

White Paper

Advanced Import

Version 4 of ImageWare's IDentifier for Windows offers "Advanced Import," a powerful feature not present in previous versions. The addition of Advanced Import allows users to "map" disparately named database fields from one file to another, eliminating the need for tedious text editing and possible data manipulation prior to a successful import. In addition, Advanced Import allows users to import portrait and signature images from an external database into an existing IDentifier database. In short, the Advanced Import function essentially provides the ability to perform two new tasks: **Merge multiple IDentifier databases into a single database file** , and **import data and images from an external database into an IDentifier database**.

There are some critical issues that must be addressed and managed, however, before Advanced Import can successfully merge or import text data and images. Therefore, this document attempts to alert users to the basic issues that make the IDentifier's Advanced Import successful or not. Any attempt to perform an Advanced Import without reading, understanding, and taking the steps outlined in this document may result in undesirable results. It is recommended that users make verified backups of all database files before proceeding with Advanced Import.

Key Concepts

Before beginning this discussion, several key concepts must be enumerated:

IDentifier Application, IDentifier database—ImageWare Systems (IWS) develops a variety of software applications for creating digital photo ID badges. Two of the more popular "families" of products are **IDentifier for Windows** and **ID Server**. In addition, IWS develops similar products bearing different names for OEMs and other resellers. Some or all of these products *may* offer Advanced Importing.

Source— "Source" refers primarily to the *external* file whose data you wish to import into an IDentifier database. The word "Source" may additionally be used to refer to the images associated with that external file (the *source* images), as well as "path statements" directing the IDentifier Application to them.

Destination—"Destination" refers primarily to the IDentifier database into which you wish to import data. Additionally, the word "destination" refers collectively to all files or path statements related to the IDentifier database into which you wish to import data.

Primary Key, Key Field, Unique Key Field—Relational databases require a field which uniquely identifies each record in a database table. The "value" in this field must be unique—no other record in the given table may have the same value. Establishing "uniqueness" ensures that individual database records are not inadvertently related to the incorrect record in another table. Several terms are used interchangeably to denote this field: Key Field, Primary Key Field, and Unique Key Field.

IDNumber—IDentifier automatically generates serialized numbers for every record in its database, whether the record was created within IDentifier using the CREATE button, or by importing records from a Source file. Even if the record had an IDNumber in the original Source file, a new, unique number will be generated for it in the Destination file when it is imported.

ITCFields table—This is a table generated by IDentifier and is used for a variety of purposes. It records, for example, the date a record was created, how many times a badge was printed, and a serialized number uniquely identifying each record in the table, etc. **IDNumber** is the key field in the **ITCFields** table.

White Paper

UDFields table—This is a table modifiable by the end user and contains the variable data he or she wishes to record in the database, such as first and last names, social security number, and address, etc. The “UD” in the table name is an abbreviation for “user defined”—meaning the user can define not only which fields are in the table, but many of the individual field properties. Text data imported from a Source file is written to this table.

ImageID—The IDentifier Application uses a database field named **ImageID** in the **ITCFields** table to store the *name* of portrait, signature and fingerprint image files created by IDServ.exe. (IDServ.exe is the program running in the background that creates and saves image files when a user captures portrait, signature or fingerprint images.) Before an image is captured (saved to disk) for the first time, IDServ.exe generates a serialized numeric name for the image (e.g., ...121, 122, 123, 124...) and writes it to the **ImageID** field. It then uses that value as the actual filename when the image is saved to disk. IDentifier only generates a name *if the ImageID field is empty*. (Images captured after a value has been entered into the **ImageID** field simply adopt the value that is already present and are *not* renamed with a new serial number at each subsequent capture.)

Import—Importing data means “reading data from one file and writing it into another.” IDentifier can read data residing in two common file formats: comma delimited ASCII text files and data tables in Microsoft Access 97 or 2000 databases. The IDentifier Application then writes the data into its own data tables. (Some caution must be made regarding importing data from comma delimited ASCII text files. Ideally, the original text file should enclose all the data within quotation marks. Otherwise, commas located within the actual data [such as the company name “The Gap, Ltd.”] might create unwanted errors.) There are essentially two common approaches to importing data employed by most databases: APPEND and UPDATE.

- APPEND-type imports query the source data file and compare the value of its primary key to a primary key in the IDentifier database—any records found with *different* primary keys will be added or *appended* to the IDentifier database.
- UPDATE-type imports query the source data file and compare the value of its primary key to a primary key in the IDentifier database—any records found in the Destination database with *identical* primary keys will be *over-written* with the data from the Source file.

