Accessing Pervasive Data From ODBC Or Other Applications

A White Paper From

For more information, see our web site at
http://www.goldstarsoftware.com

Accessing Pervasive Data From ODBC (or Other Applications)

Last Updated: 03/16/2010

When it comes to "frequently asked questions", this one takes the cake, to be sure. It seems that people are used to databases which provide ODBC access natively (such as FOXPro, Access, etc.) and get confused when they cannot see their Pervasive/Btrieve data. Then, they start Btrieve-bashing because they can't get their data out. This document was created to provide some explanation as to what the issues are when it comes to ODBC access and your Pervasive data. The material is taken mostly from the Pervasive PSQL Service & Support classes provided by Goldstar Software.

Database Engine Type & ODBC Drivers

Old DOS applications can be using Btrieve 5.x or older for DOS, 5.x for Windows, or even Btrieve 5.x or 6.10 for NetWare. These engines do NOT contain an ODBC driver and will not support 32-bit ODBC access. (MS Access 2.0 had limited support for Btrieve 6.10, but only from the 16-bit world.) If this is the case, then you will need to purchase a new database engine to support this access. The Pervasive PSQL v9 Workgroup Engines (WGE) is the best deal (only $49), as it supports older operating systems, 16-bit applications, and it also has the latest tools. Pervasive PSQL Summit v10 is a better version if you don't need Win98 or 16-bit application support. For the SAFEST access, you may wish to put the engine on a machine not currently used to access your database.

Please note that the PSQLv9and PSQLv10 database engines will NOT allow you to write to Btrieve 5.x data files, but these newer versions WILL read them just fine.

If you have Btrieve 6.15, you may or may not have an ODBC driver. If not, then you will need to upgrade your engine, as mentioned above. If you do have an ODBC driver, you will want to determine the version. The latest officially-released version was v2.04. In any case, these older ODBC drivers can be slow & cumbersome. If upgrading is an option, I'd highly recommend it.

If you have Pervasive.SQL 7, Pervasive.SQL 2000i, Pervasive.SQL V8, Pervasive PSQL v9, or Pervasive PSQL Summit v10, then you have all you need, and no upgrade is necessary! Of course, if you want to upgrade the environment to a newer release to take advantage of newer tools, faster SQL engines, and a better ODBC driver, then an upgrade may still come in handy.

Why Can't I See My Data?

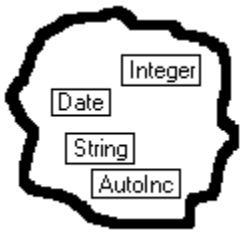
You have all of this important Btrieve data just sitting there in files from your application, yet you can't get to it from ODBC. WHY NOT???

To understand the reason, you must understand the architecture and philosophy of Btrieve. Put simply, Btrieve is NOT a relational database manager like MS Jet, FoxPro, SQLServer, Oracle, Sybase, etc. Instead, it is a *low-level record manager*. What this means is that the Btrieve programmer gives a Btrieve record over to the database and asks it to store the data. Then, it can read back, delete, and/or update those records one at a time.

What exactly is a record? A Btrieve record looks like a "bag-o-bits":

Simple, right? We have a binary object (a.k.a. blob) with a given size. Note the wonderful form, the features, the structure? Yup -- none of that exists. It is up to the *programmer* to decide what goes in this blob. This provides total freedom for data types, field types, field lengths, arrays of structures, etc. Every record in a file can have the same structure, or every record can have a different structure. The point is, BTRIEVE DOESN'T CARE. This is the basis for the extreme speed & flexibility for which Btrieve is known the world over. Without any worry about the structure of the data, the database engine can simply store the data (or read it back), extremely quickly.

Some people would argue that there IS a structure to Btrieve data. Don't we have indices in the file? Shouldn't our blob actually look more like:



OK, I can agree, but only to some extent. These index definitions actually imply that the data in those locates is of those specified types and lengths. This MAY be correct, and is certainly very useful information to have, but it is not the entire picture. There is still no type-checking and no data-checking done by Btrieve, which ensures that complete flexibility that developers relish. In fact, let's take the example of the DATE key above. Btrieve date fields are 4 bytes in length, comprised of a two-byte year and one byte each for the month and day. However, what would happen if the developer stores four ASCII spaces (0x20) in this block? Would Btrieve report an error? No way! As far as Btrieve is concerned, those spaces actually represent the date 32/32/8224, and it will be collated

properly in sequence after 12/31/8224 and before 01/01/8225. Again, BTRIEVE DOESN'T CARE.

