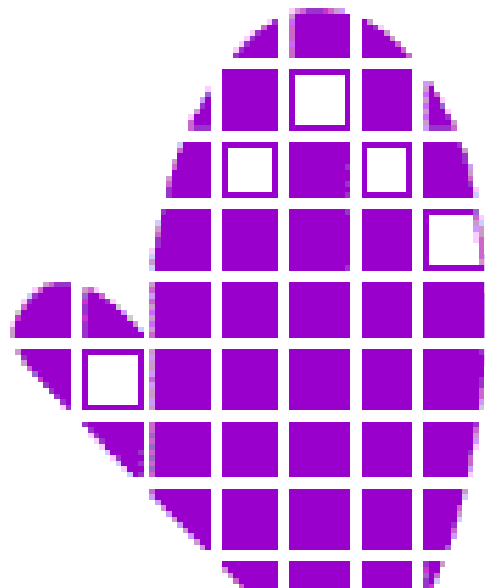


**Mitten Software**

**Proudly presents...**

**The Dictionary  
Enhancement  
Foundations, version 3.0**

For Clarion4 and Clarion5, using ABC



## ***License Agreement and Legal Stuff***

Hi there! Now pay attention. Yeah, I know this is boring. Since you've seen it before, I though I'd play around with it a bit. It's still perfectly legal, so don't let the informal language fool you.

FIRST! If you don't accept and agree to the terms of this agreement, do not use this product. Of course, you don't know yet. You still have to read the terms of the agreement. *Anyway...* if you don't think that you can keep yourself from copying the software, giving it out as Christmas presents, or otherwise stealing it from me, please just erase it off your hard drive now. Put the disks you got with the product under a fridge magnet, and throw them and the documentation in the trash. If you bought the product less than thirty days ago, contact whoever you bought the program from, and get your money back.

The Feds, under United States copyright law, protect this software, the Dictionary Enhancement Foundations. We, Mitten Software, own this software. That's right, it's ours. You, however, have oh so kindly purchased a license to use the software. Have we told you lately that we enjoy your business? Anyway, I digress. This software is licensed to you under these conditions:

1. With the exception of your own backup copies, you can't copy this software. Well, you can, you're just not allowed to. I know your Mom, your Uncle Harry, Aunt Louise, and cousin It all desperately want their own DEF set as birthday presents. Buy them one each. If you make copies for any reason other than your own backup, like I just said, it's a federal no-no, and you will feel guilty for a long time. Trust me. Your conscience will nag at you. You won't be able to eat or sleep. You'll spend your remaining days an empty husk. Is it worth it? I didn't think so.
2. You can give your license to the Dictionary Enhancement Foundations to someone. You can sell your license if you wish. If you do, you don't have your license anymore, so you must destroy the software. Before you give or sell your license to another person, make sure the other person reads this piece of paper and agrees to these terms as well. If they don't agree, don't give it to them! Just say No!
3. If you use this product, any applications you make with it are duty-free. Or is that royalty-free? That's it. You don't owe us a nickel if you sell applications made with these templates. More power to ya! Literally!
4. LIMITED WARRANTY!! (The lawyer made me yell like that.) That's right, this product has one. Now, if you're like me, you're wondering what that means. Well, it's like this. You bought the license, see. We sold it to you, see. It does what we said, see. If you try to get it to do something other than what we say it does, or if it stops working because of strange, other-worldly phenomenon, such as an upgrade to some other software product, or if it just plain quits on

ya, that's tough. OK, if it's within 30 days of when you purchased it, we'll replace it or send your money back. If you're nice and it's just a little past 30 days, maybe. But really, if aliens eat your disk, or a new version of another product breaks ours, or if you didn't understand what it does when you bought it, we'll apologize. Maybe. It was, after all, working when we sold it to you.

5. LIMITED LIABILITY!! (Don't you hate lawyers? Just kidding, David.) Like we said, we're selling you the license to this product. The license comes with its very own As-Is guarantee! That's right, just like Honest Frank down at the local used-car lot, you bought it like it is. Wait. That doesn't sound quite fair. How's this: If you bought it and it doesn't do what you want or you can't get it to work, and you come to this decision within 30 days of purchasing the license, send it back and we'll replace it or send your money back. Of course, don't make that decision on the 29th day and wait for five months to send it back. What we won't do is be liable if your company goes bust because you tried to get this product to work. You can only get from us a replacement of the program, or your money back within the first 30 days. We cannot be held liable for any consequential, incidental, or indirect damages. (Can you tell that a lawyer dictated that last line?) Anyway, what do you think? Will Honest Frank match that? I didn't think so.

Now, you might be asking yourself; "Who's to know?" We will. We know what you're thinking. OK, we don't. You're going to do what you want. Just know this: If you break either the letter or the spirit of this agreement, you lose your license. No upgrades for you, Bucko! Understand?

### ***A Plea for Help***

Every one of you who buys and uses this product is a software developer. You know what happens when people steal your software. You lose, your customers lose, everybody loses.

If you purchased this product, thank you. We enjoy doing business with you.

If you didn't purchase this product, and are using it anyway, well shame on you.

### ***Copyright***

The Dictionary Enhancement Foundations is copyright 1996 - 1998, Mitten Software, Inc. All Rights Reserved.

## Contacting Mitten Software

### E-Mail

[Mitten@MittenSoftware.com](mailto:Mitten@MittenSoftware.com)

### Snail-Mail

Mitten Software

1865 West Wayzata Blvd.

Long Lake, MN 55356

### Voice

800.825.6541

612.745.4941

### Fax

612.745.4944

### Web Site

<http://www.mittensoftware.com>

# What's covered in this mighty tome, and where...

<b>GETTING STARTED WITH THE DICTIONARY ENHANCEMENT FOUNDATIONS (DEF)</b> .....	<b>1</b>
REGISTERING THE DEF TEMPLATES IN CLARION FOR WINDOWS .....	1
NOW THAT THESE TEMPLATE THINGIES ARE INSTALLED, JUST WHAT DO THEY DO? ..	2
CODE GENERATION BASICS - PART 1, THE APPLICATION TEMPLATE .....	3
CODE GENERATION BASICS - PART 2, THE #PROGRAM TEMPLATE .....	4
CODE GENERATION BASICS - PART 3, THE OTHER STUFF .....	4
<b>USING THE DICTIONARY ENHANCEMENT FOUNDATION TEMPLATE SWITCHES</b> .....	<b>5</b>
USING THE DEF DICTIONARY SWITCHES .....	5
<b>SWITCHES FOR FILES</b> .....	<b>6</b>
FILE SWITCHES .....	6
Switch .....	6
Requires .....	6
What it means .....	6
<b>SWITCHES FOR DATA</b> .....	<b>8</b>
NON-FILE RELATED DATA IN THE DATA DICTIONARY .....	8
SWITCH.....	8
REQUIRES .....	8
WHAT IT MEANS .....	8
<b>FILE LOADED QUEUES</b> .....	<b>10</b>
<b>COMPUTED DATA FIELDS, VIRTUAL RECORD EXTENSIONS, AND FILES</b> .....	<b>12</b>
<b>GLOBAL DATA AND THREADS</b> .....	<b>13</b>
<b>GLOBAL DATA AND CONTROL TEMPLATES</b> .....	<b>13</b>
<b>DEF AND INI FILES</b> .....	<b>13</b>
HOW DO I NAME MY INI "FILE"? .....	14
HOW DO I GET THE CONTENTS OF AN INI "FILE" FROM THE DISK? .....	14
<b>DEF, INI FILES, AND VERSION/PRODUCT SPECIFIC INFORMATION</b> .....	<b>14</b>
INI DATA AND NEW PRODUCT VERSIONS .....	15
INI DATA AND NEW PRODUCTS .....	15
<b>SWITCHES FOR EQUATES</b> .....	<b>17</b>
<b>USING EVALUATED FILE NAMES IN DEF</b> .....	<b>18</b>

