1 Introduction 3

1.1 Glossary 3

1.2 General Flow of Data 4

1.2.1 Firewall Issues 4
1.2.2 Internal Components 4
1.2.3 Third Party Components 4
1.2.4 Installation Issues 4
1.2.5 Installation tree for iDT 4

2 iDT Program Flow 5

2.1 File Selection 5

2.1.1 File Types 5
2.1.2 File Types File for iDT 5

2.2 Validation (iDT) 5

2.2.1 Validation Definition File Format 6
2.2.2 Turning off validations 6

2.3 Validation (iDTin) 6

2.3.1 Validations using the Validation file 6
2.3.2 Validations using the Customer Registration file 6
2.3.3 Customer Registration File 6

2.4 Compression 7

2.5 Encryption 7

2.5.1 Blowfish 7
2.5.2 Other Encryption Options 7

2.6 Transmission 7

2.6.1 Creating the Transmitted File 7
2.6.2 Transmitted File Name 7
2.6.3 File Header 7
2.6.4 File Signature (Hash Value) 8
2.6.5 Control Numbers 8
2.6.6 Test vs. Production Transmission Types 8
2.6.7 Transmission Protocols 8
2.6.7.1 FTP 8
2.6.7.2 HTTP 8

3 iDTin Program Flow 9

3.1 Internal processing 9
3.2 Getting files from the FTP/HTTP server 10
3.3 Sending processed files to the final host 10
3.4 Keep-Alive File 10

3.5 IDTin Cycles 10

4 Configuration File 10

5 Log Files **11**
5.1 Standard Log 11
5.2 Debug Log 11
5.3 FTP/HTTP Log 11
5.4 Audit Trail 11

6 INI file **11**
6.1 Hashed passwords 11

7 Miscellaneous Features **12**
7.1 User Logins (iDT) 12
7.2 Error File Archiving (iDTin) 12
7.3 Error Message emailing (iDTin) 12
7.4 Customer Verification (iDTin) 12
7.5 Instances/Concurrent FTP Transfers 12
7.6 Firewall Authentication 12

8 The iDT HTTP servlet **13**
8.1 Program Flow **13**
8.1.1 Initialization 13
8.1.2 HTTP Post Processing 13
8.1.3 HTTP Get Processing 13
8.1.4 Temporary File Cleanup 13

9 Troubleshooting **14**
9.1 Typical problems with iDT 14

10 Appendix A: Error Messages .... Error! Bookmark not defined.
10.1 Startup (iDT) .... **Error! Bookmark not defined.**
10.2 Sending (iDT) .... **Error! Bookmark not defined.**
10.3 Validation (iDT) .... **Error! Bookmark not defined.**
10.4 Audit Trail .... **Error! Bookmark not defined.**
10.5 Dial-Up (RAS) (iDT) .... **Error! Bookmark not defined.**

Introduction

This document is intended to serve as a Technical reference of MSA's iDT software. It is not a user guide and it is not a how-to guide. It documents many of the processes and internal workings of the application so that potential users IS groups can gain a more detailed understanding of the Applications overall processes. Note that the assumption is being made that the user level documentation (help file and/or user manual) has been read, and that the reader has a basic understanding of what is in those documents.

1.1 Glossary

Blowfish – A private (or single) key encryption algorithm in the public domain. Considered very secure. Supports 448 bit encryption keys.

Clear Files – Term used to indicate the files resulting from the decryption and uncompression algorithms. The "clear" files are identical to the original files on the client machine. Note that while the term "text" or "clear text" files is sometimes used in iDT documentation, it is not accurate since the file in question could be a binary file and not text.

DES – A single key encryption algorithm, used for many years by the government. A variation of it is called "triple-DES".

EDI – Electronic Data Interchange is the general term used for sending data electronically. Often refers to using the data interchange format defined by the ANSI committee (X12), but also used commonly to refer to Electronic Commerce (EC) in general.

FTP – File Transfer Protocol is an Internet standard protocol to send files between computers. TCP/IP based.

Hash – Any one-way function that takes some data and converts it to a "unique" value. This hash value can be used to determine whether the data has been corrupted. Although the hash values are not really unique, any change to the data will produce a different hash value. A very commonly known hash mechanism for files is a checksum.

HTTP – HyperText Transfer Protocol used to send data to and from Web servers. While it is mostly used to send web pages back and forth, it can also be used to transfer files. TCP/IP based.

MD5 – File hashing algorithm developed by RSA. Because of the long value produced (32 bytes), it is highly unlikely to produce duplicate hashes.

PGP – Pretty Good Privacy is a free public key encryption algorithm. Although not as secure as the RSA MD5 algorithm, it is nevertheless "pretty good".

Private-key encryption – Any encryption algorithm that requires the same (private) key to both encrypt and decrypt the data.

Public-key encryption – Any encryption algorithm that uses separate keys for encrypting and decrypting the data. The encryption key is usually "public", allowing anybody to encrypt the data, but only the recipient to decrypt it.

RSA – Developers of what is considered the premier public key encryption algorithm. This is the algorithm used by all the major browsers for their secure HTTP servers. Until recently, they held a patent (may have expired in 2001?).

Seed – A value passed into an algorithm to make the results be more random (e.g. in random number generators). Using the customer IDs as a seed value will result in different encryption or hash values, even if two customers start with the same original file.

Server – Any computer running services full time, such as an FTP server, a web server, or iDTin.

Signature – The file signature is a hash value for a file, which can be used to determine whether the received file is identical to the original file.

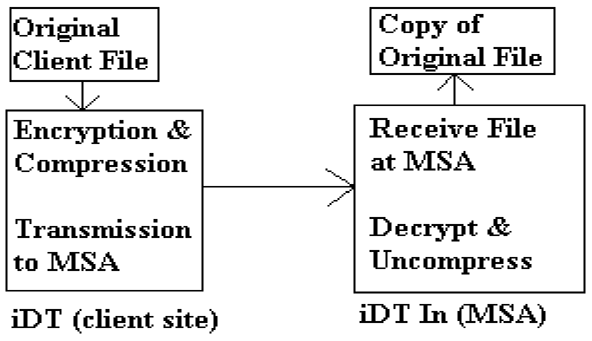
TCP/IP – Transmission Control Protocol/Internet Protocol is a transport mechanism for sending data through a network. Most of the Internet applications (email, web servers, etc.) use TCP/IP as the underlying transport.

WATCH – One of the file formats used by MSA to collect distributor weekly sales data.

24/7 – Term used to refer to a machine/program/service that is running full time, all the time. Sometimes referred to as 24/7/365.

1.2 General Flow of Data

In a very simple simplified manner, the flow of data goes like this:



1.2.1 Firewall Issues

iDT had to be designed in a way that would maintain the security required by IS groups.

1.2.2 Internal Components

emsock32 – is an internal library (DLL – not Active X) written in Delphi. It includes much socket functionality (TCP/IP) such as FTP, STTP transfers, and SMTP (email).