As you can see, there is no structure to the Btrieve data as evidenced by the file. It is up to the programmer to decide how to interpret any given stream of bits. How can a generic ODBC application like MSAccess read this data and display it in a human-readable form? The simple answer is that IT CAN"T -- without having a lot more information. This information is made available to the engine in the form of *Data Dictionary Files*.

What about simply importing the data into other applications? Sure, this is possible, but you MUST know the structure to do so. Even if you do not create data dictionaries, you'll still need to understand the layout information typically contained in these files.

What Are Data Dictionary Files?

Data Dictionary Files, or DDF's, are really nothing more than Btrieve files which have a very specific file structure. This structure is well-known to the relational engines and can be easily read by those engines. This structure contains the definitions for each table in the database, the definitions for each field of each table in the database, and additional information about indices and other database constructs.

In effect, there are several different files that comprise a “set” of DDF's. For a full listing, check the manual or attend one of Goldstar Software's Pervasive Service & Support classes. For the purposes of this discussion, we'll check out the three core DDF's here, specifically FILE, FIELD, and INDEX.

FILE.DDF
This DDF file contains information which links logical table names to physical Btrieve data files. A logical table name is a name (that will be known to SQL) in the dictionary which is linked to a physical data file, which may be in a specific location. This allows the SQL engine to access each Btrieve file via a logical name, which is especially important for some accounting packages, which use less-meaningful Btrieve file names like SY00000.BTR. Each table will have one entry in this file which includes a FileID, Table Name, File Location, and Flags.

FIELD.DDF
This DDF file defines the fields (or columns, in SQL vernacular) included in the tables. Each field definition includes a unique FieldID, FileID (to relate it to a table), Field Name, Data Type, Position, Length, and some other details. Every byte in the data file record should be defined by fields in FIELD.DDF. As mentioned above, this is the first time anything like a field is being applied to a Btrieve file, so having this information correct is important.

INDEX.DDF
This DDF file contains index definitions which allow the relational engine to extract data based on existing indices. This provides fast, random access to any record which can be accessed on a Btrieve key. Without INDEX.DDF, the ODBC driver will need to do a

linear scan to locate records, which can really make ODBC access quite slow. This file contains items like FileID, FieldID, IndexNumber, Segment, and Flags. These flags are the key to making sure everything works correctly, and is the most common point where DDF's can be broken.

An interesting aside is that the MS Access 2.0 drivers for Btrieve NEVER accessed the INDEX.DDF file, which then required a table-scan for EVERY lookup. This is one reason why it MSAccess 2.0 hitting Btrieve was always very slow.

There are other DDF's that you may see, including VIEW, ATTRIB, USER, RIGHTS, and others, but these are typically optional and outside the scope of this paper.

In conclusion, then, you see that DDF's are optional for a Btrieve application, but that once DDF's are available, the data can be accessed from the relational engine (and thus ODBC).

How Can I Obtain DDF's For My Files?

Where do DDF's come from? They should come from the software vendor of the application. Your first method of obtaining DDF's should always be to ask the vendor for updated DDF's. Some vendors will provide them with the application, or they may provide them as a separate download, either for free or for some fee. Other vendors, however, will NEVER provide DDF's, citing database security, lack of controls & auditing, or other support issues. THESE ARE ALL GOOD REASONS!

WARNING -- WARNING -- WARNING -- DANGER, WILL ROBINSON, DANGER

Once you have DDF's, it is VERY easy to change data in your database. While SQL access allows you to easily report on data and maintain your database, it can also be used to quickly erase data. All of it. At once. Without a backup. Bye bye. You'll be fired. You'll lose your house. Your dog will run away. Etc. Pervasive Software is NOT responsible for anything you do. Goldstar Software is NOT responsible for anything you do. In other words, don't come crying to us if you break your database! If you mess with the data and the software vendor denies support (or charges you for it), you probably deserved it. That's life. Survival of the fittest can be cruel. Sometimes the dragon wins.

Enough warnings, though. The other side of the coin, of course, is the premise of "It's my data and I'll change it if I want to" -- meaning that you should have full access to your own data for your own purposes. As long as you heed the above warnings, go for it!

What do you do if the vendor will not provide the DDF's? In this case, start by trolling the newsgroups and websites or meetings where users of your application come together. Oftentimes, other users will have already gone through the trouble of building DDF's for your application. If you can pay them for the files, you'll save a lot of time and headaches -- $500 is a great deal for something like this, and it will save you countless hours of your own effort! If this doesn't work, try other resellers of the application, as they may have created some for internal use as well.

If you are unable to find pre-built DDF's, then you'll have to shoulder the burden of building them yourself.

How Can I Create DDF's On My Own?

