MIGRATING TO

Revision 9

Technical Support:
Internet Address (URL): http://www.dataaccess.com/support
Newsgroup: news//news.dataaccess.com/dac-public-newsgroups.visual-dataflex-support
FTP Site: ftp.dataaccess.com
e-mail Address: support@dataaccess.com
Revision Date: February 27, 2003

Part Number: 006990.MGR
COPYRIGHT NOTICE

Copyright 2003 DATA ACCESS CORPORATION. All rights reserved.

No part of this publication may be copied or distributed, transmitted, transcribed, stored in a retrieval system, or translated into any human or computer language, in any form or by any means, electronic, mechanical, magnetic, manual, or otherwise, or disclosed to third parties without the express written permission of Data Access Corporation, Miami, Florida, USA.

DISCLAIMER

Data Access Corporation makes no representation or warranties, express or implied, with respect to this publication, or any Data Access Corporation product, including but not limited to warranties of merchantability or fitness for any particular purpose.

Data Access Corporation reserves to itself the right to make changes, enhancements, revisions and alterations of any kind to this publication or the product(s) it covers without obligation to notify any person, institution or organization of such changes, enhancements, revisions and alterations.

TRADEMARKS

DataFlex is a registered trademark of Data Access Corp.

Crystal Reports is a trademark of Crystal Decisions / Seagate Software, Inc.

All other company, brand and product names are registered trademarks or trademarks of their respective holders.
# TABLE OF CONTENTS

Introduction .................................................................................. 5

Migrating from Procedural DataFlex ........................................... 6

Migrating from Character-Mode OOP ........................................ 7

What it takes to be a happy VDF Developer ............................ 8
  Relational, Transactional, Multi-user, Data-Design Knowledge .... 9
  Understanding the Mechanics of the IDE and Database Builder .... 9
  Understanding of VDF Language ................................................. 10
  Understanding the Classes ......................................................... 10
  Understanding of OOP ............................................................... 11

Data Dictionaries—a Business-Knowledge Repository .......... 11
  Defining Data Dictionaries ........................................................ 12
  Implementing Data Dictionaries ............................................. 12

Getting Started ........................................................................ 15

The Visual DataFlex Environment .......................................... 15

Get to Know 95/NT ................................................................. 15
  Study the Windows Interface ............................................... 15

Get to Know a VDF Application ............................................. 16
  Pay Attention to the “Standard” VDF Features .................... 16
  Recognize What Is Different ................................................. 19

Run through the IDE and Database Builder .......................... 20

Quick Start & Developing Applications in DataFlex ............. 20

Migrating ApplicationS ....................................................... 23
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database and DDs</td>
<td>23</td>
</tr>
<tr>
<td>Data-File Compatibility Issues</td>
<td>23</td>
</tr>
<tr>
<td>Creating DDs</td>
<td>28</td>
</tr>
<tr>
<td>What the DDs Do for You</td>
<td>29</td>
</tr>
<tr>
<td>If you are moving from 3.x Object-Oriented</td>
<td>35</td>
</tr>
<tr>
<td>Creating Views—the Visual Part</td>
<td>35</td>
</tr>
<tr>
<td>Views—Simple and Complex</td>
<td>36</td>
</tr>
<tr>
<td>View-Construction Tips</td>
<td>37</td>
</tr>
<tr>
<td>Reports and Batches</td>
<td>39</td>
</tr>
<tr>
<td>A Typical VDF Report/Batch: UI + Process</td>
<td>39</td>
</tr>
<tr>
<td>Business Processes</td>
<td>40</td>
</tr>
<tr>
<td>Migrating Existing Report/Batch Processes to VDF</td>
<td>41</td>
</tr>
<tr>
<td>Crystal Reports for DataFlex as an End-User Tool</td>
<td>43</td>
</tr>
<tr>
<td>Deployment</td>
<td>43</td>
</tr>
<tr>
<td>Programming in VDF</td>
<td>44</td>
</tr>
<tr>
<td>Custom Coding</td>
<td>45</td>
</tr>
<tr>
<td>New Class Names in VDF</td>
<td>46</td>
</tr>
<tr>
<td>“Specialization of Labor” with VDF</td>
<td>49</td>
</tr>
<tr>
<td>Frequently Asked Questions (FAQ)</td>
<td>51</td>
</tr>
</tbody>
</table>
Introduction

With the release of Visual DataFlex, increasing numbers of DataFlex developers, facing the growing popularity of Windows, are asking questions like this:

We’ve been using DataFlex for years. Over those years, we have created some rather-sophisticated programs. Our current application is pretty big and complicated. It consists of hundreds of small program files that are all connected together with chain and chain wait statements. The application has a lot of files and some complicated relationships. Over half of the application consists of reports and batch processes. The rest consists of data-entry screens. Most of those screens are maintenance-related. These are simple and are not heavily used. However, we also have a couple of entry screens/programs that are complicated. These are used all the time and we’ve spent a long time getting them right. We have spent a lot of time putting this system together. We’ve developed our own set of standards, tools, and tricks. The application runs, and it runs well.”

We now have to move this application to Windows and we have to do this soon. Our current customers have been asking us about Windows for several years. Potential new customers will not even talk to us unless we have a Windows solution. It does not matter how good our product is. If it is not in Windows, we cannot compete.”

Our questions are:

1. If we move to Visual DataFlex, how much of our existing application will we be able to take with us?
2. How much of our existing knowledge can we take over with us? Are we starting all over again?
3. How do we do it? Can you give us any advice for making this migration?”

Does this sound like anyone you know? The purpose of this paper is to answer these questions in detail.

There are different types of DataFlex developers. The in-house (or corporate) developer usually maintains a single, often very large, application for a single company. The independent developer/consultant provides custom application solutions to various clients on an as-needed basis. The vertical developer may create a single application or a suite of applications, sometimes shrink-
Chapter 1

wrapped, for a particular market. This paper is meant to address all of these developer types. For the most part, your needs are similar. You still have “customers/bosses” that you must sell this solution to (even if the boss is yourself), you still have a development environment and an end-user environment, and you still have end users. There will be some differences.

For example, an in-house developer may have more flexibility in providing transition solutions where Visual DataFlex and character-mode DataFlex programs can work side by side. A producer of a shrink-wrapped product may not have this flexibility. On the other hand, a shrink-wrapped product may not have to deal with legacy database design, while an in-house developer may be forced to retain older, less-than-optimum file structures. As you study this material, we understand that we cannot provide a “one size fits all” migration strategy. We hope that you will be able to apply your own custom “filter” to the material presented.

Migrating from Procedural DataFlex

If you have an existing procedural DataFlex application, we expect that you are faced with two choices. Either you migrate your existing application to Visual DataFlex or you re-write it under a completely different system. A few years ago, you might have had the additional option of continuing to maintain and improve your existing procedural character-mode application. In today’s market, this option is seldom viable.

When you first look at Visual DataFlex (VDF), you may think that you have been abandoned. As you will learn, this is not the case at all. A number of design decisions were specifically made to make it as easy as possible to migrate your application over to a Windows-based, event-driven system. In fact, many of VDF’s features such as the incorporation of a data dictionary, a visual data-dictionary builder (Database Builder) and a visual application designer (the IDE) actually make it easier to move a procedural program to VDF. It would be more involving to move the same application to character-mode object-oriented DataFlex.

In addition to making product decisions based on the need to migrate applications, we also developed a suggested strategy for making this move. The strategy will be described in this paper.

The term “procedural program” is applied to any application that is primarily based around DataFlex 2.3 style code. The actual program may be running under Revision 2.3 or any of the 3.x platforms (3.0, 3.05, 3.1). These programs may make limited use of objects.
Migrating from DataFlex to Visual DataFlex

If you are moving from procedural DataFlex to VDF, your migration steps will consist of the following steps.

1. Study your data-file structures to determine if any file changes will be required. If required, make these changes.

2. Move all of your database rules into centralized data dictionaries. This is a “gathering and organizing” process. While your current application certainly has database rules, these are most-likely spread all over your application. This step provides you with the opportunity to consolidate and formalize these rules.

3. Create your data-entry views and lookup lists. You are going to want to completely redesign your current data-entry screens. If your data dictionaries have been properly designed, you will find that this can be a surprisingly fast process. At first, many developers think that the redesign involved here is mainly visual—simply changing the appearance of your screens (character-mode to GUI). In fact, the real change is operational. Windows is a mouse-driven, user-driven, non-modal environment. This is a big change. Fortunately, the high-level classes and the Application Framework method will make it easy to make this change.

4. Convert your reporting and batch-processing programs. These types of programs often consist of quite complex processing rules coupled with a simple visual interface. Often you can keep the processing code and simply dress up the user interface.

5. Create new, better-looking reports using the GUI report-building tools. Some of these tools are provided with VDF; others may be purchased separately.

Migrating from Character-Mode OOP

If your character-mode application is already written in object-oriented DataFlex using the Application Framework, you have already done most of the migration work. Your data files are already proper, your data sets contain most of your database rules, and your entry screens have already been converted to friendly, non-modal views of Windows. Your conversion will involve the following steps.

1. Convert all data-set classes to data-dictionary classes. Move any rules that remain in your entry objects into these data-dictionary classes.

2. Convert your views from character mode to GUI. In this case, you really are just changing appearances. The underlying view-based behaviors
will remain unchanged.

3. Convert your reports.

**What it takes to be a happy VDF Developer**

You have probably heard the stories about the object-oriented learning curve. Just how long is it going to take you to become productive? How much do you really need to know? Those of you who have followed the progress of object-oriented DataFlex will recognize that with each new revision of DataFlex, we have significantly flattened the learning curve. In Visual DataFlex, we have attempted to take this one step further by removing the learning curve completely. It is our goal that a developer is able to write basic data-entry applications within days. In addition, we have set a high standard for the term “basic.” We are referring to an application that:

- is multi-file
- is multi-user
- is fast
- supports full data-dictionary-based rule validations
- consists of user-directed independent entry views
- supports automatic user verification for all-important events such as saves, deletes, data loss resulting from clearing data, closing views, or exiting applications
- supports standard Windows 95/NT features such as menu bars, tool bars, status bars (with automatic display of status help), floating (right-mouse) context menus, forms, combo forms, spin forms, multi-column/row grids, buttons, checkboxes, radios, and tab dialogs
- supports many sophisticated ways to find records including finding keys, find buttons, popup lookup lists (very powerful), and drop-down combo lists
- is fully object-oriented (making it much easier to maintain)

In addition, this application can be completely maintained using visual design tools. Your programs are still code-based (and you can edit the code), but the code is created for you by the visual design tools.

Below is a list of skills you will need to work in Visual DataFlex. These skills are ranked in the order of their importance.
Relational, Transactional, Multi-user, Data-Design Knowledge

This is, by far, the most-important skill required to successfully use VDF or, for that matter, to create database applications. Another way of stating this is that you need to understand what database programming is all about. An application is only as good as its database and its underlying database rules. We assume that you already have this knowledge.

There has always been a DataFlex “style” of creating programs. The good DataFlex developer knows how to properly normalize files, create file relationships, and create the right indexes for rapid record retrieval. This, used in conjunction with the multi-user paradigm of “lock/reread/update/save/unlock,” allows one to create high-volume, very fast, very large database applications.

All of these skills will be applied in VDF. There will be two main differences in VDF. First, many of these techniques are now automatic. For example, all multi-user processing is handled automatically by the entry objects. Secondly, the techniques that are not fully automatic are centralized in one location—the data-dictionary classes. For example, a field’s capslock setting, which would be distributed over many entry programs in previous revisions of DataFlex, is now located in one single place.

Understanding the Mechanics of the IDE and Database Builder

Most of your development work will occur within two design tools. The Database Builder is used to maintain your data files and fields (just as DFile did) and is used to maintain your data-dictionary classes. The IDE is used to design your visual components (views, lookup lists, pop-up dialogs, and programs). The IDE, in particular, requires a little bit of practice to master. It is a little bit like learning to use a new word processor or spreadsheet. When you are starting out, all of the new features may make you feel a little bit overwhelmed and unsure of where to start. The payoff is in the results! After you have created a data dictionary with Database Builder and after you have created a few views with the IDE, stop and look at the source code that you generated. This is elegant object-oriented code. You did not have to write this! Keep telling yourself this!

Learning to use these tools will probably require hours, but certainly not days, of practice.

Once you have acquired these first two skills, you have everything you need to start writing applications. At this point, you are a productive VDF programmer.
Understanding of VDF Language

We expect that you are suspicious of our claims. Perhaps you do not really believe that you can write a real application without needing to resort to some kind of customization. After all, this has always been the failing of RAD (rapid-application-development) tools. You are correct. At some point, you are going to need to start writing code. In fact, both the IDE and Database Builder provide methods for adding and editing source code from within the tool.

In object-oriented DataFlex, a program consists of objects. One way objects are customized is by creating procedures and functions that are called during specific operations. The object can predefine these operations or the developer can define them. An example of a predefined operation is the data-dictionary procedure update and backout. These are called during a save. It is up to you to provide behaviors for these “hook” procedures. An example of a developer-defined operation would be a procedure that you would create and then assign to a key. When the key is pressed, the procedure is executed.

In both cases, you need to provide the code inside these procedures. The code that is needed is the same DataFlex code you have been writing for years. Procedural DataFlex still lives. It is now just packaged inside the objects’ procedures and functions.

