

Clarion

Getting Started

COPYRIGHT SoftVelocity Inc. All rights reserved.

This publication is protected by copyright and all rights are reserved by SoftVelocity Inc. It may not, in whole or part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine-readable form without prior consent, in writing, from SoftVelocity Inc.

This publication supports Clarion. It is possible that it may contain technical or typographical errors. SoftVelocity Incorporated provides this publication “as is,” without warranty of any kind, either expressed or implied.

SoftVelocity Inc.

www.softvelocity.com

Trademark Acknowledgements:

SoftVelocity is a trademark of SoftVelocity Incorporated.

Clarion™ is a trademark of SoftVelocity Incorporated.

Microsoft®, Windows®, and Visual Basic® are registered trademarks of Microsoft Corporation.

All other products and company names are trademarks of their respective owners.

Printed in the United States of America (0609)

Table of Contents

Table of Contents	iii
Getting Started	1
Getting Started Lessons	3
Exercise One - Create the Dictionary	5
Exercise Two - Import Your Data	8
Exercise Three - Relate Your Tables	11
Exercise Four - Add a Lookup File	13
Exercise Five - Create the Lookup Relationship	19
Exercise Six - Set the Lookup Validity Check	21
Exercise Seven - Fine Tuning the Customer and Orders Tables	22
Exercise Eight - Generate your Program using the Wizards	25
Summary	31
Clarion Programming Concepts	33
What's Next?	41
Index	43

Getting Started

Introduction

Clarion is an infinitely adaptable environment, written to offer fast solutions at every skill level: from the business owner to the enterprise development team. Whatever your level coming in, Clarion will help you take control of your company data—more cost-effectively, and up to ten times faster than any other product out there today. Here's why...

Advanced code generator

Simply point to an existing database, or define a new one, and then use Clarion wizards to generate a full-featured business database application with advanced user interface functionality. The generated code includes complex features like multi-table joins, user authentication and access control, as well as functionality like filtering and sorting and reporting on any combination of database tables and records.

When you need a "business" application to maintain a database, you can literally do the job in minutes using Clarion. The key is the database dictionary. If the Application Generator knows what files or tables you want in the application, and how they're related, it can build an application. This is true regardless of where the files originated or in what format they are. So all you need to do is select one or more files, then indicate (when there are two or more files) the type of file relationships. The short Getting Started lessons herein demonstrate just how easy this whole process is.

The Application Wizard can then create a full-featured application using the Clarion default application metaphor. We call this the browse-form metaphor, and it extends directly from the structure of your database. The application works like this: (1) The end user navigates the database—all or part of it, one data file or multiple related data files—by scrolling through a list box, within which each item represents one record. The window in which this takes place is called a "browse." (2) The end user selects a specific record in the list to perform an action, such as editing the data. This generally occurs in a separate window, in which the database fields appear in separate edit boxes. This is called an update "form." A form may also accept new data. (3) Optionally, the end user can look up a value from a related table during form entry. This opens another browse in a separate window. The end user can select an item, closing the new window and placing the value in the edit box on the form, in one step. This is called a "lookup."

That's the most general description. In addition, each browse window, navigable from the toolbar, opens on a separate thread with its own record buffer—which provides safer data handling. You can select among multiple key orders by CLICKING a tab. DOUBLE-CLICKING a record in the list opens an update form, with automatic concurrency checking (support for multi-user situations), and optional Referential Integrity constraint support (maintaining your database relationships).

Anybody can do this. It just starts with picking a data file from a list.

It's a visual development environment

With Clarion, dropping a control in a window gives you a lot more than other Rapid Application Development tools, which typically let you add a user interface control, but then expect you to write the code to implement its associated functionality. With Clarion, you add a template, which contains the control, and all its required data elements and executable code. That means you don't have to write code—one CLICK places a complete business solution: a user interface control, and the code that enables it to do its job. Moreover, each template has its own user interface. When you view the properties for the template, you'll see an "Actions" tab. By checking a box, choosing a dropdown list item, or filling in an edit box, you can customize the behavior of

the template so that it meets your needs exactly. You'll set "actions" for the templates at many places in the upcoming lessons.. The lessons that follow in this document introduce you to all of the Clarion RAD tools.

When you use the template interface to specify these behaviors, the Application Generator writes the code (Clarion language source code) that implements the behavior for you—and the code it writes for you is object oriented, built from Clarion's Application Builder Class (ABC) Library, so the code is very compact and efficient. Using the templates, you can do an awful lot of custom programming without writing a single line of source code by hand.

The breadth and scope of the Clarion language can seem imposing at first. The Language Reference is several hundred pages. One of the great advantages of having the template system plus the ability to write code by hand is that you can ease into the language—as slowly as, as quickly as, as much as, or as little as you wish. If you don't like the way the templates solve a particular problem—you can use it at first, just to have something to do the job. Later, you can use the template interface, to modify it a little more to your liking. Finally, when you know the Clarion language, you can write a solution yourself, and have complete freedom to do it your way (Note: you can also buy third-party templates to solve problems yet another way).

Power users, who may not normally write programs, can easily do this.