COMPOUND FILE NAMES, ALIASES, AND "IMAGE" FILES .....	19
<b>SWITCHES IN FIELDS AND EVALUATED VALUES .....</b>	<b>21</b>
<b>A FEW OTHER FIELD SWITCHES .....</b>	<b>27</b>
FIELDS AND FIELD POOLS.....	27
FIELDS, FILES, AND TRIGGERS .....	27
CALCREQ(FIELD).....	28
ALTDEC(XXX).....	29
AUTO .....	30
<b>THE DICTIONARY ENHANCEMENT FOUNDATIONS DICTIONARY SORT UTILITY.....</b>	<b>31</b>
<b>MULTI-APP DEVELOPMENT AND APPLICATION ORGANIZATION .....</b>	<b>33</b>
USING DEF IN SINGLE APPLICATION SYSTEMS.....	34
USING DEF IN MULTI-APPLICATION SYSTEMS .....	34
USING THE DEF TEMPLATES ON AN EXISTING APPLICATION OR SYSTEM.....	35
Converting an existing application to a DSA .....	35
Converting existing multi-app system Add-on applications to DEF Add-on applications .....	36
USING DEF ON A NEW SYSTEM.....	37
Creating a New DSA .....	37
Creating a New Add-on application.....	38
USING DEF WITH OTHER THIRD PARTY PRODUCTS .....	39

## Getting Started with the DEF

# Getting started with the Dictionary Enhancement Foundations (DEF)

Welcome! You're a wonderfully intelligent person, and we appreciate you a whole bunch! After all, you realized what a terrific purchase the Dictionary Enhancement Foundations is, so you bought 'em. You're ready to begin saving time now!

A quick word before we install the Dictionary Enhancement Foundations. DEF comes in two install packages. The first is DEF4.EXE, and the other is DEF5.EXE. These are versions of DEF for Clarion4 and Clarion5, respectively. These versions are ***not*** interchangeable. If you install the incorrect version of DEF, the Clarion Application Generator will fail in a manner that is not to your liking (How do you spell GPF?). Please, please, please make certain that you install the correct version of DEF for your Clarion. The feature set is the same for either version

The installation process does the following:

1. Creates a directory under your Clarion4 or Clarion5 directory, called DEF, that contains the Dictionary Enhancement Foundations.
2. Installs DEFILE.INC and DEFILE.CLW to your \LIBSRC directory.
3. Copies the Dictionary Sorter to your Clarion directory.
4. Modifies the Clarion INI file, adding the Dictionary Sorter to the Clarion menu.

Note: For the dictionary sorter to be properly installed, you must not be using Clarion when you install DEF. When Clarion exits, it totally rewrites its INI files which, in effect, uninstalls the Dictionary Sorter.

And that's it. The rest is up to you. What you have to do now is make the DEF templates visible to Clarion. So, without further ado...

### ***Registering the DEF Templates in Clarion for Windows***

Registering your DEF templates in Clarion for Windows is a pretty simple process. There are only a few steps to follow:

1. Load Clarion, but don't load anything else. No Apps, no Dictionary, No text file, nothing.

## Getting Started with the DEF

2. Select Setup ⇒ Edit Redirection File. The redirection file contains the directory locations that Clarion uses when it looks for stuff. Kinda like a "Super-Path".
3. Look for the line that starts \*.\* and add ; %ROOT%\DEF to the end of the line.
4. Select File ⇒ Save, followed by File ⇒ Close.
5. Select Setup ⇒ Template Registry. The Template Registry window appears.
6. Click on the Register button, and the Template File finder box appears. Navigate this box to the place where your DEF templates live. Select DETChain.TPL, and click on the OK button.
7. Select Setup⇒Application Options. The Application Setup window appears.
8. Click on the Registry Tab.
9. Select DEF from the Default #APPLICATION drop list.
10. Click on the OK button to save the Application Options.
11. And there you have it. The DEF templates are registered and rarin' to go!

### ***Now that these template thingies are installed, just what do they do?***

One very frequently asked question that I get is "What the heck are these templates doing?" That's a good question. Here's a little summary of where we've come from and where we're going.

Clarion is an extremely powerful programming system. Like any system, however, it has some limitations. Third-party template and class sets, like these, are developed to bridge the gaps in the shipping Clarion templates. DEF is intended to fill a few very important gaps in Clarion. Some of these gaps are:

- There is no way to include Global, Module, or Procedure Data in the Data Dictionary. This applies to both GROUPs and QUEUEs.
- You cannot put formulas or lookups in the dictionary. Each lookup must be done on a procedure by procedure basis.
- You cannot declare single record or INI files in the Data Dictionary. Saving configuration information for your program is exclusively hand-code.

# Getting Started with the DEF

- You cannot specify computed file locations in the Data Dictionary, such as CLIP(DataPath) & \PHONES.DAT'. The only way to use computed file names is to give a file a variable name, such as PhonesLocation, and provide the naming code in an EMBED point.
- Multiple Application (DLL and EXE) systems are extremely difficult to set up. Even the most experienced CW developers have problems with this.
- If you manage to get a Multiple Application system working, the Relational Integrity (RI) and Standard Functionality code is generated and compiled in each module. This impacts the developer two ways. First, Generation and Compile time for each Application must include all of the code for each file used. In large dictionaries, this can be quite long indeed. Second, since each App generates its own, you can generate different RI code in each APP.

DEF was written to fill the gaps specified above. The DEF templates also provide the framework for the rest of the DEF system. This is version 1 of the DEF system, which was itself built upon version 2 of the Dictionary Enhancement Toolkit for CW20. Features added in this version include the ability to put field pools and triggers in the dictionary. The control templates that were part of the Dictionary Enhancement Toolkit have been removed from DEF. Very few developers were using them, and coding them properly in C4 with ABC was quite difficult.

The DEF templates change the way that code is generated by providing a substitute Application and Program template.

## ***Code Generation Basics - Part 1, The APPLICATION Template***

Code generation of a Clarion application is controlled by the *#APPLICATION* template. In the CW templates, this application is called the CW Default Application. An application template is responsible for...

- Prompting the user (you) for all of an application's global properties
- Declaring all of the symbols used globally by the templates.
- Controlling generation of all of the source code your application needs. This includes the source code that you designed in the Application Generator, as well as any supporting source code.
- Controlling generation of support files, such as Export files (necessary to link your program into executables or DLLs) and SHIP lists.

# Getting Started with the DEF

- Ensuring (either directly or indirectly) that the project system has all of the information necessary to make your program into an EXE or DLL.

The DEF application template is an alternate to the CW application template. It's important to note that using the DEF templates you will never lose functionality. All of the EMBED points provided in the Clarion templates are also in the DEF templates. The DEF application template simply generates the code necessary to provide the functionality provided in the rest of the templates.

## ***Code Generation Basics - Part 2, The #PROGRAM Template***

When the time comes, the application template issues a `#GENERATE(%Program)` command. This instructs the Application Generator to generate the global module of your program using the *PROGRAM* template. The program template that shipped with Clarion for Windows is called, appropriately, CLARION. In the same spirit of naming, the DEF program template is called DEF. Because the DEF template significantly changes your program's code generation, it was necessary to provide a replacement program template.

As with the DEF application template, you will never lose functionality using the DEF program template.

