exists, Insert, Select, Update, Delete) via the base data access class in unison with the auto-generated stored procedures. Additional functionality required by each entity will be specifically implemented in each class.

The Business Logic layer (namespace: Biz) provides state cursor access for data returned from the data access layer calls. Each primary entity object exposes strongly typed properties which represent the underlying data table column values. The business objects provide a mechanism to read and write data values in memory and persist the changes in the data store. Related entities are referenced as typed business object properties. Again, Expedition makes significant use of the DesignedNet base business object class to implement the majority of transparent requirements.

The User Interface layer provides a graphical representation of the business objects. Expedition requires two distinct user interfaces, each with specific functionality. The Windows forms interface (namespace: Win) is used to synchronize path and file information, process batch operations and verify database integrity. The Web browser interface (namespace: Web) implements the functionality for viewing item, grouping files and browsing folders. Due to the highly specific nature of user interface requirements across applications, the DesignedNet framework does not provide a base class for common implementation of this application layer. Multiple helper objects are provided and will be used extensively throughout however.

To expedite the development process Design Studio, the DesignedNet code generation application, will be utilized. Once the data model is complete in a SQL server database, the schema is used to generate stored procedures, data access objects and business logic entities. Crude user interface files are also created to provide a head start on form development by providing common preconfigured controls for data input, validation and updates.

Properties and methods automatically generated from the database schema are not documented in the following section. All generated code is assumed to have a default implementation and usage. New functional requirements beyond the base DesignedNet implementation are detailed below for each primary entity. Please refer to the "DesignedNet Framework Architecture" document for additional details on base class and generated code functionality.



## File Item Entity Import Workflow Process

The process to import a File into the Expedition database requires the creation at least a Path, a File and an Item record. Orphaned files, which do not have at least one associated Item, will be removed by the database file system integrity checking process. The first step to importing an Entity is to create a File record with an associated (new or existing) Path record. The Path business object provides an ImportPath() method to create a new Path record given the root url. The name and root url are automatically populated and the record is persisted to the database, the method returns a Boolean value.

The File business object provides multiple ImportFile() methods for accomplishing this task. The base version accepts a specific existing Path record and a file url relative to the referenced root. The file system is checked to verify that the path and file exist, then a new File record is created and any supported primary File Type (ie: Item Type) is auto-detected and assigned. The File record is saved to the database and a Boolean value is returned. The extended version accepts a full url to a local file. It attempts to locate an existing parent Path in the database or imports a new root Path to represent the immediate directory where the file resides. Once the Path is established, the base ImportFile() method is referenced to complete the import process.

After the file has been created, the import process continues with the Item import. The Item business object which exposes multiple ImportItem() methods. The base version accepts an existing File record and a specific Item Type to import. The File record is then passed to the respective Import method for the Item Type. The Import method on Entity business object creates its respective record and returns control to the ImportItem() method. The display name for the Item is populated from the default Entity property value. Next, the Item record is saved and then the File is associated with the Item. Again, a Boolean value is used as a return. The extended version accepts only a File record and attempts to auto-discover the correct Item Type to import given the file extension and properties.

Each Entity import method provides logic specific to that Item Type. If the File Type property of the File record is updated to reflect a more specific supported File Type it will be automatically persisted by the Entity object. Later, an optimization pass will be made to all import functions to reference the state mode of the object. If the mode is changed to Inspect, then the same DataSet will be used for all business object state and no database updates will be executed on an individual record basis. Instead, the persistence of all data created or modified during the import process will occur at the end using the SaveAll() method for each associated business object.

The overall import process requires the selection of at least one existing system Path from the available list of Paths maintained by the database overlay. The types of Items to look for are specified by a list of checkboxes of supported Item Types. Additionally, the file search may be limited by a date range of file creation/modification or by preventing recursion of the subdirectories of the selected Path. Each file which is discovered must be checked for existence in the database. If a File and Item record already exist then the import process will continue checking the next available file.



## Future Functional Requirement Considerations

Considerations for future enhancements include the ability to filter folders or files which fit, or do not fit, a wildcard pattern. Additionally, functionality will be added to facilitate the moving, copying or renaming of files and folders into a secondary physical directory structure by selecting a destination path. Each item imported would be automatically moved or copied to this new path with various options for creating destination directories based on file name patterns, ie: intelligent identification of sequential file groups.

