

Understanding DataEase

A Primer

Graham Smith
PLM Consulting, Inc.
gsmith@plmconsulting.com
August 2005

Table of Contents

Introduction – an overview of the product and it's history.

The Multiview – a discussion of Relational Theory and how it is used in the DataEase product line.

How Things Work – a technical discussion of how DataEase is put together. Topics include Object Oriented design and how DataEase implements database structure.

Documents – the fundamental structural components of the user interface in DataEase.

Reports – the primary reporting tool in DataEase.

Procedures – how the DQL language is implemented in DataEase.

Lessons Learned – how DataEase has changed the way I work.

Introduction

DataEase, like most other business software programs, is a tool. And a tool can only be as good as the person wielding it. The better you understand the tool, the better job you can do. Not only do you have to know *how* to use the tool, you have to know *what* to do with it. For example, I'm pretty good with a saw and a hammer when it comes to rough construction work, but I'm a basket case trying to use those same tools to build a cabinet.

It has been said that if the only tool you are familiar with is a hammer, everything tends to look like a nail. One of the fundamental goals of this article is to show people that DataEase is not a hammer.

If all you are trying to do is cobble together a rough database with a couple Tables and a few reports, then you might as well stop here and save yourself some reading. But if you have any plans for something larger, and hopefully more elegant, then this is written for you.

The well known author Arthur C. Clarke may have said it best when he said. "Any sufficiently advanced technology is indistinguishable from magic." Relational database design is not magic, but it can often look like it to some. And as if that wasn't bad enough, we work in an increasingly complicated development environment known as Microsoft Windows. It just doesn't seem to get any easier, no matter how long you have been doing this.

Into this complex mix, we now introduce a product that has been growing and evolving for over 20 years from a simple DOS program to a rich Object Oriented Windows program. And, for good or for bad, most of the original programs philosophy and concepts are still in there under all the graphic trappings of a modern program.

The purpose of this paper, is to foster a deeper understanding of how DataEase works in the hopes that this will improve the readers ability to use the program. Along the way, we will delve into some Relational Theory; discuss the basic concepts of Object Oriented Programming; pass through the realm of the Multiview; and dissect the layers of Forms, Reports, and Procedures to reveal how they are all inter-related underneath. And while the main focus of this paper will be prior DataEase for DOS users, it should also prove to be enlightening for new DataEase users as well.

We start, as is appropriate in cases like this, at the beginning. And to do that, we need to explore the history of DataEase. It has often been said that the only way you can know where you are now is to know where you came from.

A Brief History of DataEase

A long time ago, in a galaxy far away... In 1977, Apple introduced what has been, arguably, called the first mass-market personal computer. While it was aimed primarily at the hobbyist market, it soon found a niche in the office as well. In 1981, IBM introduced the PC which was aimed at the business market. At that point, all computer products were similar in their capabilities, and in their limited selection of business software.

Arun Gupta, the founder of DataEase, started writing software in an attempt to provide his brother (a doctor) with some software to help run his business. It soon became apparent to him that this software could be turned into a marketable program. A company called SoftEase was founded with the goal of creating a business suite of software programs, not unlike the office suites available today. The first program created was a database they called DataEase. While some other programs were started, it quickly became apparent that they would be better off concentrating on the database market since word processing and spreadsheet programs had started to appear.

The concepts found in DataEase were modeled around the works of Codd and Date in Relational theory. They took the unusual (for that time) approach of designing a program along the lines of what has now become known as Rapid Application Development (RAD). Their goal was to provide an interface that did not require the user to know how to write code, something they referred to as “programming by exception”. As we will see later, this concept served both to make the program easier to use and to limit its acceptance in the broader market.

In the early 1990's, Microsoft Windows was beginning to find its way onto computers. And while DOS programs would still work in this environment, it was clearly time to create a Windows version of DataEase. But Arun had bigger plans than just creating a Windows port of the DOS program. Object Oriented Programming had also begun to emerge at that time and Arun saw in this a way to make a much richer program than a simple DOS port would have been.

The first version of DataEase for Windows was released under the name of DataEase Express. This version was very similar in appearance and function to the current version of DataEase, but it lacked Procedures (DQL). Like some other relational advocates, Arun believed that a complex reporting language was not necessary – that everything could be done through relational constructs and a basic reporting tool.

But DQL was both a reporting language and a processing language, and for all his genius, Arun had often underestimated what people would use his program for. The lack of DQL proved to be a flaw which seriously hindered wide spread acceptance of Express.

Dissatisfied with the direction things were heading, Arun started work on a new product (based on Express) that had no native data engine - it was strictly intended to be used as a

front end for SQL databases. This project led to the creation of a new company named Rapid Enterprise International which was eventually bought by Symantec. So Arun left DataEase to work on the new program. Ironically, Symantec was never very successful in marketing this program and eventually, the program returned to DataEase and is currently sold under the name of DataEase NetPlus.

Arun continued to work in the database field but never had the success he had found with DFD. In retrospect, it would seem that he was just too far ahead of his time since many of the concepts he pioneered along the way took another 5 – 10 years to begin to show up in the market. Arun Gupta died in 2004 as the result of an accident.

After Arun left DataEase, the job of head of development for the Windows product fell to Pete Tabord. It was decided that if DataEase for Windows (DFW 5) was ever going to sell, DQL would need to be fitted into it somehow. Express was not completely without provisions for a programming language, but this was planned to be considerably different than the DQL found in the DOS product. As we will see later, this turned out to be a difficult task.

DFW went through two major releases before it was decided to take the next logical step and migrate the program to a newer 32 bit compiler. This would make it more compatible with the newer versions of Windows and would allow for the future changes that were going to be needed.

By the time that DFW 6 was released, work had stopped on DFD and no further DOS versions were planned. For that reason, it was decided to stop calling the program DFW and just use the name DataEase.

DataEase 6 (DE6) was more than just a 32 bit product port, however. The new port alone should have been enough to encourage most professional developers to buy the upgrade, but it was determined that at least a few new features were needed to make the upgrade worthwhile to the broader market.

What was added was something that had been in the program all along, but not “switched on”. As was mentioned before, Express did have a programming language built in. It was called OML Scripting and was intended to serve as both a processing language and for object manipulation. And while it had been there for quite some time, it was not complete and had never been fully tested. This led to some interesting discussions during the Beta testing as people tried to figure out how to use this new feature and what to use it for.