## ***Code Generation Basics - Part 3, The other stuff***

Now, I'm sure you understand there's more to it than just providing application and program templates. The DEF templates generate classes that are either derived from the ABC base classes, or that are entirely new. It's important to remember that none of the Clarion shipping templates or base classes have been touched by DEF.

# Using the Dictionary Enhancement Foundation template switches

Ready to get to work? In this section I show you how to use the major boon part of the Dictionary Enhancement Foundations, the **COOL DICTIONARY STUFF!** How cool can a dictionary be? Way cool. Not just a little cool. Way cool!

**The key to using the cool dictionary stuff in DEF is located on the "Options" tab of a dictionary objects properties window. See that big block called User Options? That's where you make the DEF settings. So what are the settings? I'll address these in two separate sections below. First, though, a quick explanation on using DEF switches.**

### *Using the DEF dictionary switches*

As I said just a few lines ago, the User Options block (located on the Options tab of a dictionary component's properties page) is where you will put all of the DEF switches that let you declare global data, QUEUES, calculations, INI files, any of these things. The switches themselves are cumulative. That is, you can include more than one at a time.

For example, let's say that you have a file declaration that you want to designate as an INI file, which would be the INI switch. Looking at the table, you can see that the INI setting requires the DATA setting as well. Therefore, to set up an INI file in the dictionary, you would define the file, go to the File Properties window, click on the Options tab, and enter DATA,INI in the User Options box.

It is very important to separate the switches with a comma, and **only a comma**. No white space!!!! If you don't, well, let's say that DATAINI doesn't mean a cotton-pickin' thing to the DEF templates. Funnily enough, DATA, INI doesn't mean a thing either. The only combination that does is DATA,INI.

It is not, however, important that the switches go in any particular order. DATA,INI and INI,DATA mean the same thing to the DEF templates.

## Switches for Files

Well, now that you know how to use switches (you did read the previous page, didn't you?) it's time to talk about the switches that affect the file declarations in your system.

With DEF, the structures that you define in the dictionary can be files, or they can be something different. What can they be? Well, they can be groups, queues, equates, or types. Even the files are a bit different, with single record config files and INI files provided automatically.

When we talk about "Switches for Files", we're talking about switches that can be used in the User Options box of a structure defined as a file in the data dictionary. These switches are available if and only if the switch DATA *is not* found in the File's User Options box.

### File switches

<u>Switch</u>	<u>Requires</u>	<u>What it means</u>
CONFIG	None	<p>The file is written as a single record configuration file. The single record is read when the program is started via the method FileName::ConfigOpen. The record is written back using the method FileName::ConfigClose.</p> <p>Under normal circumstances, during the ConfigOpen procedure the file is opened. If the file doesn't exist, it is created. Then the record is read. If the record doesn't exist, it is created with the default values.</p>
CLOSEAFTERREAD	CONFIG	<p>The config record is closed after config values have been read. When it's time to write config values back to the config record, the file is reopened and the record written.</p>

## Switches for Files

<u>Switch</u>	<u>Requires</u>	<u>What it means</u>
READONLY	CONFIG	The config record is read from the disk but never written back
NAME(Value)	None	Generates the NAME attribute for the file with a variable (File:FileName) used as the name attribute. The value declared as the parameter is used as-is to compute the value for the name attribute variable at run-time.  NOTE: When the NAME attribute is used, the DEF templates ignore what you've entered into the Full Pathname field of the File's properties.
FIRST	None	Generate the file's declaration on the first declaration pass. Files with this attribute will be generated after the first data declaration pass.
LAST	None	Generate the file's declaration on the last declaration pass. Files with this attribute will be generated after all other files and data have been declared.
NOGEN	None	Suppresses generation of the structure. The structure is not generated in any application under any circumstance.
ExternalDLL(Library)	None	Generates the file structure as EXTERNAL in all locations <i>including</i> the DSA. This lets you use file structures that are native to other libraries. The single parameter contains the name of the library that corresponds to the DLL in which the file is declared.

## Switches for Data

As with "Switches for Files", Switches for Data are switches that can be used in the User Options box of a structure defined as a file in the data dictionary. These switches are available if and only if the switch DATA *is* found in the File's User Options box.

### ***Non-File related data in the Data Dictionary***

With the DEF templates, you can code non-file related data, such as GROUPs, QUEUEs, and EQUATEs in the data dictionary. It's easy as pie. The following switches are available for use with data in the data dictionary:

<b><i>Switch</i></b>	<b><i>Requirements</i></b>	<b><i>What it means</i></b>
DATA	Nothing	The file's structure will be generated as data. If the QUEUE switch is not used. The structure will be generated as a GROUP.
INI	DATA NO QUEUE	The contents of the GROUP are read from and saved to an INI file upon application load and exit, respectively. The procedures FileName::INIGet and FileName::INIPut are used to, well, get and put the INI information for the file. <b>NOTE: THIS SWITCH CANNOT BE USED IN CONJUNCTION WITH THE QUEUE SWITCH!</b>
OVER(xxx)	DATA NO QUEUE	The GROUP is declared with the OVER(xxx) attribute. Of course, xxx would never be the parameter of the OVER attribute. Instead, the structure that the GROUP is over is used.
VIRTUAL(File)	DATA NO QUEUE	The GROUP is generated as normal. The big difference is that the group functions as a virtual record extension. That is, whenever something happens to the file, the virtual record extension is affected as well. This is important when using evaluated fields, which is

## Switches for Data

<i>Switch</i>	<i>Requires</i>	<i>What it means</i>
		discussed in a later section of this document.
QUEUE	DATA	Instead of generating a GROUP, the structure is generated as a QUEUE.
FROM(File)	QUEUE	The queue pre-loaded from the file before the first procedure is called. When a record from the file is updated, the appropriate update is made to the QUEUE. More on this after the table.
FIELD	DATA	The first field in the structure is the actual declaration, rather than the file structure itself. The File's label is ignored and the first field's label is used instead. This switch was added to allow DEF to work with products that have FIELD prompts to select QUEUES, like the CPCS report templates.
TYPE	DATA	The structure is generated with a TYPE parameter. This lets you use the structure to type other structures passed as parameters. Is this too structured?
FIRST	DATA	Generate the declarations on the first declaration pass. Data with this attribute will be generated before any files are generated.
LAST	DATA	Generate the declarations on the last declaration pass. Data with this attribute will be generated before files with the LAST attribute are generated.
MODULE	DATA NO INI	Generates the GROUP's or QUEUE's declaration in the MODULE in which the structure is used. The structure is not declared globally, nor is it declared

## Switches for Data

<i>Switch</i>	<i>Requires</i>	<i>What it means</i>
		in the DSA.
PROC	DATA NO INI	Generates the GROUP's or QUEUE's declaration in the procedure in which the structure is used. The structure is not declared globally, nor is it declared in the DSA.
NOEXTERNAL	DATA NO INI	Generates the GROUP's or QUEUE's declaration without the EXTERNAL flag. In other words, the data has a unique buffer in each application in which the data is used. The data is not declared in the DSA.
NOGEN	DATA	Suppresses generation of the structure. The structure is not generated in any application under any circumstances.
ExternalDLL(Library)	DATA NO INI	Generates the data structure as external in all locations including the DSA. This lets you use data structures that are native to other libraries. The single parameter contains the name of the library that corresponds to the DLL in which the data is declared.

## File Loaded Queues

One of the new features of DEF with version 2.0 is File Loaded Queues. I know, Huh? Here's what File Loaded Queues are...

They're queues loaded with files.

Easy, right? No? OK. Let me try again. Many times I have a set of files that are used in drop-lists or for lookups, but rarely, if ever, for edit. These files are usually called lookup or dictionary files. You've used these files before. We all have.