CidtServlet – is a Java servlet that manages HTTP transmissions on the MSA web server.

1.2.3 Third Party Components

CIPack – is a collection of Internet Active X controls written by Crescent. We use only the RAS control to dial up (using Dial-Up Networking) when necessary.

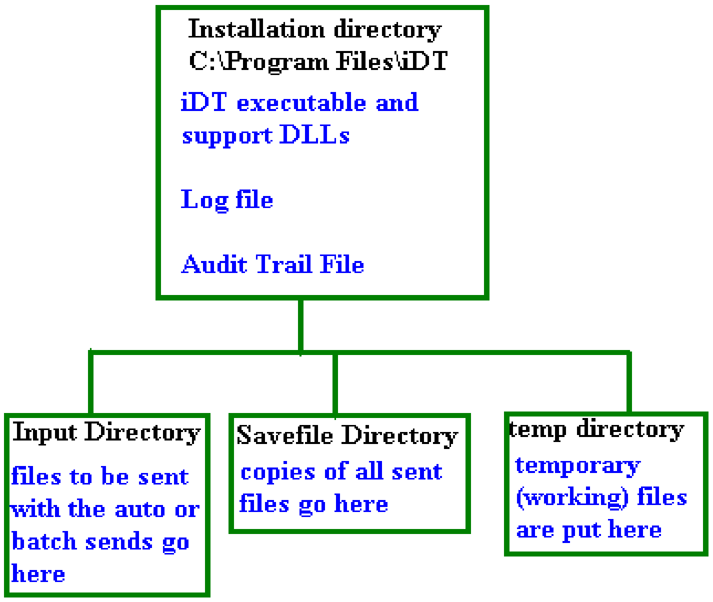
Spread – is a third party spreadsheet, more functional than the VB built-in one.

1.2.4 Installation Issues

The installation for iDT is done using Wise.

1.2.5 Installation tree for iDT

When iDT is installed, all the files and directories are kept together in the same directory tree. Note that the location of the files (log and audit trail), as well as the input, save and temp directories are under the main installation directory (and with those exact names), but they can be changed through the UI.


Installation directory C:\Program Files\iDT
iDT executable and support DLLs
Log file
Audit Trail File
Input Directory
files to be sent with the auto or batch sends go here
Savefile Directory
copies of all sent files go here
temp directory
temporary (working) files are put here

2 iDT Program Flow

When a file is selected for processing it goes through various steps. The file is validated to determine whether it should be sent at all (not used for ALL Clients). Then the "hash" is calculated. The file is then compressed and encrypted. A header is created containing various pieces of information, and it is also encrypted. Finally, a clear-text header is added. The resulting file is then transmitted. Each of these steps will be described below.

2.1 File Selection

Files can be selected either manually or automatically. Manual selection is done with a multiple selection standard File Open dialog. Note that while you can select multiple files, they all have to be of the same File Type. Sending multiple File Types would require doing separate sends (by default, the standard build only allows for a single File Type). The default directory for the file open dialog is the "Load" directory specified in the setup dialog.

The Auto Send command automatically selects and sends all files in the "Load" directory. They are all sent using the default File Type specified in the options dialog. You cannot send multiple File Types using Auto Send unless it is done through validation described in section 2.2.4 (again, by default, the standard build only allows for a single File Type).

Note that in interactive mode (**not in batch mode**) if you select or have more than 50 files in your load directory, the program will ask for confirmation before processing the files. This is intended to protect users against the possibility of setting their load directory to something that they should not (e.g. the Windows directory, the TEMP directory, etc.) in which iDT could RENAME and MOVE any files that are selected and validated – this will be described in grater detail in the following sections.

2.1.1 File Types

File Types were added into iDT for version 3.0. The idea behind file types is that different users/projects would be able to use iDT to send data to MSA. When iDTin processes the files, it knows to put them in their own place based off their file type.

2.1.2 File Types File for iDT (MSA's Processing End)

Any files sent with the older version of iDT will default to having a filetype of "IDT" when they are processed.

The preserve name flag is set **only** for filetypes that need the original filename preserved.

The FTP Transfer Mode defines the transfer mode for the transfer of the clear (decrypted) file to the final host. Since these files are usually text and the final host may need to translate end-of-line characters, the default is TEXT mode. For binary files, the keyword BINARY should be used. Note that this format will be used for all files of this type. Only the "BINARY" keyword is recognized. ASCII, blank, or anything else will default to an ASCII transfer.

The Ignore Test Flag can be used to force all test transmissions to be treated as production transmissions. Instead of going to the test directory, they would go to the appropriate production directory. The only recognized value for this flag is "IgnoreTest".

2.2 Validation (iDT)

The original iDT program was intended to send "watch" files as slated to be named "WatchOut". The "watch" format is a specific format supported by the MSA Data Factory. There was some validation included to assure that only watch files were sent and that they had some basic data (product code, Week Of date). During that early development, the goal changed so that the program would be able to transmit any type of file. The name was also changed at that point to iDT. The validations were removed, although some code existed to extract the project codes and dates from a few known formats (watch, stars and ewatch formats).

In the latest generation of the program (3.01), validation was added back in, and made much more flexible. Rules for validation are now defined by an external file that allows file types and checks to be added without having to make code changes. Validations include checking for certain characters in the file, location and format of the Week Of dates, record lengths and project codes. Files that fail validation will not be transmitted. This was done to protect from sends to the MSA that would flood the system (someone mistakenly sending their entire mainframe on to MSA ☺).

When validation is tightly enforced, any file that does not meet the validation requirements will not be sent. Any unrecognized file (based on the validation rules) will not be sent. There is an option in iDT to be able to disable validations (i.e. allow unknown files and/or files that fail validation to be sent anyway). This option will be in the INI file and will have a UI in the setup screen, but not visible or editable to the end user. Note that this override will allow files to be sent, but will still generate the error messages for failed validations. To allow any file to be sent without any validation, the validation rules file must be set up to allow such. MSA maintains the ability to adjust/modify Validations. Again, the end user has no ability to augment these settings.

2.2.1 Validation Definition File Format

Note that only the first "line" of the file to be checked is read before doing the validation check. The first "line" is defined as the first N characters of the file. N is the longest record length specified in the validation file, so we are sure to read enough characters to determine the file types. For files with shorter records, we may be reading more that one physical line. The reason for this is that if a binary file were selected, the program would crash if we just tried to read the full first physical line (which could be the whole file).

The first five fields in the file are used to determine the file format. These fields must exist. If these fields were both blank/zero, then any file would match that format description (turns off validations). We identify a file format based on the "match" characters (somewhere on the first line of the file) and the record length. Either or both items will be used, depending on the format definition in the validation file.

2.2.2 Turning off validations

There is the possibility that MSA may want to allow a Client the ability to send any files. In this case, the validation file can be set up with a single entry that will allow any type of file (even binaries) to validate cleanly, and show up with a "forced to" project code. Note that this could be combined with other "known" types so that the known types would get validations, but anything not known would be allowed through without errors. Again, MSA maintains the ability to adjust/modify Validations. The end user has no ability to augment these settings.