The next big change came with DE6.5. This version fixed a number of issues with OML and added both an OLEDB client and provider to the product. This allowed DE6.5 to attach to a wider variety of external data sources, including DFD5 databases.

As this is being written, work is underway on DE7. This next release is being developed

with a newer compiler that will better integrate with newer Windows programs and allow for a future porting to a 64 bit program. There are also several new features being planned, but at this time, no public releases have been made of just what those features will be.

The Price of Backward Compatibility

Every new version of DataEase that has been released, from DFD version 1.x onward, has attempted to be backward compatible with previous versions. While new features were being added in, old features continued to be supported. With most new versions, an older database could be opened with the new version and, with only minor tweaks, it would run just as it had before.

This worked fine until the change from a DOS character based interface (CUI) to a Windows graphical user interface (GUI). At that point, it became nearly impossible to make a clean upgrade without the necessity of doing a lot of additional work.

But, to make the process easier DFW5 had the ability to migrate most of a DFD4.5 database to Windows. For this reason, some features that would have been better off being left out of the product were kept in. This will be seen most acutely when we get to Procedures (DQL).

It became increasingly clear with time that the transition from DFD to DFW was going to be difficult. Many existing users never made the transition and their applications remain in DFD to this day. Others moved their applications as is to DFW without taking advantage of new ways of doing things. Some users found it possible to abandon their old applications and build new ones. Most fall somewhere in between.

The Multiview

Designing a Database

Everyone reading this should have a pretty good idea already what a database is. After all, you either already have one built or are getting started building one. Surely, we don't need to take up time discussing something as basic as database design. Well, we're going to do it anyway!

DataEase is a multi-user, Relational Database Management System (RDBMS) tool. As the name implies, a RDBMS is used to create and manage a relational database. But what exactly IS a database and what makes it a Relational Database?

That is a surprisingly difficult question to answer and the arguments over the answer are nearly legendary in some circles. The concepts that lie behind Relational Theory are largely taken from mathematics (set theory) and logic (predicate logic). But we're going to skip all that for now and simply put it like this: A Relational Database is a collection of two or more data Tables and a set of pre-defined Relationships.

A Table is a collection of data elements, often referred to as columns or, in the case of DataEase, as fields (an unfortunate choice of terms we will discuss later). Data is contained in rows or records. A simple version of a Table is a spreadsheet. Columns go across, rows go down. Just to keep things interesting, many database developers refer to a Table as a database. The reasons are complicated but it is worth keeping in mind that many of the first DBMS programs were not relational. And many more that said they were relational really were not.

That brings us to the "R" part of an RDBMS. What makes a Relational Database different is a structural component called a Relationship that links one Table to another Table based on columns from one Table having matching values in columns in another Table. Let's design a simple database to show how this works.

When you go into a store and pick up something off the shelf, you will note that it has a bar code on it. This bar code represents a number called the UPC, universal product code. This is going to be our first Table and we will call it Products. The Table will contain a ProductCode (the UPC) and a Description.

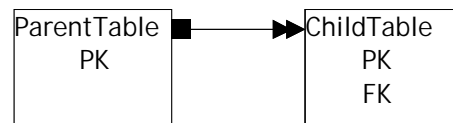
Our second Table is going to be Orders. This Table, among other things, will also contain a ProductCode. We will then define a Relationship between Products and Orders based on the ProductCode. The ProductCode column is the same in both Tables and thus we can quickly locate all the orders for any given product using this pre-defined Relationship.

That, in a nutshell, is what a Relational Database is all about. Of course, that only scratches the surface and we will be discussing things more later. In the mean time, if you

want to read an interesting book, try to find a copy of one of the books (he has written three that I know of) by Fabian Pascal. I had a chance to talk with Fabian around the time his first book was published, and he said that he considered DataEase the one program on the market that most closely adhered to the Relational Concepts that he wrote about. Even now, fifteen years later, the fundamental concepts behind DataEase remain true to the same Relational Theory.

Entity Relationship Diagrams

When most database developers think about the structure of a database, they think in terms of an Entity Relationship Diagram (ERD). An ERD is a way of creating a visual representation of the basic components of a database. Entities are the database Tables and Relationships are, well, Relationships. Consider the following



The ParentTable has a unique field which is referred to in Relational Theory as the Primary Key (PK). The ChildTable also has a PK, but it also has a Foreign Key (FK). The FK in the ChildTable relates to the PK in the ParentTable. This is the most common type of a Relationship and is known as a One-to-Many. For any one record in the ParentTable, there can be many related records in the ChildTable. Conversely, for any one record in the ChildTable, there can be one and only one related record in the ParentTable.

Relationships also denote dependency. For example, while it is possible to have a record in the ParentTable without having any related records in the ChildTable, it is not possible to have a record in the ChildTable without having a related record in the ParentTable (at least it should not be possible in a well designed system). Records in the ChildTable are dependent on the existence of a record in the ParentTable. This the corner stone of Relational Integrity.

From the ERD to the Multiview

An ERD is not a database anymore than a blueprint is a house. But an ERD is a representation of a database that everyone can look at and discuss and understand. It's our ability to mentally "connect the dots" between things that give us the ability to have this discussion in the first place.

While the ERD construct works well enough for a simple database, it starts to get a bit troublesome when you start getting into a larger structure. Trying to layout all those Tables and Relationships on a flat sheet of paper can create something that looks more like a child's scriblings than an ordered set of Tables and Relationships. This is because a sheet of paper

is two dimensional while a database isn't. Another factor is that an ERD is only a representation of the table structure – it doesn't really convey any information about the data stored in those tables.

When it comes right down to it, a database isn't exactly three dimensional either. It isn't really dimensional at all, but we are probably better off speaking of it in those terms since that is close enough for our purposes.

To truly understand DataEase, you must first understand something that the original creator, Arun Gupta, called the Multiview. Let's change the analogy for a bit. Let's talk about the stars.

Every kid at one point or the other, learns that there are Constellations in the sky. There is an entire pantheon of characters and beasts outlined in the heavens for all to see. But those Constellations are not really there. They exist only because of our ability to “connect the dots”. They are also dependent on our position in space relative to the rest of the stars. If we were able to travel freely among the stars, we would see that the patterns start to change as we move about. This is the important bit, the stars don't change – the patterns change. Our perceptions of them change.