Understanding the Classes

The IDE generates source code that defines objects. Database Builder generates source code that defines data-dictionary subclasses. These objects and subclasses are customized by setting properties (think of this as setting switches) and by creating/modifying procedures and functions. The most-common customizations are handled within the IDE and Database Builder directly—you select options from menus, set checkboxes, etc. These objects and data-dictionary subclasses are based on powerful classes provided with VDF. This library of classes is your programming tool kit. These classes are capable of a great deal of customization. You can learn about these classes and their capabilities by using their on-line documentation. It is not expected that you will sit down and attempt to memorize all of the properties, procedures and functions in a class. The most-common behaviors are set within the visual builder. The less-common behaviors may be researched, as they are needed. The on-line help reference is designed to be easily accessed (within the IDE, context-sensitive class help can be invoked with two mouse clicks), and this information is provided so that you can easily find the information you are looking for, as often as you need it. Windows’ copy and paste capabilities make it easy to copy code samples from the help
system and paste them into the IDE or Database Builder.

**Understanding of OOP**

At some point, you may want to start understanding why this all works. You will want to start acquiring more knowledge about object-oriented programming and the DataFlex implementation of OOP. This involves such topics as subclassing, inheritance, delegation, augmentation, and encapsulation. The more you know about OOP, the more powerful VDF becomes in your hands.

We will end this section by restating that you do not need to be an object-oriented guru to use Visual DataFlex. You do not need to stop everything you are doing and spend six months learning OOP. You already have the knowledge you really need (understanding how to build a database). You need to learn how to use the development utilities. Other knowledge can and will be acquired as needed.

**Data Dictionaries—a Business-Knowledge Repository**

The primary purpose of this paper is to discuss the procedural-to-VDF migration issues in practical terms. This section will bend the purpose just a little bit. Before we discuss some of the specifics of how data dictionaries will make it easier to rapidly develop applications, you need to understand the importance of data dictionaries at a more-abstract level. The employment of data-dictionary technology affects far more than your VDF application. It will change the way you manage and control business data.

One of the advantages of a data dictionary (DD) is that it provides a single place to store all of your database rules. These database rules are your business rules. This could be abstracted one more level to define data dictionaries as being a *business-rule repository*. You are storing the knowledge of your business inside your DDs.

It is generally understood that data dictionaries are separate from your user interface. One of the clear advantages of a DD is that it finally allows you to remove your database rules from your user interface. It should also be recognized that DDs are also separate from your data files. Data dictionaries do not belong in the database. Data dictionaries exist as a layer between your user interface and your physical database. All communication between your database and your user interface should brokered by data dictionaries.

We must concern ourselves with two levels of data-dictionary usage:
Defining Data Dictionaries

In theory, one of the tasks of database-application development is to define your target user’s business rules. While this is true, it overstates the scope of current database-application technology. In realistic terms, the task of database-application development is limited to defining business rules as they apply to a company’s database. An efficient way to do this is to define business rules for each data file. Rules for a data file are defined at three levels: field, file, and database. Data dictionaries provide a convenient method of grouping your business rules. In the rules for all of your data dictionaries combined, you have created a business-rule repository.

The DataFlex data-dictionary class and the DataFlex language are ideally suited for describing database business rules. The data-dictionary-class interface allows you to define rules using a syntax that is descriptive, concise, and unambiguous. These rules are visually defined and viewed using the Data-Dictionary Builder within Database Builder. Most of these rules can be defined and understood using a simple (no-programming-required) interface. Other, usually more-complicated, rules are defined through custom-source-code areas within the builder. The ability to “drop down” to source code at any point is an extremely powerful feature of the DataFlex Data-Dictionary Builder. This means that all rules, simple and complex can be expressed inside a data dictionary.

Even if you did not use data dictionaries within a program, their very existence as a repository of knowledge would be of great value. Your business rules are now defined, organized, and cataloged.

This makes these rules usable, maintainable, and extendible. Even if VDF was not going to use the data dictionaries within applications, the DataFlex language is a good choice as a “people” language for describing your rules. A language like English, while descriptive and easy to understand, is too verbose and tends to be ambiguous. A language like C++, while concise and unambiguous, is neither descriptive nor easy to understand. The DataFlex language has the advantage of being concise (but not too concise), descriptive (but not too verbose), and unambiguous.

Implementing Data Dictionaries

As the previous section implies, simply having your rules stored within data
dictionaries is of great value. These DDs are of even more value because VDF knows how to take full advantage of this information. VDF is referred to as being highly “data-aware.” More accurately, it should be described as being “highly DD-aware.” Both the IDE and your deployed application will use your business rules, expressed inside classes. The DDs help you write your application and they help you run your application.

The IDE uses data-dictionary knowledge to help build your entry views. In particular, The IDE automatically handles all of the internal connections required to make a program function properly. It does this by creating and connecting data-dictionary objects to create a structure (DDO structure). When visual entry objects are then added to a view, they are properly connected to the DDO structure. All of this non-visual structuring is handled automatically within the IDE, allowing you to concentrate on the visual aspects of your views.

When a program is run, each view will contain a DDO structure. This structure was created by the IDE. When a view is used, the DDO structure will handle all communications between your database and your visual objects.

Within a view, the DDOs will control how the visual objects appear. Your data-entry objects (DEOs) will request information from a DDO to determine how they should look and behave. If a view’s form uses a field that has an DD-assigned lookup list, a prompt button will be displayed with the form. Combo forms within a view will get their list values by asking the DDO for them. A form’s display format (capslock, currency formatting, etc.) is determined by asking the DDO for these formats. All of these requests occur automatically and require no programming on your part. The DEOs knows how to acquire this information from the data dictionaries. This is why we refer to the DEOs as being “data-aware.”

All saves and deletes are controlled by the DDO structure. The DDOs determine if a save or delete can take place (validation rules) and how a save or delete will take place (update rules). A rule is defined once within the DD class. Every view that contains a DD object based on that class would implement that rule unless an exception to it is explicitly coded in the view.
Getting Started

The Visual DataFlex Environment

First make sure that you have an appropriate environment for running Visual DataFlex and that you’ve got the right VDF components loaded on your machine.

- Visual DataFlex should be loaded on a computer running either Windows 95 or Windows NT. Most users have reported quite satisfactory performance on a Pentium processor (90 MHz or better). Any computer that is suitable for Windows 95 or NT will be suitable for VDF. As with any Windows application, the faster the processor, and the more memory, the better VDF performs.

- Make sure you are running the latest version of Visual DataFlex. Each newer revision contains minor improvements, bug fixes, and documentation enhancements. From time to time, you may wish to check the Data Access Web and FTP site (www.dataaccess.com) to make sure that you’ve got the “latest and greatest.”

Get to Know 95/NT

Before you worry about migrating your application to the Windows environment, you want to make sure that you understand what the environment actually is. Probably the most-important requirement for your application is that it “feels” like a Windows application. It does not matter how good your application is; if it does not “feel right,” it will not be a success.

This has been a hard lesson for many developers to learn. There are virtually no standards in character-mode applications. Many procedural DataFlex applications based their success on the fact that their application had a unique look and feel. In Windows, this is not an advantage! You do not want to be different. You do not even want to be better. You want your application to act like Windows.

Study the Windows Interface

Even if you have been running Windows on your computer for a while, you may wish to review the interface. As you study this, keep in mind that your application is going to have to fit the Windows mold. Start by looking at
Chapter 2

Windows itself. Load some property panel sheets. Look at the different types of controls and see how and when they are used. Load some other Windows applications. Some of the applications in MS Office are a good choice. Load Word or Excel and see how a Multiple-Document-Interface (MDI) application should behave. Take note of the size, location, and appearance of the various controls.

There are documents available that define standards for all of the above features. Microsoft ignores many of their own standards. For example, Microsoft publishes a suggested size for a button and uses a different (yet consistent) size in all of their applications. For this reason, it is very important that you study actual applications and infer your standards from them.

Get to Know a VDF Application

After you have spent some time looking at Windows, you should load one of our sample applications and study it quite carefully. Your program will look and operate a lot like these samples. Start with “Big.” This represents a typical multi-view, multi-file application. As you review this application, keep in mind that we consider this the type of application that any VDF developer could create.

Pay Attention to the “Standard” VDF Features

One of the benefits of a VDF application is that a number of very powerful features are a standard part of any application. The implementation of these features requires little or no work on the part of the developer. Below is a list of some of the most-important features.

MDI-Style Application

A VDF application is an MDI-style (multiple-document-interface) application. This interface consists of a main panel that contains a menu bar, tool bar, a status bar, and a work area. The work area is used to place your “documents.” In VDF, those documents are your views. Each view can be opened (displayed), closed, minimized, and arranged in various ways. If you attempt to move a view off of the VDF main panel, it will be clipped. This is a very standard Windows interface model and should be quite familiar to any Windows user. We expect your applications to be built using this model.
View-Based Independence
Each view is meant to operate independently from all other views. Views may be loaded and placed anywhere within your workspace area. When a user closes a program, the location of each view will be remembered. Once a view is displayed, users may switch between it and other displayed views at any time. The contents of one view do not necessarily determine the contents of another view. For example, a customer view may display information for Customer A while an order-entry view will display information for Customer B.

If you are accustomed to writing more developer-driven systems, this style may seem alien at first. Do not fight this model. Users really do like it. It is the basis of the framework method and has been proven in both the character-mode and Windows environments.

You will find the view-based development model significantly simplifies development. The independence of views makes it easy to create and debug each view as a standalone component. When all of these components are combined into a single large program, they just work.

Tool Bars
VDF applications use tool bars, which give users access to buttons for the most-commonly used functions (save, clear, find, etc.). Notice the popup “tool tip” help that is presented when you move the mouse cursor onto a tool button. This support is automatic. The tool bar you see in the sample is the standard VDF tool bar. This is available to any program with no additional programming. The tool bar can be customized.

Status Bars with Status Help
The status bar at the bottom of the application displays help text for whatever object (form, menu, tool button, etc.) happens to have the focus. Status help text is set with a property. Most entry controls within a view are associated with a data file and field. The status help for these fields are acquired from the data dictionary of the file. Status help for fields is defined within Database Builder. You create this status help in one centralized location and all views that use that field will properly display the help text.

Notice that the style of the status bar changes when you use it for view status help (it appears as multi-pane status bar) and when you use it for menu status help, (it appears as a single, simple bar). This support is a Windows standard and is automatic in VDF.
Prompt Lists for Files

Note that several entry forms’ fields contain prompt buttons located to the right of the form. These are used to pop up lookup lists. There are two things to note about these lookup lists.

1. The data dictionary determines which fields have prompt lists. Within the DD, each field may be assigned to a lookup list. The lookup lists are created using the IDE. Once the list is created and the link in the DD is assigned, prompt lists are automatically made available to all forms within all views that use the field.

2. Prompt lists are very powerful search tools. When invoked, the list starts in a position determined by the underlying view’s contents (the list is seeded). You may jump to any position in the list by typing in a search value. When you switch columns, the list can be redisplayed and is ordered by an appropriate index. These behaviors are all automatic.

Lookup lists are ideal for searching for a record when the data file is large and dynamic (multi-user).

Combo Forms and Lists for Static Choices

Some of the views have combo-forms. These are used when you are choosing from a smaller, static list of choices. In Data Dictionaries, you may assign a field to a validation table. Each validation table may consist of a list of database values and, optionally, descriptions for each value. Normally the combo forms are used to display the description value. This is a powerful new extension of DataFlex. A field’s data value is read and written to the data-file while the field’s description is displayed and selected by the user.

The connection between combo lists and a field’s validation table are automatic - the DD will fill the combo list for you.

Floating Context Menus

If you press the right mouse button on any entry form, a popup menu will appear at the cursor position. These menu choices (cut, copy, paste, prompt, save, etc.) are shadowed and un-shadowed dynamically. This menu is provided automatically.
Built-in Confirmations
Confirmation dialogs are presented before proceeding with any important database operation. These verifications occur before a save, a delete, and a clear or a close that would result in data loss. The verifications you see in the samples are a standard feature of VDF. You may, of course, change or suppress any of these verifications.

Rule-Based, Validation-Enabled Views
One of the most-important parts of an application is the part that you do not really see in the samples. All database activity is handled through the data dictionaries. These DDs make sure that all database rules are enforced. This includes update rules such as the updating of relational balances or the auto assignment of key IDs. It also includes validation checking, such as making sure that all fields are valid before a save or making sure a record deletion will not result in “orphan” records in a child file. For example, load the customer view, find Customer Number 0001 and attempt to delete this record. An error message will be generated telling you that you cannot delete a customer that has child records. This protection is by the data dictionary and is a good example of what DDs do for you.

Recognize What Is Different
As you study these applications, it is very important that you recognize some of the differences between a Windows application and your current procedural application. Understanding and accepting these differences will make your conversion much easier. Note the emphasis on the word “accepting.” Changes in the Windows interface may require that you throw away standards and techniques that you spent months, if not years, perfecting. Accept it!

In particular, notice that Windows applications are non-modal. You can usually switch from one part of an application to another at any time. On occasion, a modal pop-up dialog will appear that must be closed before moving to another part of your application. These occasions are rare and rarely occur more than one level deep. For example, you do not often see a modal dialog, invoke another modal dialog, which invokes yet another modal dialog. On the other hand, procedural programs used this type of “drill-down interface” all the time. The chain wait command made this easy to do. A modal drill-down interface is not appropriate in the Windows environment (which is one reason why the chain wait command is not used nearly as much in VDF).
Because it must be easy to switch from one part of an application to another, navigation must be user-directed and not program- (or programmer-) directed. The mouse makes this requirement obvious. This forces a major change in the way program navigation is approached. In procedural programs, the program controls navigation.

Users can usually go forward or back unless the program has special functionality to allow them to navigate to some other location. In Windows, users should be able to navigate anywhere they want unless the programmer adds special functionality to disallow them to navigate to some other location.