So, you've made the decision to mess with the data, and the vendor can't (or won't) provide DDF's. Congratulations -- you are about to get a crash course in database theory, database design, and real-world programming! In other words, you must now start the long arduous task of doing it yourself.

There are many different ways to create DDF's. Several popular tools were available in the past, including Xtrieve, DDFBuilder, DDFSniffer, and more. Additionally, some vendors actually used Btrieve calls to create their own DDF files manually, or had special-purpose tools which created DDF's from a data modeling language. Unfortunately, most of these mechanisms had some problem or another, which explains why most DDF's in the world today are not compatible with the newest relational engines.

If you have a database which has bad or incorrect DDF's, or no DDF's at all, you have your work cut out for you, and it may be easier to scrap the existing DDF's & start over.

There are a few common mechanisms to create DDF's. This section is not meant to be all-encompassing -- contact Goldstar Software for help in moving forward beyond this information if needed. We have an expert staff that has completed many projects to build sets of DDF's, ranging across all kinds of applications.

- Use the Pervasive Control Center that comes with Pervasive PSQL to define the tables manually. One of the features inside the PCC is called the "Create Table Wizard", and it provides a procedural GUI interface to help you to figure out what is in the file piece by piece. With Pervasive PSQL v9, this is in a separate download called the DDF Builder (which is included with PSQLv10). Older versions of this tool were fairly poor, so you should make sure that you have the latest Service Pack before attempting it. Please note that this tool requires you to define the ENTIRE file at once -- there's no way to easily save your work & go back to it later. The information provided by the tool is very limited -- you must possess a detailed understanding of the data structures, or have access to source code. The advantage with this method is that just about anyone can do it. The disadvantage is that it is a very slow & tedious process, and the GUI interface is truly awful! You will be constantly switching from the mouse to the keyboard, even for a table with only one column in it. Note that if you run into any "legacy" data types, such as "Note", you may get stuck and be unable to continue with this tool. When you're all done defining a file, you will see a CREATE TABLE statement generated for you. This statement is NOT completely accurate at all times and may require minor modification, but you DEFINITELY want to save it for future reference.

The Pervasive Control Center from PSQLV8 and earlier can also be used to modify the table design after the fact. This should always be done in "UNLINKED MODE" to change **just** the DDF's. Of course, if you make a major mistake, just remember that you can always blow away the definition and start over again. Be sure to have a backup, though, of both the database file AND the DDF's before even thinking about this option.