Back down to Earth. If we can now look at a database as a kind of three dimensional ERD model complete with data, we can see that by moving around in the model, we can change the way in which we see the Tables and Relationships. We can start to create our own patterns and views of the objects that reveal things we never knew were there. This is our first hint at the nature of the Multiview. As we change our point of view we change our view of the information contained within the data. The data does not change, the information it provides changes.

So far, we have been discussing ERD's as a relatively static collection of Tables and Relationships. That is only the database component of a DataEase application made up of two things – Tables and Relationships. We also have the ability to create subsets of data through the use of queries. Now, the ERD starts to take on a more malleable nature. Not only do we have the relatively static components (Tables and Relationships) but we also have these extracted collections of data elements – extensions of the underlying ERD into new views of the data.

Arun intended the Multiview to act as a level of separation between the user interface structures and the underlying table structure. In this way, he was already anticipating a time when the product could be made data-engine independent. The Multiview would be the middle tier of a three tier architecture: Data Engine – Multiview – User Interface.

The Multiview, underlies every DataEase application and everything you do draws on it. It is a very powerful structure, but like all powerful things, you have to know how to control it. If you can create anything you want, if you can mold the structure of the database to

serve your needs, you are ultimately responsible for the results. If you don't understand what it is you are doing, then how can you hope to understand the results. It's like the old saying, "A computer will do exactly what you tell it to do, whether you want it done that way or not."

Theory vs Implementation

It behooves us now to make a very significant but difficult to explain distinction between Tables and Relationships as ways of describing how the data is organized as opposed to physical structures.

In DataEase, a Table corresponds to a physical file. And the way the file is laid out corresponds to the order in which the columns are defined. Relationships are entries in a System level Table and have no physical connection to the data files at all. But these facts really have little to do with how the developer and end user deal with Tables and Relationships and Data! It is the job of DataEase to deal with how the data files are structured and stored.

This is where DataEase has traditionally run into problems with market acceptance. Most early DBMS products, like dBase, were based on physical files as tables and were procedurally based. That is, the database was simply a collection of data files that used a common format and the developer had to write code to do most of the work. When these developers looked at DataEase, they tried to carry over the same ideas that they used in dBase. In other words, they didn't understand DataEase and constantly fought against it in an attempt to make it do things the way they were used to doing them.

One of the fundamental precepts of Relational Theory is that the physical structure of the data is irrelevant. If you start thinking about a Relational Database in terms of its physical structure, then you will quickly get lost in the details. This was one of the things that Arun was trying to accomplish with the product. He wanted a database *front-end* that was relatively *back-end* independent. He wanted the developer to be able to concentrate on design without having to be saddled with writing code to do everything.

This idea was something that Arun referred to as "programming by exception." By that, he meant that most of what you needed to do was already built into the program. Bear in mind, this was long before anyone had ever heard of Windows or Wizards or "fuzzy logic." The fact is that he both succeeded and failed in implementing this. He succeeded in building in a large number of features that removed the need to do programming. He failed because there were always going to be people who wanted to do things differently.

Of Dreamers, Pragmatists and Philosophers

We are all, in our own ways, all of these things. This is as much true for us now as it was

for people like Codd, and Date, and Arun Gupta twenty years ago. It is all well and good to be able to conceive of the perfect tool set, but what is perfect to one person may be seriously flawed to another. Many people consider DataEase to be a nearly perfect RDBMS, others find it lacking. Most, myself included, fall somewhere in the middle.

One of the most difficult and frustrating things anyone can do is to attempt to push a piece of string up a hill. And yet, people using DataEase have done this for years. The reasons are endless, but often come down to, "I want to do it *this way*." For others, it often comes down to a lack of understanding of how things are supposed to work.

DataEase is one of the best RDBMS tools that has ever been made – IF you understand and agree with its design philosophy. You can either work with that philosophy or work against it. You can either pull the string up the hill or push it.

DataEase is NOT a Programming Language!

If you ever want to get Pete Tabord (head of product development at DataEase) really going, just compare DataEase to a programming language.

For years, people have made comparisons between DataEase and programs like Visual Basic and Delphi. The problems with these comparisons is that they completely miss the point that these other products are not DBMS's and that DataEase is not a programming language. Never the less, the comparisons continue to be made and people continue to insist on trying to evaluate and treat DataEase as if it were the same thing when the only thing they really have in common is that they can both access database files.

I'm not going to belabor the point any further, but at any time in the future, if you find yourself making such a comparison, just remember – you have been warned.

How Things Work

Object Oriented Design

It is not possible here to do more than touch the surface of Object Oriented design. If you want to know more about it, there are numerous books on the subject. But we need to make certain things clear about how DataEase uses Object Orientation from the outset or it may get more confusing later.

Object orientation can be loosely described as “the software modeling and development disciplines that make it easy to construct complex systems from individual components.” In other words, you make little pieces of things that you then build larger things out of. Here is a quote from Pete Tabord on the subject:

“Object oriented is difficult, largely because no-one even agrees what it actually means. I think for DataEase purposes its best treated as relevant in two ways. It was the fashion when DFW was developed, and appeared to offer better internal design - more elegant code - than the way DFD was written - and second, both OS/2 and Windows had an object oriented user interface with which we had to interact.”

It is important to distinguish between the tools used to develop the DataEase program and the tool that is DataEase. DataEase is written using the C++ programming language, but it is not an Object Oriented Database – that is a horse of a completely different color. There are, none the less, aspects of DataEase that are Object Oriented, and for that reason, there are some ideas and words that need to be looked at more closely. Before we get into the words, let's look at a practical example of how OO can work.

Before OO became as popular as it is now, PageMaker (a page layout program) was using OO concepts to format text. PageMaker uses text styles that are built up on upon another. For example, there may be a style called TitleText that is defined as BodyText + 14pt + Bold, where BodyText is defined as TimesRoman + 12pt. TitleText is based on BodyText. If you change BodyText from TimesRoman to Garamond, then you have also changed TitleText since it is based on BodyText. This is one of the primary ideas behind OO – the concept of Inheritance. Here are some more:

- **Class** – this is the basic building block in a OO system. It is the basic prototype of an object and is sometimes referred to as a template. Some examples of DataEase classes that the user would see with are Font, Color, and Border.
- **Instance** – once you create a class, you can use it by creating an instance of it. This is sometimes called cloning.
- **Instantiation** – the process or act of creating an instance.
- **Object** – an object is an instance of the underlying class and inherits it properties. An

object is also something that you can physically place on a document. For example, a field, a box, or a picture. This is the first place at which the develop or end user will directly contact the OO interface.