A typical logic mistake procedural programmers make when moving to a user-driven system, is that they assume that a user traversing from Point A to Point C, will have to travel through Point B. This assumption can no longer be made. These are the types of differences that require changes in the way an existing application operates.

For an application to be truly non-modal, most of your application needs to be running at one time. This is one reason why VDF programs will get quite large. The more you can place in a single application, the more flexible it becomes. Moreover, yes, VDF programs can be very big; and, yes, this works just fine.

**Run through the IDE and Database Builder**

After you have studied the sample applications, you may wish to run both the IDE and Database Builder. The IDE and Database Builder are complicated programs. They are not typical of the types of applications a database developer needs to create, they were written by very experienced VDF developers, and they did take a long time to write. They are, however, written entirely in Visual DataFlex. While you will probably never need to write applications like this yourself, they should give you some idea of the power of the product. There is plenty of room to grow here.

**Quick Start & Developing Applications in DataFlex**

VDF is shipped with two printed tutorial manuals: *Quick Start* and *Visual DataFlex Developing Applications*. You should read both of these manuals cover to cover (they are short). *Quick Start* contains a simple tutorial and focuses on the IDE. While we tell you that you should start a development project with the data dictionary, we know that the first thing you really want to do is to “test drive” the visual design tool. *Quick Start* will let you do that. We provide the data dictionary for you and provide systematic instructions that
show you how to create a view, a lookup list, and the application to house these components.

*Developing Applications* provides a more in-depth review of VDF and its development tools. It introduces the language, OOP, data dictionaries, the classes, Database Builder, and the IDE. It also includes a more-advanced tutorial sample. The sample, an order entry system, focuses on creating data dictionaries. It concentrates more on “why” and less on “how.” *Developing Applications* also will show you how to create reports, show you how to use the automatic features of the IDE and Database Builder, and show you when and how to add custom code. If you read the entire manual and follow the tutorial sample, you will know how to develop a VDF application.

These tutorials are not just beginning primers. They will show you how you really develop *real* applications within VDF. They are a tremendous resource—use them!
Migrating Applications

You are now ready to start working on migrating your application. The three steps of migration will be:

1. Set up your database file and your data dictionaries.
2. Set up your visual entry views.
3. Set up your reports and batch processes.

Each of these steps will be discussed in their own section.

Database and DDs

Before you design any visual screens, you need to get your database in order. This consists of two tasks:

1. **Make any required data-file changes**: First, you must make sure that your data files are properly designed and normalized. Most data files will require little or no change to be VDF-ready. If your existing data files do not use standard DataFlex file relationships, you are going to have to make some changes. Depending on your need for backward compatibility, these changes may be made in the data files or within the data dictionaries. You will also want to make sure that all of the dates in the files are in four-digit-year format to properly handle the Year 2000 and beyond.

2. **Create your data dictionaries**: Right now, your database rules are spread all over your application. Your job is to find those rules and place them all in one centralized location—your data dictionary. This is probably the single most-important step in your conversion process.

Data-File Compatibility Issues

The data files used in VDF are completely compatible with your current DataFlex data files. This compatibility is very important to you. A VDF program may read and write to the same files that your existing procedural program is using. This will allow you to run your new and old applications in parallel. This will also allow you to phase in your new VDF programs gradually. Keep in mind that under Windows you may even run multiple versions of any DataFlex program (VDF or character-mode) at the same time. It should also be possible to launch a character-mode application (using
runprogram wait or runprogram background) from within a VDF application, or vice-versa.

If your existing program is running under DataFlex 2.3, there is one compatibility consideration. When DataFlex 3.0 was released, the internal format of data files was changed. All revisions of DataFlex 3.0 and up (including VDF) are capable of reading and writing to data files in 2.3 and 3.0 format. Therefore, as long as a file remains in 2.3 format, VDF and your existing application may use it. A data file is converted to 3.0 format when it is changed with a 3.x version of DfFile (for character-mode) or it is changed by VDF’s Database Builder. If you need to maintain file compatibility, you will have to be careful.

You can still use Database Builder to maintain your data dictionaries, but you must make sure that you do not change the file itself with Database Builder. It is easy to do this accidentally, so you will probably want to build your DDs using test data files.

Obviously, it will be easier if your character-mode program is running under a 3.x runtime. If you have not made this runtime upgrade, you may wish to consider doing so. This would allow you to use the Console-Mode runtime, a special character-mode runtime that runs under Windows as a native Win32app. It is provided as part of the character-mode (Rev. 3.1c) product. It is not part of VDF.

Also in Rev. 3.1c are cross-century date capabilities similar to those in VDF. If you run VDF with cross-century enabled (it is disabled by default) on data shared with a character-mode runtime, that character-mode runtime must also have its cross-century enabled. Moreover, cross-century capability was not present in the original releases of character-mode runtimes. Only Rev. 3.1c (and Rev. 2.3 Enhanced) have it.

**Relationships**

Data dictionaries require that your relationships between files be defined in the database. Normally such relationships are established within DfFile (now Database Builder). This has been the recommended method for using DataFlex since its inception. While it is possible to use data dictionaries without these built-in relations, you will have to work a lot harder to do so and the result will not be as flexible.

Some older DataFlex programs may not use built-in file relationships. In these applications, all finding of parent records (relate) and all movement of parent-field data into child-file fields (attach) is handled manually. If your existing application contains any such manual relationships, you will need to employ
one or more of the strategies listed below.

**Add the Missing Relationships to the Data File**

Manual file relationships can be properly represented in the data file. Often the developer simply chose not to do so. (The normal reason we hear is, “I was getting confused with attach, relate, save and save record. It was easier to just do it manually”.) Standard DataFlex relationships can support your data files wherever a single child field (usually non-uniquely indexed) points to a single parent field (with a unique index). Since this is the essence of a good database design, most file structures support this. In such a case, the simplest approach is to add the relationships to your data files using Database Builder.

In some cases, you may need to add a field, create an overlap, or change an index to make this work. The only disadvantage to this approach is that this change may affect your existing procedural application. If you are going to need to run your old application in parallel with your new application, you may wish to consider the next option.

**Add the Missing Relationship using Set_Relate**

If you do not wish to establish “hard” relationships in your data file, you may create run-time-only “soft” relationships using the set_relate command. Once executed, this command establishes a relationship between two files. This relationship is identical to a hard relationship in every way except that the relationship only exists within the program while the program is running.

If another program (like your old procedural application) opens the same file, it will not see the relationship. A set_relate relationship can be defined in your data-dictionary subclass. Once defined, all programs that use that subclass will use this relationship.

The set_relate command was created specifically to make it easy to establish proper relationships within your new program without upsetting the operation of existing programs.

**Create Custom Relationships**

Data dictionaries allow you to define custom relationships. Special hook procedures (relate_main_file and attach_main_file) are provided with which you can relate and attach records any way you wish. This can be used to handle conditional relationships as well as relationships that are not based on a single field.
There are two main disadvantages with this method: it is complicated, and it is limited. Creating custom relationships requires custom coding within your data dictionary. It will require extra effort to get this working properly. Custom-coded relationships do not have all of the flexibility of built-in or set_relate relationships. Built-in relationships can perform optimized top-down constrained finds. In such a find, the child records are limited to records that actually relate to a selected parent record. This called a “relates-to” constraint. This type of relate is often used to display all related child records in a table or grid. The VDF classes are designed to automatically handle this type of display. Custom relationships do not have this flexibility.

**Code Lists**

Very often, a field’s value needs to be limited to one of several values. For example, an order may have several possible status values (open, closed, new) or an order may have several possible terms’ values (COD, Pre-pay, 30-day, 60-day). Good DataFlex programmers would place each of these values in separate related data files, making it easy to add or change a value. This had the advantage of flexibility and the disadvantage of creating and opening many additional data files. In VDF (and 3.1), you add these files to a single file called a code list. A set of values for a given field is defined as a type. Each type will have a list of valid codes and descriptions. All codes, for all types, for all fields, for all files may be contained within a single data file.

A single code-maintenance utility program is provided to maintain these values. The field-to-type connection is defined within your data-dictionary class. Within data dictionaries, these are referred to as “validation tables.” Values within a validation table can be defined within the code maintenance file or may be directly defined within the data-dictionary subclass (i.e., hard-coded).

Validation tables are a new type of relationship/validation technique not available in your existing application. Its existence may solve some otherwise-complicated transition problems for you.

**Alias Files**

Sometimes an application needs to use the same physical data file more than once within a file structure. For example, an order may need to relate to both a shipping address and a billing address, both of which reside in the same physical file. One solution to this has been the use of the alias file. This was an unofficial technique where several different entries in your file list opened the same physical file.
In other words, several logical files use the same physical file. This technique is now officially supported within data dictionaries.

**Dates and the Year 2000**

If you are using two-digit years for your dates in your current application, your application will probably not be ready for the Year 2000. You need to address this in your current application and in your new VDF application. We now recommend the following strategy for all versions of DataFlex:

- All of your dates should be stored in full four-digit-year format (YYYY).
- Dates may be entered using either two or four digits.
- When a two-digit-year date is entered, it will be internally converted to a four-digit-year. An epoch setting, which is programmer-definable, will determine if the entered date should be preceded with a 19 or 20. For example, with an epoch setting of 30, the date 5/6/52 will be converted to 5/6/1952 and the date 5/6/25 will be converted to 5/6/2025.
- All existing data files that contain two-digit-year dates must first be converted to four-digit years.

Updated releases of DataFlex 2.3 and 3.1 also support this automatic conversion. This conversion is handled in the runtime and may be employed on existing programs without compiling. To do this, you must get the new runtime and run a date-conversion utility program on all of your data files. This converts all two-digit-year dates to four-digit-year dates.

Therefore, if you need to run your existing application in parallel with VDF, you will want to get the updated runtime and convert the dates in your data files. If you do not need to run in parallel, you will want to convert your dates before running it against VDF. Although VDF will still support two- and four-digit-year dates, it is strongly recommended that you only use four-digit-year dates.

If you have already planned and your dates are all stored in four-digit-year formats, you will not need to make any changes.

When the IDE creates a new program, several lines of code are generated to determine if two-digit year dates are supported. The default values for these commands are determined by a configuration setting in the IDE (a “Use Epoch” checkbox, which we recommend, be checked).
Chapter 3

Creating DDs

You should create a data-dictionary subclass for each file in your application. The data dictionary is a code-based file. Think of this .DD file as an extension of your data file much in the same way that .DEF and .FD files are.

Normally, you will use Database Builder to maintain your DD file. When you open a data file, the first three tab-dialog pages are used to define your traditional database settings (field names, types, indexes, etc.). The remaining tab pages are used to define your data-dictionary classes. When a data file is saved, the .DD file is saved along with other data-file information.

When you compile your VDF program, your DD files are compiled into your program. If you make a change within the .DD file, you will need to recompile your program for it to take effect. This is consistent with prior revisions of DataFlex—when a data file’s structure is changed, your programs must be recompiled. Because the .DD file is stored as standard source code, you may load this file with any program editor, review it (a great learning tool), and if needed, change it. When a .DD file is loaded from within Database Builder, the source is read back in. As long as your manual source changes are properly formatted, you will have no problems.

This is a powerful feature of VDF. While it is usually easier to maintain your .DD files from within Database Builder, there are times when it will be better to work with a source-code editor (e.g., making a quick one-line change during a debugging session). With VDF, you have a choice. The same functionality is provided within the IDE—your visual programs can be modified within the IDE or with a program editor.

How to Define your DD Rules

We assume that you already know how to define your database rules. You can use any method you wish to research, develop, and assemble these rules. Your local bookstore probably contains a shelf of books that deal with this topic. While we cannot tell you how to define your rules, we can give you a few hints that will make this process go a little bit faster.

- **Use Database Builder to guide you through the definition process:** Database Builder makes it very easy to build data dictionaries. Just load your data file. If a DD subclass is not found, you will be asked if one should be created. Answer yes, and you have your basic data-dictionary subclass. Now on a field-by-field basis, check and set the properties of your DD. You will see what setting you need to set. It is easy to assign a lookup list to a field, create and assign a validation table to a field, assign
field status help, declare key fields, declare required parent and child files, etc.

- **Search your existing programs for database rules**: Your existing application probably has your database rules spread throughout its source. You will need to search through your programs, find this code, and determine what rule it was meant to express, and determine how to represent this within the data dictionary. This may be a painful process.

No one likes to revisit old code. At this point, you may find that you have some very ill defined database rules in your existing application. Now is the time to correct this. When your DDs are complete, you will probably have a much-better-designed system.

- **Look for feasibility problems**: As you build your rules, you may find some areas that you are not sure how to represent within data dictionaries. It is best to identify these problem areas sooner rather than later. Often you will find that these problems arise from improper design or inadequate definition of the problem.

  The best solution here is to correct the problem. In other cases, you may have a requirement that does not seem to fit the DD model. Ask for help. Post a message on the DataFlex on-line forum. A number of experienced developers visit this forum that know how to make the VDF DDs do wondrous deeds.

- **Start at the top of your database hierarchy and work your way down**: If you strictly follow our advice, you will create all of your data dictionaries before you start creating any visual screens. If you are like most of us, you may have a hard time waiting to see the results of your labor. If you are impatient (i.e., human) you can often build a data dictionary and then use the IDE to create the DD’s lookup lists and entry view. You can do this if you start at the top of your database hierarchy and work your way down. Usually the DDO structure in a view requires, at a minimum, the view’s main DD and its entire parent DDs. Starting at the top ensures that the parent DDs are already created. In addition, the files at the top of the database hierarchy tend to be less complicated, requiring simpler DDs and simpler views. This allows you to work your way up to your more-complex tasks.

