Design Document

Travel Agent File Transfer Software

"The Journey project"

Mark Cunningham
James Dyer
Zhili He
Cory McKay
Maru Newby
Jonathan O'Hara
Tristram Southey
Sitai Sun
David Tarc
Xinli Wang

Table of Contents

I. Introduction

This is the specifications document for the Journey software project. It is essentially a client that will allow files to be transferred from remote servers to local computers using Windows File Sharing, FTP or telnet connections. It will have an easy to use and intuitive GUI and will have complete error protection that informs the user of errors in a precise and understandable manner. One interesting characteristic of this program is that it will allow users to form a queue of files that will be automatically downloaded at a later date.

There are four main sections to this project. The first is the front end of the GUI, which will include no functionality beyond supplying the users and conveying their requests to the other parts of our program. The second section is the backend of the GUI. This section will take the commands of the users and make sure that they are processed properly. All state information will be stored here. The next section, the adaptor, will act as an interface between the requests from the backend of the GUI and the low level interactions from the three communication protocols. It will allow the backend of the GUI to issue requests and receive responses in ways that are independent of which communications protocol is being used. There is also a final section to this document that outlines the protocols that will be used.

II. GUI

The first action required from the user is a login, automatically prompted upon the execution of the program. This will ask for user ID, password, host name, and finally transfer method. The transfer method describes whether the user specifically wishes to implement ftp, telnet or the windows file-sharing mode for transferring their files. The default value for the transfer method is "Automatic", allowing the program to pick an appropriate method. During the session, the user can change to another remote machine by using the connect menu item from the connectivity menu, which will again bring up the login window. Passwords are not displayed when they are typed in.

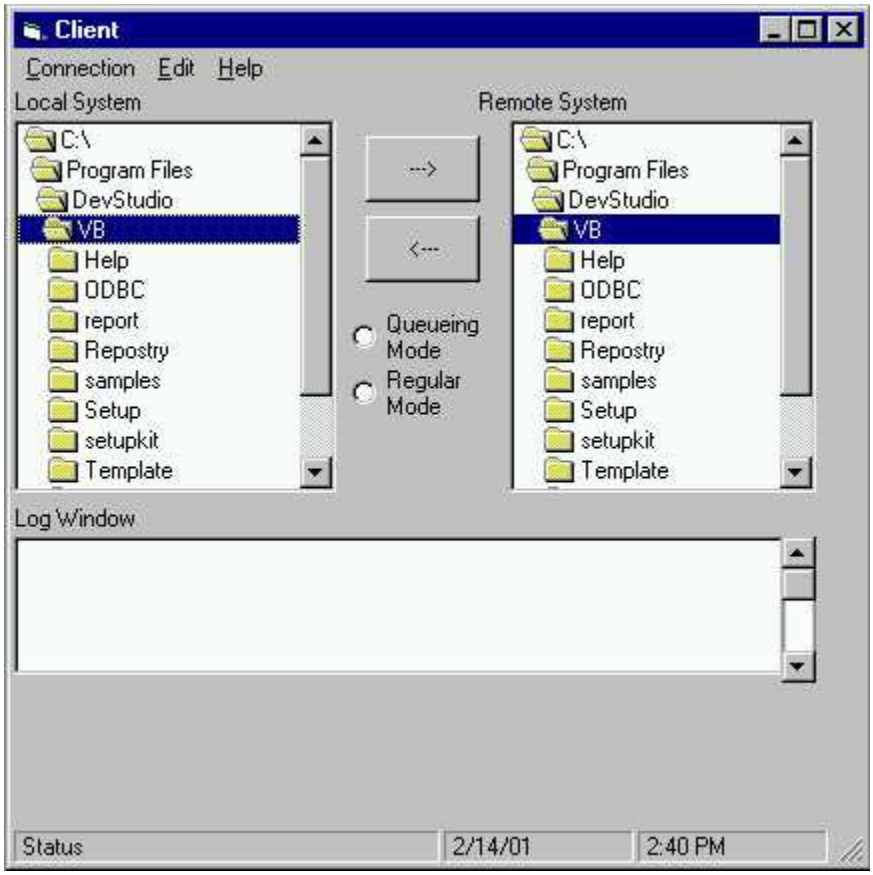
During the session the user will be able to select files on either the local or remote machine, and transfer them using the arrow buttons separating the two file trees. File selection will be similar to the methods used in most windows applications: shift and control can be used to select multiple files at once. If the user wishes to create a queue for overnight down loading this can be accomplished by pressing the "Queuing Mode" option. When this is activated then the "Queue" window will appear at the bottom of the screen. Now instead of transferring the files immediately, all transfers are simply placed in the queue. These transactions will be listed in this window and can be removed if desired, by selecting the transfers to be cancelled and pressing the delete button. When the "Regular mode is selected" or the queue window is closed the queue will be maintained, and should queuing mode be reinitiated this session these files will reappear. The transfers can also be initiated by using the appropriate button on the queue window. When the list of files has begun transferring, successful completion, errors, or progress will be reported in the log window. The transfers in the queue list will show the complete path of the file to transfer, and the direction of the transfer. Other functions available in the queue window are move a file up or down in the list, and delete a transfer from the list.