There is a section of the DataEase help file (the OML Guide) that defines some of these terms and gives examples of how they are used. However, until you are ready to get into OML, you would be best off just glancing through this section. If you do any outside reading on OO, bear in mind that it may be discussing OO from a programming and/or from an ODBMS perspective which can be quite confusing since it doesn't really apply to our discussion here.

Terminology

At this point, it would be common practice to establish some additional terminology. But we need to see why establishing a vocabulary is almost as difficult as understanding it.

The first problem is that the team that originally implemented the code that lies behind DataEase never really developed a suitable set of terms to describe things – at least not one they shared with the outside world.

The second problem is that something as complex and malleable as the Multiview is difficult to describe in everyday terms.

The third problem is that the only way we can see the Multiview is the rather limited view we can get of it via a Document.

The fourth problem is that, lacking an established set of terms, the original developers invented some of their own or used terms that were already familiar. Hence we have terms like Table, Form and Field being carried forward when newer terms might have been more appropriate.

Lastly, where no terms existed to describe things, database developers invented their own. Truth be told, I am one of those people and I am guilty of perpetuating the use of a few terms that don't really adequately describe things. None the less, these terms are now all there and we have to use them. But that does not prevent us from defining what we mean by them in the context of DataEase!

One of the best examples of where terminology can lead us astray in DataEase is the term Table. As we have already seen, when we speak of a Table in DataEase, we are referring to a lot more than someone else, a SQL programmer for instance, might be thinking of. It gets even stranger if we are speaking to a former DFD developer since to them, Form and Table are the same thing (more on this in a moment). In fact, most never use the term Table at all.

It is tempting to try and set down here a vocabulary list that we can refer to as we move forward, but that would be futile since most of the terms we are going to be using need to be

defined in context as we come to them. So, with that in mind, let us move on and define, or redefine, what various terms mean when we get to them.

Tables vs. Forms - The War of the Words Begins

It is tempting to think of Tables and Forms as being the same thing. In DataEase we use Forms to create Tables; there is often only one Form per Table; and we almost never directly use the Table after it is created. But the two are not the same thing. Even in DataEase for DOS 5 (DFD5), where they were more closely linked, the Form and the Table were not the same thing. You would have to go back to DFD4.x before you would reach a point where they were, for all intents, the same thing.

Why is this important? Because I am about to tell you something that you may at first reject – what we think of as a Table in DataEase is not what is classically referred to as a Table at all.

I expect at this point that you are convinced that I have really lost it. Of course a Table is a Table. Why would they call it that if it wasn't? The answer is the same as to why everything in DFD was called a Form when it really wasn't. We call it that because that's the way the product referred to things. And we call things in DataEase Tables because that's how the product refers to them. But calling a horse a car doesn't make it one, now does it.

A DataEase Table is to a database Table as a house is to its foundation. A DataEase Table is built upon and contains within it a database Table but it is potentially much more than that. It is an extension of the Table into Multiview.

To understand this, we need to turn to SQL for a brief while. A SQL database Table consists of a set of columns. A column has a set of attributes such as the data type, size, and whether the column is required or not. SQL Tables can also have indexes and uniqueness and default values but these are generally thought of as being “add-ons” and not an integral part of the table.

But DataEase one ups SQL products by expanding a Table to include what are called Business Rules. Things like Prevent Data-Entry, Derivations, Validations and Virtual Fields. These enhancements, being at the Table level, are a part of the Multiview, and are therefore available to all Forms, Reports, and Procedures. This is a critical and significance between DataEase and other databases.

Going back to the ERD analogy for a moment, you can readily see that there is no easy place to put these additional enhancements. This is where the real limits of the analogy fall down. In fact, it becomes increasingly difficult to find an analogy that can be used to easily express the concepts contained in the Multiview. For that reason, we are going to just try and think of the Multiview as being a kind of three dimensional lattice work, not unlike an ERD on steroids.

One-Tier, Two-Tier, Three-Tier, More?

Let's take a short detour to try and bring things into a new perspective. Let's talk for a while about architecture. In specific, database software architecture.

Database software is generally made up of a series of layers or tiers. Each tier deals only with the tier directly beneath it. To the end user, DataEase would appear to be one-tier. The reason being that you deal with everything by way of an integrated user interface. But looks can be deceiving.

In a classical two-tiered architecture, the system is broken down into a back-end (the data) and a front-end (the user interface). In file-system databases (like dBase and DataEase) the front-end acts as both the data-engine and the user interface.

In client-server databases (like SQL Server) the back-end also contains a data-engine. This allows the front-end to be somewhat back-end independent and to leave most of the work to the back end. At least in theory. In practice a SQL front-end is usually ends up doing far more work than it is supposed to.

Three-tiered architecture was designed to overcome some problems and limitations created by two-tiered architecture. In particular, it was supposed to create an insulating middle-tier that allowed the front-end to do less work. But, in the long run, the front-end still ended up doing more work than it was supposed to.

The interesting thing about DataEase is that, even from it's earliest days, it has always been a multi-tiered application. Just how many tiers is where it starts to get tricky because they don't follow the traditional boundaries and tend to merge into one another.

As we have already discussed, DataEase Tables are much more than SQL tables are. They combine the traditional first-tier table elements with some of the traditional middle-tier elements. It is so difficult for the end user to separate these two layers that they appear to be one and the same. But deep inside the program, they are distinctly different. You can really only see this if you start to work with non-native data via ODBC or OLEDB.

In a similar way, it is difficult to distinguish where the middle-tier leaves off and the third-tier begins. The two seem to be so tightly linked that there is no clear dividing line. But, like the difference between tiers one and two, two and three also have their division inside the program.

For now, let's try and make an arbitrary division. There is the data at one end, and the user at the other, and everything else falls somewhere in the middle. That may not seem too helpful right now, but by the time we are done it will.