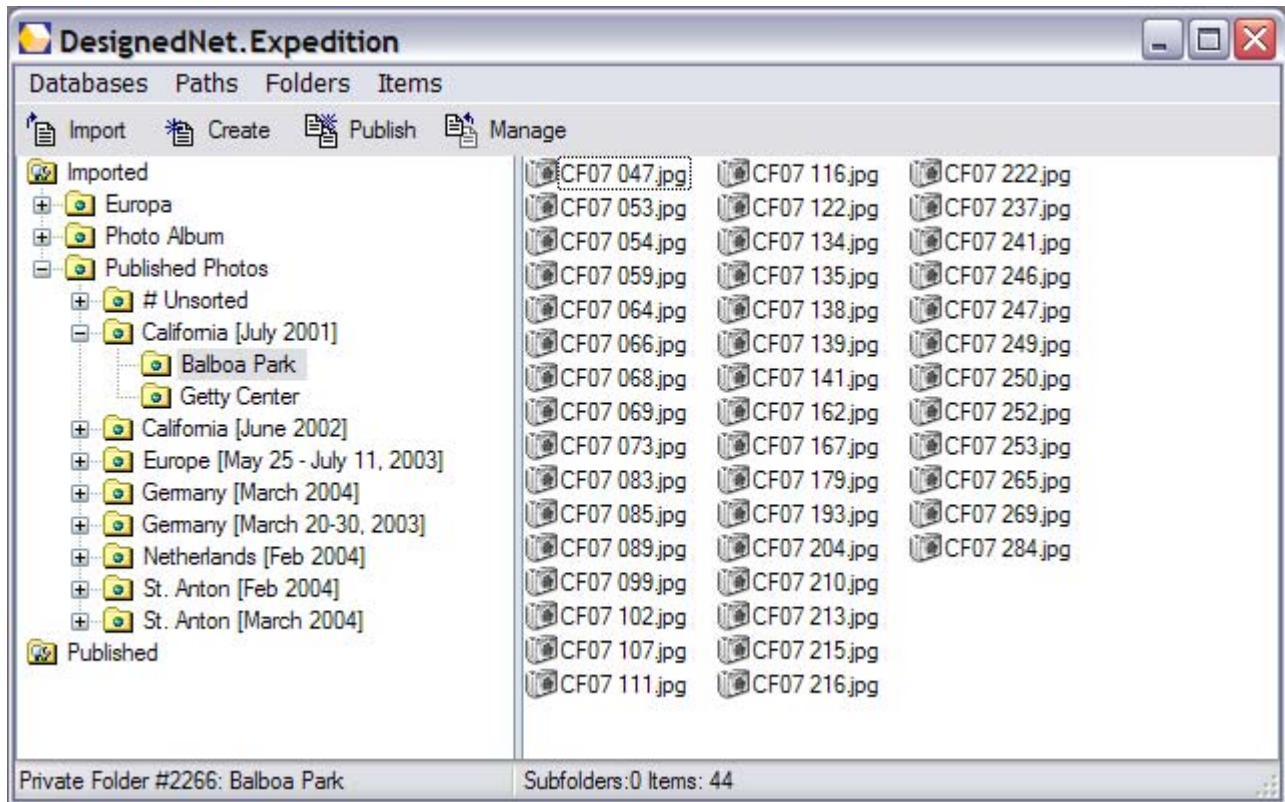
Future versions of the file import application will be able to monitor the file system in real time by attaching to NTFS folder events. Batch process functions will be identified and developed to handle common file maintenance operations such as folder synchronizations across path locations, backup of files for archival and file information reporting. Eventually, a web service interface may be developed to allow for clients to synchronize data with a remote server.

**Once a sufficient application infrastructure is developed and the requirements and functionality has been verified and validated, work will begin on a Windows Explorer like interface to allow intelligent browsing of the physical directory structure and logical folder structure in parallel.**