Each time a file or queue is being downloaded there will appear a dialog containing a status bar showing the progress of the transfer of the current file, as well as a status bar showing the percentage of all transfers so far completed. This dialog will also contain three buttons: "Pause" to pause or resume the transfer, "Cancel" to completely cancel the transfer of the current file, and "Cancel All" to cancel all downloads currently queued.

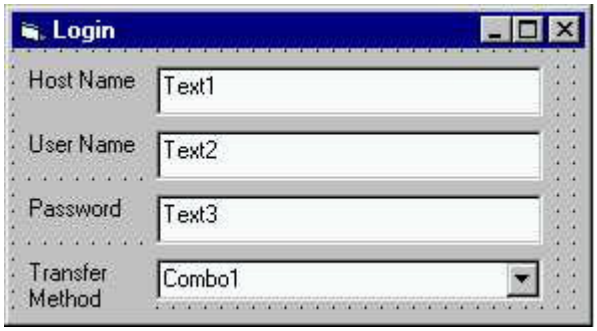
The "Connection" menu option can be used to connect to or disconnect from a server, and exit the program. "Connect" opens the login window allowing the user to connect to the given host. "Disconnect" closes the connection with the current host. "Exit quits the program.

The "Edit" menu option can be used to call the queue window, and change the default directory. "Display Queue" will open a window allowing the user to see a list of all the files they want transferred. "Change Default Directory" will change the starting path of the client to whatever the user specifies.

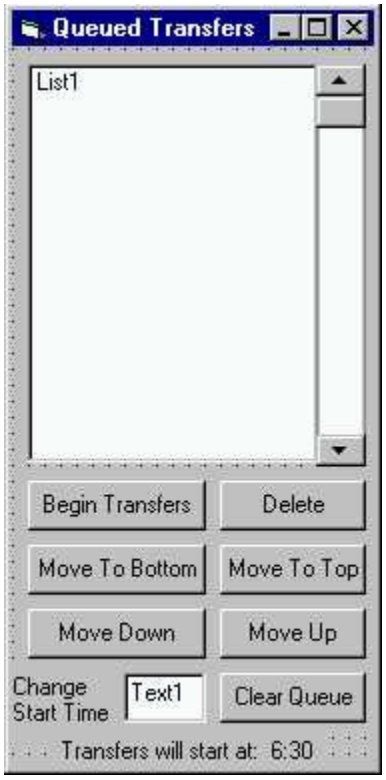
Finally, using the "Help" button can access help. This is discussed further in the Help section of this document. Also, "About the Journey" will just inform the user about the program and who made it.



Main Client Window: Will list files along with their name, size, and path.
Will support directory maneuvering as well.



Login Window: Allows user to type in host's name, user name and password,
Also allows user to specify which transfer type to use (defaults to FTP)



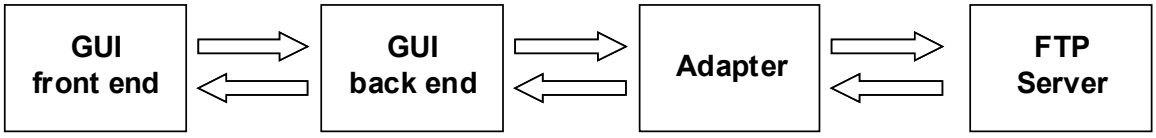
The Queue Window: Here the user can see the files they are uploading and downloading
The user can change the transfer list or remove files from it.