Fields vs. Columns - the War of Words Continues

This is another place where the choice of words in DataEase has lead to no end of

confusion. To almost any non-DataEase developer, a field is something that is strictly a form level object. To a DataEase developer, it is something found in a Table. In other words, we use the same term, Field, for both the Table level component and the Form level component. As is the case with a Form, in DFD this distinction was relatively meaningless. But in DFW and beyond, it makes a really big difference.

In DataEase, Forms and Fields can have properties that are independent of the underlying Table. Forms and Field are Instances of the underlying Table elements. They inherit the properties of the Table elements, but have properties of their own. For example:

- A Form may contain from one to all of the Fields from the Table.
- The order of the Fields in a Form is independent of the Table.
- A Form can contain virtual Fields that are not a part of the Table.
- A Form can contain a number of objects (lines, buttons, etc) that are not a part of the Table.
- A Field has visual properties that are not a part of the Field definition.
- A Field can have script elements that can do many of the same things as a derivation but are distinct and belong only to that Field on that Form.
- A Table can have multiple Forms built over it with different properties.

Projections

At the risk of confusing things further, I want to touch on a term that you may hear used from time to time – Projection. In the context of this particular topic, a Field on a Form is a Projection of the Field in the Table. There is a subtle but important distinction here. Despite the names being the same, a Field on a Form is not the actual Field found in the Table any more than the Form and the Table are the same things. They are both Projections of the underlying objects. In OO terms, they are Instances of the objects and Inherit their properties.

Think of it this way – the data lies at the bottom tier of the database. The MultiView (middle tier) incorporates, extends, and expands the data and then projects images (instances) of the data into the user interface (top tier).

Documents

Forms and Reports

Forms and Reports are pretty much the same thing. The differences mainly come down to the fact that you can't use a Report to enter or modify data. In almost every other significant way, they are the same. This largely has to do with the Object Oriented nature of DataEase.

Forms and Reports are both part of a larger class called Documents. Technically, Procedures (DQL's) are also Documents, but they are different enough from Forms and Reports that they need to be discussed separately.

One property that Documents have in common is that they are data-connected. That is, they are built on top of a Table. The exception to this is the Menu Document. The main reason that this Document type exists is that it is a carry over from DataEase Express. It was added to that product to facilitate the DFD453 to Express conversion utility. While you are free to use this Document type if you wish, you will find that you are much better off by creating a Table that can be used by Menus.

Documents and Object Orientation

To fully understand why DataEase and Documents are so closely connected, you need to look at Windows and the way in which it uses Object Orientation.

If we go back to DFD for a moment, that program used the Form as the basic building block. It even went so far as to use forms (system forms) as the interface for the developer. This concept made the program very easy to use since everything shared a common interface. But it also put some limits on what you could do since it was difficult to work outside the form model.

Fast forward to DataEase Express. Developers were faced with a decision on whether to use a single building block for Forms and Reports, or to develop a separate model for Forms and Reports – the choice seemed obvious. By making the Document a common interface for both Forms and Reports, they not only simplified the product but also gave the user a consistent interface to work with. But, as was the case with DFD, this did create some limitations.

But that still doesn't really answer the question of how these two things are intertwined. Here is what Pete Tabord had to say on the subject when I asked him about it.

“Hmm. It's difficult to explain precisely because MS [Microsoft] don't make it explicit. I guess they would say 'that's just the way it works'”

Windows has two basic elements for screen display, the modal window (ones with x in

the top right corner) and the non modal window (the pop up type typically seen as dialogs)

These elements/objects can be built on, in the normal OO way, and MS recognizes certain extra more complex types which have some direct support in the API.

Four of these elements are the application, the document, the dialog and the control. in general, applications have menus/toolbars and dialogs, documents have controls. These are thus OO type elements, in that what element you choose to map to in turn decides what other elements are available to you.

Given that the root of a program, in Windows terms, must be an application, DataEase itself has to be an application. So we are into confusion already, as the `_DataEase_` application is not that at all! Our application (database) doesn't map to any Windows concept. In Windows terms, our application is a collection of documents, and we decided there would in essence be only one document type in DataEase , which is sub-classed into forms, reports, procedures and menus.

You can of course have windows which are your own type, and that is what we do to create an application - there is an application object/window which is the 'root' of any application. It has some special 'types' of our own, such as the catalog window, although as time has gone by and Windows has introduced more standard controls we have, where possible, replaced our own custom things with Windows ones. Really, both us and MS are doing the same things - customizing elements that exist in the operating system, just as in DFW you never can create a new object, you are customizing an instance of one already defined by us.”

While that certainly goes a long way to explaining it, it still left me puzzled for a time until I began to think back to the OO hierarchy. According to OO, all things must have a root Class to which they belong. And while “Document” is not a Class, it does act like a class in some ways. It is really very much like a word processing document, in that you have to first create an instance of the default document before you can start to put any objects (words) on it. In DataEase, Documents are used as the means to access data – so you always have to start by creating a new document.

The Model

Documents are the point at which the Multiview is brought into focus on the computer screen. This is a bigger trick than you might imagine because, as pointed out before, the screen is two dimensional while the Multiview is not.

If you open a Document in Designer View and go to Query By Model, you will be looking at a kind of mini-ERD called The Model.

Documents are a subset of the Multiview. The Model is the graphical representation of

the subset of the Multiview that comprises the Document. Documents are also a projection of the Multiview, like the light on the wall from a flashlight. Don't confuse the light on the wall with the flashlight – they're not the same thing.

Documents can extend the Multiview. Documents are not limited to projecting the elements of the Multiview, you can add objects to Documents that are limited to that document. You do this when you add a Document Only Virtual to a Form or a statistical operator to a Report.

In Object Oriented terms, Documents instantiate an instance of the Multiview. When you open a Document, you are pulling together the elements of the Multiview that make up the Document. Unlike other programs that use compiled forms and reports, DataEase uses a runtime interpreter. This means that each time you open a document, you are re-instantiating an instance of the objects contained within it. This also means that if there have been changes in the Multiview, those changes will be incorporated into the Document when it is opened.

The Document and its Model are interactive. Change one and you change the other. This is because Documents and Models are different views of the same thing – the subset of the Multiview that they represent. And this is pretty much what this whole discussion has been leading up to. But a Document and the Table may or may not be interactive.