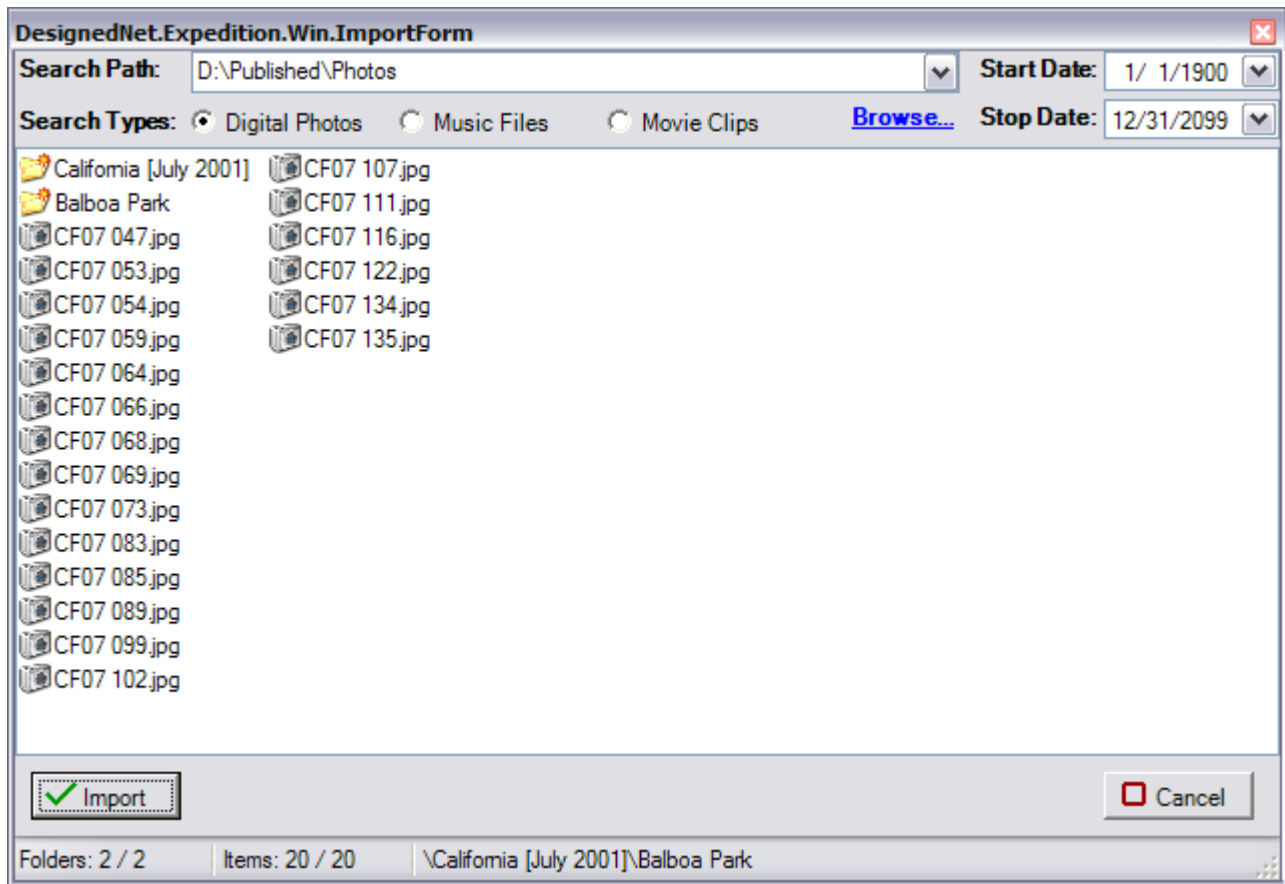
# Windows Interface Functional Design Requirements

The main application form consists of a Folder tree view and Item list view similar to the Windows Explorer interface. A splitter in the center allows both controls to be resized to fix the specific desire of the user. The Folder tree view is automatically loaded from the default database and rendered with the Imported and Published tree nodes expanded. When a Folder is selected the details on the Folder is displayed in the status bar and the Item contents of the Folder are listed on the right. Selecting an Item from the list on the right will display the Item details in its respective status bar. Double clicking the item will launch an Item specific dialog window.



# Windows Interface Item Import Requirements

The Import Form provides an existing Path selection box. If a local path is entered into the Search Path dialog then a new Path record is created in the system if a suitable parent path cannot be determined. A start and end date range filter is provided to filter for files created or modified in the specified time range. Only one Item Type may be imported at a time. Progress is indicated in the status bar. The folder display counts the number of new Folder records created compared to the number of physical directories encountered. The item display counts the number Item records created compared to the number of physical files located. The current relative search path is displayed on the right for status purposes.



## Windows Interface Photo Form Design Requirements

The Item Form Photo Tab displays the specific attributes for the Photo entity. The form allows the Title, Author, Date/Time Taken, Subject and Description to be edited and updated. The thumbnail screen shows a thumbnail of the highest resolution File associated with the Photo Item. The default thumbnail size is 320 x 240 pixels and preserves the correct aspect ratio for rotated or odd sized images.