Merge—Merging data is “sort of” like importing. The process of merging, like importing, reads data from a source file and writes it to an IDentifier database. However, there is a critical distinction between the two: when merging, IDentifier will recalculate and write new values in specified fields when the Source file is imported.

- If a primary key value in the Source file *is not identical* to that in the IDentifier database, IDentifier will *append* those records to its database *but create a new primary key value based on its own internal numbering scheme and enter it into the IDNumber field of the ITCFields table*.
- When merging images, IDentifier will generate a unique serialized numeric name for the Source images and 1) write the name in the Destination **ImageID** field, and 2) rename the image with the same value when it is copied to the Server Data Path. Even if the Source image had a “name” or value in an equivalent **ImageID** field, IDentifier will generate a new ImageID in the Destination file and rename the copied image file accordingly.
- IDentifier will only generate a new **ImageID** for a Source image if the **ImageID** field in the Destination file is empty when the images are merged.

Field Definitions—“Field Definitions” refers to the collective description of the field names and field properties of an IDentifier database. More specifically, versions 3 and 4 of IDentifier allow users to customize many aspects of the individual database fields within the application itself. For example, users may add or delete fields, rename field labels, or modify individual field properties such as a field’s INDEX, UNIQUENESS, INPUT MASK, REQUIREMENT status, etc. IDentifier stores all the information about these field names and properties in a special database table called **UDfieldsdef**. IDentifier uses the information stored in this table to reconstruct the database’s FORMS (the graphical windows displaying the data) and **UDfields** table (the table containing the actual variable data, such as individuals’ first names, last names, etc.).

White Paper

Case Study: an example of how these issues present potential problems

A small school system wishes to maintain a single, central database of its students and teachers. However, it must send laptop computers and small digital cameras to various remote locations to “enroll” those individuals concomitantly with the enrollment of the population on the main campus.

The problem: The Identifier database on the main campus generates serialized IDNumbers (stored in the **IDNumber** field) for every record in the database. The number in this field uniquely identifies each record. However, Identifier running on each of the portable laptops *also* generates serial numbers for the records it creates, and populates the **IDNumber** field accordingly. As you can see in the graphic below, though the records describe different individuals, IDNumbers are duplicated in each of the databases.

If the central Identifier Application tried to merge the data and images from the two laptops into its own database, the “merge” or “import” could fail for the following reasons:

- **Situation 1:** The operator specifies the **IDNumber** field in each database as the key field to uniquely identify each record. In instances where the value in the **IDNumber** field is the same in both databases, 1) an Append-type import will refuse to add records from the Source file because it assumes that identical IDNumbers denote the same individual, and 2) an Update-type import will overwrite good data in the Destination file with the data from the Source because it is assumed that identical IDNumber values refer to the same individual.
- **Situation 2:** The operator decides to use a field *other than* the Identifier **IDNumber** field (for example, a field containing each individual’s Social Security number) as the key field to uniquely identify every record. However, even though it is *expected* that the Social Security number is unique, data entry errors result in the same Social Security number being entered more than once, or being omitted (left blank) altogether. Records where the value in the “unique key” field is missing or not unique are rejected for merging or importing.
- **Situation 3:** The operator specifies a database field in the Source file which contains unique values not duplicated in the Destination file. However, as one of the import options, the user elects to import data from a Source field into the Destination’s **ImageID** field. When the operator attempts to merge the images in a later step, Identifier fails to generate new values for the **ImageID** because the **ImageID** field already contains data from the previous text import. Therefore, when the Source images are copied to the Destination Server Data Path, they overwrite any *previously existing Destination images of the same name* that have an earlier date of creation.

How the Identifier Application Implements Importing

The Identifier Application presents a series of “user prompts” or “dialogs” to assist the operator in negotiating the various import and merge options available to him or her. Because there is some cross-referencing in this discussion between the various steps of the import process, we employ an *outline* format to assist you in relating one step to another.

Import Text Wizard: User Prompts

Step A: (select file-type) - consists of a user prompt displaying four types of files that may be imported:

A-1: *Compatible IFW text file*—a comma-delimited ASCII text file generated by an Identifier Application. It is assumed that the Field Definitions of this file are identical to the Field Definitions of the Destination file. This file will always use **IDNumber** as the key field ensuring successful APPEND-and UPDATE-type imports.

A-2: *Text file*—any other comma-delimited ASCII text file generated by a third party application. The operator will be required in Step E to specify a key field.