Table Owning Forms

One other major difference between Forms and Reports is that a Form can define a Table. These types of Forms are known as Table Defining Forms or Table Owning Forms. While Defining is the more proper term, Owning is often used to keep the abbreviation of TDF (Table Defining Form) from being mixed up with the DataEase file type of TDF (which is the actual Table definition). Confusing, isn't it?

The concept behind the Table Owning Form (TOF) goes back to DFD and our discussion of Forms vs Tables. Having a Form define a Table is a part of the Rapid Application Development (RAD) concepts that DataEase embraces. The downside to TOF's is that changes in the Form can change not only the Model, but the underlying Table.

Table Using Forms

A Table Using Form (TUF) is exactly what the name implies – it uses the Table without Defining it. While there can only be one TOF per Table, there can be any number of TUF's per Table. This can create some interesting possibilities. For example, you could have one Form for one set of users and a different Form over the same Table for a different set of users. The two Forms need not have the same fields, the same subforms, the same security, or the same properties.

The ability to use multiple Forms over the same Table is probably the biggest change from the DOS version, yet it is also probably the least used feature. The bulk of the Forms built are TOF's and these are the same forms that the users are given. This is also the most unfortunate consequence of the RAD concept of simultaneously building Forms and Tables.

The primary reason for not giving the user the TOF is that it is entirely too easy to inadvertently mess up the Table while changing things around in the Form to suit the user. This is not the time to go into a dissertation on "best practices", but it is worth noting here that many people (myself included) have learned this lesson the hard way.

Data Filters

There are two ways to set a Data Filter in a Form, either at the designer level or at the user level. The only type of filter we are going to discuss here is a designer level filter.

If you open any Form in Designer View and go to Document / Query By Model, you will see a button there labeled "Select this Table's Records If:". This is referred to as setting a Filter. A Filter set here cannot be changed or removed by the end user. For example, you can set a Filter on a form so that the only records that will be displayed in that form are those with a particular State, or City, or ZipCode, or almost anything else you want.

Setting a Filter in a Form effects only that particular Form. It does not effect the underlying table, nor does it effect any other Forms that use the same Table.

Containers

To understand Containers, we need to look at a Form from an OO perspective. Open any Form in Designer View and click on View / Form/Record Labels. You will see that there are now at least two "bars" added at the top of the Form. One labeled TableName Main form and one labeled TableName Record. These labels represent the Form Object and the Record Object that all the Fields sit upon. As a general rule, you should leave these labels off since it will make it difficult to line things up correctly when you switch to user view, but for this section it will help to leave them on.

If you click on either of these labels, you will generally find that neither will move. The reason is that a Form is generally laid out as one record per page and the Record is expanded to it's maximum dimensions so it has no where to move. However, if you change the layout to more than one record per page, the Record will shrink and you will now see that you can move the Record Object inside the Form Object. Once you have separated the two, you will now see that you can click anywhere on the Record and be able to move it.

The reason you can move the Record is because it is setting within the Form. You will not be able to move the Form because there is nothing below it to move it about in. This is where the term Container originated. The Record is *contained* within the Form.

If we now turn to the objects (Fields, Labels, Buttons, etc) that are within the Record, you will see that the Record contains them just as the Record is contained within the Form. Try as you will, you cannot drag an object out of the Record and onto the Form. However, you can create a Text Object on the Form and then drag it so that it is on top of the Record which can make it look like it is in the Record even if it is not.

The concept of Containers will become more clear if you have a form with a subform. On the body of the Form the Record Object will contain the subform. There is not only no way to move the subform container out of the record container, it would make no sense to do so since the records of the subform are subordinate to the main form records – it cannot be any other way.

The term Container originated when developers (myself included) were trying to discuss Form and Report layout. It's not easy to discuss something like this without reverting to familiar words and the word Container just seemed to fit. It is not, however, a perfect term.

Object Manipulation Language (OML)

From the DataEase Help File. © DataEase International Ltd:

DataEase 6 introduces the ability to define OML (Object Manipulation Language) Scripts which can be attached to objects, thus customizing an object's response to selected events. An "Event" might be a mouse-click, or a field value changing. When an Event takes place, the relevant OML script is triggered and runs like a piece of DQL.

Event Scripts can alter the appearance and contents of any object on your document. As an extreme example, you could write an OML Script which - when a button is pressed - alters the size, shape, colour, font, screen co-ordinates, and contents of every single object on your document.

Additionally, OML Scripts can run a sub-set of DQL, so they can enter/modify/delete data, call programs and procedures, and carry out most other tasks which are usually assigned to a DQL procedure. (In fact, OML Scripts are part of DQL, but because the syntax is different, it is convenient to refer to OML Scripts and DQL as if they were separate scripting languages).

OML Scripting is pretty much a topic unto itself. While I would like to take the time here to delve into it, it is really beyond the scope of this article. Therefore, I will leave you with this advice and follow up later with another article: Until such time as you have mastered the other aspects of DataEase that are discussed in this article, you would be best served leaving OML alone. There are many reasons, but suffice it to say that getting into this too soon will simply frustrate you.

Reports

In addition to the things discussed in the Documents section, there are a few more things that set Reports apart from Forms. Not functional differences, mind you, but differences in the way they are used. For the most part, Forms are setup to display one record per page. Further, just as developers seldom create more than one Form per Table, they seldom set any Filters in Forms.

Reports, on the other hand, are generally laid out with the columns going across and rows going down, much like a spreadsheet is laid out. Further, there is generally some kind of filter set that restrict the records that appear in the Report.

Interactive Layouts

The report layout (body) is the first place where people are likely to have problems with DataEase. The reason is that the QBM and the layout are interactive. They are both representations of the same thing and both can manipulate the same thing – the Model.

Here is an interesting experiment you can do to see how this interactive process works. Create a report using a small table. Once the layout is done, open the QBM dialog and notice that each field in the table that is contained on the layout is in bold. Now go back to the layout and remove a field, then look at the QBM again. You will notice that the field you just deleted is no longer in bold. Now try adding a field to the layout and look at the resulting QBM.

What you are seeing here is the interactive nature of Reports. In fact, even with their differences, you will see that changes to Reports and Forms are largely the same. The reason is that Reports and Forms have some common Class inheritances. They behave the same because, just below the surface, they are the same.