The normal way to use these files is with the Clarion FileDrop or FileDropCombo control template. These templates load an image of the file into a queue every

## Switches for Data

time a procedure is entered. If you use the new button navigation tool, you can load the same queues from the file dozens of times in a matter of minutes. Not exactly speedy.

Another problem with the FileDrop and FileDropCombo control templates is that they really don't support primary keys well. You can get them to support primary keys, just not well. The process involves declaring a temporary display variable, and doing a whole bunch of other stuff.

With DEF's File Loaded Queues, you can declare a QUEUE the way you want the information displayed. The queue is filled when your program is loaded, then you can use the file in any normal list control, as is. Don't worry about the QUEUE being synchronized with your data, because as your user adds, changes, or deletes records, the QUEUE is kept up to date. No muss, no fuss.

Setting up a File Loaded Queue is a piece of cake. To set up a File Loaded Queue, follow these steps:

1. Highlight the file for which you want to make a File Loaded Queue.
2. Copy the file to the clipboard.
3. Paste the file back. I change the name by adding :Queue to the end, and I add a Q to the end of the prefix. You can use whatever convention turns you on.
4. Highlight your new "file", and click on the Properties button.
5. Select the Options tab.
6. ADD **DATA,QUEUE,FROM(File)** to the User Options box. "File" is the label of the imaged file.
7. Save the file properties window by clicking on OK.

The rest is optional.

8. Click on the Fields/Keys button.
9. In the Fields window, remove any fields that you don't want to use. You should always, by the way, delete all MEMOs and BLOBs.
10. Reorder the fields any way you see fit.
11. Add any evaluated fields that you might want to use for each record of the QUEUE. Refer to the "Switches for Fields" chapter for more on this.
12. When done, save the file's fields by clicking on the OK button.

## Switches for Data

13. Save the file definition.

A File-Loaded QUEUE creates an instance of a DEFQueueManager. This is a similar class to the DEFFileManager, with one difference. Yep, it manages a QUEUE.

As said above, File Loaded Queues are loaded on entry to the program. The templates provided keep the queue current within the templates themselves. If, however, records are changed to the file outside the template's purview, the queue will be out of date.

In short, there is no multi-user concurrency built in to the File Loaded Drop system. You can, as one Beta Tester has, build in your own simple messaging system with a data and time last modified flag in a control record for each file. When the flag changes, reload the queue. If, however, you are writing a single user system, and your disk writes are all taken care of by the templates, you won't need to do anything special when using File Loaded Queues.

## Computed Data Fields, Virtual Record Extensions, and Files

As we discuss in a few pages, you can now put computations and other cool stuff in the dictionary for fields. Computations in a file's fields are made when a record is written. Computations in normal data are made when a field from the data is displayed. If, however, you want to link computed fields in data (as opposed to files) to a record of a file, you need to make a "virtual" record extension. This virtual record extension will have values computed for every instance of the parent record. This is extremely useful for browses and reports.

For example, let's say that you have a customer file, with a first, middle, and last name field. Every time that record is read, you want a field called combined name that contains a complete name. This field can then be populated into browses. Every time a record from the customer file is read, the combined name in the virtual record extension of the customer record is computed. You populate the field once, and every time a new customer is encountered, the field is computed with new values.

To make a DATA "File" a Virtual Record Extension, you need to do two things:

1. Name the "File" with the name of the file being extended, followed by ":Virtual". For example, if you want an extension to the Customer file, create a file called Customer:Virtual.
2. In the File User Options block of the Virtual Record "File", put the keyword DATA.

## Switches for Data

That's it. The :Virtual addition to the name tells the templates what the file is. The templates take over from there.

## Global Data and Threads

I get asked often if global data is threaded. Actually, that depends on you. If when you define the data in the dictionary you check the Open in Current Thread checkbox, then the group or queue is threaded.

**NOTE:** INI setting groups are **NEVER** threaded.

## Global Data and Control Templates

You cannot use a file with the DATA flag as the primary or secondary file for a template, ever. If you do, the templates might try to do something silly like try to write a record to a GROUP. See, that even sounds silly.

So, the question comes up: "**How do I make the fields in one of my data "files" visible to a procedure?**" Simple. Add the data "file" to the Files button of the procedure's properties window in the Other Files section. If you attach the file to a control template as a primary, code generation will fail with an error message. This is a bad thing.

## DEF and INI Files

One of the nicest features of DEF is the ability to specify INI files in the dictionary. INI Files, for those of you who don't know, are basically a Windows standard "config" text file. By default, INI files are located in the Windows directory. Though you can put your INI files anywhere, **I recommend letting INI files go to the Windows directory.** This way Windows can always find your INI file.

If you put your INI file in another directory, you're defeating half of the purpose of INI files, which is providing a place for your program to go for configuration data that lets your program be run from anywhere.. The other half of the purpose, which is to provide a structure independent configuration source, is still maintained wherever you put your INI file.

The information in the INI file is segmented into sections. If you look at an INI file, you can see that section names are surrounded by square brackets. Take a look at the following INI excerpt.

```
[UserInformation]
DEF:UserName=Tom Moseley
```

## Switches for Data

```
DEF:UserCompany=Mitten Software
```

```
[CurrentSettings]
```

```
DEF:LastDictionaryProcessed=
```

```
DEF:DictionaryFile=C:\CLARION5\DEF\EXAMPLES\FIELDTST.DCT
```

In this case, there are two sections. The first is called UserInformation; the second, CurrentSettings.

By default, all information in a DEF INI file goes into a section called Setup. You can set up sections in your INI "File" through the use of GROUPs. A "first level" group (not inside another group) will create a section name that is the label of the group (as shown above). A group within the group will create a section file the label of the first level group, followed by a colon, followed by the next level group label..

### ***How do I name my INI "file"?***

You name your INI file like you would name any file. If you specify a file name, that name is used. If you don't, the first eight characters of the file label are used, and a .INI tagged onto the end. Remember, by default INI files are saved in the WINDOWS directory.

**NOTE:** You cannot use compound file names for INI groups.

### ***How do I get the contents of an INI "file" from the disk?***

In most cases you don't have to. If you find that, for some reason you want to save or re-retrieve data from the INI, you can use the Access:INIFilename.FillGroup or Access:INIFilename.DumpGroup methods. Of course, replace the word INIFilename with the label of the INI "file" as specified in the Data Dictionary.

## **DEF, INI Files, and Version/Product Specific Information**

As we discussed just a bit ago, you can separate your INI files into sections. This feature, in conjunction with the fact that INI files in DEF are really just groups, makes it easy to put information specific to each version of your product, or to information specific to different products, in the same INI file.

## Switches for Data

### ***INI data and new product versions***

If you want to hold information specific to different version of a product in the INI file, keep the information in a version GROUP. With each new version of the product, rename the GROUP to the new version.

For example, let's say your product is called Ducky. You create an INI file, DUCKY.INI, to keep configuration information. In the INI file, you create a group called "UserInformation", and another group called "DuckyVersion100". UserInformation contains the data concerning who owns the license to Ducky. DuckyVersion100 contains the information specific to version 1.0. After 1.0 goes gold and ships, you begin working on version 1.1 of Ducky. Since users could, at first, be using Ducky 1.0 and 1.1 in conjunction, or because they may not like Ducky 1.1, you want to keep the 1.0 configuration information separate from the 1.1 info. You go into the dictionary, and rename the DuckyVersion100 group to DuckyVersion110.

