Foreword by Marco Cantù

I don’t remember when I met Cary (and his wife Loy) for the first time, but it must have been one of the early Borland Conferences, in which Cary was still focused on Paradox and I was focused on C++ development and the good old Object Windows Library. That was in the early ‘90ies, the years before Delphi was released. So Cary and I came to Delphi from two different angles, database development for him and language and Windows API development for me.

Over all of the years we have known each other, I always sought Cary’s insights on the database side, and even if I ended up with a significant experience in this area through consulting work for large projects, I think few in the Delphi community have worked on so many and so large Delphi database projects as Cary. We have met many times over many years, at conferences where we’ve both been speakers and while organizing the Delphi Developer Days events together for a few years. And we still meet at conferences and have online discussions now that I’m working for Embarcadero Technologies as Product Manager. I certainly still value Cary's advice and ideas in shaping Delphi’s future, and Delphi's data access future in particular.

Now what is relevant is that Cary is an expert is most areas of the product, from language to visual component libraries, from REST architectures to core RTL, but if I had to pick an area I know he truly masters unlike anyone else, this is certainly the database access side of the product. And given that database access has been a cornerstone of Delphi since its inception, this is not a side feature, but one most, if not all, developers have to learn about and delve into.

Starting with the good old BDE technology, in fact, and through a variety of data access options over the years (including dbGo, dbExpress, and FireDAC) and now even open to cloud and REST data sources, database access and database data manipulation have been among the most precious and more relevant features in Delphi. If the BDE is now officially retired (as it doesn’t ship directly with XE7 any more), all of the other database technologies in Delphi keep providing a solid foundation and offer great data access power, a

complete design-time experience, data binding using either the classic data-aware component model or new Visual Live Bindings, and portability among platforms.

Considering the different architectures Delphi allows you to create (from local data, to client server, to multi-tier), the fact you can connect to many different backend relational database engines, and the complexity of these different scenarios, a single book covering database access in Delphi won't probably suffice. But there is indeed a single component that has been at the heart of Delphi database architectures of different types for many years now, and this is the ClientDataSet.

The ClientDataSet component has a long history and emerged from earlier Borland in-memory table technologies. For many years, this was one of the only few components in Delphi not to ship with the source code: While the source code of the component is now available, most of what it does is calling an underlying library called MIDAS (originally an acronym of "Multi-Tier Distributed Application Services Suite"). Now it is true that the source code of this library is currently available with the product, but that source code is written in C++, not in Object Pascal.

This hasn't prevented an evolution of the platforms this library and the ClientDataSet component can be used on: Windows 32-bit, Windows 64-bit, Mac OS X, and also iOS and Android platforms. The practical effect is that the ClientDataSet component can be used outside of Windows for truly single-source multi-device applications running on your phones, tablets, and desktop PCs.

With this long history behind, the ClientDataSet component is an extremely powerful component, used by countless Delphi developers, but much more powerful and feature rich than most Delphi developers know. This is the reason the ClientDataSet is the only component in the product to be the main focus of a full-length book—Cary's first edition of this volume—and well worth its own book and this new edition.

Not only having a book for the ClientDataSet component helped many developers understand its power and features, but also Cary's work certainly helped underlining the incredible role of this component in many modern Delphi architectures, ranging from local database storage to complex multi-tier and web service architectures.

Now the book does a terrific job delving into each and every of the complex and assorted capabilities of this component, many of which are largely unknown to Delphi developers, like aggregates and cloned cursors, to name just a couple out of many. While focusing on the ClientDataSet component, the book offers a lot of general advice and information for Delphi database development in general, and it goes in depth into the overall architecture of the base TDataSet class of the Delphi database runtime library. This includes covering features that are not unique to the ClientDataSet component, like the dynamic fields management introduced in Delphi XE6.

The fact that a single component has the focus of an entire book is in itself a testament of the amount of features and of the flexibility the ClientDataSet component has. But also a clear indication of the knowledge Cary has about all of the details of this component, combined with his ability to explain core concepts and build great demos to showcase them in practice. The book, in fact, is not short of usage scenarios and direct advice most Delphi developers can apply in their day-to-day coding.

While in the future other technologies, like the FDMemTable component of the FireDAC library, might end up taking its role, the ClientDataSet is a component every Delphi developer should know about. Until there is a book on FDMemTable, learning about ClientDataSet will help you understand many core ideas that other in-memory database tables share.