What You Want, Where You Want It

Simon Irwin of Sapphire coined this phrase as far back as DataEase Express. What he was referring to by this was the concept that you need to create a model that looks like what you want your format to look like. The failure to follow this simple concept has created more work and problems for people than anything else that I can think of. Here are two simple examples of this at work:

You select the fields to include in a report by clicking on them in the QBM. The order in which you click on them is the order they are going to appear on in the body. A little forethought here will save you a lot of rearranging later,

Groups are one of the more mis-understood things in DataEase. Most people think of groups as just another way of ordering the information. In fact, the ordering of data is largely just a useful by-product of the way groups work. The command “In Groups” is,

actually, a Relational Operator.

Grouping creates what could be, arguably, called a pseudo-table. I say arguably because Pete Tabord has gone to great lengths to explain to me and others that no matter what it looks like to us, it is not a pseudo-table. Never-the-less, if you create a Group in a Report, you will see that it creates a new table-like structure that, in many ways, is analogous to a Form – Subform construction. It even uses those terms in the report body. A good term to describe this from the perspective of the Model is a node.

Turning our attention now to the body of the Report, we will see that we have four containers. The Main Form, the Main Form Record, the Group Subform, and the Group Record. Whatever fields we chose in the Main node in the QBM will be in the Main container in the Body. Same for the Group node/container. This said, it becomes rather self-evident that to produce a Report Body with things where you want them, you have to select those things in the proper nodes in the QBM. But you would be amazed at how long it can take to fully grasp this simple concept.

The Report and the Multiview

To get technical for a minute, a Report projects or instantiates an instance of the Multiview. As Tables and Fields are selected, the Report builds up a model complete with all the necessary information to create a kind of mini-Multiview. When the Report is executed, it navigates the nodes of that Multiview in much the same way as you would if you were moving through records in a form.

As has been previously stated, Documents can manipulate and extend the Multiview and create structures that are not a part of the underlying Multiview. Grouping is an excellent example of this and why the term pseudo-table is inaccurate.

Reports - The End All and Be All?

As was previously mentioned, Reports were all you had in Express – there were no Procedures. This was viewed by many as an extremely short sighted approach and was in no small way responsible for the failure of the DataEase community to accept Express. However, in retrospect, Reports were no where near as limited as first appeared. In fact, even today, they are an extremely powerful tool and well worth the time it takes to learn how to use them.

It is not too much of a stretch to say that, if you haven't spent time working with Reports, you shouldn't even be thinking about touching Procedures. We'll see why next.

Procedures (DQL)

And So It Begins...

Procedures are, arguably, the most difficult aspect of DataEase to explain and to use. This is particularly true when it comes to long time DFD users. The reason relates to the fact that they were not a part of the original design and are thus considerably different in many respects from their DOS counterpart.

DQL is the scripting language of DataEase and is used both as a procedural language (enter, delete, modify records) and as an advanced reporting language. It is this latter role that we will go into here.

As has been discussed, Form and Report Documents are interactive between the Model and the Document Body (Format). Procedures, however, add a third layer to this equation that pretty much pushes all the previous rules out the window. In other words, Procedures have their own sets of rules as to how they behave and how you must deal with them.

To understand why this is true and how it came to be, we have to put ourselves in the shoes of the design team tasked with fitting DQL scripts into Express when DFW5 was being created.

Get Out the Shoehorn

Fitting DQL into Express was only going to be accomplished in one of two ways. One was to more or less abandon the QBM-Document mode and create a completely new way of producing a report format. The other was to find a way to combine DQL with a Report. It has been argued that the prior method may have been the better choice, but it was simply not practical at the time. This left the team with the second approach.

The solution was at once ingenious and at the same time, a ticking bomb. The DQL component would be grafted on to the QBM-Document model in such a way as to let the DQL create the Model which would, in turn, create the Body. Simple!

Why Does The Body of My Procedure Keep Changing?

When a simple Procedure is first created, the layout created is generally fine. The first problem to surface comes when users try to reproduce DFD DQLs in DataEase. Since most users have not spent much time working with Reports, and few have ever heard of "What you want, where you want it", the results are almost predictable. They are going to have a lot of trouble.

The fact of the matter is that DFD allowed users to pretty much do whatever they wanted to do in the format with relative impunity. That is not the case with Reports and is even more emphatically not the case in Procedures. The reason is that the way in which

Procedures create a Body makes the interaction of the Model and the Body impractical.

We can skip over some of the more egregious problems that came up when DFW5 was first released and instead concentrate on DE65 and how changes in the DQL can effect the Body of the Procedure. The most important thing to remember here is that Procedures are a one-way street when it comes to the Body. Changes in the DQL will necessitate changes in the Model which will necessitate changes in the Body. This is simply a fact of life and no amount of cursing will change that. Like everything else in DataEase, the solution lies in understanding how it behaves in order to get it to do what you want it to do the in the first place.

What You Want, Where You Want It, Revisited

There is no better way to illustrate the issue of how the DQL effects the Body than to have another look at a simple Grouping DQL.

```
for Orders ;  
list records  
  CustomerCode in groups ;  
  CustomerName ;  
  OrderDate .  
end .
```

The desired output for this DQL is to have the CustomerName appear once per group and at the same level as the CustomerCode. If this DQL were created in DFD, one would simply take the CustomerName out of the .items area and move it into the .group header. If you do the same thing in DataEase you will be OK until such time as you go back to the DQL and make a change. At that point, DataEase will take the CustomerName out of the group container and move it back into the record container. The reason is that this is where the DQL says it is supposed to be. In other words, it is putting it exactly where you tell it to put it.

If you want to put the CustomerName into the group container of the body, you need to do that in the DQL itself.

```
for Orders ;  
list records  
  CustomerName ;  
  CustomerCode in groups ;  
  OrderDate .  
end .
```

When you specify “in groups”, you are creating a new node in the Model. Look at the QBM of either a Report or a Procedure that groups and you will see that additional node. The Group: node will contain all the fields that belong in that container. Anything that is before the “in groups” in the list records belongs in the group container. Those things after it belong in the group sub-container. If something is in the wrong container on the Body of

the report, you need to return to the DQL in order to move it.

If you go back to our discussion of Groups in the section on Reports, you will see that we are discussing exactly the same thing. You need to think of a DQL in terms of the Model it will create, because that Model will dictate how the Body is going to be laid out. Pull the string, don't push it.