Does this mean you have to change your programs? Nope, Ducky.INI works as a group. No conversions or nothing. The difference is how Ducky.INI stores the information on disk. After running Ducky 1.0 and 1.1 on the system, your INI file will look something like...

```
[UserInformation]
DUK:UserName=Another User
```

```
[DuckyVersion100]
DUK:VersionSpecificInformation=Goes Here for version 1.00
```

```
[DuckyVersion110]
DUK:VersionSpecificInformation=Goes Here for version 1.10
```

When your user runs Ducky 1.0, the DuckyVersion100 section is read. The DuckyVersion110 section is read when Ducky 1.1 is used. The user just thinks that you're the bees knees, you like DEF more, tell your friends, and we all get rich!

### ***INI data and new products***

Now that Ducky is in version 1.1, you start working on the next product in the Ducky system, the communication package called Quackers. Do you need a separate INI file for Quackers? No!

In the new Quackers dictionary, create an INI file called Ducky.INI, just like you had before. In fact, since you still need the user information, you add a group called UserInformation just like in the Ducky dictionary. Instead of creating a DuckyVersion100 group, though, you create a QuackersVersion100 group. This

## Switches for Data

group contains the information for version 1.0 of Quackers. After running Quackers version 1.0, Ducky.INI looks like...

```
[UserInformation]
DUK:UserName=Another User
```

```
[DuckyVersion100]
DUK:VersionSpecificInformation=Goes Here for version 1.00
```

```
[DuckyVersion110]
DUK:VersionSpecificInformation=Goes Here for version 1.10
```

```
[QuackersVersion100]
DUK:CommPort=Com1
```

### Switches for Equates

The last switch you can use in a File's User Options box is EQUATES. This causes the contents of the file's declaration to be declared as a series of equates. When this happens:

- The prefix of the "File" is ignored. Only the ID of the field, without the prefix, is generated in your code.
- The declaration of equates goes before the MAP in global code.
- The "Initial Value" field of each field's properties window is used to supply the EQUATED value.

## Evaluated File Names and DEF

### Using Evaluated File Names in DEF

With DEF, you can specify an evaluated name for any file in your dictionary. By compound name, I mean a name such as **CLIP(INI:DataPath) & 'PHONES.DAT'**. This name is evaluated at run-time.

With the DEF templates, compound file names are computed immediately before the file is opened. What's more, variables used in the compound file name can be variables used in an INI group or in a config file. There are two ways to use compound file names in the data dictionary:

1. If you want to use a compound file name in a File's "Full Pathname" prompt of the file's properties window, you **MUST** precede the name with a double exclamation mark. For example, **!!CLIP(INI:DataPath) & 'PHONES.DAT'** tells the DEF template that the file has a name that is computed as **CLIP(INI:Datapath) & 'PHONES.TPS'**. If you use only a single exclamation mark or none at all, your program won't work!
2. You can use the NAME(value) switch in the File User Options section. For example, in the example above you'd add the switch **NAME(CLIP(INI:DataPath) & 'PHONES.DAT)** in the Phones file's User Options box.

The second option is useful if you use Clarion for Windows' file browser from the dictionary. The browser, you see, can't understand the double exclamation mark file names, so it elegantly dies when you try to browse a file with this name type.

In fact, over time a really useful tip has come out several times. When you want to use variable filenames to compute a file's location, put the computation in the NAME switch in the file's user options block. Hard-code the usual location into the Full Pathname field of the File's properties window. What does this do? Well, when you try to browse the file, the browser reads the Full Pathname field to access the file. When DEF makes the app, the pathname specified in the NAME attribute is used. You get the best of both worlds.

#### Example: Using Compound File Names and INI files to easily customize data locations

One of the most common uses for INI files is the storage of a data path for files. In Clarion for Windows, to be honest, this is a chore that makes even the most experienced programmers weak at the knees. Not so with DEF. If you want to use an INI file to store a data path, it's as easy as 1-2-3(-4)...

1. Create a DATA,INI group (in the dictionary), called for example, INIFile.
2. Inside this group, declare a variable called DataPath.

## Evaluated File Names and DEF

3. Code all of the File Names in your dictionary with a path similar to **!!CLIP(INI:DataPath) & 'PHONES.DAT'**. Now, the double exclamation mark might have caught your eye. As discussed above, the double exclamation mark (!! ) tells the DEF templates to compute this file name at run-time.
4. If the INI:DataPath variable doesn't have a value when the program starts, present your user with a "Where do you want the data" File Dialog, which fills the INI:DataPath variable.
5. Nothing. There is no step 5.

That's all there is to it! When the file's are opened, the names are computed automatically. When the program ends, the data path is saved automatically, and retrieved when the program starts. What could be easier?

### ***Compound File Names, Aliases, and "Image" files***

As you know if you've ever used an alias, you must have a file name, a Full Pathname, defined for an alias to work. The file name can be a literal, a variable, or a compound file name. Under normal circumstances, the alias will have the same name as the aliased file, as must be for the alias concept to work.

The situation came up a while ago where a user wanted the ability to specify an image file. That is, a file that is an exact image of another. In this example, the user wanted to create an archive file and move records verbatim to the archive file. The archive file needed to be an exact image. He couldn't cut and paste, because AppGen would lose its links. Maintenance was a headache. He asked for help.

DEF solves this very simply. If you want to create an image file, follow these steps:

1. Create an alias for the imaged file. You might need to provide a full pathname for the aliased file. If you don't have any special name, just put in the name that the file driver will use. In other words, the first eight characters of the file label, with appropriate extension.
2. Highlight the alias file, and click on the Properties button.
3. Go to the Options tab.
4. Enter the new name using the NAME() file switch. For example, if you want the file to be called FileImag.TPS, put NAME('FileImag.TPS') in the file user options block. Note that this text will be evaluated directly in code, so literal text must be enclosed in single quotes.

## Evaluated File Names and DEF

5. That's it. There is no number 5. Save the file and dictionary, and you're good to go.

# Switches in Fields and Evaluated Values

When using Clarion, if you want to have a "Full Name" available for every name of a customer file, you need to code the full name computation every place you want it displayed. On a Browse, make the computation. In a FileDrop list, make the computation. On a report, make the computation. Another report... You get the idea.

Well, let's think about this. The computation is really something related to the current record of the file, isn't it? The computation is truly part of the data definition of the file, though not part of the file's structure. Why, then, can't you write the computation in the dictionary where it belongs, rather than writing it over and over in your app?

And this doesn't even address the headaches when you want to change the computation!

Lucky for you, DEF comes to the rescue. You can now specify an evaluated value for any field in the data dictionary. What's an evaluated value? Simple. **A field can have a value that is computed at run-time, where the evaluation is specified in the data dictionary!** These are not simple computations such as X times Y, but lookups, conditions, and totals as well.

NOTE: You can only use DEF Field switches in global structures. That is, you can only use them with fields in files, or fields in DATA structures that do **NOT** have the MODULE or PROC switches.

Why is this? Well, it's because the computations are made in global procedures, called CalcField:Field (where Field is the label of the field calculated) . If the data is module or procedure data, the calc procedures won't know what the heck you're talking about when you use your procedure declared field, since the field hasn't been declared yet. The compiler will give you messages that say, to paraphrase, "Huh?", and you'll call me to say "Huh?" and I'll say "Read the documentation".

Before we get into some of the nuances, let's talk about the switches you use. All of these switches go in the User Options box of the field's properties window.

When I describe these switches, I need to tell you about the parameters the switches are using. If a parameter is optional, I'll put it in square brackets "[ ]". If you can have multiple sets of an optional parameter, I'll use a double set "[ [ ] ]" of square brackets. If you have a choice of a set of literal parameters, I'll put those in curly braces. **Don't put these symbols in your parameter section. DO NOT PUT ANY OF MY INCLUDED TEXT INTO THE ACTUAL PARAMETERS.** For example, I might say that a parameter is ELSE EXPRESSION. This means that

## Switches for Fields

the ELSE portion of an IF or CASE goes here. **DO NOT PUT ELSE IN THE PARAMETERS THEMSELVES!** Got it? Good. On with the switches. First switch:

CALC(Expression)

Written as:

Field = Expression

Example: CALC(ITE:Quantity\*ITE:UnitPrice) for the field ITE:ItemTotal would generate...