**What the DDs Do for You**

Below is a partial list of what can be done with data dictionaries. While the
functionality provided is quite advanced, none of these features requires any advanced implementation techniques and can all be implemented directly from within Database Builder. In most cases, these features are enabled by clicking on a checkbox, selecting an item from a list, or by entering text into a form. As long as you understand conceptually what you need to do, the implementation is straightforward.

As you review these features, consider your existing application. Do you currently support the same capabilities? How, and where, are these features implemented in your existing application? Most of these features are probably currently implemented within your visual entry programs. Imagine how much simpler your entry programs will be in VDF once all database-related material is centralized within the data dictionary.

**Multi-user is Automatic**

Data-dictionary saves and deletes are fully automatic. Files are locked, records are reread, records are saved and files are unlocked as required. This lock process is optimized (see smart file mode) and can be reversed (see rollbacks). No extra code or knowledge is required to make this work.

**Smart File Mode is Efficient**

In previous revisions of DataFlex, advanced developers created techniques that would optimize the record locking and re-read process. Before a save or a delete operation, they would carefully change the filemodes of files so that only the required files would participate in the lock and reread. When the operation was complete, the filemodes would be restored.

Data dictionaries now do this for you automatically. Before a save or a delete, the data dictionary will determine which files need to be locked.

All other files’ filemodes are set to read-only (so they are neither locked nor reread). This process is optimized; creating different locking schemes for saves, deletes with no cascade child-record deletion, and deletes with cascade child-record deletion. System files are added to this lock list. You can even specify when system files will be locked (any combination of new save, edit save and delete). Smart file mode will dramatically improve database-processing speed over networks.

**Cascade Delete Protection**

When a record to which records in child files relate is deleted, one of two things should happen. Either the delete should be disallowed or all of the child
records (and all of their descendant records) should also be deleted. Data
dictionaries support both modes. Your preference may be set within the data
dictionary (a single checkbox) and may even be changed at runtime by
changing a property. Either way, you are assured that the deletion of a record
will not result in orphan child records.

Key Fields Can Be Protected
In a data dictionary, you may specify a field (or set of fields) to be the primary
key. This key may then be protected against change. This ensures that an
inadvertent change of a key field will not result in child records pointing to a
non-existent record. Key fields are defined and protected by setting various
checkboxes.

Validation Exists at Many Levels
Data dictionaries provide multiple levels of validation.

Structure Validation
Each data dictionary contains lists of required support files. Each DD
maintains a required list of server (parent) files, client (child) files, and extra
(often system) files. These lists are used to control smart file mode locking,
cascade child-delete protection, and DDO-structure validation. Before a save
or a delete occurs, the current data dictionary object structure is checked to
make sure that the entire structure is complete.

Field Validation
Validation may be assigned at the field level. This ranges from simple pre-
defined validations (range values, check values, etc.) to complex user-defined
validations (where a user-defined function is called). Field validation is used
whenever a record is saved or deleted. All fields in all participating files are
validated. This ensures that all files are valid before any database change
takes place.

In previous revisions of DataFlex (both procedural and object oriented),
validation was applied to the visible input item and not the field. If a field was
not represented visually, it could not be validated. DD field validation
represents a major step forward here.

Field validation is also used during keyboard navigation, allowing users to see
their errors when they occur. This gives you the best of both worlds; errors
are reported as they are encountered, and all fields are validated before a
Chapter 3

Migrating Applications

Locked Validation
Field and structure validations occur before the data files are locked and reread. This way, these validations have minimum impact on other users. Some validations must be performed in a locked state. For example, you can only verify that an inventory quantity is sufficient if you know that no-one else can change it. The function validate_save is provided for this purpose. It is called when the data files are locked and all fields are updated. This function is sent to all participating data dictionaries. A similar function, validate_delete, is sent during a delete.

Rollbacks
If for any reason a save or a delete is canceled, the entire transaction is rolled back. All changes in all files are restored. The cause of the cancellation may be expected (a validate_save function fails) or unexpected (a program error is encountered). In either case, all files are restored to their prior condition.

Relational Balances (update / backout)
In procedural DataFlex, the ENTER and ENTERGROUP macros had subroutines for maintaining relational balances (entry.update, entry.backout, etc.). Data dictionaries provide the same capabilities with procedures (update, backout, deleting, creating). These are most-often used to maintain a parent file’s balance field. Unlike procedural DataFlex, these relational rules are coded in one single location.

In addition, unlike procedural DataFlex, these relational balances are properly maintained, even if the parent record is switched. A balance will be removed from the old parent and added to the new parent.

Auto Assign Key Values
Very often, a record’s primary key will be system-assigned. An ID counter from one file (often a system file) is incremented, saved, and assigned to the new record. In data dictionaries, this process may be automated by assigning an auto-increment file and field.

Prompt Lists
Prompt lists provide an extremely powerful way to search for records. They
allow you to easily search for records within large data files. Lists may be viewed in various index orders and it is possible to jump to any location with a list.

Each field in a data dictionary may be assigned a prompt (or lookup) list. Prompt lists are usually assigned only to fields that are indexed. Once assigned, all entry views that use the field will have access to the prompt list. A prompt button will be automatically added to the window to indicate that a prompt list is available and provide a means of getting it displayed.

Validation Tables
Each field in a data dictionary may be assigned to a validation-table object. This object consists of a list of valid field values and descriptions. This list can be used for display purposes and, optionally, for validation. Validation tables are most-often represented visually with Combo-form objects. If a combo form is assigned to a field that has a validation table, the combo form will be filled with the validation table’s description values. If a form or a grid column is assigned to a field with a validation table, a prompt list of valid values and descriptions will be presented.

When used for validation purposes, the data dictionary will make sure that the current window value matches a value from the validation table. A validation table can be static, dynamic, or custom. A static table consists of a table with values defined directly within the validation object (i.e., the items are coded into the object). A dynamic table consists of values maintained in the codes master file. Custom tables are customized—the values are defined however you want them to be. The contents of static and dynamic tables are maintained directly within Database Builder.

Field and Foreign-field Options
All prior revisions of DataFlex supported item options. Options such as capslock, autofind, findreq were assigned to the entry item. While item options are still supported, they will now be rarely used, having been replaced by data dictionary field options. When an option such as capslock is defined within your file’s DD class, all entry items that use that field will acquire that option. The advantages of this are obvious. You are applying your rules once at the database level and not multiple times at the data-entry level.

Data dictionaries recognize two types of field options, field, and foreign field. Regular field options are applied when your field is being accessed through views to which the file is the main file. When the field is being accessed through a view to which the file is a parent (actually ancestor) file, the foreign-
field options are added to the field options. If you look at your existing application, you will probably see that your parent fields have special settings: the index fields tend to be find-only (NoEnter) and non-index fields tend to be display only. Foreign field support is added to make this type of model easy to implement.

**Currency, Numeric, and Date Masks**
Visual DataFlex supports display masks for numbers and dates. Dates may be displayed in a variety of formats including numeric (e.g., 5/6/52, 05/06/1952) and text (e.g., May 6, 1952). Numbers may be displayed in a variety of numeric or currency formats which include thousands separator, decimal separator, currency symbols, and positive/negative indicators). Normally you will assign the mask types within your data dictionary. A field may be assigned a mask type or a custom mask. For example, if a numeric field is assigned a currency mask type, all entry items using this field will acquire the standard currency mask that is applied to all currency fields (e.g., £2,345.99). You may, of course, customize the default masks. This numeric field (or any field) may also be assigned a custom mask which will used for this one field (e.g., 2.345,99Cr).

**Field Status Help**
Each field may be assigned one line of status help. Once this is defined in the DD, all entry objects using that field will automatically display the appropriate status help.

**Default Appearance**
Each field may be assigned a long label, a short label, and a default visual control. When a data aware entry control is created within the IDE, these values will be used to create the object.

Long labels are used for form based entry (where labels usually appear to the left of the entry control). Short labels are used for grid entry (where labels usually appear at the top of each column). If labels are not defined in the data dictionary, the IDE will still assist you by creating an intelligent label based on the field name. For example, the field ZIP_CODE would be converted to “Zip Code.”

The default visual control determines what type of control will be used for a field. For example, you might want a date field to always use a spin-form
control (dbSpinForm) instead of a regular form (dbForm). These values are only default values. Once the IDE has created the control based on these defaults, you may change them.

If you are moving from 3.x Object-Oriented

If your current character-mode application is written using framework-style object-oriented DataFlex 3.x, the task of creating DD classes will be quite a bit easier. You already have your data-set classes. You simply have to convert them to data-dictionary classes. Here is how you would go about doing this:

1. Open a file within Database Builder. It will ask if you wish to create a new DD for the file. Answer yes.
2. Open your old data-set class with a program editor. Copy and paste all of the custom procedures and functions from the data-set class into Database Builder’s method editor.
3. Look at these methods. In some cases, they will require minor editing. Usually that minor editing is removing entire procedures and functions. Many of the tasks that required manual code in data sets are now supported directly from within a data dictionary. Examples of this are locking of system files, auto assignment of key field Ids, and key-field protection.
4. Set all the other standard DD settings.

Creating Views—the Visual Part

When creating your views and lookup lists, your goal is to be able to concentrate entirely on the visual aspects of your program. Because you have data dictionaries, you can do this. If you have done a good job of creating data dictionaries, you can now focus all of your attention on the visual aspect of your application. When you create a view, do the following:

1. Create the view container
2. Build a data-dictionary object structure. You do this by selecting DDs from a list and designating one of the DDs as the “main DD.” The IDE will connect the DDOs into a structure and properly set up any child to parent constraints as needed.
3. Add visual entry objects. Each of these data-entry objects (DEOs) will be assigned a file and field. This creates a connection to the appropriate DDO. At runtime, the DEO will use the services of the DDO to determine how the entry object should appear (e.g., display only,
Migrating Applications

Chapter 3

capslock, currency masks, prompt buttons, and contents of a combo list). All data-file requests (save, delete, clear, find) are passed from the DEO to the DDO. The most-common types of entry objects are forms, combo forms, checkboxes, and grids. In addition to the actual entry objects, you can also create grouping (container) objects. The three most-common types of containers are the tab dialog, group, and raised container. Your views may also contain non-data-aware objects such as buttons that are used to perform special tasks (e.g., a “close” button).

Keep in mind that each view that you create is fully independent of your other views. Each view can be written and tested by itself. Each completed view is a component. When you put an application together, you simply select your components from a list.

Views—Simple and Complex

Usually views fall into one of two categories: Simple maintenance views and complex entry views. If you look back at your existing application, odds are you will find that you have a number of simple entry programs and a few very-complicated programs. Those complicated panels get used all the time and therefore a great deal of time and cost has been invested into getting those complicated panels “just right.” If you are like most of us, you will probably discover that those complicated panels required more time to build than anyone possibly imagined at the time.

Most of your views will be simple maintenance views. These can be created within the IDE with remarkable speed. This assumes, of course, that you have properly created your data dictionaries. Everything comes back to the data dictionaries. The most-important part of visual design is having first created proper DDs.

Your complex views will still require extra time and effort. Their development, however, should be much easier than their procedural counterparts. The main reason for this will be ... (Can you guess?) data dictionaries. As views get more complex, the intermixing of visual design and database rules becomes harder to manage.

The more complex the view, the greater the contribution of the data dictionary. In addition, the non-modal data-entry model tends to simplify your complicated views by flattening them out and distributing their functions to other views. A typical procedural entry program often tried to support every type of data entry within a single program. In addition, the developer had to
Migrating from DataFlex to Visual DataFlex

decide where users would most likely need a particular entry functions and add the capability to just that right spot. This resulted in very-complicated, very-deep programs.

By deep, we mean you might have a pop-up screen that invokes another pop-up screen that invokes another screen and so on. (You know you’ve got this type of application if a technical support calls consists of “Where are you? Ok, you’re in the wrong place. Press escape, now press escape, now press escape, now press escape….”) You needed to create applications like this because it was too inconvenient to exit your current program so you could gain access to a maintenance panel. That overhead no longer exists. You do not need to exit views; you simply switch views and then switch right back. This reduces the need to add everything to a single view. It also creates a much more-consistent user interface.

View-Construction Tips

Creating views is primarily learning how to use the IDE. As you build your views, the following advice may help.

• **Think non-modal. Do not try to convert what you have.** The procedural entry programs of your existing program are probably not good models for VDF views. In particular, fight the urge to put all of the functionality of your current complex procedural views into your new VDF view (see above discussion). Often developers feel that the freedom of a user-directed system is too complicated for most users. This is simply not true. Users very quickly adjust to a flexible yet consistent interface. Look at the success of Windows!

• **Stick with the standards.** Today’s Windows panels and dialogs are quite simple. A good-looking Windows application usually has few different fonts (often-just one), colors that are consistent with the rest of Windows and consistent sizing and locating of visual objects. By default, the IDE adheres to these standards. For example, a VDF button is by default a predefined size (14 x 50 dialog units). A form is always a standard height (13 dialog units). Forms are horizontally spaced at a predefined distance (2 dialog units). All objects use the system’s standard font and the appropriate system color. This makes the application look like a “real” Windows application. You may customize any of these settings. Doing so requires extra effort and usually results in views that just do not look quite right.

• **Take advantage of multi-page tab dialogs containers.** Generally, it is hard to copy a complicated character-mode screen to a VDF view. Doing
Chapter 3

so usually results in views that look crowded and busy. You also want to stay away from screens that overlay information or present information in a cascade of pop-up panels. They just will not look right. Instead, you want to use tab dialogs. Tab dialogs represent an excellent way of presenting information. In VDF they are quite easy to create and very flexible. If you have too many tab pages to display in the available space, your buttons at the top of the page will scroll.