2.3 Validation (iDTin – MSA's End)

iDTin also does some validations, but different from the ones done by iDT. It uses both the validation file and the customer registration file. The validation file is used to collect info about the file, while the customer file is used to determine if the customer is allowed to send that file. This is done so that a customer cannot send a file with information that belongs to another customer, unless they are permitted to do that specifically.

2.3.1 Validations using the Validation file

When iDTin reads an incoming file, it checks it against the validation section of the configuration file. However, it does this only to extract some information from the file (file type, ID, etc). One of the items it extracts is the "File ID". The file ID is the customer ID contained within the file. While the customer ID in the file name and file header is the ID of the sender, the ID inside the file is the ID of the customer that the data belongs to. This could be different from the "sending" customer ID, since the same iDT could be used to send files from different customer IDs.

2.3.2 Validations using the Customer Registration file

There are three types of validations that are done if Customer Registration validations have been enabled. The first is to check if the customer is in the list of registered customers. The "sending" customer ID is found on the filename, and on the file header. This ID must exist, or the file will immediately fail and not be decoded.

If the sending customer ID is valid, the file will continue to be processed. Once it has been decrypted and uncompressed, the "file ID" will be extracted, using the field defined in the config.dat file. This "file ID" is the customer ID that the file belongs to, and may not be the same as the sending customer ID. If the file ID is the same as the sending customer ID, then the file is processed. If not, it is checked against all additional file IDs that the sending customer is allowed to send (this list may include wildcards as defined below). If the file ID is in the additional list (or matches a wildcard in the list), then the file is processed. Otherwise, it becomes an error transmission.

Finally, there is a TEST override in the customer registration file. The customer may be set up so that any valid transmission for that customer becomes a TEST transmission, even if sent as a PRODUCTION transmission. Note that this override will apply to all file IDs that the customer is allowed to send. This is used by MSA if in the event an End User starts transmitting "Validated" bad data. MSA can force data to prevent automatic processing until such time that MSA Helpdesk representative has worked out data integrity issues with the End User.

The TEST override for a customer will take precedence over the filetype ignore test flag.

2.3.3 Customer Registration File

The file is a comma-separated values (CSV) file (customers.dat) that contains information on all customers allowed to send files via iDT. If Customer Registration checks are enabled, only customers in this list are allowed to send files. Files from these customers will be processed, while others will be treated as errors.

This file will not be included in the iDTin "setup.exe" executable, and will not be automatically installed by the installation program, even if included in the CD or installation directory. The file can be created and maintained through the UI of the program however. Note that if the file is changed through the UI, the new values are reloaded automatically.

2.4 Compression

The files are compressed using ZIP compression. The zip files generated can be opened with any PKZip compatible zip program, like WinZip. However, this zip file is only temporary, and will be deleted after being used. Note that if ZIP is used as the encryption algorithm, a zip password is put on the file, but that option (i.e. using a zip password as the encryption method) will not be used, except for testing in emergencies.

2.5 Encryption

Once the file has been compressed, it will get encrypted. While technically the program supports various types of encryption, it is always used with the most secure encryption we support, which is Blowfish. Other encryption options exist only for testing.

2.5.1 Blowfish

Encryption is provided via the Blowfish encryption algorithm. The key length can be up to 448 bits which is far outside of the near real time crack range which is around 56 bits as of January 1, 2003. When developing iDT, we looked at various encryption algorithms, including PGP, RSA DES, and others and eventually decided on Blowfish for its level of security and adaptation.

Blowfish is a private (or single) key encryption, which means that the same key must be used for encryption and decryption. We have the option of changing the way we generate the key. Should we change the encryption method in a future version, this can be used to maintain backward compatibility. iDT currently utilizes the maximum encryption level of Blowfish (448 bit encryption).

2.5.2 Other Encryption Options

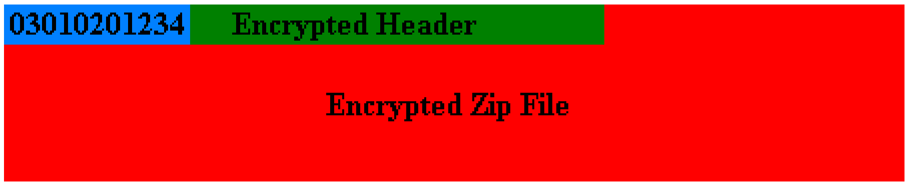
You can set up the program to have no encryption at all (just send the zip file), or just to have a zip password added to the zip file (this password is hard-coded). There is no UI to change this, and it exists only to allow overrides for testing, should there be a problem with a customer. Customers should NEVER be sending files without full encryption.

2.6 Transmission

2.6.1 Creating the Transmitted File

The actual file to be transmitted has three parts. The first 10 bytes of the file are a version header (unencrypted) that tells iDTin what version of iDT encrypted the file. This version header includes the iDT version number (to determine what encryption was used) and the length of the header (so it can be stripped off and decrypted separately from the rest of the file). The next piece is the encrypted file header (described later). Finally, the encrypted zip file is appended.

So the final transmission file looks like this:



The first 6 bytes (unencrypted) are the version number for iDT (in this case 03.01.02). Only two digits are used for the build number, since the format was defined before we went to 4 digit build numbers. The next 4 bytes are the length of the encrypted header. In this sample, the header is 1234 bytes long. The rest of the file is the encrypted version of the compressed original file.

2.6.2 Transmitted File Name

The transmitted filename specifies the Customer ID (needed for decryption, and to verify the header information), the file type, the project code (or "none" for file types with no specific project code), the control number, and whether the transmission was a test or a production transmission.

2.6.3 File Header

The file header is added to the file to allow for "extra" information to be sent in the transmission. The header includes things like the file type and the customer id. While these values also exist in the transmission file name, they are put in the encrypted header as an extra security measure to insure that the filename has not been tampered with). The header also contains the file signature or hash (for checking the file integrity), the original filename (in case the preserve option is set in iDTin), and the customer email (for auto-notification), the timestamp when the file was sent, and the hash value for the intermediate zip file. The header is variable length and consists of comma-delimited keyword=value pairs. It is limited to 9999 bytes.

2.6.4 File Signature (Hash Value)

One of the things we transmit in the header information is the file signature or "hash" value. The file hash value is a type of checksum for the file (like a CRC checksum). After a file is decoded, its hash value is calculated and checked against the original file hash value. If they do not match, the file is considered corrupt. It serves as a type of digital signature that ensures that the file has not been corrupted or tampered with.

2.6.5 Control Numbers

Control numbers are used so that every transmission will have a unique filename. Separate control number sequences are used for test and production transmissions i.e.. if you send 5 test transmissions (e.g. 11-55), the next production transmission will not follow that number, but go to the next production control number (e.g. 23). So the same control number may exist for test and for production. However, the filenames are still unique since they include a "T" or "P" in the name.