ITE:ItemTotal = ITE:Quantity\*ITE:UnitPrice

## Switches for Fields

```
IF(condition,TRUE expression, [[ELSIF condition, ELSIF Expression,]] {ELSE Expression})
```

Written as:

```
IF condition
  Field = TRUE Expression
[[ELSIF Condition
  Field = ELSIF Expression]]
[[ELSIF Condition
  Field = ELSIF Expression]]
ELSE
  Field = ELSE Expression
END
```

Example: IF(CUS:Gender = 'Male','Dear Sir',CUS:Gender = 'Female','Dear Madam','To Whom it may concern') for the field CUS:Greeting would generate...

```
IF CUS:Gender = 'Male'
  CUS:Greeting = 'Dear Sir'
ELSIF CUS:Gender = 'Female'
  CUS:Greeting = 'Dear Madam'
ELSE
  CUS:Greeting = 'To whom it may concern'
END
```

## Switches for Fields

CASE(Evaluation, [[Value, Expression,]] {ELSE Expression})

Written as:

CASE Evaluation

[[OF Value  
Field = Expression]]

[[OF Value  
Field = Expression]]

ELSE

ELSE Expression

END

Example: CASE(CUS:Gender,'Male','Dear Sir','Female','Dear Madam','To Whom it may concern') for the field CUS:Greeting would generate...

CASE CUS:Gender

OF 'Male'

CUS:Greeting = 'Dear Sir'

OF 'Female'

CUS:Greeting = 'Dear Madam'

ELSE

CUS:Greeting = 'To whom it may concern'

END

## Switches for Fields

LOOKUP(Assignment, Key,[[Linking Fields]])

Written as:

```
CHECKOPEN (Looked-up File)
Key Fields = Linking Fields
GET (Looked-up File, Lookup Key)
IF ERRORCODE ( )
  CLEAR (Field)
ELSE
  Field = Assignment
END
CLOSE(Looked-up File)
```

Example: LOOKUP(HON:Gender,HON:Primary,CUS:HonorificID) for the field CSV:Gender would generate...

```
CHECKOPEN (Honorific)
HON:ID = CUS:HonorificID
GET (Honorific,HON:Primary )
IF ERRORCODE ( )
  CLEAR (CSV:Gender)
ELSE
  CSV:Gender = HON:Gender
END
CLOSE(Honorific)
```

NOTE: Do not use Lookup calculations in a system that uses a Supplemental DSA. The files needed may not be visible.

## Switches for Fields

TOTAL ((COUNT|SUM|AVERAGE),Parent File, Totalled Field, [Condition])

Written as:

CHECKOPEN files

Totalled key fields = Parent key fields

Clear total accumulators

SET(Linking Key)

LOOP

  NEXT(Totalled File)

  IF ERRORCODE() THEN BREAK.

  IF Totalled key fields <> Parent key fields THEN BREAK.

  [IF Condition THEN CYCLE.]

  Add Totalled Field to Total accumulators

END

Resolve Totals

Close files

Example: TOTAL(AVERAGE,Invoice,ITE:ItemTotal) for the field  
INV:InvoiceTotal would generate...

CHECKOPEN (Item)

CHECKOPEN (Invoice)

ITE:InvoiceID = INV:ID

INV:InvoiceTotal:Total\$ = 0

INV:InvoiceTotal:Total# = 0

SET(ITE:Foreign:Invoice,ITE:Foreign:Invoice)

LOOP

  NEXT(Invoice)

  IF ERRORCODE() THEN BREAK.

  IF ITE:InvoiceID <> INV:ID THEN BREAK.

  INV:InvoiceTotal:Total\$ += ITE:ItemTotal

  INV:InvoiceTotal:Total# += 1

END

IF INV:InvoiceTotal:Total# = 0

  INV:InvoiceTotal = 0

ELSE

  INV:InvoiceTotal = INV:InvoiceTotal:Total\$ / INV:InvoiceTotal:Total#

END

CLOSE(Item)

CLOSE(Invoice)

NOTE: Do not use Lookup calculations in a system that uses a Supplemental  
DSA. The files needed may not be visible.

NOTE: If ITE:ItemTotal was computed, as in the first example, then before the  
total was accumulated a call to the procedure CalcField:ITE:ItemTotal would  
have been made to gather to correct value. Therefore, you can have a running  
total of a value not directly in the file. Cool beans, huh?

Dictionary Enhancement Foundation Documentation

Page 26

### Fields and Field Pools

One of the features new to DEF customers is the addition of field pools. A field pool is declared a file, with FIELDPOOL added in the file's User Options box. This tells DEF not to declare the file at all. Instead, it turns this file into a repository of field formatting information.

To have a field from your dictionary use a field pool field as a reference, you simply put POOL(*field pool field*) in the field's user options box. The picture (PICTURE), justification (JUST), justification offset (OFFSET), case (CASE), type mode (TYPEMODE), Help ID (HELPID), tooltip (TOOLTIP), message (MESSAGE) and list box heading (HEADING) will be derived from the field pool field's definition.

If you don't want to use all of the attributes of the field pool field, you can INCLUDE or EXCLUDE whichever ones you like. For example, if you have a field pool field called FLD:Date, and you want a field to use only the picture from the field pool field, you would put, in the User Options box of the field derived from FLD:Date...

```
POOL(FLD:Date,INCLUDE,PICTURE)
```

To use the picture, justification, and offset, you would use...

```
POOL(FLD:Date,INCLUDE,PICTURE,JUST,OFFSET)
```

Exclude works the same way, except all of the attributes are included if not in the EXCLUDE list.

### Fields, Files, and Triggers

The other thing new to DEF from the Dictionary Enhancement Toolkit is triggers. Triggers are basically code that is executed when certain events take place. The triggers that DEF supports are:

- PREADD, POSTADD, PREPUT, POSTPUT, PREDEL, POSTDEL – These triggers are available on files or fields. The single parameter of these is the code that you want executed when the activity takes place. For example, PREADD code takes place before a record is added.

```
PREADD(Case GLO:UpdateStatus
  OF 1
    FLD:Status = 'DONE'
  OF 2
    FLD:Status ="READY'
  END)
```

## Switches for Fields

- **ONSELECTED, ONACCEPTED** – These triggers are available on fields only. They are triggered when the **SELECTED** or **ACCEPTED** event is triggered for a field that uses the field. The single parameter of these is the code that you want executed when the activity takes place.
- **VALIDATE** – This trigger is available for fields or files. This trigger is activated whenever disk activity is taking place for the file.

The first parameter is the evaluation to take place for validation. An evaluated value of **FALSE (0)** indicates that the record passes validation. A non-zero value indicates an error condition. The validation parameter is, basically, the right side of an **IF** statement.

The second parameter is the text displayed in an error message if the validation fails.

There are five optional parameters for the **Heading**, **Icon**, **Button List**, **Active Button**, and the **button**, if any, that signals that the record can be written with validation failing.

### ***Trigger Procedures and Functions***

The trigger code entered in the dictionary can be straight Clarion code. Additionally, it can contain procedures and functions written in the Dictionary Support Application, which is where all of the trigger code is contained.

## A Few Other Field Switches

There are a few other switches that you can place in the Field User Options box.