It's a complete programming language

At the language level, you can quickly code an application using Clarion's fourth generation programming language. Clarion has a high level of abstraction, so that it's very "readable," and a compact database grammar, so that you can easily manipulate data with standard language functions like ADD, GET, PUT and DELETE. You don't have to know Clarion to create an application... but if you do know how it works, it helps you understand what your applications are doing, and that helps you make better applications. One lesson has you write a small amount of your own Clarion code into the template-generated source, and another lesson introduces the Clarion language at a fully hand-coded level.

Professional developers will really appreciate the Clarion language. It was designed from the ground up for business programmers. Yet for its relatively quick learning curve (as a high-level 4GL) you'll get blazing performance. The SoftVelocity compiler technology turns all of your code—whether you wrote it or the Application Generator wrote it for you—into highly optimized machine code.

Note:

No matter which level you intend to work at, you're going to work a lot smarter if you first work through all of the lessons all the way to the end.

Getting Started Lessons

Clarion is a data-centric application development tool. Businesses rely on data. That means Clarion is optimized to create business programs. Clarion's data-centric approach begins with the Data Dictionary.

In Clarion, you can create a Data Dictionary using the Dictionary Editor and import your desired business data. You can then create a working application based on that data with the help of the Application Wizard—all with no coding required, and in less than 10 minutes.

We will demonstrate this, and more, in this lesson!

In this section:

- You'll use the **Import Tables Wizard** in the Dictionary Editor to import a Customer and Orders table.
- You'll define the relationship between two tables in the Dictionary Editor, complete with Referential Integrity rules.
- You'll use Clarion's **Application Wizard** to create a complete relational database application from the same data dictionary.
- Finally, you'll use the Application Generator to add more functionality to the application using one of Clarion's Procedure Wizards.

This should *all* take about thirty minutes—without any "coding" on your part. By the end of this section, you'll have a complete application for a database containing three related tables.

Welcome! Let's get started!

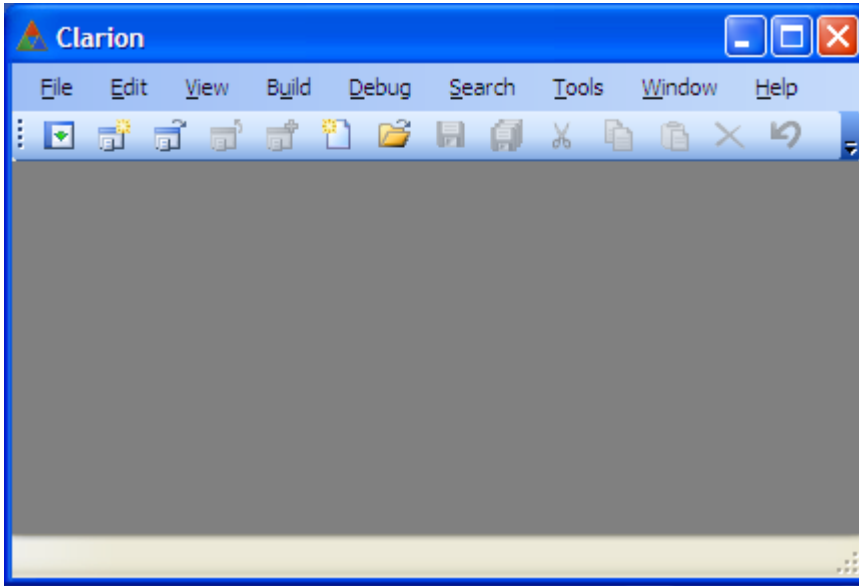
Note:

Make sure that the Clarion IDE is loaded (started). Close any windows that may be opened at this time. You can leave this help file opened, and task switch back and forth with the IDE windows.