The Photo business object provides a function to create resized Jpeg files from a source Jpeg file. The algorithm uses the highest resolution (largest) source file available for reduction and creates a new Jpeg file at the location provided. A File record must be created and added to the Item in order for the association to be persisted in Expedition.

Exporting an Entity Item from the Expedition database creates a physical file in a specific local directory of the local machine. The directory is contained within a root system Path for published items. The relative path is the source Path Name and source File RelativeUrl. The application must have full write access to the destination directory. The directory structure will be created, to mirror the source, if necessary. After the physical file is created, a File record is added to the Item.

The screenshot shows a Windows application window titled "DesignedNet.Expedition.Win.ItemForm". The window has a tabbed interface with "Photo" selected. Below the tabs are icons for "Publish", "Folders", "Files", and "Comments". The main area contains several input fields and sections:

- Title:** A text box containing "Photo.Title".
- Author:** A text box containing "Photo.Autho".
- Date Taken:** A date picker showing "6/24/2002".
- Time Taken:** A time picker showing "3:47:00 PM".
- Subject:** A text box containing "Photo.Subject" with up and down arrow buttons.
- Description:** A text box containing "Photo.Description" with up and down arrow buttons.
- File Info:** A section displaying "Canon Canon PowerShot S330", "W/H Size: 1600 x 1200", and "XY Res: 180 x 180".
- EXIF Info:** A section displaying "Map: 1 Shutter: 5 Aperture: 2 Flash: 16 Focal Len: 5 FStop: 2 Exposure: 0 @ 0 Metering: 5".

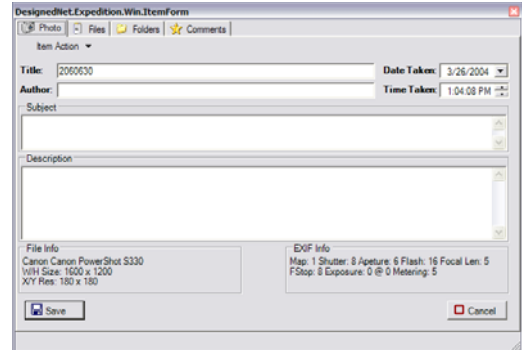
At the bottom of the window, there are two buttons: "Save" (with a green checkmark icon) and "Cancel" (with a red square icon).



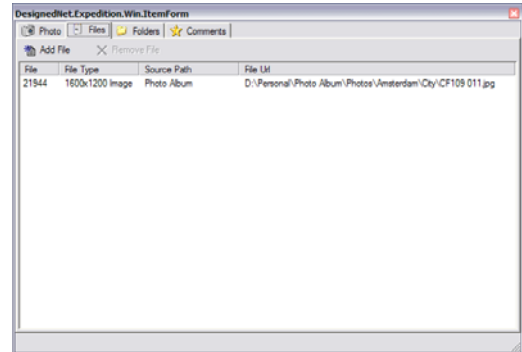
# Windows Interface Item Form Design Requirements

The Item form dialog consists of three main tabs. The first tab provides the user functionality to maintain the entity profile and execute Item specific commands.

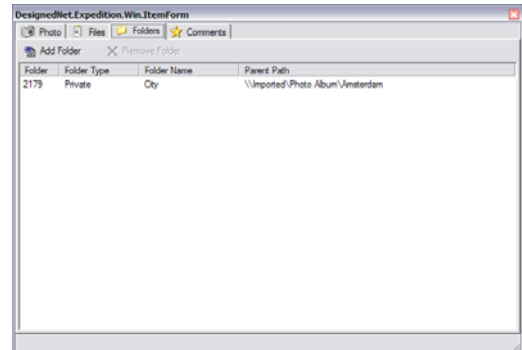
Each support Item Type has a form specific to the entity. The Photo form is displayed for demonstration purposes and allows the user to edit the Photo profile and view the EXIF data extracted from the digital photo file.



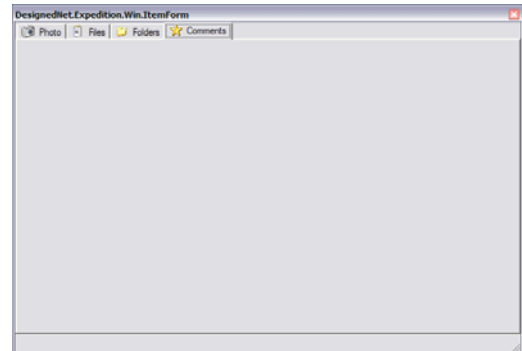
The Item File tab lists all the Files associated with the selected Item. Details for each File include the File id, type, full address, and source Path. Double clicking on a File will execute the File in an explorer shell, launching the associated Windows application for the specific file extension. A selected File may be removed if more than one File association exists. A new file may be added to an Item with the Add File toolbar command by select a File from the file system. The Type and Path will be auto-discovered.