### ***CALCREQ(Field)***

The first switch is **CALCREQ(Field,[[Field,]])**. Basically, this tells the templates to do other calculations first. For example, let's say that you have a lookup to get a customer's gender from the honorific file, and you want to use an **IF** on the gender to create a salutation. Well, you could do the lookup on **Honorific** with...

```
LOOKUP(HON:Gender,HON:Primary,CUS:HonorificID)
```

...in the **CSV:Gender** field. Then, you could use...

```
CASE(CSV:Gender,'Male','Dear Sir','Female','Dear Madam','To Whom it may concern')
```

## Switches for Fields

...in the CSV:Greeting field. Now, you can't always count on CSV:Gender being computed first. To force this, you would add...

```
CALCREQ(CSV:Gender)
```

...to the Field User Options of the CSV:Greeting field, so now it would read...

```
CASE(CSV:Gender,'Male','Dear Sir','Female','Dear Madam','To Whom it may concern'),CALCREQ(CSV:Gender)
```

...and before the CASE statement was processed, the CSV:Gender lookup would take place.

The CALCREQ switch, in conjunction with all of the other DEF Field switches, give you almost unlimited flexibility in the depth of information you can keep in your dictionary.

### **ALTDEC(xxx)**

You can tell the templates to ignore how you type a field from a file or data declaration, and use your own custom declaration. That's right, you can supply your own alternate declaration for a field. How, you say? By adding the ALTDEC() switch to the FIELD's User Options box. The value supplied as the parameter for the ALTDEC switch will be used instead of the field's parameters as defined in AppGen. You can use PRE(), you can use LIKE(), you can use any valid declaration you like.

For example:

```
ALTDEC(LIKE(ABC:Field))
```

Would generate the field's declaration as

```
MyField      LIKE(ABC:Field)
```

You might be asking yourself why this would be a nice switch? Well, because a user asked for it. But really, I can see the use of the switch. The important usage is LIKE(), where you can have a single group declaration and LIKE another group to the first.

AltDec removes the dictionary's ability to maintain law and order. If you use ALTDEC over a group, for example, the dictionary doesn't know this, and still generates an END for the group. Make certain that you keep track of what you're AltDec'ing. I suggest never using AltDec on a group.

## Switches for Fields

Beware when using this switch! AppGen doesn't know a thing about these settings. You must make certain that you never populate one of these fields, or AppGen will certainly blow a fuse. You have been warned!

### ***AUTO***

One feature that is available in the Clarion "data" field declaration window that isn't available in the dictionary is the ability to specify that a data (not file related) field is not automatically initialized, using the AUTO flag. (Doesn't it seem like that's backwards? If you don't want the field automatically initialized, maybe MANUAL would have been better. Oh well.)

How do you do this in DEF? Simply add the AUTO flag to a FIELD's User Options settings (when the field is in a DATA structure)...

## The Dictionary Enhancement Foundations Dictionary Sort Utility

How often have you wanted to change the order of the files in your data dictionary? I've needed to do this quite a lot, since I get some large dictionaries. But still, it'd be nice if you could get even a small five or six file dictionary to display in alphabetical order.

The DEF Dictionary Sorter is a solution to this problem. The dictionary sorter is a stand-alone program that you'll find has attached itself to the Clarion for Windows main menu. Click on the DEF Sorter item on your menu, and the sorter appears.

After the start-up screen, the sorter will ask you for a dictionary. You can select either a TXD or DCT file. Whichever file type you select, the original file will be overwritten by the Dictionary Sorter.

After the splash screen dismounts, you will be given a file dialog selection window. Select the dictionary you want sorted, and click on the OK button. A processing window will appear as the dictionary is being read. When that's finished, you will see a window with two list boxes on it. The left list box contains the files and data defined in your system. The right list box contains the aliases defined. The reason that these are separate is that the Clarion for Windows Data Dictionary separates files and aliases.

The files list box contains three columns:

- The file type. If the file is really a file, the driver is displayed. If it is a DEF data file, the data type is displayed.
- The file prefix.
- The file label.

You can sort your files quickly by clicking on one of the column headers. You can also use drag and drop to move files around.

# The DEF Dictionary Sorter

The alias list box contains two columns:

- The alias prefix.
- The alias label.

As with the files list box, you can sort your aliases quickly by clicking on one of the column headers. You can also use drag and drop to move aliases around.

There are three buttons on the Dictionary Sorter toolbar...

- Open Dictionary - This puts away the current dictionary and lets you select a new dictionary to open.
- Close Dictionary - This button just puts away the current dictionary. You are not prompted to "save changes".
- Save Dictionary - This lets you save the changes to the selected dictionary and keep editing or exit.

When you're finished with the dictionary sorter, exit as you would any windows program.

You must be in Clarion for Windows when using the Dictionary Sorter and you must not be in AppGen or the Data Dictionary. You will get odd behavior. Your dictionary will be fine, the program will have a hard time because of the modal aspects of the Data Dictionary.

## Multi-App Development and Application Organization

In case you haven't realized it yet, the way your applications are organized when you use the DEF templates is a little different than when using the Clarion for Windows templates. This difference is the reason that DEF is so easy to use and so powerful. The two kinds of applications that DEF uses are:

- **Dictionary Support Application.** Also called the "DSA". Every DEF system has one of these. For small systems, this may be the only application you have. The DSA is an application which:
  - Is instructed to act as a Dictionary Support Application - You must set the Application Type prompt of the DEF tab of the Global Properties window (whew!) to "Dictionary Support". This is the default setting.
  - Is generated into a DLL in a multi-app system - You have to tell AppGen to create a DLL. Do this by selecting Application ⇒ Properties and changing the Destination Type field to DLL.
  - Contains all of the standard code - Every Add-on module will call the standard functions from the DSA. This is done automatically by DEF.
  - Contains all dictionary based non-file declarations. That is, contains all of the GROUP, QUEUE, and INI declarations.
  - Contains all or some file declarations.
  - If the DSA is creating an EXE, only the files used in the DSA are generated, otherwise all file declaration will be generated in the DSA.
  - Contains all of the RI code for all of the files declared in the application. Every Add-on module will call the RI functions for these files from the DSA. This is done automatically by DEF.
  - You can include any of your own "Shared" procedures and functions in the application. If you EXPORT these (same as using the Clarion ABC templates), then they will be visible to all of the Add-on modules.

Before you make any Add-on applications you MUST first create the Dictionary Support Application. If you don't, you cannot make these applications, as they refer to variables declared in the DSA!

# Multi-App Development

**Add-on Application.** Also called simply "Add-on". These are the "multi" part of a multi-app system. These applications hold the procedures whose user interface and code you write. An Add-on application is one which:

- Is instructed to act as a Add-on application - You must set the Application Type prompt of the DEF tab of the Global Properties window to "Add-on"
- Is told the name of the DSA through the "Support DLL's LIB" prompt on the same page. Just enter the name of the DSA, with a LIB extension.
- Contains user (you) defined procedures to be made into a Clarion application system

## ***Using DEF in Single Application Systems***

There are times you'll want to use the DEF templates in a single application system - times when you just want to make a small app, but you want the ability to put global, module, and local data in the dictionary. I do this all the time, and you might as well.

Using the DEF templates for single-app development is easier than using the Clarion templates, because you have the DEF features. If you want to use the Clarion for Windows Data button, go ahead! Everything you did when using the Clarion for Windows template can be done with the DEF templates. You just get extra stuff. No penalty, only reward!

## ***Using DEF in Multi-Application Systems***

As discussed earlier, multi-app development using the stock Clarion templates can be a bear. There are a lot of little things to keep track of, and even the most experienced of us gets things wrong. I've spent hours trying to track down a single missed switch. That's why I wrote DEF in the first place.