2.6.6 Test vs. Production Transmission Types

iDT supports two different types of transmissions, test and production. Each transmission can be sent using one of two transmission protocols (FTP and HTTP). Either type of transmission can be sent with either protocol.

The only difference between a test and a production transmission in iDT (other than the control number sequence) is that the filename will contain a "T" or a "P". However, these files are treated differently by iDTin. All test transmissions go to a test directory on the back end server. Normally, new customers may be instructed to send one or more test transmissions, to make sure that everything on their end is working correctly. If these files went to the production area, they would be processed automatically. By sending them to the test area, the files can either be discarded or be processed by a test system, evaluated, and if applicable, moved to the "Production" system for automatic processing.

2.6.7 Transmission Protocols

The original iDT implementation only supported FTP transfers. While this works well most of the time, there are many customers that are behind firewalls. For them, using FTP would require either having the firewall changed (often not feasible), or having to use a separate Dial-Up connection (slow).

The HTTP transfers were implemented in version 3.0 to allow users to transmit using HTTP instead of FTP. The reason for this was that most firewalls are configured to allow HTTP traffic, either with or without a proxy server.

2.6.7.1 FTP

FTP transfers are the easiest in many ways, especially when testing. They only require that the receiving host have an FTP server with the appropriate user account and password. The account name/password can be either fixed (not changeable by iDT users) or it can be set up to be editable via UI by the iDT user. This is controlled via MSA's installation build process.

As of version 3.03 of iDT, it is possible to specify the use of "passive mode" FTP transfers (PASV). Passive mode transfers are sometimes useful in transmitting through a firewall. They should only be tried if the normal (PORT) FTP transfer is not working. ***This option, while existing in the software, is disabled (the UI for it is hidden) as it is intended for future use***.

Note: As stated earlier, iDT has the ability to be configured to supply a Firewall username and password (disabled by default) however, using the HTTP transfer may be the preferred solution when trying to go through a firewall.

2.6.7.2 HTTP

The HTTP transfer uses the same mechanism as web pages to transfer the file. It can be set up to use a proxy server. As far as the firewall is concerned, it looks just like a web page download. This means that it can usually go through firewalls (with or without proxies), since most are set up to allow web page transfers.

With the HTTP functionality, we could have eliminated FTP (HTTP will work with or without firewalls and with or without proxies). However, FTP was left in both for backward compatibility and for backup. Also, we did not want to change how current customers were sending files.

The HTTP transfers require more significant configuration in the receiving host (MSA's End). The receiving machine must be running a web server that is capable of running Java servlets.

To solve the problem of iDTin not picking up partially transmitted files, the HTTP servlet transfers the files to a different directory, and then moves them to the actual transfer directory when the file is completely transferred.

3 iDTin Program Flow (MSA's End)

The program flow for iDTin is pretty similar to the flow in iDT (except backwards, of course). This section will concentrate on the differences between the two.

The flow of data through iDTin can be broken up into three pieces. The heart of the program is the part that decrypts/uncompresses the files. This part runs totally locally on the iDTin machine. The other two pieces are to get the encrypted files from the receive host to the iDTin machine, and finally to send all the processed files to where they have to go.

3.1 Internal processing

The central part if iDTin runs locally on a single machine (any 32-bit version of Windows). It starts by getting any encrypted files in the received directory and trying to process them.

The first thing that happens is that the file name is checked for valid format. From the file name we extract the customer ID, file type, project code, control number and test/production flag.

If all is OK, the version header is read off the file, to tell us what version of iDT was used, as well as the length of the file header, which is needed to know how many characters to strip out for the header.

Next the file header is stripped from the encrypted file and decrypted. The customer ID and file types are checked against the file name, to make sure everything matches.

The remaining part of the encrypted file is then decrypted, resulting in a zip file. This file is then unzipped. If everything had gone fine by this point, this unzipped file should be the same as the original file.

The file signature for the file is calculated and compared against the saved signature in the file header. If they match, we consider the process successful, and the file is copied to the clear text directory (for production transmissions) or to the test directory (for test transmissions). A copy of the original transmitted file is copied to the encrypted files directory.

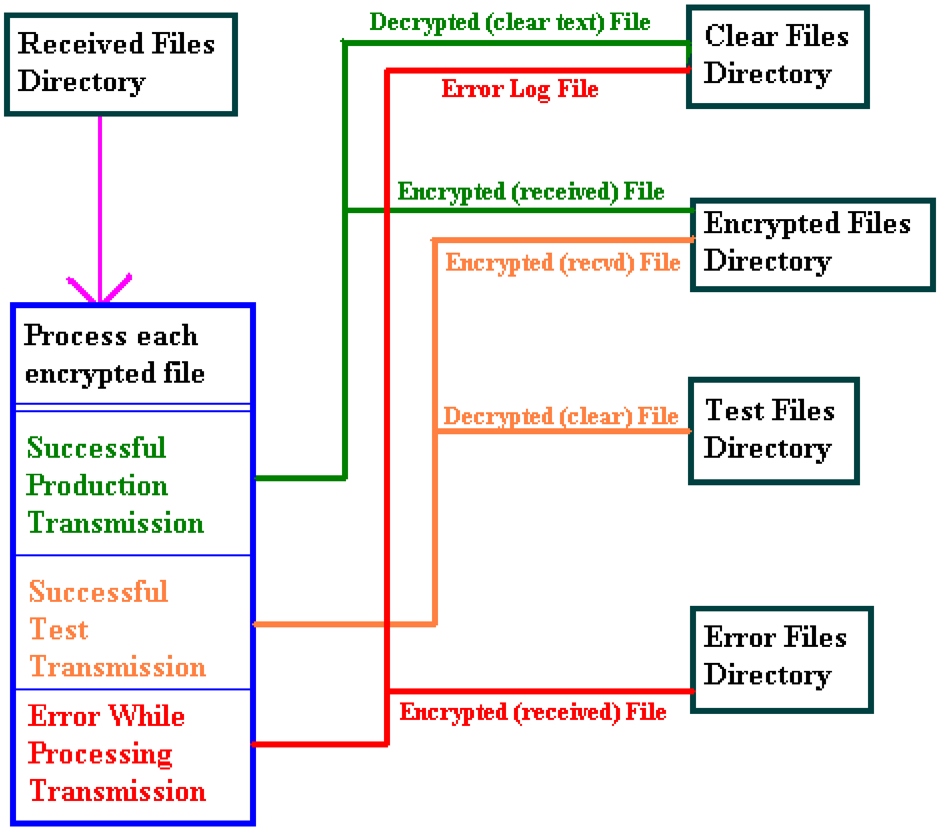
If any errors are encountered in the process, then the original file is copied to the error directory, while an error log file (with the same name as the original file, but with a ".log" extension) is sent to the clear text directory.

So for every file transferred, there should be a file (either the decoded file or the error log) placed in the clear text directory (for production) or in the test directory.