So it was with great pleasure that I accepted Cary's (and Loy's) invitation to write the foreword of this book: A great book that is focused on a cornerstone Delphi component and that I recommended to countless developers in its first edition. And while I'll be waiting for new books by Cary covering other Delphi technologies (both in the database area and outside of it), I'll make sure I keep listening to his suggestions and advice to make Delphi an even better database development tool than it is today. Today you can have a ClientDataSet component on your desktop computer, tablet and phone apps. We'll see to which platform the future will bring a ClientDataSet tomorrow...

Piacenza, December 2014

Marco Cantù

RAD Studio Product Manager at Embarcadero Technologies

Introduction

Welcome to Delphi in Depth: ClientDataSets 2nd Edition. Let me begin by thanking everyone in the Delphi community for their support of the original release of this book, *Delphi in Depth: ClientDataSets*. I appreciate the wonderful reviews that that book received, and thank everyone for their warm feedback.

When the first edition of this book was released, one of the more frequent comments I heard was "Can you believe it? Cary wrote an entire book about one component!" Yeah, it sounds funny, but it's only partially true. While the ClientDataSet is only one of the many hundreds of components that ship with Delphi, it is more than a component. It's really more of a framework, or a philosophy, even.

In some cases, ClientDataSets serve the simple role of providing a layer of abstraction between an application and its underlying data. However, as you begin to master the various capabilities of this powerful component, you are likely to find surprising uses for ClientDataSets that previously would not have occurred to you. It is from this perspective that I find ClientDataSets utterly fascinating. If I hadn't understood the power of cloned cursors, or the flexibility of nested datasets, I would not have written some of the more creative solutions to my application challenges. Indeed, to this day I marvel at the utility and flexibility of this noble component.

It's been three and a half years since Delphi in Depth: ClientDataSets was originally released. A lot has happened in Delphi during that time. FireDAC was introduced, and a new in-memory dataset has started to ship with the product: FDMemTable. There has also been the introduction of a cross-platform component library, FireMonkey, as well as a new data binding mechanism called LiveBindings.

Speaking of FDMemTable, let me make one thing perfectly clear. FDMemTables do not entirely replace ClientDataSets, they compliment them. If you are using ClientDataSets now, you are not going to simply go through your existing code and strip out ClientDataSets, replacing them with FDMemTables. For one thing, their APIs (application programming interfaces),

while similar, are not identical. More importantly, in many cases, they do what they do differently.

On the other hand, FDMemTables are very likely to play an increasingly important role in Delphi's future. For example, while ClientDataSets are still an important player in DataSnap applications, I anticipate that future implementations of DataSnap will rely on the capabilities of FDMemTables.

Which brings me back to this book. With the introduction of FireDAC, and the removal of the BDE from the default installation of Delphi XE7, I felt that it was time to revisit Delphi in Depth: ClientDataSets. In this edition, I have re-worked all of the code examples that made use of the Borland Database Engine and the sample Paradox tables that ship with Delphi, replacing them with FireDAC and the sample InterBase tables. This gives the examples a more modern feel, replacing the obsolete local Paradox database with a modern, transaction-based remote database server.

But that's not all. I have added a significant amount of additional material. In Chapter 9, *Filtering ClientDataSets*, I have added a lot of new material about special filter expressions. And in Chapter 4, *Defining a ClientDataSet's Structure*, I have included material on the new FilterOptions property added to TDataSets in Delphi XE6. You will also find a number of new code samples.

And, as you can imagine, I have gone back over every single page of this book repeatedly, and have updated and improved everywhere I saw fit. You will even find discussions of LiveBindings, where appropriate.

I have also taken some material out. Specifically, I removed the three chapters on DataSnap that appeared at the end of the original release of the book. I felt that much of that material didn't apply directly to ClientDataSets, and also covered too much of the old COM-based MIDAS technology. Don't get me wrong, I believe that DataSnap is an important technology, but it is changing for the better. In fact, DataSnap deserves to be covered in depth in a book of its own, and, time permitting, I'd like to be the author of that book.