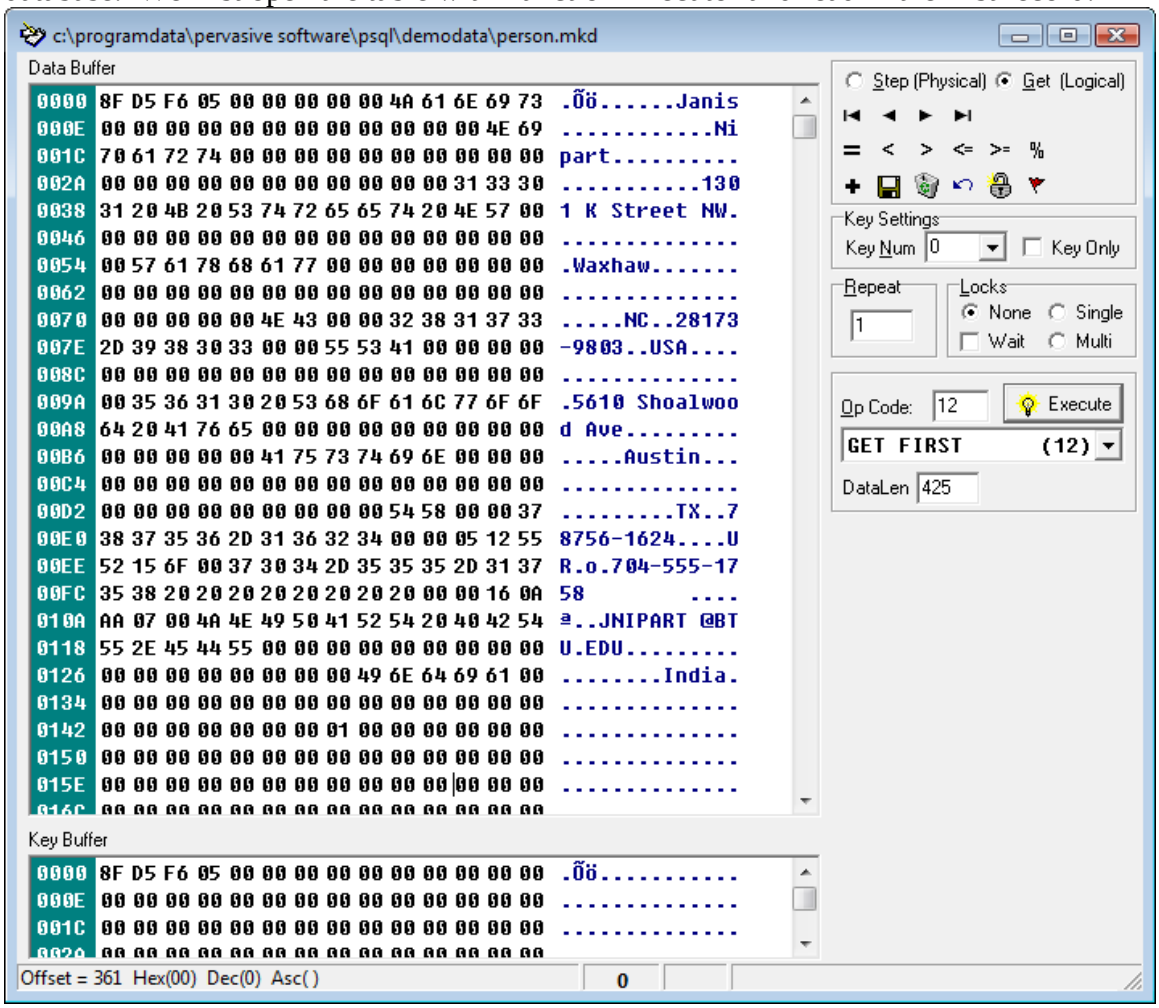
- Use the DDFEase utility from PSQL7. This is the precursor to the Create Table Wizard, and it actually worked fairly well. (I like the way it displays data a bit better, making it possible to figure some things out MUCH more easily.) It is still largely a manual process, but DDFEase can also create older DDF's with legacy data types fairly quickly.
- Use a utility called BtSearch, from Gil Nelson, available from http://www.nssdd.com. BtSearch allows the end user to analyze the structure of their data and create Data Dictionary Files. It does this by allowing you to select a single or group of characters from your file and BtSearch will then show you the values in all of the Pervasive Btrieve data types including additional types not supported by Pervasive. Once you locate the correct data type you can select it and assign a name and add it to the definition. You can also edit your DDF files after you have created them. After you have defined all of your fields you can then insert this definition into the DDF files. Once built you can export your data to dbase or ascii format. BtSearch also has additional abilities such as edit, delete, detailed query searches and more.
- Create SQL-level CREATE TABLE scripts to run from the Pervasive Control Center. This will allow you to recreate the DDF's at will, and is by far the BEST solution overall, since it is guaranteed to make good DDF's. Remember to make a backup of your data files, though, before you create the new file on top of it! Of course, the newer syntax of the SRDE (in PSQL2000i and newer) does preclude the use of some older data types and constructs which may be required for your data, such as NOTE and LSTRING, so you may need to use PSQL7's SQLScope for your database instead.
- Use the SQLScope utility to execute CREATE TABLE scripts on Pervasive.SQL 7. This is similar to the PCC above, but will create DDF's with older legacy data types. An interesting aside is that Pervasive.SQL 7 seems to be able to create DDF's for either Btrieve 6.15 (when done carefully), Pervasive.SQL 7, or Pervasive.SQL 2000i. The Pervasive Control Center creates DDF's that are only usable in Pervasive.SQL 2000i and newer.
- Use the ODBC Pass-through feature of MS Access to run CREATE TABLE scripts in Btrieve 6.15/SSQL3.01. Great for backward compatibility, but the resulting DDF's may not work with newer engines.

What Information Do I Need to Create DDF's?