The generation of an error log file for each file with an error can optionally be suppressed in **SETUP|OPTIONS**.

The actual directory locations and names can be set by the user in **SETUP|OPTIONS**.

The following diagram is not necessarily complete.


Received Files Directory
Decrypted (clear text) File
Clear Files Directory
Error Log File
Encrypted (received) File
Encrypted Files Directory
Encrypted (recvd) File
Process each encrypted file
Decrypted (clear) File
Test Files Directory
Successful Production Transmission
Successful Test Transmission
Error Files Directory
Encrypted (received) File
Error While Processing Transmission

3.2 Getting files from the FTP/HTTP server

While it is possible to select one or more local (i.e. already on the iDTin machine) files for processing by using the Open command, this is not what typically happens in a production environment. For the production environment, iDTin will get the files to be processed from the FTP/HTTP server. Note that both the FTP and the HTTP servers should be putting the received files in the same directory, since iDTin can only get files from a single location. The directory on the server must be accessible via FTP, since that is how iDTin gets the files. It will open an FTP connection and get all the files with a ".wo" extension (to avoid getting partial files or files in the middle of a transfer.

Once all the files have been moved to the local received file directory, and deleted from the server, the processing happens as described above.

3.3 Sending processed files to the final host

Finally, once the processing cycle is finished, the files in the various directories are sent to their final. FTP is again used to transfer the files from the iDTin machine to the final host. Files in the error, test, and encrypted directories are sent to the respective directories on the final host.

While there is a directory specified for the clear files, that directory is only the "root" directory for the clear files. Usually, there will be various subdirectories under that one, and files of each type (see filetypes description) will go into different directories.

Finally, some error logs where the file type cannot be determined will be put into the "root" clear directory

Note that the same FTP account is used to send the test, encrypted and clear files, but the error files (NOT the error logs but the files that caused the errors) can be set up to use a different FTP account (even a different host).

3.4 Keep-Alive File

One of the issues with iDTin is that it usually runs unattended on a server somewhere in a computer room. So if no files were received at the destination host, it would be hard to tell if there had been no files received, or if iDTin had crashed and was not processing files at all.

To solve this problem, iDTin has a "keep-alive" file. iDTin runs through a cycle every XX minutes as setup by the iDTin Administrator (described in 3.5) and generates a keep-alive.log file. This file will be placed in the clear directory. At this point, it is then transferred to the clear_text specified directory on the final host. If the file has not been updated with the XX timeframe setup by the iDTin Administrator, then iDTin is either hung up or crashed.

3.5 IDTin Cycles

Normally iDTin is running full time, checking for new files that have arrived. The way it works is by running processing cycles every X minutes (where X is settable in **SETUP|OPTIONS**). Each cycle consists of getting all the available files from the receive host, processing them all, and sending all the processed files to the final host. The timer restarts after the whole cycle is finished. So if the cycle is every 30 minutes, that means 30 minutes after the previous cycle finished, and not 30 minutes after the last cycle started.

4 Configuration File

As of version 3.03 of iDT and 4.02 of iDTin the configuration options that used to exist in various files (filetypes.dat, validation.dat, extensions.dat) are now combined in a single file (config.dat), with sections corresponding to each of the old configuration file.

This file is encrypted, and therefore, the user cannot tamper with it. The only way to create/edit the configuration file is using the Admin Tools application at MSA.
To make matters simpler, the configuration file for both iDT and iDTin (and the editor) are the same format. This means that there may be extra fields that are not used in iDT (e.g. some of the fields in the file types), or even sections that are ignored (e.g. if there is an "Extensions" section, iDT will not use it) but allows for maintenance of a single configuration file.

5 Log Files

iDT and iDTin keep various log files, to help in tracking down activity and/or problems. There is a "standard" log, which is what the user normally has access to. In addition, there is a debug log, and logs for FTP and HTTP.

Additionally, an Access Database was built into iDTin that maintains information on all files processed via iDT that can be used for querying and eventually tied in to Customer Service Databases maintained at MSA.

5.1 Standard Log

The standard log will log every line that gets displayed in the status window to the log file set in **SETUP|OPTIONS**. For iDT, this log is "permanent" (i.e. it gets appended to forever). For iDTin, the log gets archived to a zip file and restarted on a weekly basis. The View Log command works on this log.

5.2 Debug Log

The debug log is not a separate file, but rather additional information that is sent to the standard log (lines are marked with "DEBUG"). Debug information is copious, and should not be left enabled for any extended period. It should only be used when there are problems that need to be identified.

For iDT, the debug information can only be enabled by adding the "/Debug" command-line argument. For iDTin, the command-line argument will also work however, there is also a checkbox in **SETUP|OPTIONS** that allows the debug information to be toggled on/off on the fly, so the program does not have to be restarted. Note that the value of this checkbox is not "sticky". If the program is restarted, the debug information will be enabled automatically only if the command-line argument exists.

5.3 FTP/HTTP Log

There is another log (or logs) that are created by the application. These logs are created by the DLL that handles the FTP and HTTP transmissions and contains the commands sent and responses from the FTP server, as well as some other information from the HTTP transmission.

For iDT, the FTP log is a temporary file (in the temp directory) that gets overwritten every time it is used. If there is a problem, the file will contain a log of the last session only.

For iDTin, the log is more permanent. It is kept in the same directory as the standard log, is archived and reset weekly, and is added to the same zip file as the standard log.

5.4 Audit Trail

The Audit Trail file (settable in **SETUP|OPTIONS**) is a comma-delimited file that contains a single line for each file processed. It can be viewed from within the program, or loaded into Excel (or similar program) for analysis. Note that all the fields in the file are not necessarily displayed when the user looks at it through the program UI.

As of version 3.03 of iDT, the audit trail file at the customer sites is encrypted. New entries to the file are encrypted before adding them to the audit trail. The file is decrypted only when the user needs to display it and can do so only through the iDTout interface.

6 INI file

All of the options used by iDT/iDT In are saved in the INI file. For security reasons, as of version 3.03 of iDT, the INI file at the customer sites is encrypted.

NOTE: as of version 3.04 iDT contains a function to allow the End User to easily select the Audit Trail and iDT.ini files, zip them, and send them via email to an MSA representative if in the event such action is required (most useful in troubleshooting transfer issues). This function is accomplished by the End User simply clicking on the "Send Dat files" button in the GUI.

6.1 Hashed passwords

For the login passwords for multiple users in iDT (if this feature is enabled), the program uses a one-way hash, similar to what is done in UNIX systems. Only the hashed value (a large number) is stored in the INI file. You cannot reconstruct the original password from the hashed value. When the password is entered, it is hashed, and that value is compared to the saved hash value. If they match, the password is OK. This type of hash is very secure, since it cannot go backwards (from hash to password).

7 Miscellaneous Features

This section contains descriptions of some of the other features found in the program.