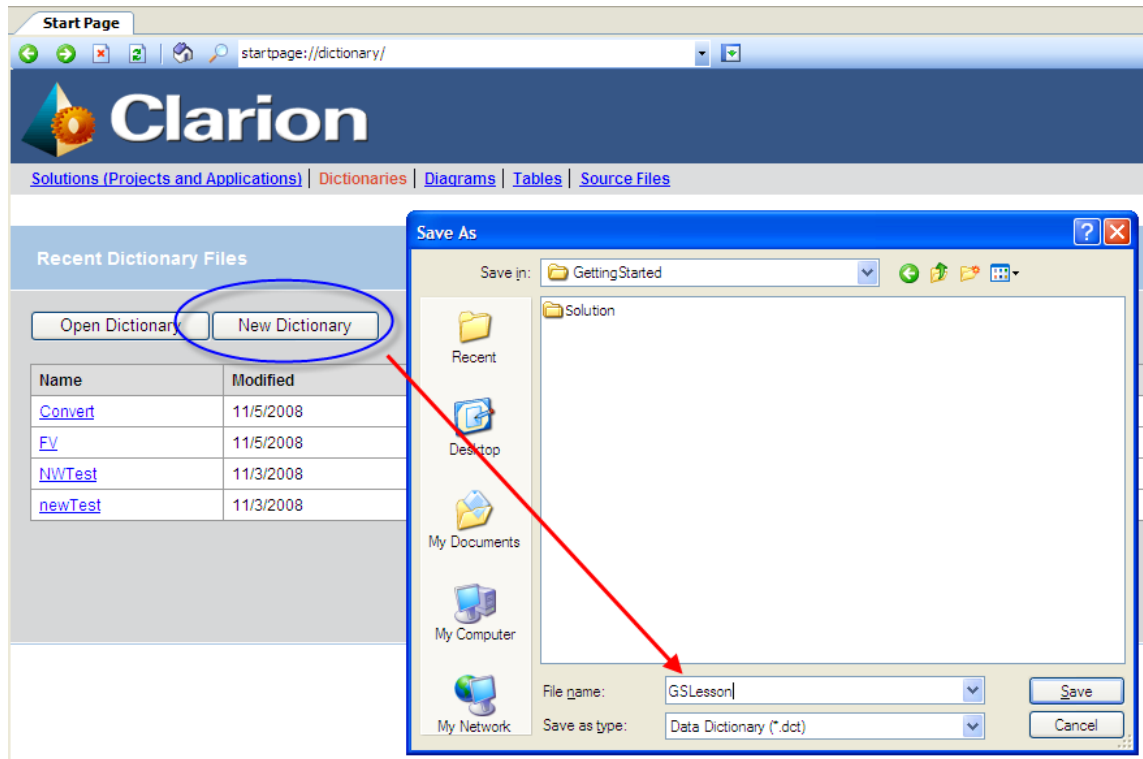
Exercise One - Create the Dictionary

Starting Point:

If not already opened, start Clarion from the install's program menu. If it is your first time, you may be required to enter your serial number. You should have the Clarion development environment open and the Start Page closed. You may optionally close all other IDE pads at this time, we will open them as needed throughout this lesson.



1. From the IDE Menu, choose **View ► Show Start Page** and click on the **Dictionaries** section.
2. Press the **New Dictionary** button, and the *Save As* dialog window appears:



3. Use the **Save in** drop list to navigate to the *Lessons\GettingStarted* folder found in your Shared Documents folder.

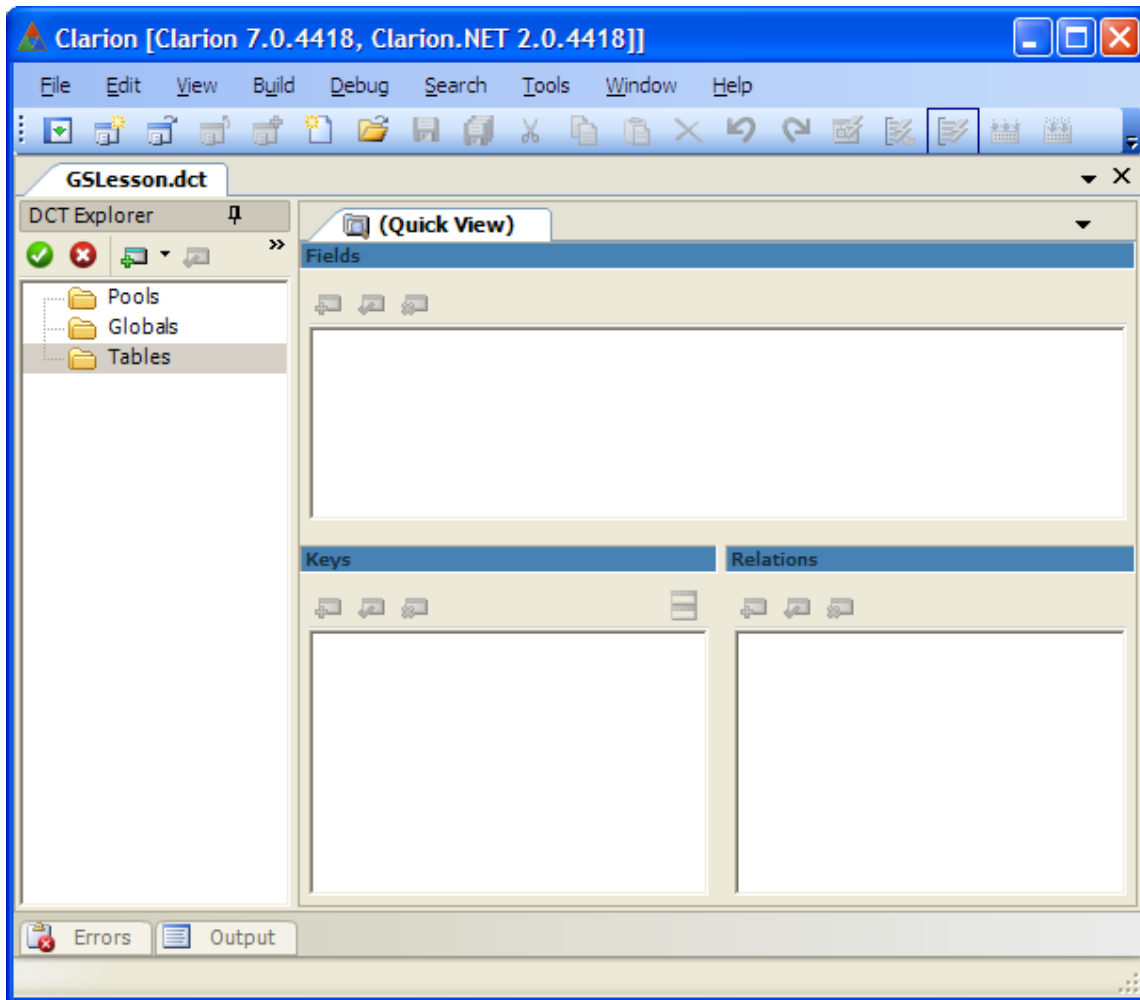
For example, in Windows XP, the Lessons folder is found in:

C:\Documents and Settings\All Users\Documents\SoftVelocity\Clarion7\Lessons

In Vista:

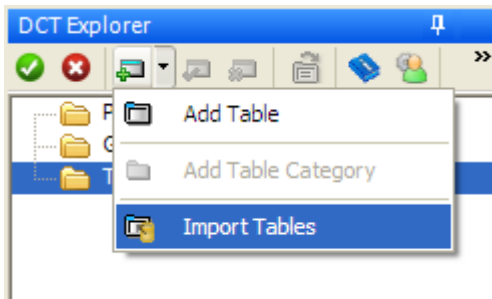
C:\Users\Public\Documents\SoftVelocity\Clarion7\Lessons

In the **File name** entry, type *GSLesson.DCT*. Press the **Save** button to load the Dictionary Editor. The Dictionary Editor's main window should now appear.



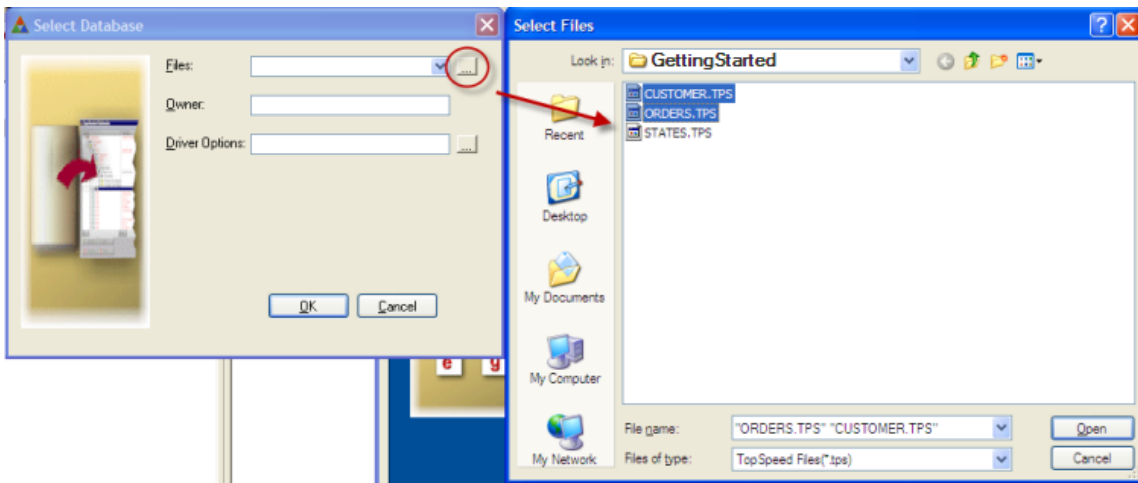
Exercise Two - Import Your Data

1. If you haven't done so already, maximize your dictionary window. It's not much to look at right now, so let's import some data definitions. In the *GettingStarted* folder, we have placed some sample data files. In this exercise, we will import a set of *customers*, and their *orders*.
2. From the DCT Explorer's toolbar menu, choose the **Import Tables** option as shown here:



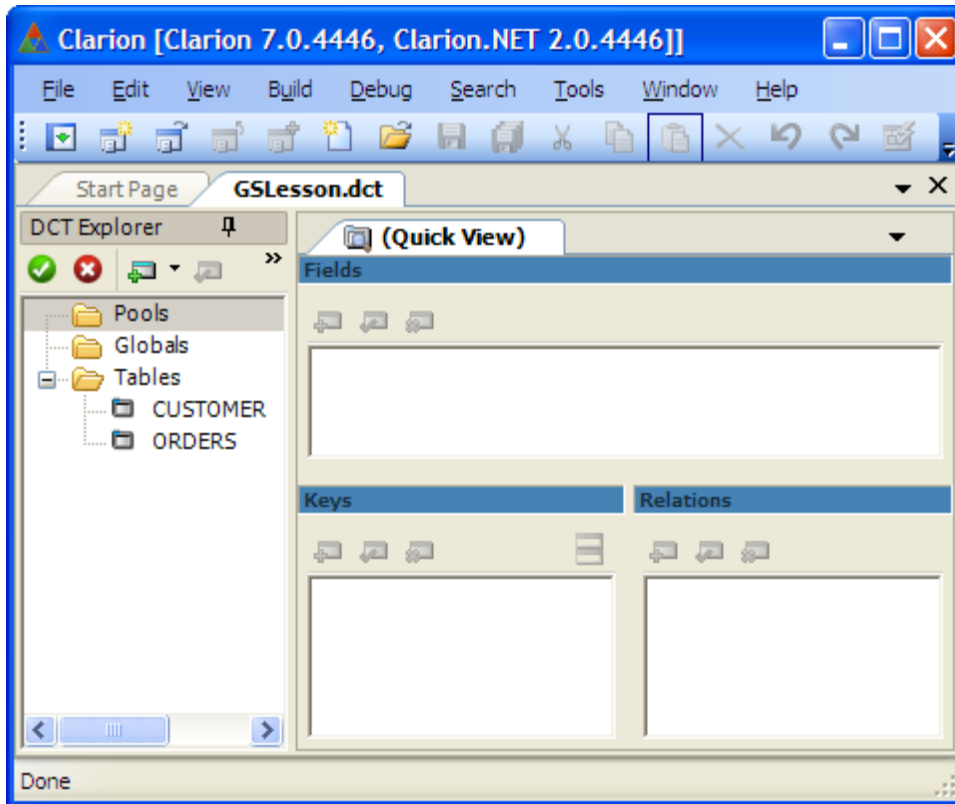
The *Table Import Wizard* window appears.