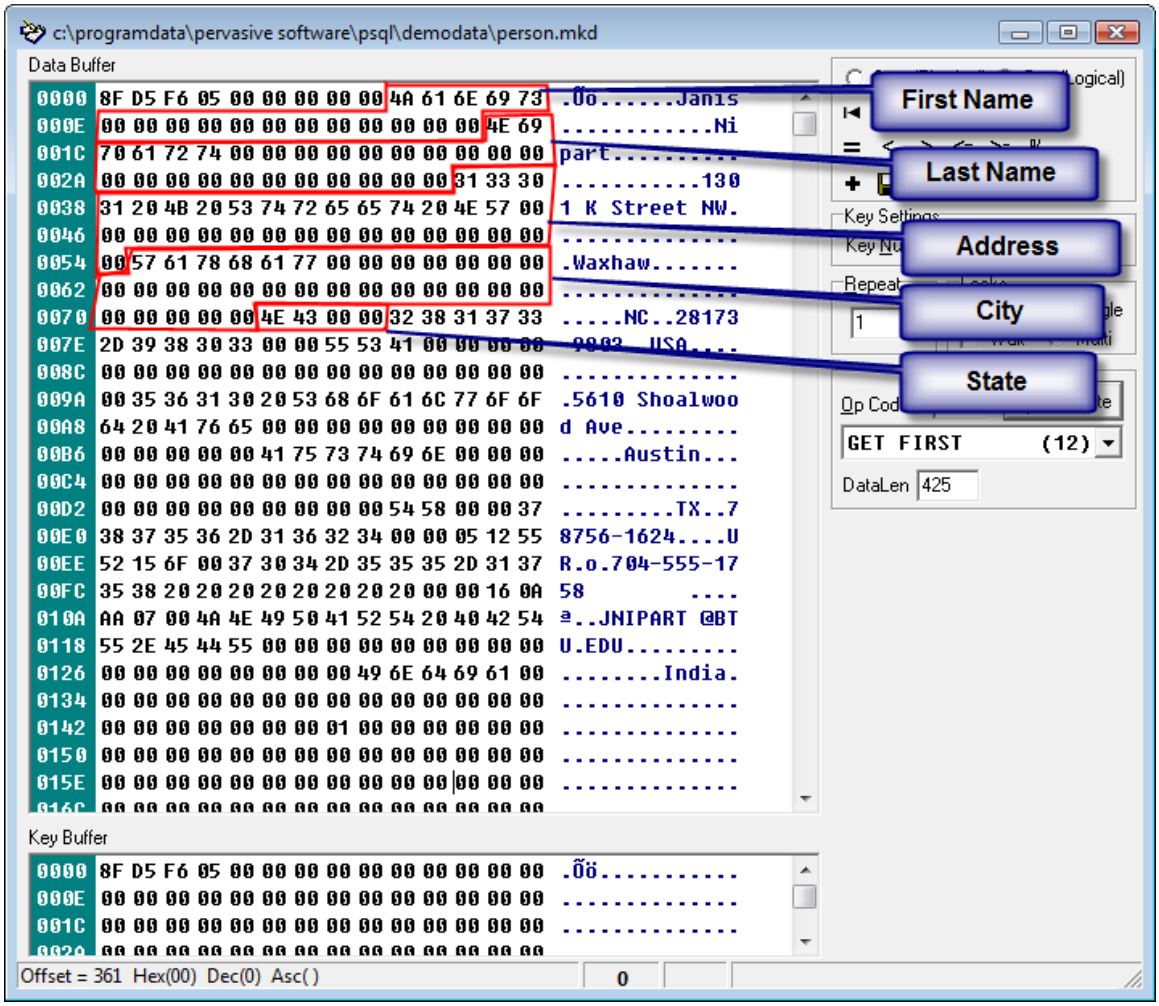
Ultimately, the tool you use is inconsequential – the hard part is collecting the metadata (the data about your data) that the DDF's must contain. If you have access to application source code, or at least very detailed data structure definitions (check your application documentation -- some vendors hide this information in the docs or in a file on the

system), this task will be very easy. If not, it will be a veritable nightmare to be avoided at all costs. Why is this a nightmare? Think about a blob of bytes in a Btrieve file where you have twenty ASCII spaces (0x20) in a row. Is this a single 20-byte space-padded string? Is this two 10-byte space-padded strings? Is it twenty individual 1-byte characters? Is it twenty 1-byte integers with the value of 32? Perhaps it is ten 2-byte fields with a value of 8224? Think about having a 175-byte string of ASCII NUL bytes (0x00)! What do you do with those? It only gets worse & worse from there.

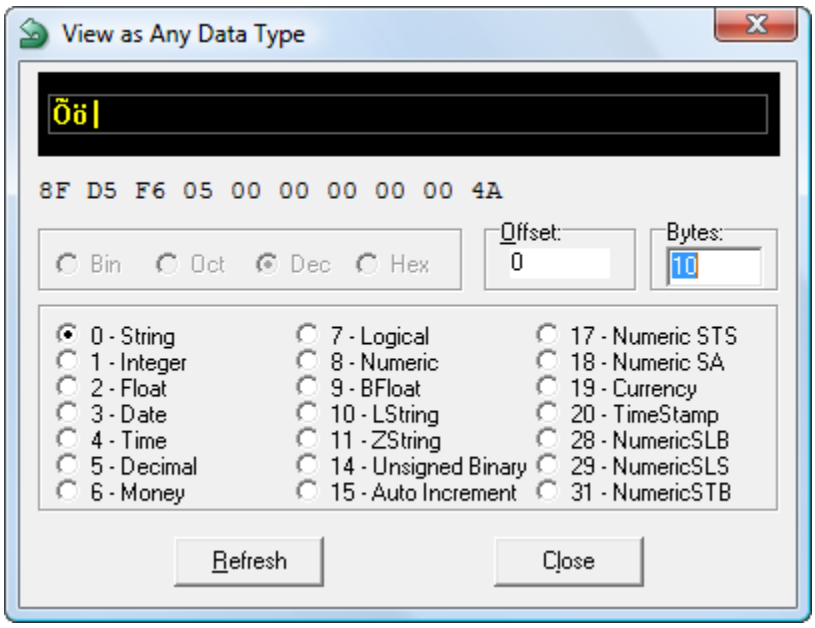
Let's look at a simple example, from the PERSON table in Pervasive's DEMODATA database. We first open the table with Function Executor and read in the first record:



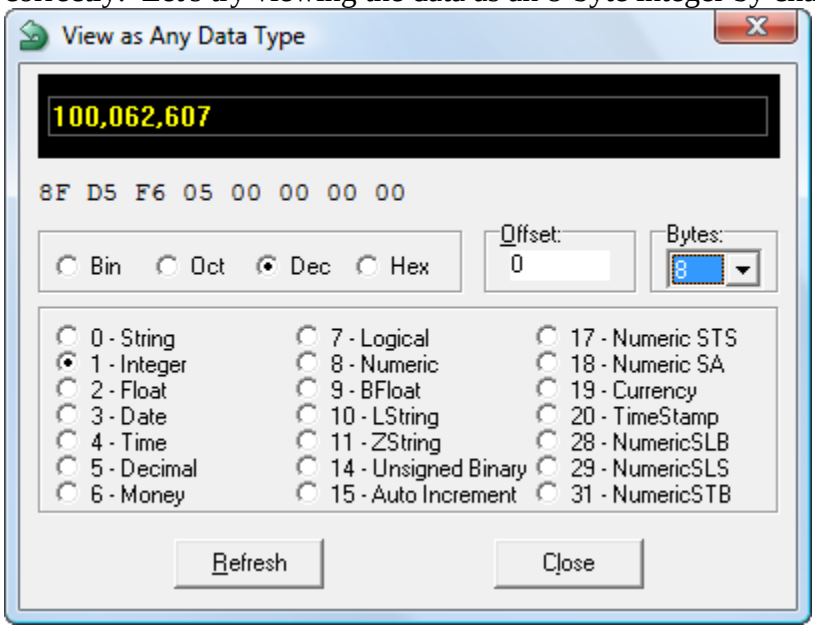
From this display, you can see a number of fields that are easily identifiable. I can see a First Name (Janis), Last Name (Nipart), Address, and many more fields. As such, I usually start by highlighting the fields that are easy to figure out. Note that I really cannot see the LENGTH of these fields, so I have to take some guesses. With some simple guesswork, I can find the locations of several text strings:



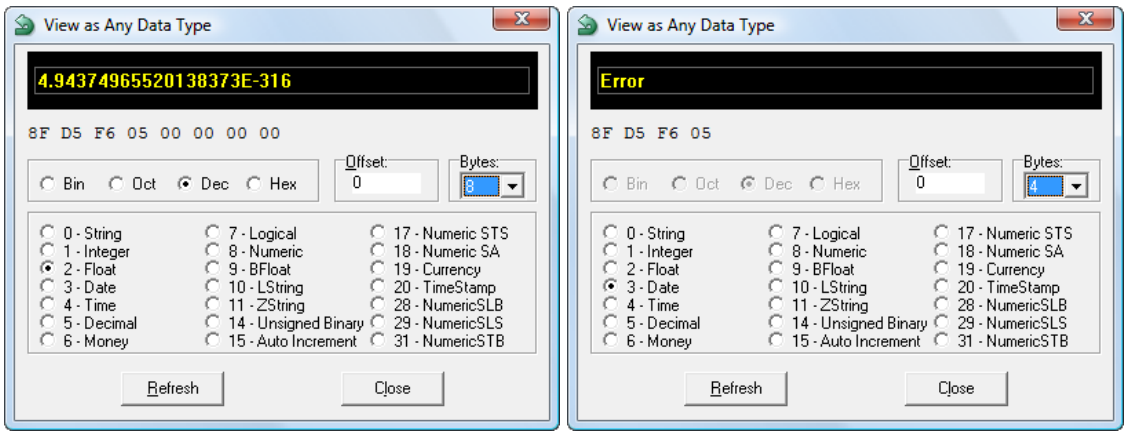
Looks easy thus far. However, the string fields are usually the easiest to do. What about the first bytes in this record? What the heck does 8F D5 F6 05 really mean? Luckily, the Function Executor has a special option that can help with this. We select the first byte, then right click and select "Show As" to get the following dialog box:



This defines the next 10 bytes as a String field -- but it obviously doesn't display it correctly. Let's try viewing the data as an 8-byte integer by changing to data type 1:



OK, looks better. What about a Float or a Date field?