7.1 User Logins (iDT)

iDT supports the ability to restrict use by user logins. This feature was added in case the program was installed in machines or locations where multiple users have access. Adding a login name and password would restrict use. The program comes pre-configured with a single user "admin" with a password "admin". This password can be changed, and other users added. Note that the list of user names and passwords is in the INI file. If the INI file got deleted, access would again be allowed. So this is not very restrictive, it is mostly administrative.

7.2 Error File Archiving (iDTin)

There exists an error archive directory, separate from the "regular" error directory. Any files with errors are copied to this directory also. It exists so that error files can be kept locally on the iDTin machine, since files in the regular error directory get sent to the error host and then deleted from the local machine.

There is a setting (in Setup | Options) that indicates how long to keep the files in the error directory (in days). Any files older than this will be deleted automatically by a cleanup process that runs when iDTin starts, and then every processing cycle. Note that the file modification date (which is used for this check) does not change by copying into the error archive directory. If a file is a week old, and then gets processed and copied to the directory, it will still be considered a week old, for cleanup purposes.

7.3 Error Message emailing (iDTin)

This feature, added for version 3.01 allows a single email address (that can also be a distribution group) to be set up to receive error messages from iDT. This address must be a valid internet format address (xxx@yyy.zzz). The program does not use MAPI nor does it require that there be any mail client on the machine.

At the end of each "session", if any errors were logged, they get sent in an email. Only one email is generated per session. The message text is exactly the same as the error messages that get displayed in the status window.

Note that the mail server can be configured in the Options dialog. The "From" address that the mail will use is also configurable and should be set to a valid address, in case the mail is undeliverable.

7.4 Customer Verification (iDTin)

This feature allows iDTin to only process transmissions for certain user IDs. Each user ID can also have a list of other IDs that it is "allowed" to send.

7.5 Instances/Concurrent FTP Transfers (iDTin ONLY)

There is an option in iDTin (as of version 4.02) to allow the program to "multi-task" the FTP transfers. When this option is enabled, iDTin uses three different executables (actually three copies of the same executable, but with different names to allow them to run concurrently). This feature provides a crude multi-tasking ability to iDTin, so that the three parts of the execution cycle (getting files, processing the retrieved files, and sending the post-process files) do not slow each other down. This way, for example, if the sending part is slow due to volume and/or network issues, the main executable can still be processing new files as they come in, without having to wait until all the files from the previous cycle finish transferring.

The "main" executable (idtin.exe) will run as usual. This is the only executable that should ever be started (manually, or via a scheduler or batch process). When it reaches the part of the processing cycle where it needs to get newly arrived files from the receive host(s), instead of doing the transfer, it will spawn a separate executable (idtinGet.exe) which will do the gets. Similarly, at the end of the processing cycle, it will spawn a separate executable (idtinSend.exe) to do the sends of all the files to the final host. There will only be a single instance of each of these separate executables at any one time (similar to how only a single instance of the main one is allowed). The main program will try to spawn these secondary executables every processing cycle, but if they are already running, the new instance will not run again.

7.6 Firewall Authentication

There is a setup screen for the authentication name and password. If these fields are filled in, then iDT will "log in" to the firewall using the authentication values, and then log in to the FTP server, using the FTP account user name and password. If the fields are empty, the authentication step is skipped. By default, these fields are hidden in the installation as the Firewall Authentication functionality does not work with ALL firewalls. Any End User wishing to attempt to utilize the Firewall functionality should contact their MSA Helpdesk Representative.

8 The iDT HTTP servlet

8.1 Program Flow

The purpose of the iDT servlet is to allow iDT to send files using HTTP instead of FTP. The servlet runs in 2 modes, based on the HTTP operation (PUT or GET). The normal mode, used by iDT, processes files sent using an HTTP PUT. If the servlet is called using an HTTP GET (typically by specifying the servlet URL in a browser), it returns an HTML page with configuration information; this can be used to ensure that the servlet is running and has been configured correctly (see section 8.2).

8.1.1 Initialization

The servlet performs one time initialization functions the first time it is accessed. It reads the configuration file (idtservlet.dat, in the same directory as the idtservlet.class file), a comma-separated file with 2 directives. The SentPath directive identifies the full path where the files sent by iDT are to be copied. The IniPath directive identifies the full path where INI files, to be sent back to iDT, are located. The servlet will also create a temporary file directory ("tmp") under the SentPath directory if it does not exist. Note that the INI file functionality is not currently in use. It was intended for automatic updates of the config.dat file (in spite of the "INI" name) but its future use is not certain.

8.1.2 HTTP Post Processing

iDT uses an HTTP Post to communicate with the servlet. It sends an op code to identify the operation to perform.

The SendFile op code is used to send a file. The servlet simply copies the file to the SentPath directory.

The GetIni op code is used to send an INI file back to iDT. iDT sends a user id to identify the file. The file named <user id>.ini, residing in the IniPath, is sent back to iDT.

The servlet always sends back a status code and status message. If successful, the status code is 0 and the message is OK. If the operation failed, the appropriate status code and message are sent.

This is also used to test whether the servlet is running. A dummy opcode is sent, and if we fail the connection command, the servlet is not responding. If it responds, then it is reachable. We ignore the error return.

8.1.3 HTTP Get Processing

A browser can be used to access the servlet, by typing in the servlet URL. The browser sends an HTTP GET, which is how the servlet differentiates between access by a browser and by iDT (which sends an HTTP PUT).

If the servlet is accessed using an HTTP GET, it sends back an HTML page with 2 tables containing configuration information.

The first HTML table identifies the paths used by the servlet – the SentPath and IniPath in idtservlet.dat and the tmp directory. It also indicates whether or not these paths exist.

The servlet also runs two tests that simulate a SendFile operation, to ensure that the permissions are configured correctly on the server. The first test sees if a temporary file can be created in the tmp directory. The second test sees if a file can be moved from the tmp directory to the SentPath. The second HTML table indicates the results of these tests.

8.1.4 Temporary File Cleanup

Starting with servlet 1.3.2 of the servlet, the servlet includes cleanup of the temp files used by both the servlet, and by the FTP server. The servlet temp files are in the tmp directory defined in the servlet configuration file. The FTP server tmp files are files in the destination directory that have a .TMP extension.

By default, the cleanup is run no more than once per day, and files older than 5 days are deleted (all files from the temp directory, .tmp files from the sent files directory). There is an optional directive you can put in idtservlet.dat ("FileCleanUp, 1, 5") to change the defaults. The first parameter is the number of days between runs, the second the max age in days of files before they are deleted. For debugging, you might want to change these values to 0. Note the values are displayed when you run an Http Get (and both the Http Get and Http Post will cause the servlet to check and see if the cleanup should be run).

9 Troubleshooting

This section will mention a few of the most common problems we often see. It is not meant to be an exhaustive list, but to give a feel for what tends to go wrong.

9.1 Typical problems with iDT

- User cannot FTP. This is often caused by a firewall. There are various solutions to this. The firewall can be tweaked to allow straight FTP access, the user can switch to HTTP (preferred solution), or the user can use a Dial-Up account.