3. In the **Select Server** prompt, select *TopSpeed Files* from the drop list (It's at the bottom of the drop list).
4. Select the **Select Dictionary** prompt, and press the **ellipsis** button (you can also press the **Next** button at the bottom). The *Select Database* window appears.



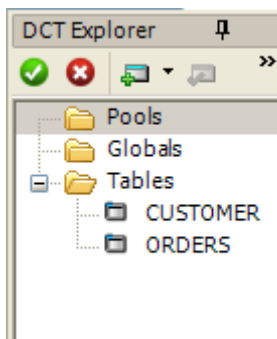
5. Press the ellipsis button to the right of the **Files** prompt.
6. Another *Select Files* window appears, but this time it is a file dialog where you can select a file name. Navigate to the \Lessons\GettingStarted folder, and holding your CTRL key down, highlight the CUSTOMER.TPS file (The TPS is the TopSpeed default file extension) and the ORDERS.TPS and press the **Open** button.
7. We are now back at the original *Select Database* window. Press the **OK** button to begin to import the files.

8. The DCT Explorer displays your imported tables.



That's the basics for a file import. Of course, there are many other options and techniques that you can use to import data definitions, but we are only getting started.

To review, the dictionary will be used with the application generator to help match and locate the data that we are planning to process.




At this time, your dictionary is auto-saved after the import, so let's proceed to Exercise Three.

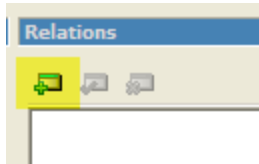
Exercise Three - Relate Your Tables

Obviously, we want the Orders table to contain the pertinent information of the Customer. This means there must be a relationship between the two tables. In this case, one Customer can have many Orders, making this a "One to Many" relationship. To define this relationship, we must link the tables together in the data dictionary to provide the Application Generator with the information necessary to access the related records.

If you first need to review the basics of database design, click [here](#), and when ready, continue through this exercise.

Set the Relationship for the two tables:

1. Examine the Main Dictionary window. You should notice in the bottom right corner the *Relations List*. Highlight *Customer* in the *DCT Explorer* list box, then press the **Add Relation** button  located directly above the *Relation* list box. This button is the only button enabled on the relation pane (right hand side).



The *Relationship Properties* dialog appears. This is where you define the relationship.

2. Make sure the **Type** drop list is set to *1:MANY*.
3. In the **Primary Key** drop list, press the down-arrow key to display the choices, highlight *KeyCustNumber*, and then press the TAB key.

This is the key on the Customer table (the One side of the relationship) that will be used to link the two tables.

4. In the **Related Table** drop list, press the down-arrow to display the choices, highlight *Orders*, and then press the TAB key.
5. In the **Foreign Key** drop list, press the down-arrow to display the choices, highlight *KeyCustNumber*, then press tab.

This is the key on the Orders table (the Many side of the relationship) that will be used to link the two tables.

Next, the linking columns in the keys must be "mapped" so the Application Generator can know exactly which columns in the two tables are related to each other. Since we used identical column names, this is easy.

6. Press the **Map By Name** button.

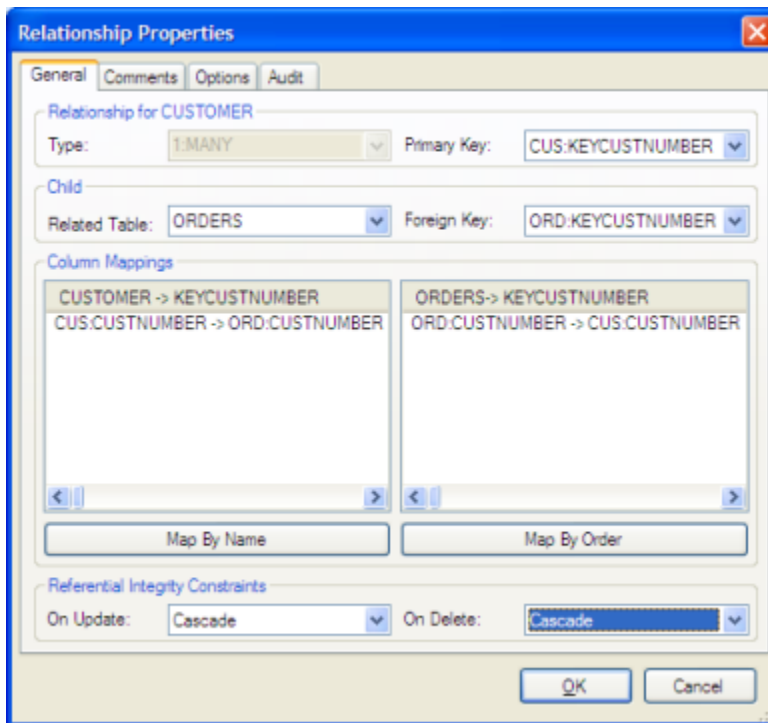
The linking columns between the two tables appear in the two Column Mapping list boxes. This is one reason to name linking columns the same.