In fact, we can go through all of the data types looking for any types that display the data correctly. Note that some types have different lengths, so we may have to try various lengths, too. Ultimately, though, we need a bit more information to track this back to the data. To get this information, we launch the application and find the record for Janis Nipart. On that data entry screen, we find that Janis' Student ID Number happens to be 100062607, which indeed matches the value that we get from either a 4-byte integer or an 8-byte integer.

Once you've figured all of this out, then you can start building a table for what you know, and what you don't know. This table looks like this thus far:

Offset	Name	DataType	Length
0	StudentID	Integer	8
9	FirstName	String	17
26	LastName	String	27
53	Address	String	32
85	City	String	32
117	State	String	4
121	Unknown	Unknown	304

Now, look at the first field, which starts at offset 0, and which we know to be 8 bytes. However, there is another byte before the FirstName field that is unaccounted for. This could be another 1-byte field of some kind, or something else. In this case, it happens to be something else -- a Null Indicator Byte (or NIB) for the FirstName Field. Since this table uses "true null support", added with the newer SQL engines, we have to alter our table a bit, adjusting forward the starting offset and decreasing the length by 1.

Additionally, notice that the string fields are not padded with spaces, but rather they are padded with null (0x00) bytes. As such, we know that these are not STRING fields, but are really ZString fields, which are Null-terminated string fields. With SQL, we ignore the null termination byte and only define the field length, so we have to decrease the lengths by 1 more. With this information in hand, we update our table a second time to get this result:

Offset	Name	DataType	Nullable	Length
0	StudentID	Integer	No	8
8	FirstName	ZString	Yes	15
25	LastName	ZString	Yes	25
52	Address	ZString	Yes	30
84	City	ZString	Yes	30
116	State	ZString	Yes	2
120	Unknown	Unknown	Unknown	304

Now, you continue digging through the remaining fields of the 425-byte record structure. If you deem something as completely unknown, give it a data type of STRING to avoid the SQL engine parsing it at all and just move along to the next field. You can always change it later on if you garner more information about the record.

Finally, we need to convert the table into a CREATE TABLE statement in SQL. This is academic at this point, since all of the hard work has already been done:

```
CREATE TABLE Person USING 'PERSON.MKD'
(
        StudentId       BIGINT NOT NULL,
        FirstName       VARCHAR(15),
        LastName        VARCHAR(25),
        Address         VARCHAR(30),
        City            VARCHAR(30),
        State           VARCHAR(2),
        Unknown         CHAR(304)
)
```

Once we have this, we can create a new dictionary in an empty directory, then run this statement to create a new file, called PERSON.MKD. Finally, we simply copy over the real data file over the top of the new file we just created, and we try accessing the data from SQL. We will see valid data in the first 6 fields, and one big field full of junk at the end. Repeat this entire process, refining the data structure definition on each iteration, until you have enough data defined to perform your export.

Internally, Goldstar Software may use any or all of the above techniques to build DDF's. We start by analyzing the data that we have, including any source code definitions or other documentation. If possible, depending on the level of compatibility needed by the end user, we will create CREATE TABLE statements in either PCC/SQLDM (2000i) or SQLScope (6.15/7). We then check the resulting Btrieve file and compare that with the source file provided, and determine if data is readable or not. We then make any needed corrections and loop over and over through the process until finished. In some cases, DDF's can be very tricky, especially since the way Pervasive.SQL 2000i creates DDF's is not always the same as older Btrieve files would like. As such, there may be a need to

manually hack at the DDF files, manipulating index definitions and field data manually. Of course, if there is no documentation on the file format, we may have to run the application, modify data to some set values, then stare at the hexadecimal bytes using the Pervasive Function Executor until it starts to make sense. Luckily, we're trained professionals, and you might not want to try this at home.

Why Are My Queries Slow or Incorrect?

Once you have valid DDF's, then you should have good access to the data. However, sometimes the queries you run may be very slow, or may not return the data as you expect it to. There can be several reasons why queries may not work correctly. This section outlines a few of those reasons.