You can even nest tab dialogs inside tab dialogs (as are done in Database Builder). Tab-dialog pages represent a great way to present a lot of information in limited space.

- **Use the right entry control for the right job.** The highly DD-aware nature of DEO controls makes it very easy to use a variety of entry controls. In some cases, the choice of control should be obvious. A text field belongs in an edit control (dbEdit). You want to use a form (dbForm) for most entry fields with the following exceptions: A checkbox (dbCheckbox) should be used for checkbox (two-choice) fields. A combo form (dbComboForm) should be used for fields that have validation tables assigned to them. You can use radio objects for check box and validation table fields, but this tends to be an inefficient use of space. You can assign default entry controls to a field within the data dictionary. This makes it even easier to ensure that you make the right choice during view construction.

- **Know when to use Grids and Forms.** Multi-row grids are a great way to present and edit information. In some cases, their format ideally models a real-world process (e.g., order entry, check writing, journal entries). Be aware that there are some downsides to the use of grids. They tend to be more complicated (both for the developer and the user), and they can be slower. The speed is an issue because you are now showing multiple records (often with multiple related ancestor records) on the screen at one time. Naturally, this is slower. A lot of procedural applications tended to use the “table-zoom-to-form model” of development where a list of records appears, the user finds a record, and then presses a key to display and alter a “zoomed” version of the data. For various reasons, this model does not work well in the non-modal Windows environment. Instead a “form-prompt-to-list” method is used where a view consists of a single record and users press a key (or click the mouse) to pop up a modal list of records.

The record is selected and brought back to the form. You are
Migrating from DataFlex to Visual DataFlex

encouraged to use this model. You could do it the other way, but you are first going to have to alter the way your classes work.

- **Know when to leave well enough alone.** The IDE when used in conjunction with data dictionaries let you rapidly create views that are easy to use and protect against data corruption. If you use the tools the way they were designed to be used, your progress will be rapid. If you insist on doing things your own way, or adding just that one extra feature, you may find the price for that addition to be quite high. At the very least, defer the implementation of these additions and add them to a list. Get your application up and running and then decide which features on the list, if any are worth implementing.

**Reports and Batches**

Migrating reports and batch processes to VDF is the area where you probably have the most flexibility. These options range from rewriting all of your reports to making absolutely no changes at all. The migration path you choose will be a function of the needs of your deployment site.

**A Typical VDF Report/Batch: UI + Process**

A typical report or batch process usually consists of two components: a user-interface (UI) component and the actual report/batch component. Typically, the UI is used to allow users to select various processing parameters and to then allow user to start the report or batch process. The report/batch component executes the actual process. In VDF, these two components are placed inside a container component called a report view. The report view and its user-interface component are created using the IDE. The report/batch component is “plugged” into the view and, depending on its type, can be created a number of different ways. The report/batch plug-in component can be one of four types: Crystal Reports for DataFlex, WinPrint, Report Object, or custom.

**Crystal Reports for DataFlex:** Not surprisingly, Crystal Reports for DataFlex reports are created using the Visual Report Writer, Crystal Reports, which is licensed and sold as a stand-alone product. This is the easiest and the most-visual method of report creation. Once created, these reports can be deployed at any runtime site. The runtime site does not need to own their own copy of Crystal Reports for DataFlex to run these reports; they can be run from VDF programs.

**WinPrint:** WinPrint uses a special version of the Report object, which allows
you to create GUI reports (multiple font size, style, weight, etc.) Writing code creates these reports. Because they are code-based, the reports can potentially be more sophisticated than Crystal Reports for DataFlex reports. WinPrint can also be used for batch update processes. Because they are code-based, they also require more effort to create. The IDE’s report wizard will make this process a little easier. The wizard will guide you through the process of creating a WinPrint report and report view. While you will usually need to customize these wizard generated reports, it makes getting started much easier.

WinPrint is included as a standard VDF component. We recommend that you use WinPrint for more complex reporting.

Report Object: The Report Object is the same report object that is provided with DataFlex 3.1. For those of you not familiar with this product, it may be thought of as supercharged, object-oriented version of the Report Macro. Just like the Report Macro, the Report Object uses images to create output. This means that this should be used for line-printer-style reports (single-font, non-proportional font). A big advantage of the report object is that you can use existing report layouts (and existing images).

Custom: This can be just about anything you want. Any type of update process can be coded and run. A process could be contained inside an object or simply placed inside a procedure. The code inside the procedure would be very similar to your existing update processes. One difference is in its ability to interrupt a report. There is no InKey or KeyPress checking in VDF—Windows does not allow it. Instead, a status panel is provided that can be used to monitor and interrupt your process.

Business Processes

Visual DataFlex introduces the concept of a Business Process. A business process is any process that changes data and is program directed. A business process can be small (e.g., marking an order as shipped and modifying required inventory counts) or quite large (e.g., an end of year update process). These processes are contained inside of single objects called Business Process objects (or BPOs). A BPO will contain a data dictionary object structure (similar to a view’s DDO structure) and a processing procedure that starts the routine. This procedure will use the DDO structure to handle all file activity. This allows your batch processes to take full advantage of your data-dictionaries.

A special class called a Business Process is used to create these objects. This class is specifically designed to handle non-interactive processes. This
Migrating from DataFlex to Visual DataFlex

makes it easy to control:

- The finding, clearing, changing, validating, saving and deleting of data – all protected by the data-dictionaries.
- Status update displays – a status update panel is easily created.
- The cancellation of a process- an optional cancel button may be added to the status update panel.
- Error handling, error reporting, error logging, and error rollbacks.
- Status logging – activity can be written to a log data-file.

Often a BPO component will be plugged into a report view, taking the place of one of the other reporting objects. Keep in mind that it can be used for far more than this. BPO components can be plugged into any other component within your application. They can be added to report views, pop-up modal dialogs, entry views, and reports. A BPO is entirely self-contained. Once created, you have a component that models a part of your business (hence, the name Business Process).

Your current application probably has many batch processes that will fit the business process model quite well. While you may not be able to immediately convert all of your existing batch processes into BPOs, it should be your long-term goal. It lets your application take full advantage of data dictionaries.

Migrating Existing Report/Batch Processes to VDF

You have many report migration choices. You can choose to migrate all, part, or none of your report and batch processes. One reason you have so much flexibility is that a running report/batch process in a procedural program and a Windows program remains much the same. They are modal processes and have a very simple interface (show progress and allow cancel). The following lists some of the different strategies you might employ when migrating your application. When reviewing these options, keep in mind that any of these can be transitional choices. You can employ different short-term, mid-term, and long-term strategies.

- Do not make any changes at all: It may be possible that there is no reason at all to convert your existing report and batch processes. You can continue to run your existing processes on your existing runtime using your existing programs. If quality of output does not matter (or there is no output) and nobody ever sees these processes being run, you could just leave them alone.
• **Use your existing reports/batch process, UI, and runtime:** If your reporting and batch processing is largely a back-room process, you may wish to continue to maintain those parts of your programs with character-mode DataFlex. You could keep these processes entirely separate from VDF or you could create VDF programs that launch these programs. From within VDF, you could use runprogram wait or runprogram background to start the character-mode runtime.

• **Use your existing reports/batch process under VDF:** In this case, you would create VDF report views and move the user interface to VDF. The report plug-in would be your old procedural process. You can choose from the following variations for this approach:

  1. Your report plug-in could be a runprogram wait/background to your old report/batch process and a procedural runtime—ideally the console-mode runtime. In this case, you are giving your user interface a facelift while leaving the underlying process unchanged.

  2. Your report plug-in could be a chain wait call to your existing report/batch process. This would still require some code changes. The method of showing progress and the method of canceling the process would need to be changed. At the very least, your old code that handles status updates and process canceling would be replaced with a status panel.

  3. You could take your old process and place it all inside a procedure. Most of the global variables used in the old process would be replaced with the procedure’s local variables. Wrapping an old process, often an entire chain wait module, is a surprisingly simple process. You may find that it is as easy to do this as it is to chain to another module. If this is the case, this option is preferable.

• **Rewrite your existing report/batch process in VDF:** In this case you would create VDF report views, move the user interface to VDF, and create a report plug-in based on one of the standard report types of VDF. You can choose from the following variations for this approach:

  1. Use the “basic” report object. It is quite easy to convert a report-macro report to a report-object report. This was a design objective of the report object. Since this would be based on a mono-spaced, single sized font, you could use your existing output images. Report output is as fast as before, or possibly a bit faster.

  2. Use a WinPrint report object. This is similar to the above option except
you will be required to redesign your output. Instead of using images to create output, you now must “describe” the output through a series of commands. This gives you the capability of selecting and changing your fonts (type, size, weight, orientation, and color). The advantages is a better-looking report; the disadvantages are a longer conversion and slower output performance. Because of the reduced performance, this approach is practically not an option for very-large reports that are run frequently or subject to a tight deadline.

3. Use Crystal Reports for DataFlex to create the report. If your report logic were simple and your output requirement complex, this would be the best choice. Even quite-complicated report processes can be written using Crystal reports for DataFlex.

4. Create Business Process objects to handle your batch processing needs. While it will take some time to convert existing processes there will be a long-term payoff when you have a library of self-contained, data dictionary enabled business process components.

Crystal Reports for DataFlex as an End-User Tool

You can create and deploy Crystal Reports for DataFlex reports without needing to purchase any additional software at your deployment site. While you do not need to purchase and install a copy of Crystal Reports for DataFlex at your deployment site, you may wish to do so. This provides your users with a powerful data-inquiry tool. The additional use of Crystal Reports for DataFlex data dictionary (not be confused with VDF’s DDs) makes it possible for you to create a report-inquiry environment that is both user-friendly and safe. Using Crystal Reports for DataFlex as an end-user tool can be an important part of your VDF reporting strategy.

Deployment

VDF deployment is discussed in detail in VDF’s documentation. We will very briefly discuss the areas of deployment that seem to cause the most confusion.

- **What to deploy:** The VDF Client installation will automatically install all the necessary files and create the appropriate registry settings that allow VDF applications to run. The Client Configuration File allows you to control various aspects of the Client installation without user interaction, so that the Client installation can be embedded in the main installation for your application. In addition to the VDF Client files, your development workspace area consists of five directories: Data, Help, Bitmaps,
Migrating Applications

Programs DDSrc, and AppSrc). You will want to deploy the first four directories. The "src" directories contain source code and are not needed at your deployment site.

- **Additional Requirements:** If you want your users to be able to create custom Crystal Reports for DataFlex reports, you will want to purchase and install the appropriate license for your deployment sites.

- **Creating an Installation program:** The documentation states that you may want to create a Windows-style installation for your application and suggests several available installation products. This has created some confusion. You do not have to use an installation program. You can install your application using any method you want (copy the files, create zipped directories, load files over the modem, etc.). If you have a shrink-wrapped application that will be widely distributed, you will probably wish to use an installation program—it is an expected part of a Windows application. If you migrate an in-house application, such an installation would be overkill.

- **Workspaces:** You will find that workspaces will make it much easier to deploy applications. Instead of creating long complicated DfPath settings, you only need to specify the name of a program’s workspace. This workspace name is automatically compiled into your program. When your program is deployed, it needs to run from that workspace. A workspace consists of logical directories that map to physical directories. You simply need to create this map. Although it is not required, you are encouraged to use the standard workspace directory structure in your deployed application. Workspace information is maintained in the Windows registry.

**Programming in VDF**

VDF is still primarily a programming language. The main difference between VDF and previous revisions of DataFlex is that we now have tools that will write much of your application for you. When you use the IDE to generate your visual components and Database Builder to generate your data dictionaries, these tools are writing your code for you. These tools have a profound effect on your development strategies. It will change how you code and when you code. It will also change how you learn, what you learn, and what you bother to remember. Below is a partial list of how these visual design tools will change your development strategies:

- They will teach you how to use the language, OOP, DDs, and the Framework. Once you understand conceptually what you need to do to
write a program, you can get started. The visual tools will guide you in
the right direction. The visual DD builder will create proper data-
dictionary code. The IDE will create proper view code. You do not need
to spend time learning syntax, learning property names, and learning
how to create and connect objects. Let the tools do this for you. We are
not suggesting that you do not want to understand the code the tools
create. This is all information you want to master. However, you can
master this while you are productively creating applications.

- They will generate nicely formatted, good-looking, consistent, code. Not
  only will you be able to understand your programs, so will the rest of
  the world. This has a huge impact on maintenance and revision strategies.
  This makes it easier for you to remember how your programs work,
  easier for you to bring new developers into your team, and easier to use
  the assistance of outside consultants.

- You do not need to remember nearly as much. There is a difference
  between understanding and remembering. VDF, like any good high-level
  OO development system, has many features. Its tool kit consists of
  numerous classes containing numerous properties and methods. If you
  had to manually code all of this, you would have an awful lot to
  remember to be at all productive. The visual design tools do most of this
  remembering for you.

- It changes the way you research and solve problems: Rather than
  committing a large body of information to memory, you need to know
  how to find it. The VDF on-line help system and its integration into the
  IDE was designed to be used for this purpose. We no longer expect you,
  or anyone, to read all of the documentation. We expect most of your
  documentation reading will be part of a “How do I do … ?” inquiry. The
  on-line documentation in particular is designed to accommodate this.