Set Up the Referential Integrity Constraints

1. In the Referential Integrity Constraints group box, choose *Cascade* from the **On Update** dropdown list.
2. In the Referential Integrity Constraints group box, choose *Restrict* from the **On Delete** dropdown list.

The generated source code will automatically maintain referential integrity between the tables.


3. Press **OK** to close the *Relationship Properties* dialog.
4. Another *Relationship Properties* dialog appears, allowing you to add a second relationship. Press the **Cancel** button to close this window.

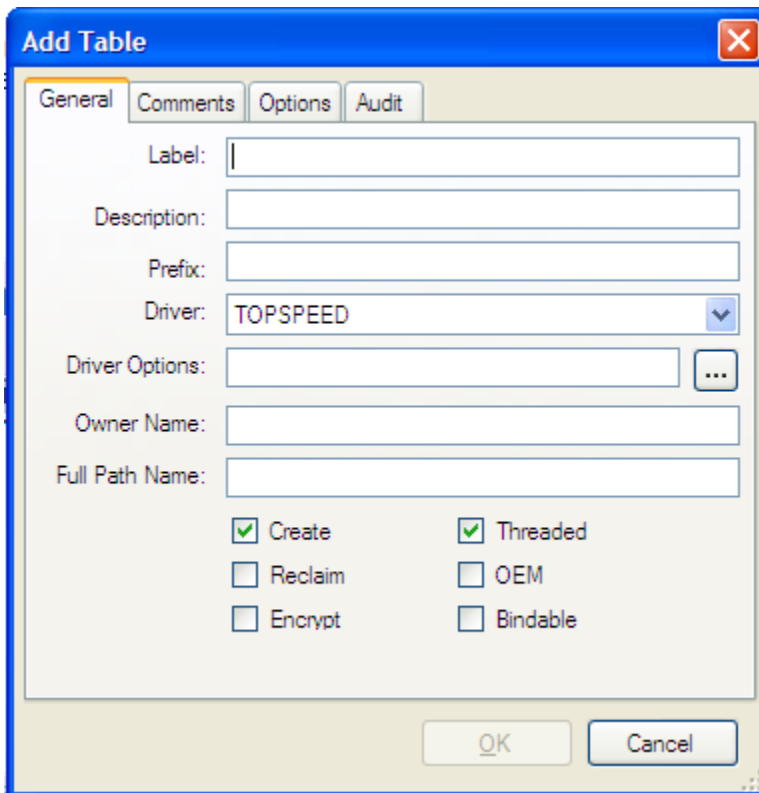


In the next lesson, let's create a new dictionary file from scratch, and examine the Dictionary Editor in more detail.

Exercise Four - Add a Lookup File

In this lesson, we will now look at "fine tuning" the dictionary in the Dictionary Editor. This will directly affect how our application will be generated in Lesson Eight.

1. Highlight the *Customer* table and examine the **Fields** list. You should notice a *State* column. In the next sequence of steps, we will create a lookup table to allow a user to select from a list of valid state abbreviations. After that, we will "notify" the state column that a lookup is active.
2. In the DCT Explorer, highlight the Tables entry, and press the **Add Table**  button, located in the DCT Explorer toolbar.




Add Table


General Comments Options Audit

Label:

Description:

Prefix:

Driver: TOPSPEED 

Driver Options: 

Owner Name:

Full Path Name:

☒ Create ☒ Threaded


☐ Reclaim ☐ OEM

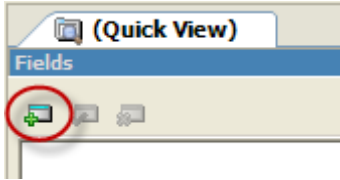
☐ Encrypt ☐ Bindable

OK Cancel

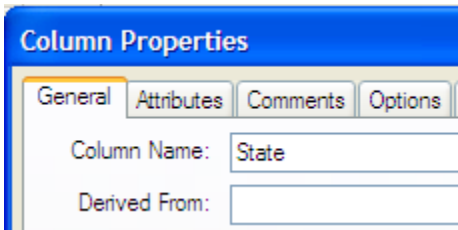
3. In the **Label** entry, enter *States* in it, and then press the TAB key.
4. Press the TAB key again to accept *STA* as the Prefix.

Next, you are prompted to choose a Table Driver.

5. Select *TOPSPEED* from the **Driver** drop list, and press the TAB key.
6. Press the **OK** button to complete the new Table.
7. In the Fields Quick View pad, press the **Add**  button, located in the toolbar.