Database Is Not Relational-Compatible

Some databases are simply not "relational compatible". The term "relational compatible" refers to data record formats which can be represented by the relational database model. In other words, every record is the same structure (i.e. database tables are made up of like rows), each field is well-defined and a known type (i.e. the record is comprised of fields or columns which are understood by the SQL engine), and no two fields are overlapping (i.e. fields must be atomic).

Databases which violate these rules are often very difficult to put into the relational model, and problems usually result. The most common issue is overlapping or variable field structures that depend on other data in the table. Another common issue is unsupported data types, like the use of a 6-byte floating point value, which was supported in Turbo PASCAL but never fully supported as a Pervasive data type.

Bad Table Definitions (Reserved words, bad identifiers...)

When defining table names and field names, care must be taken to ensure that reserved words are not used. This was not a problem in previous versions, but newer versions cannot handle reserved words as field names. Sometimes, you can enclose the field in double quotes, but this does not always work. Also, field names should be short (under 20 characters), alphanumeric (underscores are the only special character allowed), and have the first character be alphabetic. Identifiers like "Select", "Discount%", "6Pack", "Customer-Ptr", and "WhatALongFieldNameThisIs" should be avoided at all costs.

Bad Field Definitions

Having a field defined which overlaps another field is a no-no. Fields in SQL are atomic, meaning that they cannot be broken up. Also, field definitions for unknown data types are not supported in the relational model. While a Turbo Pascal 6-byte float is perfectly valid in Btrieve, the record cannot be accessed from SQL.

Bad Index Definitions

Files which have index definitions which do not perfectly match their Btrieve counterparts are another no-no. The proper flags must also be set for duplicatable, modifiable, and case sensitive keys. Mismatches can cause an index to be discarded, resulting in VERY slow response times and VERY busy servers.

Solutions and Workarounds

The first thing to do when you have a problem is to review your query and the DDF structure. Use the DDFEase utility (PSQL7) or one the Check Database Wizard (newer versions) to see if the DDF's match the Btrieve definitions. If not, try to fix them first. If they can't be fixed, try creating all new ones as described above.

OK, so I can't use ODBC. How else can I export data?

If you do NOT have DDF's, then ODBC access will be impossible, and you are left with two options:

1) **Application-Level Export**: If the application you are using allows a formatted export (i.e. comma-delimited), then you should use this as the first and least-painful option. Even with an application export that dumps a formatted text file, it may be possible to use Pervasive's Data Integrator package (or contract someone to use it for you) to extract all of the data from the report file out into a comma-delimited file, readable by almost any database application.

2) **Btrieve-Level Export**: Why not just export the raw "blobs" and go from there? Of course, you will need to know something about the data and about the application to be able to properly extract the information. However, a simple C application should be able to read and parse this information. If you are a developer familiar with digital data streams, then you should have no problem with this. Let's look at how to do this in a bit more detail:

Exporting the data with "BUTIL -SAVE" command will create UNF files which contain the binary record images. You will only need to write an application that parses the data into its fields and values. Easy to export? Extremely. Easy to decipher? No way -- but the data is no longer in a Btrieve format -- it is readily accessible to any application which can read data files. Again, if you have the metadata, you're home free!

You create a UNF file by running "BUTIL -SAVE <btrvfile> <unffile>". These files have a structure such as:

47,<47 bytes of data buffer><CR><LF>

The first few bytes will be a number in ASCII (47 above, but we'll call it "n" for this discussion since it can be any value). The value of "n" indicates the total length of the Btrieve record (i.e. the blob). Then, you'll see a comma, followed by "n" bytes of raw binary data, then a CR/LF pair. If you can figure out how the data is represented in that format, you can write an application (C works very well for this since you may need to read the data byte by byte, and languages like VisualBASIC just aren't up to that task) to pick out the bytes & import them somewhere else, or even write the data to a comma-delimited file. If not, then any hope of anyone else doing it is VERY slim, since you (presumably) know the app & would be in the best position to figure out the data structures.

For what it's worth, Pervasive's own Data Integrator product has the capability to parse records directly from the Btrieve files, but the metadata definition is still the primary factor here. In other words, this tool can save you development time, but you must still define the data structures.

What If I'm Stuck?

If you still can't get DDF's built, or if a SQL query still won't work right for you, contact Goldstar Software (www.goldstarsoftware.com) and let our professionals work with you to help! Our staff has worked on numerous applications, both with and without metadata. When provided with a data set, we can give you back a quote for creating a basic set of DDF's to meet your specific data needs. (Remember that not all data is relational-compatible, and it may not be possible to tell this from an initial review.) Some DDF's (for small, simple data sets) can be built in just a few hours. Others may require a few days of investigation to get a complete definition ready. We usually scale such projects in smaller, more manageable, fragments so that you can control the costs of the process.