How DQL's Process Data

The structure of the DQL effects not only the way the report body is laid out but the way the data is processed. Essentially, the DQL specifies the table(s) and the selection criteria that the Procedure will use. This comprises the MultiView that the procedure will now act on.

In DFD, a DQL would process the data and send the output to the device (screen, disk, printer) a record or line at a time. DataEase builds up the output as a complete set of pages which are sent to the output device all at once. This makes some significant differences when it comes to some of the tricks DFD developers use to use. Among other things it makes the kind of detailed line by line control of the output used in some DFD DQL's impossible.

There is another, far more subtle, effect that this method of processing has. The sequence of commands in a DQL can make a difference in how the final data is structured. The exact order in which things happen in a DQL is a function of both the structure of the DQL and the structure of the Model it creates. In general, you can assume that things will pretty much happen in the order they are found in the DQL. For example, moving a line that increments a variable from before a list records to after it can effect it's value since, in the latter case, the record pointer may have already moved onto the next record in the "stack" when it increments.

Data-Entry Forms

A data-entry form is used to acquire input from the user at the time a report is run. This input is most often used to set the selection criteria in the report (date range, customer, etc.) But, as with the DQL script itself, data-entry screens were not a part of Express and also had to be grafted onto DFW.

While there are almost no outward signs of problems with these constructs, they do create some internal problems for the programs developers. Just exactly what these problems are is not known to me, but Pete Tabord has stated that he would like to find a better way to implement them in a future version. In the mean time, many developers have taken to using other ways of obtaining this information and passing it on the the DQL via CDFs.

Input Using

Input Using was introduced into DFD as an alternative to data-entry forms, but it was soon being used in other ways. The interesting thing about input using is that it was a rather clever trick that was introduced in DFD but was rather more difficult to reproduce in DFW. Consequently it never really worked all that well.

It was also something that was not very widely used so making it work better in DFW was never a high priority. As it stands now, input using is on a hit list of things that the program developers would like to remove.

Output

Output was added to DFD as a means of overcoming the fact that it could not deal with either multiple all's or multiple list records. Despite the fact that the major structural changes in DFW made it possible, output was retained in DFW for backward compatibility. However, like input using, it is not a very good fit and, as it is essentially obsolete, is also on the list of things to be removed in a future version.

All vs For

The relational operator All, is probably the most used DQL construct that really has no place still being in the language. The ability to use a For loop in place of an All is also the most under utilized new feature in the language, despite the fact that is a significantly superior way of doing things. This, alone, will tell you how much current users still rely on DFD habits.

Export

In a way, I feel somewhat personally responsible for the existence of this critter. It seems that I was at DataEase working with a member of the development staff on a problem in Express when DFW was being created. While there, I had a look at something one of the developers (Bill Woods) had stuck into DQL as a debugging tool. After a brief discussion of why it was there, I immediately began lobbying for it's inclusion in the final product.

To understand why Export exists, you have to look at how the DFD method worked. In DFD, it was a simple matter of changing the output of a DQL from Printer to Disk. This is because DOS programs do not have to use any Operating System printer drivers. The only difference to DFD between output to a Disk and a Printer is that the file went to a drive rather than a printer port (internally that's pretty much the same thing). However, DataEase uses the printer functions built into Windows which do not permit this same thing.

What Bill Woods wanted to be able to do was run DQLs and send the output directly to a disk file, bypassing the Windows printer drivers completely. He also bypassed the need to

format the Body of the Procedure and could put comments and notes into the output. It should be pointed out again, that this was something he did to act as an aid to debugging the work he was doing in DQLs. It was never intended to be a part of the finished product.

But, as development time passed, it quickly became clear that the Form Export function was not going to be flexible enough to meet the needs of the development community. So, as oft-times happens with things, what started out as a quick-and-dirty add-in became a part of the final product. Warts and all.

The biggest problems with the Export function are that it is not well documented and is far from being intuitive or easy to use. In fact, because it is so closely tied in with the way DQLs process data and because so few users understand *how* DQLs process data, it is one of the most frustrating things to use in DataEase.

Learning New Tricks

The issue of crafting a DQL to produce a desired output will go far towards making Procedures easier to work with, but it will not solve all your problems. Over the years, there have been hundreds of “Tips, Tricks and Techniques” articles published for DFD, but so far there have been relatively few such articles dealing with DataEase. One reason is that many of the people who wrote the original articles are no longer around. Another is that many of those who are still around are still learning themselves. And lastly, many of those who have learned quite a bit are often too involved with projects to spend much time writing about them.

The problem with many of these tricks is that, by and large, they focus on a rather narrow issue and fail to teach the reader much about how things work. This is the old “Give a man a fish...” conundrum. Hopefully this and future articles can start to fill in this void.

Lessons Learned

If I have learned anything in the past few years working with DataEase it is that many of the old ways I had for doing things in DFD are not really applicable to DataEase. Furthermore, with each passing month I find new and better ways to use the program.

I have also learned new ways to look at and interact with the data itself. This has been the biggest challenge since I am so attuned to thinking of things in terms of how DFD worked. This has also been the biggest boon since I can now provide my clients with a better product.

The challenge that DataEase presents is a daunting one. It is at the same time a database development tool, a database engine, and a Windows user interface tool. Becoming an expert in any one of these areas is a full time job for some people. And yet hundreds of people with no formal training in any of these areas are building and using databases with DataEase.

For the experienced developer, the challenge lies in learning how to use the product more effectively. This means not only using it the way it should be used, but to find new and inventive ways to use it. This is frequently referred to as “thinking outside the box.” And when it comes to that topic, the biggest box of all is the one we bring with us from prior versions of DataEase or from other products.

Another lesson I have (re)learned is that sitting down and writing things out helps me focus my attention in on something. In this case, it has helped me put down in print some of the various techniques I have developed. Most of these things are currently for my own use – a reference library of things I want to more fully develop. Others are related to specific projects and may have no use outside that area. Others still are lists of things that I want to figure out how to do, or how to do better.

The best advice I can give to the people reading this is to start to make lists like I have done. Make lists of things you don't understand, of things you do understand, of things you want to understand better. Keep working on those lists and testing things out and experimenting. Don't be satisfied with doing things the same way today as you did them last year – find new ways. Figure out what works best for you and why.