III. Backend of GUI

3.1 Definition

The back-end of the GUI is responsible for communicating requests from the GUI front-end to the Adapter, and Adapter will connect to FTP server to finish information transfer. Back end is also responsible for communicating the response to these to the front-end. In addition, GUI of FTP shall be responsible for transfer information temporarily storing.

3.2 User requests handling model



User Requests Handling Model

3.3 Information storing.

3.3.1 Information to be stored:

3.1.3.1 user login information

hostname
username
password

3.1.3.2 file information

current directory
alarm clock

3.3.2 Data structure

3.3.2.1 Variables Used:

String hostName: remembers the hostname currently connected to.
String userName: remembers the login id used.
String password: remembers the password used.

String currentDir: remembers the current directory.
double downDone: remembers how much of file is being downloaded.
int ErrorCode: int, keeps track of error that occurred.
outStream: the output link used for uploading a file.
inStream: the input link used for downloading a file.

3.3.2.2 Data Structures:

Queue: a list of files needed to be downloaded or upload

- String name: the name of file to be downloaded or upload.
- String path: Location of server to where file is.
- long size: how big the file is.
- long done: how much has already been downloaded.
- String request: upload /download
- Server: URL of server to get the file.
- String login: ID needed to login to server.
- String password: Password needed to login.

ErrorList: a list of error messages

- Type: the type of error (int)
- Time: the time the error occurred at.

IV. User Requests Handling

4.1 Connection

4.1.1 pass connection request to Adapter

4.1.2 pass connection information to Adapter:

- hostName
- userID
- password
- Connection method: Telnet

4.1.3 check results from Adapter and reply them to GUI

```
if connected
        reply: connection by Telnet successful
else pass FTP connection request to Adapter
        check results from Adapter and reply them to GUI
        if connected
                reply: connection by FTP successful
        else pass WFS connection request to Adapter
                check results from Adapter and reply them to GUI
                if connected
                        reply: connection by WFS successful
                else
                        reply: connection failed:
                        Host not found
                        Error login name or password
```

4.2 Disconnection

4.2.1 pass disconnection request to Adapter

4.2.2 check results from Adapter and reply them to GUI

if disconnection failed

reply: disconnection failure to GUI

else

reply: disconnection successful to GUI

4.3 Download

4.3.1 pass download request to Adapter

4.3.2 pass download information to Adapter

fileName

OutputStream

4.3.3 check results from Adapter and reply them to GUI

if source file cannot be read

reply: source file cannot be read

else if source file is a directory

Ask user to download the directory and its contents or not

if user answer yes

Send start download request to Adapter

else

Send request to Adapter to stop download

else if destination file exists

if destination file is a directory

reply: destination file cannot be written

else ask user to choose overwrite/append/no and

reply: user's request to Adapter again

if user doesn’t want to go on download

send request to Adapter to stop it

else

send request to Adapter to begin

4.4 Upload

4.4.1 pass upload request to Adapter

4.4.2 pass upload information to Adapter

fileName

InputStream

4.4.3 check results from Adapter and reply them to GUI

if source file is a directory

Ask user to upload the directory and its contents or not

if user answer yes

Send start upload request to Adapter

else

Send request to Adapter to stop upload

else if source file cannot be read
reply: source file cannot be read
else if destination file exist
if destination file is a directory
reply: destination file cannot be written
else ask user to choose overwrite/append/no and
reply: user's request to Adapter again
if user doesn't want to go on uploading
send request to Adapter to stop
else
send request to Adapter to begin

4.5 List

4.5.1 pass request to Adapter
4.5.2 check results from Adapter and reply them to GUI
reply: GUI the list result

4.6 Change Directory

4.6.1 pass request to Adapter
4.6.2 pass directory information to Adapter
pathName
4.6.3 check results from Adapter and reply them to GUI
if path is an effective directory
reply: change successful
else
reply: directory change failure

4.7 Help

4.7.1 pass request to Adapter
4.7.2 pass results to GUI

V. Adaptor

5.1 Transfer Adaptors

The three protocols will each have a corresponding transfer Adaptor. Each transfer Adaptor will implement the TransferAdaptor interface, which allows client systems to ignore the details of the underlying transfer implementation.