**Custom Coding**

Both the IDE and Database Builder allow you enter custom code. This code
can be entered from within the tools by opening up a custom code editor or
from outside the tools with a program editor. Most often, custom code
consists of creating or overriding a procedure or functions. Most often, you will
want to create your custom code from within the IDE using the IDE’s code
ingitor. As you get more experienced, you will find yourself using an external
ingitor as well.

The IDE and Database Builder not only write VDF source code, they also
must be able to read it. To make sure that they can to do this with code you
add, certain coding conventions and syntaxes must be obeyed. The general
strategy is simple.

The visual tools maintain most of the code. Primarily those tools should change those areas. If you make manual changes to these areas with an external editor, you must make sure that the IDE or Database Builder will be able to understand those changes. The IDE and Database Builder allow you to open up custom-code areas. The tools will store but not attempt to process any code in these areas. Within these areas, you can do whatever you want. These custom areas are identified with special markers. If you code within the tools, those markers are maintained for you automatically.

If you choose to code with an external editor, you must make sure that you maintain these markers yourself. Usually this is quite easy since the IDE and Database Builder have already provided them for you. You just need to make sure that all of your changes remain within the boundaries of the markers.

Why you want to use an external editor at all? Sometimes an editor is a better tool for the job. This is particularly true when dealing with custom-code areas. Customizations within these areas tend to be non-visual. The current editor of choice among VDF developers is Multi-Edit from American Cybernetics. This editor has many DataFlex-specialized features such as special colors for commands. Good editor cans sometime speed up debugging. For example, Multi-Edit allows you to compile a program, display the list of compiler errors, and jump to the line of code in the package where the error occurs. When a good editor is the best tool for a task, use it. When the IDE or Database Builder is the best tool for the task, use them. You can even use the two simultaneously although this requires that the developer keep the two representations of source in synch with each other.

Sometimes you will find that the visual tool simply cannot do what you need it to do. In such a case, you would take your component as far as you can take it visually and move to an editor where you would make changes that the visual tool can no longer process. From this point on, that one component would need to be maintained by your editor, but you could still use the component within programs maintained by the tool. The fact that you can do this for any selected component in your application should be considered a strength of VDF. You are not limited by the capabilities of the visual design tool.

New Class Names in VDF

If you are migrating an existing object-oriented DataFlex program to VDF, you should note that VDF has a new set of class names. Previous revisions of DataFlex reflected the order of our internal development in the class names.
The "best" names, those that are short and easy to understand, were assigned to low-level classes that developers never used. The high level classes that are used all the time by developers were assigned rather long and cryptic names (because they were the last class names to be assigned and all of the good ones were already taken). This was even more pronounced in Windows, (just what is a DFEntry_View_Client?).

We addressed this in VDF by renaming all classes. The high-level classes were assigned the more-developer-friendly names. Whenever possible, we used names that were consistent with Windows (e.g., MenuBar and PopupMenu instead of Action_bar and Pulldown_Menu). We removed underscores from the class names (another Windows standard). We also made a clear yet simple distinction between data-aware and non-data-aware classes. Data-aware classes are all prefixed with a "db" (form and checkbox are non-data-aware "basic" classes; dbForm and dbCheckbox are data-aware).

The result is names that are easier to learn, easier to remember, and much better looking. This makes your development easier. Implied in this change is the idea that developers should not need to concern themselves as much with the class hierarchies. They should find the tool (class) they need and use it. This implication is intentional. We want to make VDF easy to use.

To prove this point, we present a list of the most-commonly-used VDF classes. The classes consist of containers (classes that are used to group other objects) and controls (classes that are used for entry). Each of these classes may be basic or data-aware (db). Look at this list. Even without documentation, you can probably guess the purpose of most of these classes.

<table>
<thead>
<tr>
<th>System Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Panel</td>
</tr>
<tr>
<td>ClientArea</td>
</tr>
<tr>
<td>ModalPanel</td>
</tr>
<tr>
<td>ToolPanel</td>
</tr>
</tbody>
</table>

---

47
### Basic Container Classes

<table>
<thead>
<tr>
<th>View</th>
<th>Group</th>
<th>TabDialog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Container3d</td>
<td>RadioContainer</td>
<td>TabPage</td>
</tr>
<tr>
<td>Container</td>
<td>RadioGroup</td>
<td>BitmapContainer</td>
</tr>
<tr>
<td>ModalPanel</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Basic Control Classes

<table>
<thead>
<tr>
<th>Form</th>
<th>Edit</th>
<th>LineControl</th>
</tr>
</thead>
<tbody>
<tr>
<td>ComboForm</td>
<td>Grid</td>
<td>SpinButton</td>
</tr>
<tr>
<td>Radio</td>
<td>Button</td>
<td>TreeView</td>
</tr>
<tr>
<td>SpinForm</td>
<td>CheckBox</td>
<td>List</td>
</tr>
<tr>
<td>TextBox</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### High-Level (DEO) Container Classes

<table>
<thead>
<tr>
<th>dbView</th>
<th>dbContainer</th>
<th>dbRadioGroup</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbContainer3d</td>
<td>dbGroup</td>
<td>dbRadioContainer</td>
</tr>
<tr>
<td>dbTabDialog</td>
<td>dbTabView</td>
<td>dbTabPage</td>
</tr>
<tr>
<td>dbModalPanel</td>
<td>dbBitmap</td>
<td></td>
</tr>
</tbody>
</table>

### High-Level (DEO) Control Classes

<table>
<thead>
<tr>
<th>dbForm</th>
<th>dbCheckBox</th>
<th>dbComboForm</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbList</td>
<td>dbEdit</td>
<td></td>
</tr>
<tr>
<td>dbSpinForm</td>
<td>dbGrid</td>
<td></td>
</tr>
</tbody>
</table>

### Data Dictionary Classes

<table>
<thead>
<tr>
<th>DataDictionary</th>
<th>CodeValidationTable</th>
<th>ValidationTable</th>
</tr>
</thead>
<tbody>
<tr>
<td>ValidationList</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
“Specialization of Labor” with VDF

One of the advantages of VDF is that different areas of development require different skills and different levels of skill. You could probably break VDF application-development work into the following categories:

- Database and data-dictionary design and implementation
- Visual Application (views, lookup lists, etc.)
- Help authoring
- Crystal Reports for DataFlex report creation
- Other Report and batch process creation - integration of all reports into the application
- Object-oriented “expert” - class designer, custom-code “guru”
- Windows low-level API expert

Let’s assume that an application team consists of the following three people. We will describe their skills and their current project responsibilities.

<table>
<thead>
<tr>
<th>DEVELOPER</th>
<th>SKILLS</th>
<th>RESPONSIBILITIES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Developer 1</td>
<td>Has used DataFlex for four years. Understands all of the database issues (files, relations, rules, etc.). Knows how to create data dictionaries and has over the last nine months become familiar with OO DataFlex.</td>
<td>Maintains all database and DD code. Assists in some of the more-complicated views and some of the more-complicated custom-code requirements. Assists with more-complicated debugging.</td>
</tr>
<tr>
<td>Developer 2</td>
<td>Just graduated from college with a computer-science degree. Has been working with Visual DataFlex for three months. Before those three months, had never even heard of the product.</td>
<td>Maintains most of the visual code. Writes some batch processing/reports and creates visual report views (interfaces) for Crystal Reports for DataFlex reports.</td>
</tr>
</tbody>
</table>
Migrating Applications

<table>
<thead>
<tr>
<th>DEVELOPER</th>
<th>SKILLS</th>
<th>RESPONSIBILITIES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Developer 3</td>
<td>Has little programming experience, less object-oriented experience, and even less DataFlex experience. Was hired originally to write documentation.</td>
<td>Writes documentation and uses various help-authoring tools to create on-line help. Creates Crystal Reports for DataFlex reports.</td>
</tr>
</tbody>
</table>

In this example, this team has chosen to use the classes and tools provided, so they do not need an in-house class designer. The application did have some custom Windows interface issues. They hired a local Windows API consultant to help them with the interface requirements. This consultant has no DataFlex experience.

It might be interesting to follow these three developers after nine months of development. Now, their skills have increased and their job requirements would have changed. There might also be some additional developers added to the team to handle some of the other more-basic requirements. Within nine months, we might expect Developer 1 to spend more time extending the various class tools and directly dealing with other types of Windows interface issues (e.g., connecting to the Internet). We would expect Developer 2 to be a master the IDE programmer (including all custom code requirements) and now assisting in the maintenance of the data dictionaries. We might expect Developer 3 to have added the IDE development to his (or her) skill list. This developer can visually create just about any type of view. When custom code is required, Developer 2’s assistance is obtained.

It should be emphasized that VDF makes it easy to assemble a productive development team and makes it easier to handle team-member turnover. (Note we said easier. Turnover, while it can be planned for, is never easy.) Not every member of a development team needs to be a VDF expert. While it is good to have someone who can tackle the hard problems and answer the hard questions, this skill is not required of every member of your team.
Frequently Asked Questions (FAQ)

The following are some of the most-commonly-asked questions concerning application migration and VDF. Some of the information here has already been addressed in the previous sections. They are repeated here for ease of reference.

**Can I use my old data? Can I run in parallel?**

Yes and yes! It is understood that most existing applications need to keep running until the conversion to Visual DataFlex is 100% complete. The data files in VDF are 100% compatible with previous revisions of DataFlex. This means that your data can be accessed concurrently using Windows and non-Windows runtimes. In fact, the same computers can safely run character-mode DataFlex and Visual DataFlex at the same time.

While the data files are compatible, there is some data-file issues that must be resolved.

- You cannot use a 2.3 runtime to access files that have been saved in 3.0 format. If you continue using a 2.3 runtime, you must be careful to keep your files in 2.3 format. It would probably be best to first migrate to a 3.x runtime to obviate this concern.
- Your existing file-relationship structures (or lack thereof) may not be proper for VDF’s data dictionaries. Most often, these structures can be corrected at run time using the set_relate command. In the worst case, you may need to change your data-file structures, which, of course, may affect the way your current application runs.
- VDF delivers solutions to the Year 2000, two-digit versus four-digit-year issue, but they do have to be activated. Solve this now, and you will not have to worry about it for another 100 years.

None of these problems is serious and they are all discussed in a previous section.

**Is the language the same?**

The language is the same. VDF supports all of the procedural-language commands found in Revision 2.3 and all of the object-oriented commands added in Revision in 3.0.
Programming in VDF consists of creating subclasses and objects. Creating procedures and functions customizes these objects. These procedures and functions contain procedural code.

In fact, in VDF it is actually more useful to understand the procedural-language commands than the object-oriented commands. The object-oriented code is created for you automatically using the IDE and Database Builder. When you need to add custom code to these programs, you will most likely be writing procedural commands within procedures and functions.

Although the procedural language is the same, it has been enhanced significantly. The good DataFlex programmer will want to take advantage of those improvements. Among those are:

- Do not use subroutines (gosub and labels). Instead, use procedures and functions. There are numerous advantages to using procedures and functions. You can pass them parameters, they can return a value, and they can use local variables.
- Avoid using global variables. Whenever possible, use a local variable inside your function or procedure. When you need to create a “variable” that an entire object (including all child objects) can access, create a property. Only create global variables when they are truly global, when all parts of your program need equal access to the value.
- Avoid using Indicators. Indicators are global and are limited in number. Instead, use Integers (preferably local Integers). They are actually more flexible than indicators anyway.
- The runtime’s logical expression evaluator is far more powerful than its 2.3 counterpart. In particular, you can use AND and OR within expressions (e.g., If (Allow_Save AND (Status="A" OR Status="B") ….)).
- There are now commands that let you access and change all information about a data file. The Get_Attribute and Set_Attribute commands can be used for this. In addition, commands exist that allow you to restructure and sort data files from within a program.

**What is a DD subclass? What is its relationship to a DataSet subclass?**

If you are asking this question, you are probably already using the DataFlex Application Framework to write your character-mode and Windows-based applications. This means that you are currently creating data-set subclasses for each file in your application.

The DataDictionary class is a subclass of the DataSet class. The data-
dictionary class is an improved version of the data-set class. The data-set class and data-set concept was good, but it just did not go far enough. Before data dictionaries, we would advise programmers to places as many rules as possible inside their data-set subclasses. We used the phrase “as possible” because it was not possible to put all your rules within data sets. Those rules, which could not be modeled within the data set, had to go in the entry objects.

This limitation was even more noticeable in VDF. The IDE is a tool for designing the visual part of an application. When used with data-set technology, the IDE and therefore the product became too hard to use. Extending data sets by creating its subclass DataDictionary solved the problem.

We now advise programmers to place all rules inside the data-dictionary subclasses.

In prior revisions of DataFlex, the Framework method revolved around creating data-set subclasses and creating views that contained data-set-object (DSO) structures and data-entry objects (DEOs). The framework specified specific rules for connecting and constraining DSO structures and for connecting DEOs to these structures. None of these rules has changed at all. We now refer to DDOs instead of DSOs and the connecting syntax has been improved.

You may also notice there is less emphasis on actual framework connection rules. The reason for this is simple. The IDE understands how to do this, and most of the time, sets all of these connections for you. This is one less thing you need to know in order to use the technology.

**Can I continue to use data sets instead of DDs?**

You could, but you probably do not want to. All of the data-entry classes will work with DDOs or DSOs. There is little reason that you should want to do this. If you are using data sets right now, it should be quite easy to convert these to data dictionaries. Here is what you want to do:

1. Run Database Builder and load your data file. It will notice that there is no DD file and will ask if you wish to create one. Answer yes. This will create a DD file for this data file.
2. Load your DS class with some text editor (like Notepad). Take your custom procedures and functions (update, backout, creating, validate_save, etc.) and copy them to the clipboard. Now, within Database Builder, go to the “Methods” tab page and paste the code into your methods section.
Chapter 4

3. Set up the rest of your DD from within Database Builder. Most of the information in the DD tab pages is not currently stored in your data-set classes. You will need to search your existing DEO views for any entry_item commands that use this file. Find the items with the {…} options (capslock, autofind, noput, iPrompt, etc.). All of these (options) should be moved to the DD class.