In its simplest form, a DEF multi-app system consists of a DSA and one or more Add-on apps. The DSA contains all of the file and global declarations. The RI and Clarion for Windows standard functions are also included in the DSA. Each Add-on application generates the file and global data declarations only for the files it uses. The files and data are declared as EXTERNAL though, since the file's true declaration is in the DSA. The RI and standard procedures are also referred to as EXTERNAL.

What does this mean to you? Well, your apps will all generate and compile faster, since they don't need to generate the RI and standard stuff. Additionally, your apps will be smaller, and your system footprint should be smaller.

# Multi-App Development

On the down side, though, the DSA contains the declarations for all of the files and global data in your dictionary. Why? Because the templates have no way of knowing which files you need where. Remember, though, that one DSA can support any number of systems. If you have ten systems out there, they can, maybe even should, all use the same DSA.

## ***Using the DEF templates on an existing application or system.***

We all have existing applications. If we didn't, we wouldn't know that something like DEF was a good idea. The question is how do you have your existing applications use the DEF templates?

There are actually three steps to making your apps work with the DEF set. First, you have register the templates. You should have done this in Chapter 1. If you haven't, please do so now.

Step 2 is to create your DSA.

- If you have a single application program, your DSA is the application itself.
- If you have a multiple-app system, then things are a bit different. Clarion for Windows Multi-App systems need you to create a central application. This app is sometimes called a "Files" or "Repository" or "Master" application. This application is what is referred to as a DSA in the DEF templates.

## Converting an existing application to a DSA

If you have an existing file application, the conversion of this app to a DSA is a piece of cake. Just follow these steps:

1. Load the Application into AppGen.
2. Select Application ⇒ Properties. The Application Properties window appears.
3. Select DEF from the Application Template drop-list.
4. Click on the OK button.
5. Click on the Module tab of the application tree.
6. Highlight the first module. This is the program module.
7. Click on the Properties button. The Program Properties window appears.
8. Enter DEF in the Type prompt.

# Multi-App Development

If you forget to do this, the consequences are dire. OK, not *dire*, but they will make you think that the templates are messed up. If you forget this step, you'll get oodles of generation errors, like "%UpdateRelationPrimary not defined", that sort of thing.

9. Click on the OK button.

Your application is now set up to use the DEF templates. Now to check the settings of your DSA.

10. Click on the Global button. The Global Properties window appears, with the DEF tab on top.

11. Make certain that the Application Type prompt says "Dictionary Support".

12. Click on the OK button to save the settings.

13. Make your application.

And that's it. Your app is now a genuine DSA. If you are converting a single application system, you're done! If you're converting a multi-app system, well, you need to follow the next set of instructions for every application in the system.

## Converting existing multi-app system Add-on applications to DEF Add-on applications

1. Load the App into AppGen.

2. Select Application ⇒ Properties. The Application Properties window appears.

3. Select DEF from the Application Template drop-list.

4. Click on the OK button.

5. Click on the Module tab of the application tree.

6. Highlight the first module. This is the program module.

7. Click on the Properties button. The Program Properties window appears.

8. Enter DEF in the Type prompt.

I'll repeat myself. If you forget to do this, the consequences are severe. OK, not *severe*, but they will make you think that the templates aren't working. If you forget this step, you'll get bunches of generation errors, like "%DeleteRelationPrimary not defined", that sort of thing.

# Multi-App Development

9. Click on the OK button.

Your application is now set up to use the DEF templates. Now to check the settings of your DSA.

10. Click on the Global button. The Global Properties window appears, with the DEF tab on top.

11. Make certain that the Application Type prompt says "Add-on".

12. Enter the name of your DSA, with an extension of .LIB in the "Support DLL's LIB".

This is the library file that contains the addresses of everything in the DSA. If you can't remember the name of your DSA, or you just want to look it up, click on the ... button next to the field. You will be given a window to locate the file. It will be located in your CLARION5\OBJ directory.

13. Click on the OK button to save the settings.

14. Make your application.

Now do this for the next app in your system, until you've converted them all.

That's it! You've converted your existing Clarion for Windows application to DEF. That wasn't so bad. By the way, did you notice how much faster the make went in step 14, after you converted to DEF? Cool, huh?

## ***Using DEF on a new system***

You won't, of course, only use DEF on existing systems. Eventually, you'll want to make new systems with it as well. Of course, the first thing you need to do is create your Dictionary Support Application (DSA).

### Creating a New DSA

1. Tell AppGen you want a new application. Go to step 2 when you have the Application Properties window up.
2. Ensure that DEF is the entry in the Application Template drop-list.
3. If you are making a single app system, set the Destination Type to Executable (.EXE). If you are making a multi-app system, set the Destination Type to Dynamic Link Library (.DLL)
4. Click on the OK button.

# Multi-App Development

5. Click on the Module tab of the application tree.
6. Highlight the first module. This is the program module. Make certain that the program module says "Dictionary Enhancement Foundations Program" next to the module name. If it doesn't, follow steps 7 through 9 below. If it does, go directly to step 10.
7. Click on the Properties button. The Program Properties window appears.
8. Enter DEF in the Type prompt.
9. Click on the OK button.

Your application is now set up to use the DEF templates. Now to check the settings of your DSA.

10. Click on the Global button. The Global Properties window appears, with the DEF tab on top.
11. Make certain that the Application Type prompt says "Dictionary Support". This is the default setting, so it should already be this way.
12. Click on the OK button to save the settings.
13. Make your application.

You are now the proud parent of a brand new baby DSA. If you are converting a single application system, you're done! If you're converting a multi-app system, well, you need to follow the next set of instructions for every application in the system.

## Creating a New Add-on application

Now that you've created a DSA, it's time to get on to creating the working apps, the Add-ons. To create an Add-on app:

1. Tell AppGen you want a new application. Go to step 2 when you have the Application Properties window up.
2. Ensure that DEF is the entry in the Application Template drop-list.
3. If the app you're making now will be an executable, set the Destination Type to Executable (.EXE). If you're making an app that contains library procedures, set the Destination Type to Dynamic Link Library (.DLL)
4. Click on the OK button.
5. Click on the Module tab of the application tree.

## Multi-App Development

6. Highlight the first module. This is the program module. Make certain that the program module says "Dictionary Enhancement Foundations Program" next to the module name. If it doesn't, follow steps 7 through 9 below. If it does, go directly to step 9.
7. Click on the Properties button. The Program Properties window appears.
8. Enter DEF in the Type prompt.
9. Click on the OK button.

Your application is now set up to use the DEF templates. Now it's time to check the settings of your Add-on app.

10. Click on the Global button. The Global Properties window appears, with the DEF tab on top.
11. Make certain that the Application Type prompt says "Add-on".
12. Tell the Add-on app where the DSA is by entering the DSA's name (with a LIB extension) in the "Support DLL's LIB" prompt. If you need to find it, use the ... button.
13. Click on the OK button to save the settings.
14. Make your application.

Congratulations, your Add-on app is done!

### ***Using DEF with other third party products***

One area of concern when writing any third-party template set is the compatibility with other third-party template sets. Each third-party writer has different styles, which makes some conflicts inevitable. That is why I'm proud to say that as of this writing we have none of these conflicts. Yep, we've nailed 'em all down.

There are some considerations, however. Some third party template sets, PowerBrowse by ToolCraft for example, declare DLLs themselves. These templates have code that senses if the app is, in effect, a DSA, and codes the library access accordingly.

These templates almost always use global extensions to add support to a multi-app system. If you have any third-party templates that have global extensions, you must add the global extensions to the Dictionary Support Application as well as to every Add-on application that uses the templates. This will let the templates handle DLL creation correctly.