The following is an ADT representation of the TransferAdaptor interface: Please note that the TransferAdaptor interface implements the ObservableTransfer interface as well, these methods and those of ancillary ADT's are found at the end of this section.

```
public void connect( ConnectionInfo info ) throws      ConnectionFailedException,
                                                       ConnectionRefusedException,
                                                       PermissionDeniedException,
                                                       HostNotFoundException;
```

Opens a connection to the server with the provided connection info object. If the method returns without throwing any exceptions, the connection was successful.

```
public VirtualFile[] list() throws                     ConnectionFailedException;
```

Lists the files and directories available in the working directory on the server, and returns them as an array of VirtualFile objects.

```
public String getWorkingDirectory( String path ) throws   ConnectionFailedException;
```

Retrieves the path of the working directory on the server.

```
public String setWorkingDirectory( String path )
   throws                                              ConnectionFailedException,
                                                       ResourceNotFoundException,
                                                       PermissionDeniedException;
```

Sets the current working directory on the server, returning the new working directory. The provided path must be the fully qualified.

public void getFile(String fileName, OutputStream out, long offset)
throws ConnectionFailedException,
ResourceNotFoundException,
PermissionDeniedException;

Retrieves the file identified by fileName from the working directory of the server and writes it to the given output stream. If the offset is non-zero, the stream must be set to the correct offset, and the method will ensure that it begins receiving data from that offset.

public void putFile(String fileName, InputStream in, long offset)
throws ConnectionFailedException,
ResourceNotFoundException,
PermissionDeniedException;

Sends the data from the provided input stream to the server, where it will be written to a file called fileName in the server's working directory. If the offset is non-zero, the stream must be set to the correct offset, and the method will ensure that it begins receiving data from that offset.

public void disconnect() throws ConnectionFailedException;

Disconnects from the server. Any errors in closing the connection will be indicated by the appropriate exception.

public static void getFile(QueuedFile file, OutputStream out, TransferObserver observer)
throws QueuedFileTransferException;

Establishes a connection to a server – file has a reference to a ConnectionInfo object – and transfers a named file from the server to the supplied OutputStream. When the transfer is completed, the connection to the server is closed. The given TransferObserver will be aprised of the transfer status.

public static void putFile(QueuedFile file, InputStream in, TransferObserver observer)
throws QueuedFileTransferException;

Establishes a connection to a server – file has a reference to a ConnectionInfo object – and transfers the file named in file from the supplied InputStream to the server. When the transfer is completed, the connection to the server is closed. The given TransferObserver will be aprised of the transfer status.

The static methods getFile and putFile are convenience methods that use a TransferAdaptor object to connect to a server, transfer a file, and disconnect, using TransferAdaptor objects. They are non-interactive by nature, and are intended to allow easy implementation of deferred file transfers.

The TransferAdaptor's regular instance methods will be utilized by the GUI back end to provide interactive functionality to the user.

The connection protocol subclasses will communicate to the adaptor the file transfer progress using an object which behaves as an observer. The adaptor will then communicate this information to to registered observers. It is intended that the GUI shall register as an observer with the Adaptor it is using to receive transfer status notification.

5.2 Exceptions:

ConnectionFailedException
- a protocol's server connection has been terminated

ConnectionRefusedException
- a server is either unable or unwilling to accept a connection

PermissionDeniedException
- the desired operation cannot be completed due to invalid security credentials

HostNotFoundException
- the given host was not locatable

ResourceNotFoundException
- a file or directory could not be located

QueuedFileTransferException
- a queued file could not be transferred for any of the above reasons

5.3 Other ADT's

ConnectionInfo ADT
It supplies all the necessary information for a connection (host machine name, user name, password) to the Adaptors.

QueuedFile ADT
This object represents a file that has been queued for a scheduled transfer. The backend of the GUI will pass this object to the static methods getFile and putFile contained in the Adaptor interface. The QueuedFile chall have public methods for accessing all the information that these static methods require.

ObservableTransfer ADT
Objects implementing this interface can register TransferObserver objects and notify them of transfer status as required.

TransferObserver ADT
Objects implementing this interface can add themselves as observers to an ObservableTransfer to be notified of transfer status.

TransferStatus ADT
Contains simple transfer status information and provides the percentage completion for the transfer in question.

VirtualFile ADT
Represents a file or directory on the remote system.