4. Save your file.

Once you have done this for a few files, you will find this to be a very fast process.

If you choose to continue using your existing data sets, you will not be able to fully use the IDE. The IDE expects data dictionaries. You could still use the IDE as a prototyping tool, but you will be required to make custom code changes to run your programs. Once these changes have been made, you will not be able to use the IDE for these views.

You are much better-off converting your data sets to data dictionaries. In the end, this will be the most-cost-effective migration scheme.

Do I have to use DDs all the time?

No and this can be very important to your migration strategy. Your visual entry programs will most likely use data dictionaries. Update processes (e.g., end-of-month updates) do not have to use DDs. You can bypass the DD layer and update your files exactly as you do right now. Clearly it is safer to use the DDs, and at some point you want to create data dictionary enabled business process objects (BPOs) and use them throughout your program. However, right now you probably have existing batch-processing code that uses existing data that has been thoroughly tested and it works. In such a case, you can continue to use your existing update processes. Usually these existing processes will be moved into VDF by placing them inside a procedure.

Can I use DDs in character-mode DataFlex?

Yes! The character-mode DD technology for DataFlex 3.1 is currently in beta. Both VDF and DataFlex 3.1 can use the same DD subclasses. This means that you can use VDF’s Database Builder to create and maintain your DD classes for either platform. In fact, Database Builder running under VDF is the recommended tool for creating DD classes for DataFlex 3.1. Do not expect to see a character-mode version of Database Builder.
The compatibility of DD classes between the Windows and character-mode platforms is an important part of the cross-platform compatibility. If you need to maintain an application that runs under Windows and character mode, you will maintain a single set of data dictionaries. The visual entry objects (views, selection lists) will be unique for each platform. The visual capabilities of a GUI and character-mode environment are so different that it is all but impossible to create screens that look good on both platforms. Therefore, the visual code is not shared, but the non-visual code is shared.

**When I change a DD, must I recompile?**
Yes. The data dictionaries are code-based. When you load or save a DD within Database Builder, you are loading or saving source code. Making a change in source will not affect your application until you have recompiled the program. Therefore, anytime you make a change in a DD, or you make a change in one of your views, you need to recompile all applications that use the data dictionary (or view).

**Why can’t I use my old code? My old programs run great.**
We do not hear this question as often as we used to. It has been included for old times’ sake.
Even if you could take an existing application and magically change it so that it looks like a Windows application, there is a 99% chance that the results will not be satisfactory. The most-important change between your old procedural program and a VDF program (or any Windows application) is the way the application operates—a Windows program is a non-modal. It should be easy to switch to any window within a view, and to switch to any window within any other view.

This type of non-modality must be designed into your application from the very beginning. Most procedural programs are far more modal. Information is entered and accessed by “drilling down” into the application. The user presses a key and a popup panel appears. They press a key within this panel and another popup panel appears. Each of these panels is modal—the user cannot get back to the previous panel without closing the current panel. This style of interface is not appropriate for the Windows environment. Even if your users cannot tell you what is wrong, they will sense it. They will tell you that “it just doesn’t feel like other Windows applications.”

This does not mean that you have to throw away years of development. You can use your old code (at least some of it). It must, however, be moved. This is why data dictionaries were created. All of your database rules will be
moved to the DDs. Look at one of your existing programs. What would happen if you could move all of the database-related code out of the program? It would probably be a lot smaller, and, it would be a lot easier to redesign in VDF. Now look at the database code you removed from your entry view. Odds are, this is your “expensive” code. You can move this code to your DDs.

What type of computer should I run this on?
Any computer that is appropriate for Windows 95/NT is appropriate for VDF. If you ask this question about any current Windows product, the answer is “the faster the better, the more memory the better.” VDF runs great on just about any Pentium-based computer. Your development computer should be more powerful than your runtime computers. The IDE and DataBase Builder are big, complex programs. Relative to other Windows-based development systems, VDF is actually quite frugal in its use of memory and other resources.

Can I run this in Windows 3.1?
No. If your application must run under Windows 3.1, you will need to use DataFlex for Windows. Keep in mind that VDF is a far more-powerful product. DataFlex 3.05 for Windows does not support data dictionaries and it probably never will. Try to talk your client or company into moving up to Windows 95 or NT. They will thank you in the end.

What compatibility is there between VDF and DataFlex 3.1?
As mentioned in a previous answer, data dictionaries are cross-platform compatible. A beta version of the data-dictionary classes is available for DataFlex 3.1. You can (and should) use VDF and Database Builder to create and maintain your DD subclasses. DataFlex 3.1c as well as VDF may use these DD subclasses that are created.

What can you tell me about speed?
We are only letting you ask this question because early revisions of object-oriented DataFlex had some performance problems and the rumors of these problems have persisted. The quick answer is that performance is quite good. The type of computer required to run Windows 95 or NT provides a good match for VDF’s requirements. If you are running VDF on a Pentium-based workstation, you should have no performance problems.
Performance should be divided into several areas: database-activity speed, visual-interface speed, and program-load/processing speed.

**Database Speed:** The good news here is that the underlying databases and more important, the underlying method of accessing your data, remain the same. A properly written DataFlex program is fast where it really counts—finding and saving records in a multi-user environment. VDF and DataFlex 3.1 have a number of optimizations that make it even faster. An indexed find does not actually load the record until a non-indexed field is required. This is automatic. Data dictionaries employ "smart file mode" technology that ensures that only files that actually participate in a save are locked and reread.

**Interface Speed:** If you are used to character-mode video-display speed, VDF may seem slow. What you really want to do is to compare the interface speed to other Windows applications. If you do this, you should discover that VDF's display speed is quite reasonable.

**Processing Load/Speed:** There is an added overhead for object orientation. This is the overhead that was so noticeable in early revisions of OO DataFlex. Since that time, a lot has changed. The biggest change is that computers are much faster. In many ways, DataFlex 3.0 was ahead of its times. The hardware available (mainly speed and memory) was not quite powerful enough to properly power an OO application. This is no longer an issue. If your hardware is fast enough to run Windows, it is fast enough to run VDF. In addition, a number of runtime and package optimizations have greatly reduced bottlenecks. VDF (and DataFlex 3.1) provide a method of deferring the creation of objects until they are actually needed. This deferred creation is easily implemented (one click of the mouse within the IDE). This speeds up program initialization.

**Could you give me a basic overview of Workspaces?**

As you move into the VDF environment, you will notice that a new concept, *workspaces*, has been added to the product. In previous revisions of DataFlex, development work areas were indirectly defined through a combination of environment (dfpath) and search-path settings. There was no standard for these settings and each developer would develop their own strategy for creating these work areas. More structure was required in VDF. The VDF directory structure has gotten more complicated, the environment is no longer a suitable place for directory-path settings, and both the IDE and Database Builder need to know where to find and where to save various files. In addition, we wanted the developer to be able to create and switch work areas easily from within the IDE and Database Builder. Workspaces have been introduced to provide this functionality.
A Workspace creates a logical development area, which consists of physical directory areas. Those areas are:

- Data
- DDSrc
- AppSrc
- Help
- Bitmaps
- Programs

Each of these logical areas points to a physical disk subdirectory. Normally there is a one-to-one mapping between logical and physical directories, but this is not required.

Once workspaces are created, it is very easy to switch between them. From within the IDE or Database Builder, you load the workspace selector and select a new workspace. This makes multiple project development quite easy. We provide a suggested physical directory for each workspace. If this suggested structure is used, it also becomes quite easy to move information around between workspace areas.

Workspace information is stored in the Windows registry. A VDF utility is provided to let you easily create and modify this information. In addition, any of these settings could be changed with regedit (the Windows registry editor).

**Are workspaces used at runtime?**

Workspaces are used at runtime. The name of the workspace is compiled directly into a program. This allows a program to be run “explorer style” by double clicking on your compiled VDF9 program. The program will use its workspace name to determine what directories it should use. This means that a program can be run from any location (such as a floppy disk or an e-mail enclosure). Although the workspace name is compiled into your program, the locations of the workspace directories are not. This gives you the flexibility to place your files in different locations.

In addition, Visual DataFlex provides a method that allows you to run the same program in different workspaces. Special program keys can be created that override the name of the workspace compiled in the program. These keys are stored in the Windows registry. They allow you to associate a key name with a program name and workspace name. When you dfrun the key (dfrun MyOrderKey) instead of the program (dfrun c:\vdf9\examples\order\order),
VDF will search for the appropriate key in the registry and run the program defined by the key in the workspace defined by the key.

**Must I use workspaces?**

You do not have to use workspaces for program development. Workspaces, however, make it a lot easier to manage and deploy your development projects.

If you plan to use the data-dictionary builder within Database Builder and you are planning on using the IDE, you will want to use workspaces.

Workspaces were specifically designed to allow developers to easily manage multiple projects within the IDE and Database Builder. The compiler and runtime do not actually use workspaces. When a program is designed, compiled, and run from within the IDE, a program registry key is created for the program. The IDE uses its knowledge of workspaces to properly construct this key. If you are not using workspaces, you can still use the compiler and the runtime. You must, however, create this program key manually.

There are two possible reasons why a developer would not want to use workspaces. Some developers may choose to employ an alternative non-framework style of development. In such a case, the developer will not use the data-dictionary builder, the IDE, or the standard VDF classes in programming. In such a case, there would be no need to use workspaces. In other cases, a developer may be using the VDF classes but not using the IDE for their development and wish to use their own methods for managing their work areas. If you are using the framework classes but not using the IDE, we still recommend that you use workspaces.

**Will “real programmers” writing “real applications” use the IDE and Database Builder?**

Absolutely! We expect that the IDE and Database Builder will be the two primary tools used to create applications. A great deal of effort went into making these tools powerful and extendible. They really work for the following reasons:

- The framework method of application development is a proven method. It works! The IDE and Database Builder are designed around this method. This greatly increases the chance that these tools will generate usable programs. Not only are these tools compatible with the Framework, they fully “understand” the method. Object structures and applying the framework rules intelligently makes connections.
• The task of developing database programs has been divided into two domains. Database Builder is used to maintain your database files and rules; the IDE is used to generate visual user interfaces. This division of labor makes it much easier for the developer to create and maintain programs. It also made it possible for us to create specialized tools that really work.

• We allowed room for expansion. We know that every application will require custom code. Both the IDE and Database Builder support the ability to enter customized code.

• Both tools read and write standard DataFlex source code. This gives you the flexibility you need as a developer. This allows you to use the IDE and Database Builder as cooperative development tools. They can be used with other tools such as your program editor. This makes their usage realistic.

• Look at the source code generated by the IDE and Database Builder. The code is clearly structured and nicely formatted. Think how much work it would be to write this code manually. If you look at a view generated by the IDE, you will note that a commented object tree is generated at the top of your program and that all objects are registered in alphabetical order. Try doing that with an editor.

**Do you have to use the IDE and Database Builder all the time?**

As mentioned above, there are times when it is just easier to work with a program editor. You can always use your favorite editor to make any change you want. As long as you obey certain source format rules, manual source-code changes can be loaded back into the IDE and Database Builder without any problems. It is expected that the experienced VDF developer will regularly switch between the IDE/Database Builder and their program editor.

**Will the complexity of the DD slow me down?**

Adding a data-dictionary layer to your program obviously adds an overhead. This overhead has been surprisingly modest. In fact, in most conditions you cannot perceive any speed difference at all. We have a very simple means of measuring this. If no one complains to us about performance, performance is satisfactory. (Whenever performance is an issue, we are always the first ones to hear about it!)

The largest processing overhead of data dictionaries occurs during a pre-save validation. Every field for every file participating in the save is validated. There are methods for optimizing this validation process but these optimizations
involve selectively disabling complete validation and should therefore be used carefully.

**What are Business Process Objects (BPOs)?**
When designing an application there is a tendency to concentrate first on the visual data manipulation processes. This makes good sense because components that allow user directed data entry are, by far, the most complicated part of an application. However, each application will also consist of a large number of non-visual data processes. Rather than allowing the user to determine how data is changed, you, the programmer, must write the code to run these processes. These are often referred to as batch processes. While not required, it is desirable that these processes use the same data dictionary technology utilized by the entry views. This is accomplished by: creating an object component that contains a data dictionary object structure and a process (a procedure) that employs the DDOs for all data manipulation. Because these batch processes now use and obey the business rules built into your data dictionaries we refer to them as business processes.

Examples of business processes are:
- Importing data from an ASCII file into a database.
- Processing an order (marking it shipped, changing inventory quantities, creating, and printing a shipping invoice).
- Duplicating a set of records (e.g., creating a new check based on an existing one).
- Deleting old records.
- Performing end of period updates

These types of processes can be very simple or quite complicated. In all cases, you want to be able to run all of these data changes through your data dictionaries.

A special class called a **Business Process** is used to create business process objects (BPOs). This class is designed to facilitate non-visual data manipulation. It has built in support for error handling (processing, reporting, logging, and rollback), visual status updating, process cancellation and status logging. Most of these features are enabled or disabled by setting properties.

When you create a BPO, you must add the data dictionary object structure and create the procedure to do the actual data processing. When complete you will have a self-contained component capable of performing a business
task. This component can now be plugged into any other component (report view, view and/or dialog) and utilized by sending it a message to start the process. Eventually you will create a library of these components.