- User cannot start the program. This is usually a VB installation issue. If any other program has clobbered a common DLL with an earlier version, the program may not run. Reinstalling often fixes this. Newer versions of the shared DLLs are usually OK.

- In an operating system environment (such as Windows 95) that runs the AUTOEXEC.BAT file at startup, some programs will modify the AUTOEXEC.BAT file by renaming the TEMP file (e.g., SET TEMP = C:\DOS_6.22). This can affect the installations of iDT and iDTin, which use the Windows TEMP directory. At the end of the installation, an error such as the following could appear: **"Could not load the DLL library C:\DOS_6.22\GLF162.TMP. One of the library files needed to run this application cannot be found."** To avoid this problem, either remove the "SET TEMP" line from the AUTOEXEC.BAT file, or change the "SET TEMP" line to use the Windows TEMP directory.

10 Appendix A: Error Messages

This section will describe the different error messages and result codes that may appear during the use of iDT and iDT In. Some of these codes may appear only in "debug" mode. Some may be specific to iDT or iDT In, and will be identified as such.

There are also some messages and codes here that should never be seen by users since they are used to flag internal problems (which should not happen in released code). But they will also be included for completeness.

The errors will be shown in the different areas where they appear.

10.1 Startup (iDT)

These errors may be displayed when the application starts up. Also see the tampering checks for other startup errors.

You have illegal values in your options. Please correct before proceeding.
One or more errors were found in the options. The specific errors for the options can be found in the Options section.

10.2 Startup (iDT In)

These errors may be displayed when the application starts up. Also see the tampering checks for other startup errors.

You have illegal values in your options. Please correct before proceeding.
One or more errors were found in the options. The specific errors for the options can be found in the Options section.

Could not reload File Types file.
Could not reload Extensions file.
These files get reloaded every run cycle (as well as at startup). Any problems loading will generate the error

Log backup operation failed.
Failed trying to back up the current log to the backup zip file.

Could not delete file from error archive: <filename>
There was a problem trying to delete an "expired" error file from the local error archive (e.g. the file is read only or there are permission problems).

Size for <directory> is too large.
The current size of the directory exceeds the limits set in the options dialog.

10.3 Sending (iDT)

These errors may be encountered while sending files from iDT.

Could not connect to any host.

This is a very generic "catch-all" error that will happen any time the program fails to contact the primary host (or the secondary, if defined). It may be preceded by another more specific error, depending on why it failed. This may happen for both FTP and HTTP transmissions.

Invalid customer number found: <customer ID>

Indicates an illegal customer ID.

File <filename> not found.
File <filename> is empty.

The file selected is either missing (deleted) or empty (no reason to ever send an empty file).

Error validating file, cannot continue.

There was a validation error (which should have preceded this message). See the validation section for those errors.

Error Zipping file, cannot continue.

The zipping process failed. This should normally never happen.

Error encrypting file, cannot continue.

The encrypting process failed. This should normally never happen.

Customer ID required to process files. Cannot Continue.

The customer ID is missing.

Failed transmitting file <filename>.

The transmission failed. May be preceded by another more informative message, depending on the transmission method and the reason for the failure.

File <filename> could not be deleted.
File <filename> could not be copied.

After sending the file, it gets copied to the sent directory and sometimes deleted (depending on the options). If either operation fails, you will get these errors.

10.4 Receiving (iDT In)

These errors may be encountered when receiving/processing files in iDT In.

File <encryptedFile> not found.
File <encryptedFile> is empty.

The file to be processed is either empty or missing

Bad filename for file: <encryptedFile>. Cannot process.

The filename for the encrypted file is not in the correct format, and cannot be processed. Usually indicates a file sent to the server manually (not via iDT).

FileType <filetype> not supported.

The filetype for the file is not in the valid filetypes list for iDT In. Indicates the user had filetypes in their filetypes files that are not in iDT In's list.

File <encryptedFile> was sent with ZIP encryption only, but processed anyway.

Old versions of iDT could send files only encrypted with ZIP passwords. Newer versions no longer allow this.

Error decrypting file, cannot continue.
Header information is corrupt. Cannot Continue.
Error unZipping file, cannot continue.
Only one file expected in Zip archive. <number of files> found.
Failed to extract file from Zip archive: <zipFile>
File <zipFile> not a valid Zip archive. Original file may be corrupt.

These errors are usually caused by a file that got corrupted so that we cannot even process them. They need to be resent.

Original and decrypted files do not match (different hash values).
CustomerID for files do not match. <customer ID> vs. <header customer ID>
FileType <header filetype> & not supported.
FileType in header does not match filename. <filetype> vs. <header filetype>

These errors indicate that the information in the header, and the information in the filename or the processed file do not match. This usually indicates file corruption, or possibly, a file whose filename was changed.

Error copying clear file to: <savedClearFile>
Error copying encrypted file to: <savedEncryptedFile>
Unable to delete file: <encryptedFile>
Unable to delete file: <decryptedZipFile>
Unable to delete file: <unzipFile>

These errors occur when there are permission problems with the files or directories.

No email address. Could not send Auto confirmation.
Could not send Auto confirmation.

These errors can occur during auto confirmation. Since this feature is currently disabled, they should never show up.

Load directory not set in configuration.

The load directory is not set correctly in the Options.

Could not contact host: <host>

This error occurs when the program fails to connect to a particular host, either for getting or for sending files.

Customer number (<customer ID>) not registered.
Customer number (<customer ID> - <name>) cannot send file with ID = <fileID>

The customer is not set up correctly in the customer registration file, or they are sending with a customer ID that is not consistent with the customer ID within the file.

10.5 Customer Registration (iDT In)

These errors may be encountered when setting up the customer registration information in iDT In.

Name Duplicates: <name1> <name2>
MID/DID Duplicated (<customer ID>) for: <customer name1> - <customer name2>

There are either duplicate company names or customer IDs. Both have to be unique.

Blank customer name found for ID = <customer ID>
Customer ID blank for: <customer name>

There are either blank company names or customer IDs. Both have to exist.

Customer ID must be 8 characters for: <customer name>

Customer IDs must be eight characters.

<number of errors> errors found in customer list

Lists the number of errors that were in the list when the check is performed.

10.6 FTP

The following are errors that may occur during FTP transfers (sends or receives).

Unable to login to FTP server <host>
Unable to get FTP data port.
Unable to transfer local file <local file> to ftp server as <temp file>
Unable to transfer local file <local file> to http server.
Unable to login to FTP server <host>
Unable to get FTP data port.
Unable to retrieve <remote file>
Unable to delete file <remote file> on FTP server
Unable to find file to rename <temp file>
Unable to rename file <temp file>
Unable to login to FTP server <host>
Could not change to <host directory> directory.
Unable to get FTP data port.
Unable to transfer local file <local file> to ftp server as <temp file>
Unable to delete file <local file>
Unable to transfer files of type <file type>, remote directory <dir> does not exist

These are basically the errors that the FTP server has returned, with a little extra information.