VI. Protocols

6.1 Protocol Overview

The user will be able to access one of the three file transfer protocols, namely, FTP, TELNET, and Windows' File Sharing (WFS) one at a time. The protocols will be implemented in three respective modules. They take the user's requests from the Adaptor, and pass them onto the server, then process and pass the answers back to the Adaptors.

The special error codes in the implementation will be passed to the Adapters, which will be mapped onto the back end of the GUI..

6.2 FTP

FTP uses the protocol "File Transfer Protocol" whose communication is established by using the TCP/IP protocols. Our implementation will support the following FTP commands.

CONNECT
Hostname, username and password are required.
If any one of these fails to be authenticated, a respective failure code will be issued.

CWD
With this command, the user can change the directories of the remote machines.

LIST
This command will list the contents of a directory on the remote machine.

RETR
With this command, a copy of the remote file shall be transferred to and stored in the user's machine.

STOR
This command uploads the user's file onto the server's machine.

STAT
This command will show the status of the operation in progress.

QUIT
This command terminates the user's connection to the server.

6.3 Telnet

With the Telnet Protocol a user with a terminal using the local telnet program is able to run a login session on a remote computer where his communications needs are handled by a telnet server program. The communication is established by using the TCP/IP protocols. The following are the telnet functionalities that the client is going to support.

OPEN
This command requests the hostname.

SET
This command sets operating parameters.

MODE
This command allows entering line-by-line or character at a time mode.

STATUS
This command prints status information.

DISPLAY
This command displays operating parameters.

QUIT
This command will end the telnet session.

6.4 Windows' File Sharing

The Common Internet File System (CIFS) defines a standard remote file-system access protocol for use over the Internet, enabling groups of users to share files across the Internet. CIFS is an enhanced version of Microsoft's open, cross-platform, Server Message Block (SMB) protocol, the native file-sharing protocol.

It illustrates a typical message-exchange sequence for a client connecting to a server, opening a file, reading its data, writing data to it and closing the file.

The supporting functionalities are as follow:

SMB_COM_TREE_ CONNECT
It transmits the name of the disk share the client wants to access.

SMB_COM_NEGOTIATE
It negotiates a file-sharing dialect to use.

SMB_COM_OPEN
It opens a file and retrieve its handle.

SMB_COM_READ
It reads a server file.

SMB_COM_TREE_ DISCONNECT
The client disconnects from the resource.

SMB_COM_CHECK_DIRECTORY
It verifies that a path exists and is a directory.

SMB_COM_SEARCH
It searches directories for a file name.

SMB_COM_WRITE
It writes data to a server file.

VII. Error Handling

7.1 Exceptional Event Handling

7.1.1 Prompting for User Decision

One exceptional event of interest and importance is when a file is attempting to be transferred, but a file of the same name already exists, occupying the desired destination. In this case the user will be prompted with a pop up box, which halts all other processing, until the user chooses from one of the three buttons, Cancel, Rename file to be transferred, or Overwrite. Hitting the rename will allow an alternate name to be entered for the destination. If there is a queue being downloaded, the system will continue to download the selected file.

7.1.2 Subject Matter Reported On:

There will be a fairly small number of errors presented the user, which will render transparent the use of the different protocols and connection mechanisms. The standard errors to be reported to the user are as follows:

- Unable to Establish Connection with Server
- Login/Password Invalid
- File/Directory Not Found (this covers such situations as changes of file structure which render previous information invalid)
- You Do Not Have the Rights to Perform that Action
- Transfer Incomplete (when a connection dies mid-transfer)
- Insufficient Disk Space to Perform that Action
- File Already Exists - Unable to Create (this will be an error limited to the log window, to happen only during non-interactive transfers)

7.1.3 Display of Error Messages During Interactive Transfers

Errors, which occur during the interactive mode of transferring, opposed to those occurring during the “batch transfer”, or “queued” transfers, will demand an interactive acknowledgement from the user. This will be implemented using a popup message box, containing the available details on the error that occurred. No further activity will be allowed until the OK button on the message box has been clicked. The message contained on the error will also be displayed in the “Log Window”, near the bottom of the GUI. The log window will hold a large number of the most recent errors during the current execution of the program. In the event of more errors being generated than can be recorded in the log window, the newer errors will bump out the older errors.