Since BPOs are non-visual, you will probably create these components using an external programming editor. You will add these components to your IDE-based visual components by including them in an object’s custom code area (e.g., `#include ImportCustomersBPO.pkg`).

**Do I have to use Business Process Objects?**

No, this should be an important part of your migration strategy. Whether you know it or not you probably already have created a number of business process components in your current application. Often these types of processes were placed in chain wait programs or contained within isolated parts of a program. If these processes already exist and are thoroughly tested, you may not want to change them right away.

One migration strategy is to create pseudo BPO components. Rather than placing a DDO structure and a procedure within the BPO, you could place your old code (moving the code into a procedure or several procedures). Alternatively, you might create a procedure that chain waits to your old program. The BPO’s external interface (the message or messages sent to the BPO to run the process) should be the same for a pseudo BPO as it would be for a real BPO. As time permits, you can convert these BPOs into data dictionary enabled components. Because the external interface will not change, the rest of the program will never know the difference.

**Must everything be in one program? What if I want to upload a change to my customer?**

If you are used to procedural applications that consist of many small .FLX files, the “all-in-one” .VDF files may seem at first to be a disadvantage. (.VDF is the new extension for compiled programs.) It is quite likely that a single .VDF program might be several megabytes in size or larger. Of course, instead of having 450 .FLX files, you probably only have several .VDF files. In many ways, this actually makes it easier to keep your application “in synch.” (How often have you uploaded five file changes and forgot about File Number Six?) The .VDF files compress down in size quite efficiently. Uploading these large programs should not really be a problem in these days of fast modems.

**Chain wait used to work so well. Why can’t we keep doing that?**

The chain wait was a marvelous construct for procedural programs. It made it
quite easy to create self-contained modules. This was particularly useful in prior revisions of DataFlex because it was not at all easy to maintain multiple self-contained components within a single program. The main disadvantage with chain wait is that it is a very modal operation. If an application chain waited many levels deep, you would need to exit from each level before switching to another entry “view.” This type of program model does not fit the Windows application style. In fact, this is one of the main reasons that an existing procedural program will not migrate gracefully to Windows. The problem is operational, not visual.

On the other hand, the object-oriented nature of VDF and DF3.1 make it quite easy to maintain multiple components within a single program. These components are, of course, objects. A typical single program will consist of data dictionaries, entry views, report views, lookup lists, popup panels and much more. With object orientation, you get all of the advantages of component modularity (encapsulation) combined with all of the advantages of a non-modal interface.

**What about a menu system?**

There is no main menu system in VDF. This may be added in a future revision. The entire concept of a main menu does not really fit well within the Windows environment.

**Development:** During development, you should think of the IDE as being your main program. All development activity can be coordinated from within the IDE. Apart from handling all visual component development, the IDE provides easy access to the most-commonly used development tools (Database Builder, Compiler, On-line Help, DFSpy, dbExplorer, Workspace Explorer, etc.). In addition, you can add custom programs to the IDE’s Utilities menu. For example, want to add menu access to your external program editor.

Often you will run wish to run several of these tools at one time. Unlike the traditional character mode menu systems, you do not need to close one tool before using another. For example, it is very useful to leave Database Builder open while working in the IDE. This way you can quickly bring up information about a file’s data dictionary.

**Deployment:** A VDF application will most likely consist of a single very large program or perhaps several large programs. For deployment, we expect that the installer will provide access to these programs by placing shortcuts to them on the desktop, or, by placing the programs within a desktop group (or shortcut to the group). Alternatively, you may also wish to create your own
custom menu program. A simple menu program is provided with VDF, which is used to run some of our sample applications. The source code for this program is included and it can be easily modified for your needs.

What do I do for a query program?
We suggest that you use Crystal Reports for DataFlex. The Crystal Reports for DataFlex report writer is licensed per user and a Crystal Reports for DataFlex report engine can be distributed with your application. This means that you can create Crystal Reports for DataFlex reports for your customers, and that they can run these reports without having to purchase an add-on product.

If your customer wants to be able to easily create their own ad-hoc (and in some cases quite-sophisticated) reports, they will want to purchase their own copy of Crystal Reports for DataFlex.

Often end users find that report query programs are too complicated to use. In addition, they also have the disadvantage that confidential database information becomes too-easily accessible. Crystal reports for DataFlex has a data-dictionary option, which should be of interest to you as a developer. This data dictionary is unrelated to the VDF DD and is used for a different purpose. Using the Crystal Reports for DataFlex data dictionary, you can set up report rules making it easier for end users to create reports and at the same time ensuring that privileged information remains hidden.

What screen size should I develop for?
There is no single fixed answer for this, other than that you should be aware that is a factor that must be considered when designing your panels. When you design your views, you should have a minimum-resolution screen size in mind. Also, keep in mind that this minimum keeps getting larger. In designing VDF revision 4, we attempted to make all of our programs and examples fit within 640 x 480 resolution. We did not always succeed. We have a few views and panels that do not quite fit. (We can take some consolation in the fact that Microsoft seems to have used the same strategy and they did not always succeed either.) In designing VDF revision 5, we increased our minimum screen size to 800 x 600 resolution, which appears to be the current industry wide standard for minimum resolution.

Often developers will work on higher resolution screens than their end users. This is desirable since it allows the developer to design an end user panel and still have screen space available to manipulate the IDE’s development panels. The IDE contains a several resolution guides that will show you how a
component will fit within other, smaller screen resolutions. You can use this to make sure that your components adhere to your minimum resolution guidelines.

You only need to worry about the screen size of your views and selection lists. The size and location of your main program (which contains all of your views) and the location of all of your views can be controlled individually on each computer. Normally a higher-resolution screen means your user can see more views at one time, or can see parts of other applications while running a VDF program.

When designing your views, take advantage of tab dialogs. This is a great way to present a lot of information in a relatively small space. Finally, keep in mind that a user can remove a program’s status bar and tool bar. This provides a way to provide a little extra space for lower-resolution displays.

**Should I read all of the documentation? What do I need to know?**

We know that nobody really reads documentation anymore and we have adjusted the way technical information is presented to adjust to this reality. Most of the technical VDF documentation is on-line. The command reference, the class reference, and most other technical reference material is available on-line. In fact, this information can be accessed directly from within the IDE. We do not expect you to read all of the information. When you need the information, just look it up. The user guides for the tools (the IDE, Database Builder, DfComp, etc.) are also presented on-line and are available in context sensitive format. Again we do not expect you to read a book about using the IDE; we expect you to call up information, as you need it.

We do provide two important pieces of documentation that we hope you will read from cover to cover. The *Quick Start* manual provides systematic instructions for building a simple visual application. This will primarily show you how to use the IDE. It will also introduce you to Database Builder, the compiler, and a running application. It also demonstrates that the term “simple” refers to creating the application and not the end results.

We know that the first thing that everyone wants to do is load the IDE and build a visual application. In the real world, this is not how a real application should be created. First your data files must be defined and created, your data dictionaries should be created, and finally you get to create some visual views. We created *Quick Start* so you could do what you wanted to do—build a visual application. The next manual, *Visual DataFlex Developing Applications*, shows you how you are supposed to build an application. This manual will provide you with all the basic information you need to develop in
VDF. You will create a more complicated (i.e., more realistic) multi-file, multi-view order entry application. In particular, we present much more information about data dictionaries. In *Quick Start*, we give you the DDs, in *Developing Applications* we make you build your own. In *Developing Applications*, we concentrate on the “why” of program development and less on the “how.” If you work through both of these manuals, you will be ready to produce applications. From this point on, we expect that most of your documentation reading will be on-line.

**How much Windows 95/NT knowledge do I need to run all of this?**

You need very little knowledge of Windows 95 or NT to successfully create VDF programs. Probably the most-important knowledge is to have a good sense of what a Windows program should look and act like. By default, VDF creates very Windows-like applications. We often get requests from developers asking how to customize the application to make it less Windows-like. This is usually a mistake.

One area of Windows you will want to learn more about is the registry. The registry is used by VDF and all other Windows applications to store set-up information. You will probably wish to use this yourself for your own custom set-up needs. The area of the registry used by VDF can be maintained from within a DataFlex program by VDF commands and can be viewed and modified from the workspace selector within Database Builder. In addition, a Windows program named regedit provides access to the entire registry. You probably will want to learn more about the registry and regedit. One word of warning: be careful when you use regedit. You can quite easily damage the registry.

Most of the DataFlex visual objects actually contain Windows objects. The interface to the actual Windows objects can get quite complicated. We hid most of this complexity by “wrapping” these unfriendly Windows objects within a more-friendly DataFlex shell. This means that you do not have to spend any time learning the Windows API (the Windows programming interface). If you choose to learn the Windows API, you can take advantage of it. DataFlex can directly send Windows messages to Windows objects. This is a very powerful, although advanced, feature. This is a bit like understanding FMAC. You did not need to understand it to use DataFlex, but once understood it was very powerful.
Can I go directly from procedural to VDF or should I first convert to character-mode OOP?

If your application is currently written in procedural DataFlex, you will probably find it easiest to jump right into Windows, bypassing the conversion to OOP character mode altogether. You will convert your application using the following steps:

1. Study your existing data files and make sure they will operate properly under a data-dictionary model. Make any required file and relationship changes to support this.
2. Create your data dictionaries. Move as many rules as possible from your existing entry programs into the DDs.
3. Create your entry views and lookup lists.
4. Transfer reports, making changes as required.

The IDE and Database Builder will provide you with most of the support needed to perform Steps 2 and 3. In other words, these tools will generate your code for you. The power of our code-generation tools (the IDE and Database Builder), the power of the data dictionaries, and a clear strategy for migration make VDF the easiest revision of object-oriented DataFlex to upgrade to ever. Moreover, the results are quite impressive.

Do I have to convert my entire application all at one time?

It is very important to understand that a VDF application will use the same database as your existing application. You can always run existing parts of your application in parallel with your new VDF programs. You can plan on running existing programs on your existing legacy hardware (i.e., non-Windows 95 / NT workstations) or you can plan on running your existing applications on your Windows workstations using the console-mode runtime (see next question).

In many cases, legacy applications are just too big to convert at one time. In some cases, you may find that there are parts of your applications that you simply cannot justify the cost of converting at all. For example, you may discover that some of your batch processes and reports will not benefit from a conversion to Windows. In such a case, you may decide to change those programs towards the end of your conversion cycle, or to not change them at all. You have this flexibility.

Application conversion can be a gradual process. One big advantage here is that you can get the most-critical parts of your application converted and
deployed right away. For example, a developer may convert all of the commonly used visual data-entry programs and integrate Crystal Reports for DataFlex into their system. At this point, the majority of the users will be operating in Windows (although there may be hundreds of reports and batch processes still running in character mode).

**How can the console-mode runtime fit into my conversion strategy?**
The console-mode runtime is not really part of VDF. It is a character-mode runtime and is included with DataFlex Revisions 3.1c and above. It allows you to run an existing character-mode application within Windows as a real Win32 application. The application can run in a window or run in full-screen mode. It properly uses the resources of Windows and is consequently much better behaved than running standard DataFlex in a DOS box. Console-mode programs can be run concurrently with a VDF program (or programs) and can be run concurrently with other console-mode programs. This provides you with a great deal of flexibility as you phase out one set of programs and phase in another.

**What advice can you give to the procedural programmer?**

- **The Language is the same**: The visual design tools will guide much of your transition to object-oriented programming. Both the IDE and Database Builder generate source code for you. Most of your custom coding will involve creating and augmenting functions and procedures within objects and classes. For the most part, the functions and procedures will contain the standard DataFlex statements you have been writing for years.

- **Embrace procedures and functions**: In procedural DataFlex, you probably made extensive use of subroutines (gosubs and labels). In OO DataFlex, you will want to use functions and procedures. They are far more powerful and much easier to use. Their main advantages are that they allow you to pass parameters, they allow you to define local variables, and they can return values. A well-written VDF program should contain no subroutines.

- **Avoid global variables**: Whenever possible, avoid the use of global variables. In procedural DataFlex, you had no choice but to use global variables. In OO DataFlex, you should rarely need to use a global. Procedures and functions support local variables and you will find that this removes the need for most global variables.

If you need to create a “variable” that is shared by an object or a group of
objects (an object and all of its children) you should define properties within the object. Between the use of local variables and properties, you will find that the only remaining variables in a program will be those that are truly global—variables that really must be used by the entire program.

- **Avoid the use of Indicators**: Since indicators are global, they should be avoided. You cannot create a local indicator or an indicator property. Instead, you will use integers. If an integer is zero (0), it is false; if it is non-zero, it is true.

**What do I do for Help in my application?**

VDF application help is Windows help. You will create your help in the exact same manner you would to create help for any Windows application. There are numerous Windows help-authoring tools available and we expect you to use one of them. Creation of help files is extensively covered in Microsoft's' Windows documentation and in third-party publications such as *Mastering Windows 95 Help* by Blue Sky Corp., ISBN 0-9647236-0-3.

Once a help file is created, it must be linked to your application. This is a very simple process. The act of creating a help file will generate a list of context IDs. You then assign these context IDs to different areas of your application creating a context-sensitive help link. The degree of context sensitivity is up to you. You can provide a single link for an entire application, a unique link for every object in your application, or anything in between. The IDE, Database Builder, and Database Explorer all present their help using exactly this method.

Your help author does not need to be a DataFlex developer. Anyone who knows how to write Windows-based help (or can learn how) can be used to author your help. The program author will be needed to define the context links in your application. If the help files have been created and a list of context links have been created, this is a very fast process. This gives you the opportunity to allocate your development resources better.