White Paper

A-3: *Compatible IFW database*—a Microsoft Access database created by IDentifier. It is assumed that the Field Definitions of this file are identical to the Field Definitions of the Destination file. This *.mdb file will always use **IDNumber** as the key field ensuring successful APPEND- and UPDATE-type imports.

A-4: *MS Access table*—any other Microsoft Access 97 or 2000 *.mdb file.

NOTE: Case A-1: **IDNumber** field must be present in the Source file. Case A-3: **ITCFields** as created by IDentifier must be present in the Source file. The fields in the **UDFields** table must be identical in both Source and Destination files. The operator selects his or her Source file's file-type.

Step B: (enter Source file name) - presents a standard Windows Open dialog in which the operator navigates to the Source file.

For Cases A-1 or A-2: The IDentifier Application reads the data in the ASCII text file and writes it to a temporary table in the application.

For Case A-3: The IDentifier Application links the **ITCFields** and **UDFields** tables to temporary tables in the application.

For Case A-4: The IDentifier Application presents a subsequent user prompt allowing the operator to select the specific table within the Microsoft Access database containing the data to be imported. The IDentifier Application links the table to a temporary table in the application.

Step C: (select import-type) - presents a user prompt allowing the operator to specify the type of import to be performed.

C-1: **Add only**—adds (APPENDS) records from the Source file where the data in the primary key field does not exist in the Destination file.

C-2: **Update only**—over-writes (UPDATE) data in the Destination file with data from the Source for those records where the data in the primary key field is duplicated.

C-3: **Add or update**—performs both steps C-1 and C-2.

Step D: (field mapping) - addresses the need to map fields between the Source and Destination files. The IDentifier Application needs to know where in the Destination file to write the data it reads from the Source.

For Cases A-1 or A-3: IDentifier assumes that the database Field Definitions in both Source and Destination files are identical. It therefore reads data from a Source field and writes it into the identically named field in the Destination file *without presenting the field mapping dialog*.

For Cases A-2 or A-4: IDentifier presents a user prompt allowing the operator to manually “map” database fields between the Source and Destination files. This dialog offers a point and click interface instructing IDentifier to write the data it reads from the Source file into specified fields in the Destination file. (See **Note 1** in **Step F** below.) If the user selects an APPEND import-type (cases C-1 or C-3), any database fields in the Destination file designated as “required” *must* be imported from the Source file.

That is, when the operator is presented with the field mapping dialog in this step, he or she *must* map a Source field to all *required* Destination fields.

Step E: (select Source key field) - IDentifier presents a user prompt for the operator to select a key field to compare the Source and Destination records. This prompt only shows the fields selected from the Source file that were actually mapped to a field in the Destination file. If the Source file contains a field named **IDNumber**, the operator should only select the **IDNumber** field in this step if the Destination file is currently *empty*. (The ability to append or update records might be jeopardized if the Source and Destination files contain duplicate IDNumbers for records that *do not refer* to the same individual.) Select any other field known to contain unique data (e.g., Social Security or Employee ID Number). If the operator mapped a Source field to the Destination **IDNumber** field (in Step D), IDentifier will use the value from the Source file as the new IDNumber in the Destination file. If *nothing* was mapped to the IDNumber field, IDentifier will generate a new serial number based on its internal numbering scheme. **Note:** Use of the **IDNumber** field as the primary key results in a faster import.



White Paper

Step F: (anticipate image import) - Step F presents a user prompt which anticipates the possible importation of external images from a Source database. The user prompt is in the form of a single question: “Do you want to import the **ImageID** and **ImageDate** fields?” (See **NOTE 1** below.) Answer NO if you will want to import images (in a subsequent step) in addition to text. If you answer NO, IDentifier *will not* import the data from the **ImageID** and **ImageDate** fields in the Source file, leaving those fields empty in the Destination file when the text import is finished. (Empty **ImageID** and **ImageDate** fields allow IDentifier to generate new, unique values for these fields when the operator imports the images later.) Answer YES only when the Source ImageID’s are known to be unique numbers that are not duplicated in the Destination file. (Generally, the operator answers YES at this prompt only in the cases where 1) he or she might select **IDNumber** as the key field to be used for matching because the Destination database is empty, or 2) the operator does *not* want to import images.

NOTE 1: In version 4.02d and earlier, the user prompt in Step F only appeared if the operator selected A-1 or A-3. Operators who selected A-2 or A-4 were *not* presented with this dialog, and *had to know NOT to map the ImageID and ImageDate fields* if they wished to subsequently import images. Versions 4.02e and after always present the user prompt in Step F, regardless of Source-type.