An error occurred while processing file type: <file type>, cannot continue

This occurs if there were errors trying to process a particular type (i.e. trying to send files of a particular type to the final host).

FTP processing cancelled by user

The FTP processes was interrupted manually.

Unable to open data channel
Please specify an FTP Import location using Setup Options
Unable to obtain directory listing
Unable to establish connection
Password was not accepted
Unable to set transfer data type
Connection established to FTP Server
Getting FTP Server directory listing
Obtained directory listing for
Logged out of FTP Server

These are the actual errors returned by the FTP server. Usually you will not see these except in debug mode.

10.7 HTTP (iDT)

These are the errors returned by the HTTP servlet.

SendDataToServer failed.

This error usually indicates a failure to connect to the servlet.

Did not receive data file from server. Reason: <return message>

If you do connect to the servlet, or if you encounter an error, this error is displayed with the error message generated by the servlet.

10.8 Email (iDT In)

These errors may be reported when trying to send email from the application (both error emails and auto confirmations).

Unable to connect to SMTP server <server>

The program could not connect to the server at all. Can be caused by the server name or address being wrong, or by problems with the network or the server itself.

Email server did not accept message text
Email server did not accept MAIL command
Email server did not accept recipient: <email address>
Email server did not allow message data to start
Email server did not accept from/to/subject
Email server did not accept comments
Email server did not accept message boundary
Email server did not accept message body

These are internal errors generated by the email server. You only see these in debug mode. They usually indicate server problems.

10.9 Validation (iDT)

The following errors can be generated when a file is validated using the validation rules in the validation.dat file.

File Format is not valid.

The file format was not found.

WeekOf date: <date> is not valid.

The weekOf date was missing, malformed, etc.

ProjCode: <code> is not valid.

The project code in the file is not allowed or is missing.

Illegal characters found in Customer ID#. Only letters and numbers are allowed.
Illegal characters found in Customer ID#. Only numbers are allowed.

These two errors are for the generic and standard versions. The standard version only allows numbers, the generic one letters and numbers.

10.10 Encryption (iDT In)

These are errors that may occur while decrypting files in iDT In.

File was sent with no encryption: <encrypted file>
File was sent with ZIP encryption: <encrypted file>

File was sent with a low level or no encryption. This should never happen, unless the user has a very old version of iDT.

Could not get BF password for version <version>
IDT In cannot unencrypt files from iDT version <version>

iDT in cannot recognize the iDT version that sent the file. Indicates file corruption.

File header seems corrupt.

The file seemed to decrypt, but we cannot read the header. Indicates file corruption.

Error in filesize. Not a multiple of 8.

The file size is not consistent with the encryption method. Indicates file corruption.

10.11 Audit Trail

These errors may occur when displaying the Audit Trail.

Illegal status in Audit Trail: <status>

The status code for the entry was not recognized.

Error while reading Audit Trail: <error number> - <error description>

There was some error (exact error will be reported) trying to load the audit trail file. Indicates that the audit trail file is corrupt.

10.12 Error Codes and Statuses

The following is a list of status codes. These statuses appear in the error .log files sent to the clear directory by iDT In and in the audit trail for both iDT and iDT In.

	ErrorCode	Description
00	STATUS_ILLEGAL	Illegal Status
01	STATUS_SENT	Sent
02	STATUS_RECEIVED	Received
03	STATUS_PROCESSED	Processed
04	STATUS_FAILED_CONNECT	Failed Connecting
05	STATUS_FAILED_TRANSMISSION	Failed Transmission
06	STATUS_FAILED_VALIDATION	Failed Validation
07	STATUS_FAILED_ZIP Failed	Compression
08	STATUS_FAILED_ENCRYPT	Failed Encryption
09	STATUS_FAILED_DECRYPT	Failed Decryption
10	STATUS_FAILED_BAD_FILENAME	Bad Filename
11	STATUS_FAILED_FILE_NOT_FOUND	Filename not found
12	STATUS_FAILED_EMPTY_FILE	Empty File
13	STATUS_FAILED_AUTO_CONFIRM	Failed Autoconfirmation
14	STATUS_FAILED_CORRUPT_HEADER	Corrupt Header
15	STATUS_FAILED_HEADER_CHECK	Mismatch in Header
16	STATUS_FAILED_VERSION	Bad Version
17	STATUS_USER_NOT_REGISTERED	User Not Registered
18	STATUS_USER_EXPIRED	User Expired
19	STATUS_BAD_FILE_TYPE	Bad File Type
20	STATUS_DIRECTORY_DOES_NOT_EXIST	Directory does not exist
21	STATUS_CUSTOMER_CANNOT_SEND_ID	Customer cannot send files with this ID number

10.13 Dial-Up (RAS) (iDT)

These errors can occur when iDT dials out to the Internet.

Could not dial up.
Could not connect to any host.

Any time the dial-up fails, you just get the two generic errors above.

However, while dial-up is working, it displays the status on the status line, and that will give you more details. The Status line display is as follows:

Dial-Up: <process> / Status: <result>

Example:
Dial-Up: OpenPort / Status: SUCCESS
Dial-Up: PortOpened / Status: SUCCESS
Dial-Up: ConnectDevice / Status: SUCCESS
Dial-Up: DeviceConnected / Status: SUCCESS
Dial-Up: AllDevicesConnected / Status: SUCCESS
Dial-Up: Authenticate / Status: SUCCESS
Dial-Up: AuthNotify / Status: ERROR_AUTHENTICATION_FAILURE

10.14 Options (iDT)

The following errors may occur in the Setup Options screen. When the window is closed, various checks are performed.

Cannot select Dial-Up option without having a connection selected.
Log file name is required.
A default file type for AutoSend is required.
Valid e-mail address is required for auto confirmations.
The primary host name (or address) is required.
The proxy host name (or address) is required.
The proxy port number is required and must be a number.
The HTTP servlet URL is required.
The HTTP servlet port number is required and must be a number.

10.15 Options (iDT In)

The following errors may occur in the Setup Options screen. When the window is closed, various checks are performed.

Log file name is required.
Audit file name is required.
The Auto Process Interval must be a number.
The Archive Log Interval must be a number.
The Archive Log Dir Max Size must be a number.
Archive directory must exist.
Text directory must exist.
Test directory must exist.
Error directory must exist.
Load directory must exist.
Load archive Zip directory must exist.
Error file archive directory must exist.
The Error Archive Cleanup Interval must be a number.
The Error File Archive Dir Max Size must be a number.
Archive directory cannot be the same as the installation directory.
Text directory cannot be the same as the installation directory.
Test directory cannot be the same as the installation directory.
Error directory cannot be the same as the installation directory.
Load directory cannot be the same as the installation directory.
Load FTP directory cannot be the same as the installation directory.
Error email address is required if sending error mail.
email address is not formatted correctly.
SMTP server is required if sending error mail.
Error email from address is required if sending error mail.
email from address is not formatted correctly.