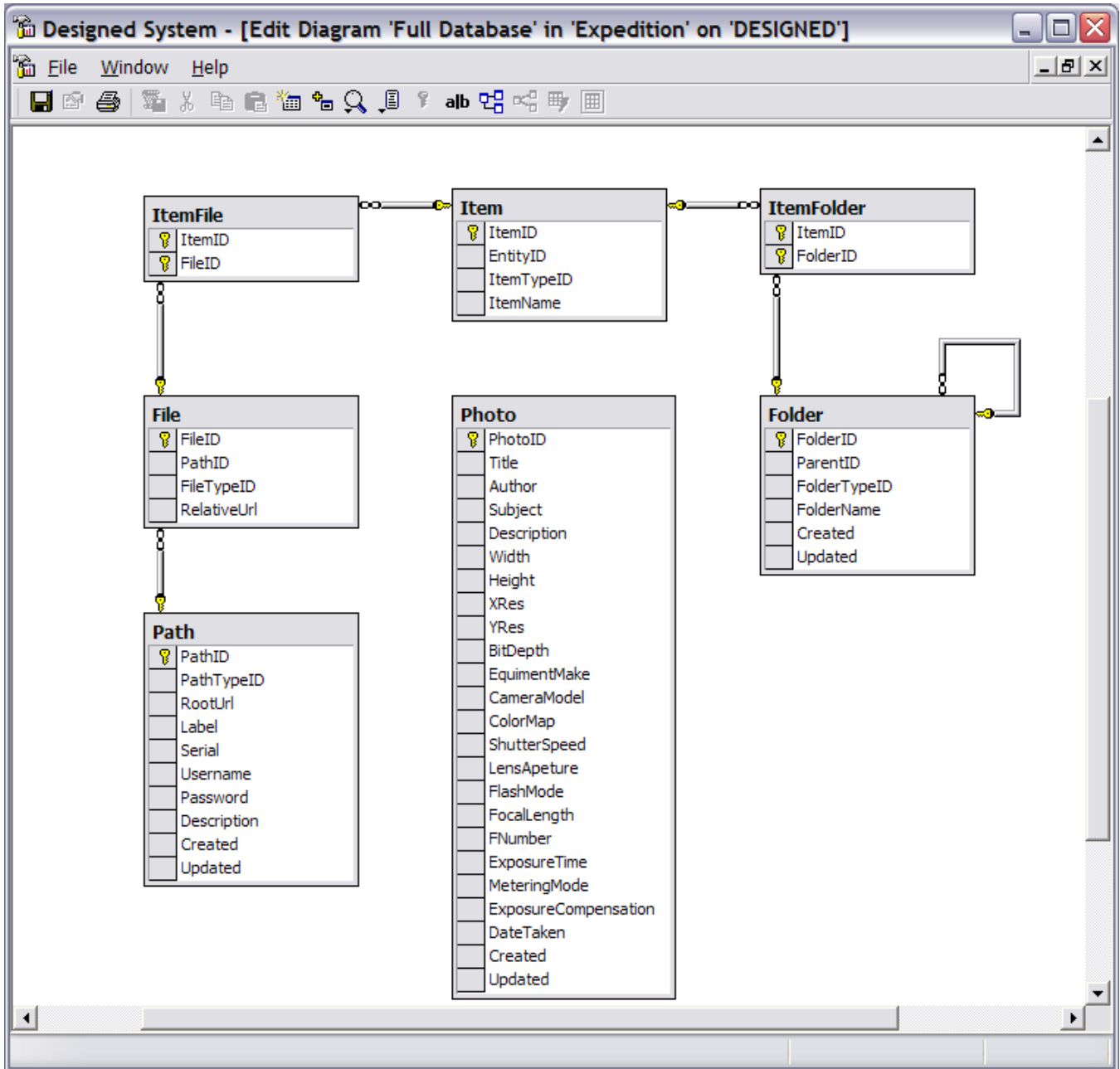
The Item Folder tab lists all the Folders associated with the selected Item. Details for each Folder include the Folder id, type, name and full path for parent Folder. A selected Folder may be removed if more than one Folder association exists. A Folder may be added to an Item with the Add Folder toolbar command by selecting a Folder from the Browse Folder form.