Import Text Wizard: What it does

IDentifier first determines the number of records to be APPENDED and/or UPDATED. The Application uses special queries to compare the values in the “key field to be used for matching” (Step E) between the Source and Destination files. Records in the Source file that *do not* have identical key field values in the Destination are considered “new” and eligible for addition. Records in the Source file that *do* have identical key field values in the Destination are considered eligible for updating. (Adding and/or updating is attempted depending on the operator’s selection in Step C. Source records conflicting with append or update requirements are ignored.)

Update:

UPDATE records are processed one at a time. An UPDATE for a specific record is not performed if the value in the **ChangeDate** field in the Source file is older than the **ChangeDate** in the Destination file. When an UPDATE is performed, the **ChangeDate** in the Destination record is set to the current date. If an error occurs, the entire transaction is rolled back.

Append:

If the operator has selected **IDNumber** as the key field in the Destination file, APPEND records are processed in a fast batch update. (When the records in the Source file are imported, the IDentifier Application will quickly assign new serial numbers, beginning with the next consecutive number. Additional queries or calculations are unnecessary.) Otherwise, a slower one-at-a-time APPEND method is employed (requiring numerous queries) and new unique values for **IDNumber** will be generated.

Merge Images Command

Typically, an operator invokes the **Merge Images** command following the Advanced Import’s **Import Text** command. Answers to steps A, B and C of the Merge Images Wizard will be the same as answers to steps A, B and E of the Import Text Wizard. **Merge Images** does not have steps corresponding to steps C and D of the Import Text Wizard because Merge Images is only an update process and field mapping is unnecessary.

Merge Images Wizard: User Prompts

Step A: (select file type) - consists of a user prompt displaying the following four types of files the IDentifier Application can import:

White Paper

A-1: *Compatible IFW text file*—a comma-delimited ASCII text file generated by an Identifier Application. This file can be trusted to include a key field named **IDNumber** which contains unique values for each record, as well as **ImageID** and **ImageDate** fields.

A-2: *Text file*—any other comma-delimited ASCII text file generated by a third party application.

A-3: *Compatible IFW database*—a Microsoft Access database created by an Identifier Application. This *.mdb file can be trusted to include a key field named **IDNumber** which contains unique values for each record, as well as **ImageID** and **ImageDate** fields.

A-4: *MS Access table*—any other Microsoft Access 97 or 2000 *.mdb file. The operator selects the file-type of his or her Source file.

Step B: (open file) - presents a standard Windows Open dialog in which the operator navigates to the Source file and/or table.

Step C: (select key field) - Identifier presents a user prompt for the operator to select a key field to compare the Source and Destination records. This prompt only shows the fields selected from the Source file that were actually mapped to a field in the Destination file. The operator should select the same field used in Step E of the Import Text Wizard.

Step D: (open images) - A standard Windows Open Dialog opens to allow you to navigate to the location of the Source portrait images. Repeat for each of the image types (Portraits, Signatures, and Fingerprints) as selected in IDServer Setup.

Merge Images Wizard: What it does:

If a date in the Source **ImageDate** field is equal to or newer than the date in the Destination **ImageDate** field, the Destination image is overwritten with the Source image. Otherwise, the image is *not* imported. If the **ImageID** field in the Destination file is empty, Identifier will generate a new, unique serial number in the **ImageID** field when the image is imported. It will then create a copy of the Source image and name it with the value it created in the **ImageID** field. (Identifier saves the image in the file format specified in IDServer Setup, and saves it to the “path” enumerated in IDServer Setup’s “Server Data Path.”) IDServ.exe requires an appropriate dongle to perform this operation. This step is repeated for each image type specified in IDServer Setup.

Miscellaneous Notes

When Identifier is attached to a third-party data table through a *nonnumeric primary key* field, importing is disabled within Identifier. Importing data in this case must be performed by a non-Identifier Application. Prior to importing, pay close attention to any text fields in your Source file which might contain numbers with leading zeros and hyphens (as in some Social Security numbers), or other non-numeric characters. Microsoft Access, when it attempts to import the data into the Identifier database, may strip the leading zeros from the number, treat the hyphens as mathematical operators (the “subtract” sign), or perform other unexpected conversions. The best way to ensure that data in an ASCII text file is imported exactly as it is found in your Source file is to make certain that your Source data is enclosed within quotation marks in the text file.