8. Type *State* in the first row of the **Column Name** entry, and press the TAB key.

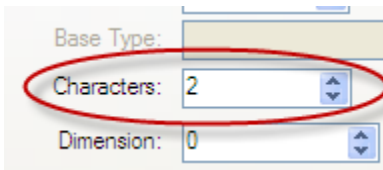


This creates a column named *State*. This column is the link to the *State* column in the *Customer* table. Using the same column names makes it easier to link the two tables in a relationship. In the generated Clarion code, adding the table's prefix to column labels (separated by a colon) creates unique names for columns with the same name in separate tables. Therefore, this column will actually be called *STA:State*, while the column in the *Customer* table is called *CUS:State*.

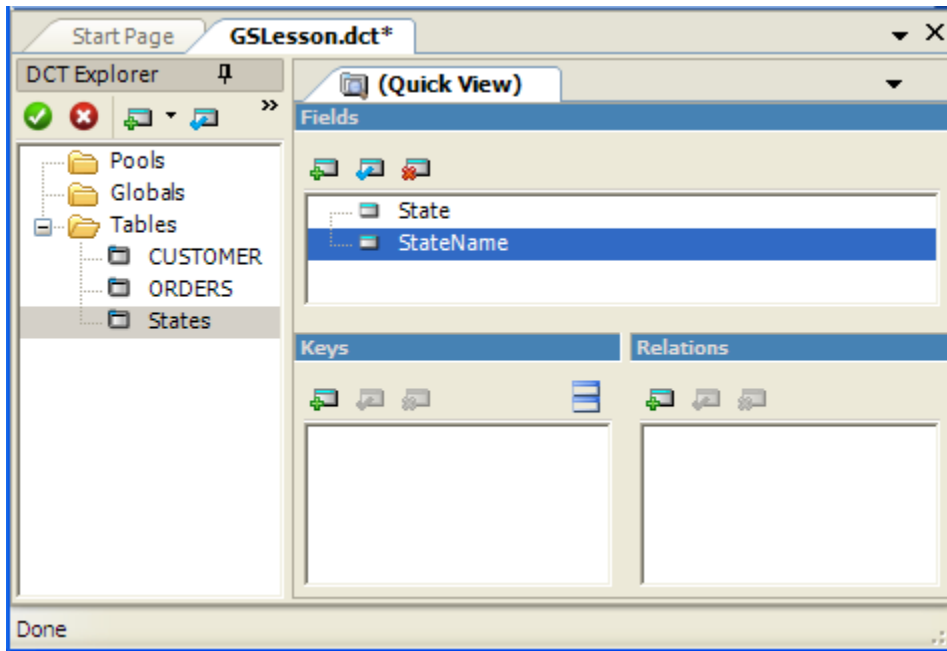
Note:


Clarion also fully supports a naming convention called Field Qualification Syntax. The prefix format above (*STA:State*) can also be represented as *States.State* (*file label.field label*).

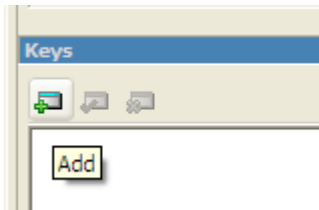
9. In the **Characters** entry, type 2, or use the spin control, and press the TAB key.




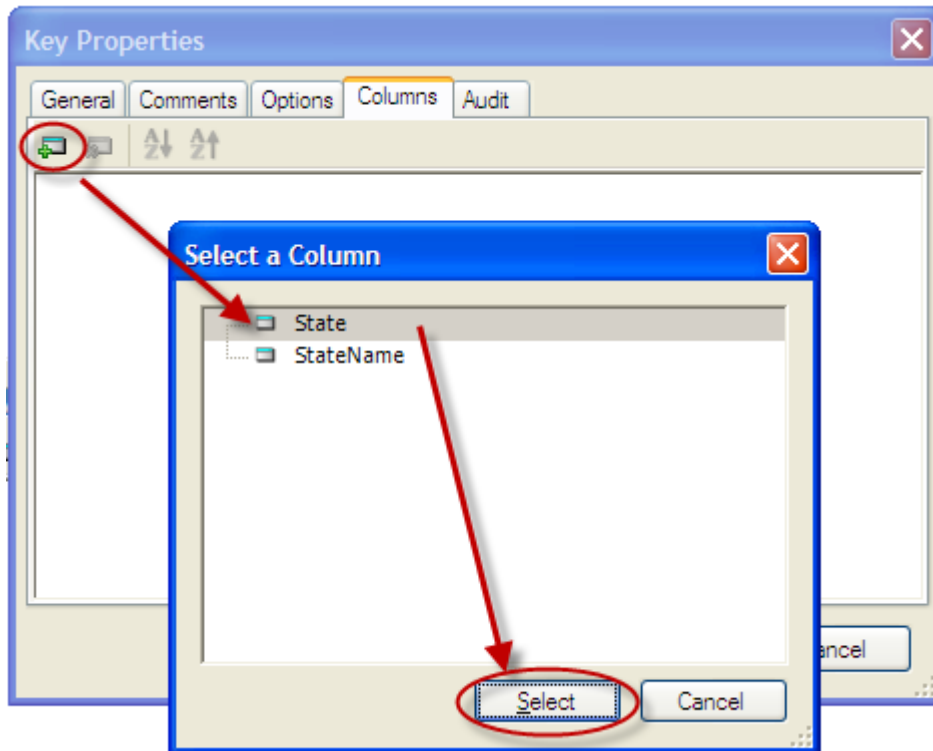
10. Accept all other defaults, and press the **OK** button to complete the new column.
At this time another *Column Properties* dialog opens allowing you to enter a second column.
11. In the **Column Name** column, type *StateName*, and then press the TAB key.
12. In the **Characters** column, type 30, then press the OK button to complete this column.
13. At this time another *Column Properties* dialog opens allowing you to enter a third column.
We are finished with this table's columns, so press the **Cancel** button to exit this dialog.