While there are many updates and improvements to be found in this book, I want to be honest and say that much of this book is unchanged, fundamentally, compared to the first edition. I worked hard to get the first one right, and more than three years later I've been pleased to find that much of what appeared in the original manuscript was no different than what I would write were I starting on this book from scratch today. I did not fix what was not broken.

What to Expect

This book takes an in-depth look at ClientDataSets. It begins with an overview of what a ClientDataSet is, and describes some of the ways that ClientDataSets can be used in your Delphi applications. It continues with a close examination of one of the more common uses for a ClientDataSet, reading and writing data from an underlying database through its interaction with a DataSetProvider.

Next you learn how to define the structure of a ClientDataSet, a process that can be undertaken both at design time as well as at runtime. The relative roles of FieldDefs and Fields are discussed, and the roles of persistent versus dynamic Fields are considered. How virtual Fields differ from data Fields is also explained.

Chapter 5 introduces indexes, including the difference between persistent and temporary indexes. The relationship between a ClientDataSet's indexes and those of an underlying database are also considered.

In Chapter 6, I take a long look at the change cache, the mechanism responsible for caching updates. Here you learn how to enable and disable the cache, how to detect changes to it, and how to modify the contents of the cache.

Chapters 7 through 9 introduce topics that will be familiar to many Delphi database developers, including how to edit, navigate, search, and filter data. These discussions, however, demonstrate the flexibility that ClientDataSets bring to these features, providing you with capabilities not available in other DataSets.

Chapters 10, 11, and 12 introduce some of the more interesting features found only in ClientDataSets. These features include aggregates, cloned cursors, and nested datasets.

Class Name Conventions

Consistency is important when writing a book. And while I know that there are always some inconsistencies that slip between the cracks, a lot of effort went into using terminology consistently in this book.

One of the areas that was most challenging in this respect was how to refer to objects. By convention, Delphi classes begin with an upper case T, and in most of my Delphi writings, I have carried that convention into the discussion of

objects. For example, I might talk about ClientDataSet instances as TClientDataSets.

I did not adopt that convention in this book, however. I did not like how it looked to be talking about TClientDataSets. I though that referring to instances of the TClientDataSet class as ClientDataSets read better and looked more natural. As a result, when I refer to an instance of the class, I omitted the T. In those cases where I refer to the actually class declaration, I used the T.

While this approach worked well, for the most part, there are some places were it made the prose a little more difficult to read. This is especially true in the discussion of Fields, instances of classes that descend from TField. Frankly, those discussions, especially those in Chapter 5 dealing with ClientDataSet structure, were challenging to write clearly, since sometimes I was referring to a field (lower case, and referring to a column of data from a database table), sometimes to a Field (an instance of a TField descendant), and other times referring to a TField (the actual class definition).

The Database

There was another significant challenge in this book, besides how to refer to objects, and this was related to the code samples. In short, I felt that it was extremely important to provide code samples that made few assumptions about your Delphi installation, and required little or no additional configuration.

One place where this was especially difficult was in the selection of a database. Many, but not all, uses of a ClientDataSet involve interacting with a database. Where would the database come from, and what data access mechanism should be used to interact with that data?

When writing the original version of this book I settled on using Paradox tables and the DataSets of the Borland Database Engine (BDE). I received a few criticisms about this decision, but I stand by it. The BDE shipped in all versions of Delphi to that point, and could be employed in the samples with virtually no effort on the reader's part. While I totally agree that even then the BDE was outdated, that had no bearing on the value of the examples I needed to create for the book.

With the release of Delphi XE7, the BDE, which has been deprecated for a long time, was officially removed from the default installation of Delphi. (Users of the BDE can still download the BDE for installation, but it cannot be known

how many future releases of Delphi will permit this). It became time to use something else.

What I settled on was using FireDAC and InterBase, both of which are installed by default in the most recent releases of Delphi. FireDAC, which was released with Delphi XE3, is a comprehensive TDataSet implementation that supports a very wide range of databases. InterBase is Embarcadero's powerful remote database server. Together, and along with the dbdemos.gdb InterBase database that ships with every version of Delphi, I found the perfect combination of power, real-world configuration, and convenience.