The Comments tab will be described in detail later in the document.



# DesignedNet Expedition Database Model Diagram



# DesignedNet Expedition System Data Dictionary

## Item Types

0	Path	Item.EntityID = Path.PathID
1	Photo	Item.EntityID = Photo.PhotoID
2	Audio	Item.EntityID = Track.TrackID
3	Video	Item.EntityID = Movie.MovieID

## File Types

0	Unknown Source File	Any unrecognized file format
1	Image Source File	Any compatible image file format
2	Audio Source File	Any compatible audio file format
3	Video Source File	Any compatible video file format

## Path Types

0	System Path	System path used by application
1	Local Directory	Directory located on fixed local disk
2	Network Share	Share located on a networked computer
3	Removable Media	Removable media from CD or DVD drive

## System Paths

0	Application installation local directory
1	Published Photo Album site virtual directory
2	Published Music Album site virtual directory
3	Published Movie Album site virtual directory

## Folder Types

0	System	Folder contents not available to any account
1	Private	Folder contents available to admin accounts
2	Protected	Folder contents available to agent accounts
3	Public	Folder contents available to user accounts
4	Published	Folder contents available to public users

## System Folders

-1	Virtual system tree root folder
0	Expedition path import destination root folder
1	Published Photo Album site navigation root folder
2	Published Music Album site navigation root folder
3	Published Movie Album site navigation root folder



# Data Access Layer Functional Requirements

## Expedition.Dal.DalPath

int FindPath(string rootUrl) Returns the primary key of the first path with matching url

## Expedition.Dal.DalFile

int FindFile(string fileName) Returns the primary key of the first file with matching name  
int FindFile(string fileName, int pathID) Restricts file search to specified path

int FindPath(string relativeUrl) Returns the primary key of the first file with matching url  
int FindPath(string relativeUrl, int pathID) Restricts file search to specified path

int ListByItem(int itemID) Lists all file records associated with the specified item

## Expedition.Dal.DalItem

int CheckEntity() Returns true if item entity record is located (check integrity)  
int FindEntity(int ID, int typeId) Attempts to load the item represented by the entity specified

int AddFile(int fileID) Adds the specified file to the item file collection  
int RemoveFile(int fileID) Removes the specified file from the item file collection

int AddFolder(int folderID) Adds the specified folder to the item folder collection  
int RemoveFolder(int folderID) Removes the specified folder from the item folder collection

int ListByFolder(int folderID) Lists all item entities containing the specified folder reference

## Expedition.Dal.DalFolder

int Delete(int folderID) Cascaded deletion of parent folder records

int ListByItem(int itemID) Lists all folders containing the specified item



# Business Logic Layer Functional Requirements

## Expedition.Biz.BizPath

BizFile Files Returns all the files associated with the selected path  
int FindPath(string rootUrl) Returns the primary key of the first path with matching url

## Expedition.Biz.BizFile

BizItem Items Returns all the items associated with the selected file  
int FindFile(string fileName) Returns the primary key of the first file with matching name  
int FindFile(string fileName, int pathID) Restricts file search to specified path  
int FindPath(string relativeUrl) Returns the primary key of the first file with matching url  
int FindPath(string relativeUrl, int pathID) Restricts file search to specified path  
bool ListByItem(int itemID) Lists all file records associated with the specified item

## Expedition.Biz.BizItem

BizFile Files Returns all the files associated with the selected item  
BizFolder Folders Returns all the folder associated with the selected item  
bool CheckEntity() Returns true if item entity record is located (check integrity)  
bool FindEntity(int ID, int typeId) Attempts to load the item represented by the entity specified  
bool AddFile(int fileID) Adds the specified file to the item file collection  
bool RemoveFile(int fileID) Removes the specified file from the item file collection  
bool AddFolder(int folderID) Adds the specified folder to the item folder collection  
bool RemoveFolder(int folderID) Removes the specified folder from the item folder collection  
bool ListByFolder(int folderID) Lists all item entities containing the specified folder reference

## Expedition. Biz.BizFolder

BizItem Items Returns all the items associated with the selected folder  
BizFolder SubFolders Returns all the sub folders of the selected folder  
bool Delete(int folderID) Cascaded deletion of parent folder records  
bool ListByItem(int itemID) Lists all folders containing the specified item