7.1.4 Reporting of Errors During Queued Transfers

7.1.4.1 Prevalidation of chosen files to transfer

While the client is in the process of choosing files to transfer, even though the transfer will not happen immediately, what validation can be done immediately will be done. This includes the case of the file of the same name and path already existing in the queue or on disk. The user immediately be prompted by a standard Skip File, Rename or Overwrite dialog as described above. Disk space limitations will also be checked. In the event that a file comes into existence between this validation and the time the transfer actually takes place, that file transfer will be aborted, and the abortion noted in the log window.

7.1.4.2 Errors Occurring During Queue Mode

Contrarily to the way error messages are handled during interactive transfers, the very nature of the queued transfers necessitates a different approach. Errors of the very same type as during interactive transfers are liable to occur, yet the application will take note of the type of error that occurred, and report the error solely in the log window, just as is the case with the interactive transfers, but there will be no pop up message box demanding user input. Failure of one item in the queue will be recorded, and the next queued file will be attempted, automatically. Any transfer, whether it terminates in failure or a success will only be attempted once.

7.1.5 Additional Considerations for Error Reporting: Observer Mechanisms

One of the strategies that will be implemented to ensure prompt reporting of errors will be the observer design pattern (Gamma et al 1995). This observer mechanism will be used whenever possible in place of throwing exceptions to allow the running methods to continue their work uninterrupted. Some circumstances will demand the use of exceptions because of their critical nature, and the impossibility of continuing normally. The observer makes use of a notify method to alert the calling method, where the error handler will reside, that a specific error condition has occurred. This will allow errors to be reported immediately without the running method relinquishing control of the execution sequence. The observer mechanism will also be used for other purposes such as real-time updating transfer progress, as detailed elsewhere.

VIII. Help System

This is the general layout of the help system. There will be two types of help available from the help menu:

- Help Topics
- About Journey Ftp

The Topic item will spawn a window containing a list of several topics in a side frame and large text frame to its right. As the user selects a topic from the side bar, the relevant help information will appear in the text frame. This information will be broken down into numbered steps, which will walk the user though, the use of the relevant function. The list of help topics will include:

- Introduction to file transfer concepts
- Connecting to server
- Receiving files
- Sending files
- Setting up the queue
- Troubleshooting

The About Journey Ftp menu item will spawn a window containing information about the serial number of the Journey Ftp program being used and the creators.

IX. Test Provisions

In order to ensure system quality, a comprehensive regime of testing will be implemented. The modularity of the described system will allow for automated testing (unattended execution) of the unit and functional tests, but the integration tests will be executed manually due to their GUI nature.

Automated testing will allow suites of tests to be run successively without user input, ensuring that testing is performed in a consistent fashion. Unit tests will have to complete successfully without error before the functional tests will be allowed to execute.

9.1 Unit Tests

Unit tests are required for any classes that contain non-trivial operations. A non-trivial class can be described as any that operates on data. A trivial class is one whose only purpose is to contain data.

Each unit test must fully exploit the functionality of each class to ensure that the class performs as specified. Potential error conditions must also be tested, to ensure that the class handles potential problems as specified.

When writing tests, the developer is encouraged to modify the tests to fail under normal conditions. In many cases, this can function as a sanity-check, to ensure the test is written correctly.

9.2 Functional Tests

Rather than testing individual classes, functional tests are geared towards verifying class interaction in each sub-component (GUI, Backend, Adapters) and their environment. This will largely be centred on verifying that the back end and each adapter can perform their intended functions of communicating with a server and transferring files.

9.3 Integration Testing

Integration testing will ensure that the GUI can integrate with the rest of the system and provide all the functionality provided in the requirements document. User requirements centre on providing functionality to the user of the software, and all functionality must therefore be evident in the GUI.

GUI testing will be a manual process, and as such must be specified in great detail. Testing procedures for each requirement must be laid out in a step-by-step fashion to ensure testing continuity. Detailed testing procedures will assist in reproducing any inconsistencies that are uncovered by providing the sequence of steps necessary to reproduce the problem.

X. Conclusion

So concludes the journey. With that the four main sections, front-end GUI, backend GUI, Adaptor and Protocols make up our project. Each section will be designed with the utmost care to achieve our goal. To make a program that will download all needed travel agent advertisement files. The program will work with FTP, Telnet and Windows File Sharing to ensure relative ease of the user to collect their data.