The code samples that accompany this book, whose download is discussed in detail in Appendix A, are configured to work with Delphi XE5 and later. While FireDAC was released with Delphi XE3, and which required a separate install, still used the component names and unit names of its previous incarnation as a third-party data access framework (AnyDAC). I anticipate that this book's code examples will also work with versions of Delphi that follow XE7 (since between XE5 and XE7, I only needed to include a single {$IFDEF} in the sample uses clauses). Nonetheless, if future versions of Delphi need additional updates to the code samples, I will post information about that on this book's web site. Once again, see Appendix A for details.

If you are using Delphi XE4 or earlier, you should consider using the code samples that make use of the BDE and Paradox tables. These are still available from this book's web site, and with almost imperceptible exceptions, are no different, from a demonstration perspective, from the FireDAC/InterBase versions. For XE3 and XE4 users, you are welcome to make changes to the FireDAC examples to accommodate the component and unit names found in those versions, but I don't think it would really be worth your while. Either use the BDE/Paradox versions, or upgrade to a later version of Delphi.

There is one database that is used in some of this book's examples that did remain the same between book editions. There is a file-based ClientDataSet, named BigCDS.cds, and it is found in both the FireDAC and BDE code sample downloads. This is a database of 25,000 records that I generated, and which you are free to use in conjunction with this book's code samples (please see the readme.txt file found in that database's directory for copyright information). If you extract the sample code files from the code sample downloads, retaining the directory structure of the original ZIP file, you should require no additional configuration to use this database.

Let me say one more thing about the InterBase database used in the FireDAC code samples. *Almost every version of Delphi has located these files in a different directory.* While the same is true about the Paradox tables, those are associated with a BDE alias, or database name (DBDEMOS). Regardless of which version of Delphi you are using (Delphi XE6 and earlier), that BDE alias is defined during installation. Since the BDE/Paradox code samples use that alias, you do not have to worry about the path to the sample Paradox tables (although you may have to run Delphi with administrative privileges in Windows Vista and later in order to access the PDOXUSER.NET file).

While the InterBase dbdemos.gdb database is located in the same directory as the sample Paradox tables, the FireDAC FDConnection component must be configured to point to this directory. Since the path to this directory has changed with every recent version of Delphi, this poses a particular challenge. Fortunately, Delphi's configuration includes a number of environment variables that are defined when you run a project from within IDE (integrated development environment). And, the variable DEMOSDIR points to the parent directory of the folder named Data, and this is where the sample database files reside.

The OnCreate event handler of the main form in every FireDAC/InterBase code sample includes code that will point the FireDAC FDConnection to the data folder at runtime. This code looks something like this:

```
var
  DataDir: string;
begin
  DataDir := GetEnvironmentVariable('DEMOSDIR');
  if DataDir.IsEmpty then
  begin
    ShowMessage('DEMOSDIR environment variable is not set. ' +
                'Cannot continue. See Appendix A, Setting the ' +
                'DEMOSDIR environment variable for more info');
    exit;
  end;

  FDConnection1.Params.Clear;
  FDConnection1.Params.Add('DriverID=IB');
  FDConnection1.Params.Add('Database=' +
    GetEnvironmentVariable('DEMOSDIR') + 'Data\dbdemos.gdb');
  FDConnection1.Params.Add('User_Name=sysdba');
  FDConnection1.Params.Add('password=masterkey');
  FDConnection1.Params.Add('Server=localhost');
```

As you can see, the Windows API call GetEnvironmentVariable is used to retrieve the value of DEMOSDIR. If this environment variable is not found, an error message describes what steps you can take.

If you want to connect to the dbdemos.gdb database at design time, and you are not using Delphi XE7, you will have to point the FDConnection Database parameter to the correct directory manually. To do this, right-click the FDConnection, select Connection Editor, and then click the Database parameter to see an ellipsis that you can click to navigate to the correct directory. In Delphi XE7, this file is located in the following directory:

```
C:\Users\Public\Documents\Embarcadero\Studio\15.0\Samples\Data
```

In Delphi XE5, the directory path looks something like this:

```
C:\Users\Public\Documents\RAD Studio\12.0\Samples\Data
```

In Closing

If you haven't worked with ClientDataSets before, get ready. I expect that you'll begin to look at data differently once you learn about the various advantages of in-memory datasets. If you have, I truly hope that you will gain new insight into ClientDataSets, and will be eager to add new features to your applications by leveraging the unique capabilities of these classes.

Note: You can download the code samples from this book's web site.
http://www.jensendatasystems.com/cdsbook2/

See Appendix A for details.