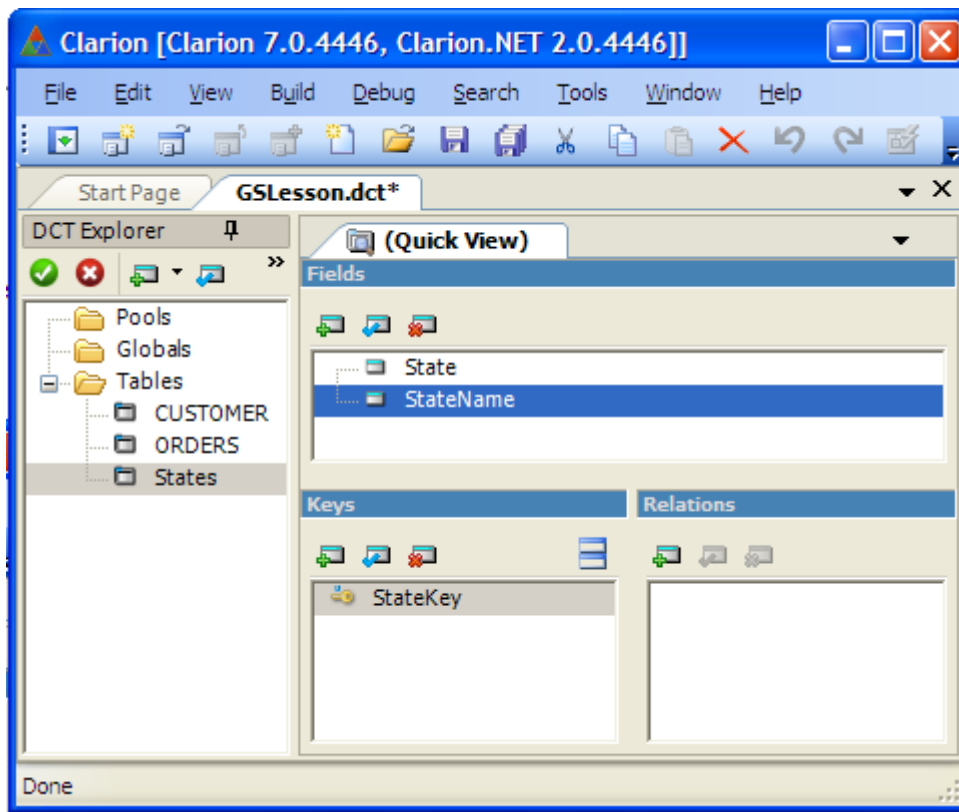
14. Next locate the *Keys Pad*, and press the **Add**  button, located in the toolbar.




15. In the **Label** column, Type *StateKey*.
16. In the Attributes group, check the **Require Unique Value** check box.
- This specifies a key that does not allow duplicate entries, enabling you to have a unique state reference file for your customers. This allows us to create a Many to One relationship between the two tables (Many Customers live in one State).
17. Click on the **Columns** tab, and press the **Add**  button, located in the toolbar. In the *Select a Column* dialog, highlight *State* and press select as shown on the next page:




18. Press **Cancel** in the *Select a Column* dialog to close this window, and press the **OK** button to save this key. Finally, press **Cancel** again in the *Key Properties* dialog to stop adding additional keys and return to the main Dictionary IDE.



19. Press the **Save** button  in the IDE toolbar at this time to save your work, and stay in the Dictionary Editor.

Exercise Five - Create the Lookup Relationship

Set the Relationship for the Customer and States tables:

1. Highlight *Customer* in the *DCT Explorer*, then press the **Add Relation**  button located directly above the *Relations* view.

The *New Relationship Properties* dialog appears. This is where you define the relationship.

2. Make sure the **Type** drop list is set to *MANY:1*.
3. In the **Foreign Key** drop list, press the down-arrow key to display the choices, highlight *None*, and then press the TAB key.

This is no key required on the Customer table (the Many side of the relationship). You will always be "looking into" the States table and never the other way in this example.

4. In the **Related Table** drop list, press the down-arrow to display the choices, highlight *States*, and then press the TAB key.
5. In the **Primary Key** drop list, press the down-arrow to display the choices, highlight *StateKey*, and then press the TAB key.

This is the key on the States table (the One side of the relationship) that will be used to link the two tables.

Next, the linking columns in the keys must be "mapped" so the Application Generator can know exactly which columns in the two tables are related to each other. Since we used identical column names, this is easy.

6. Press the **Map By Name** button.

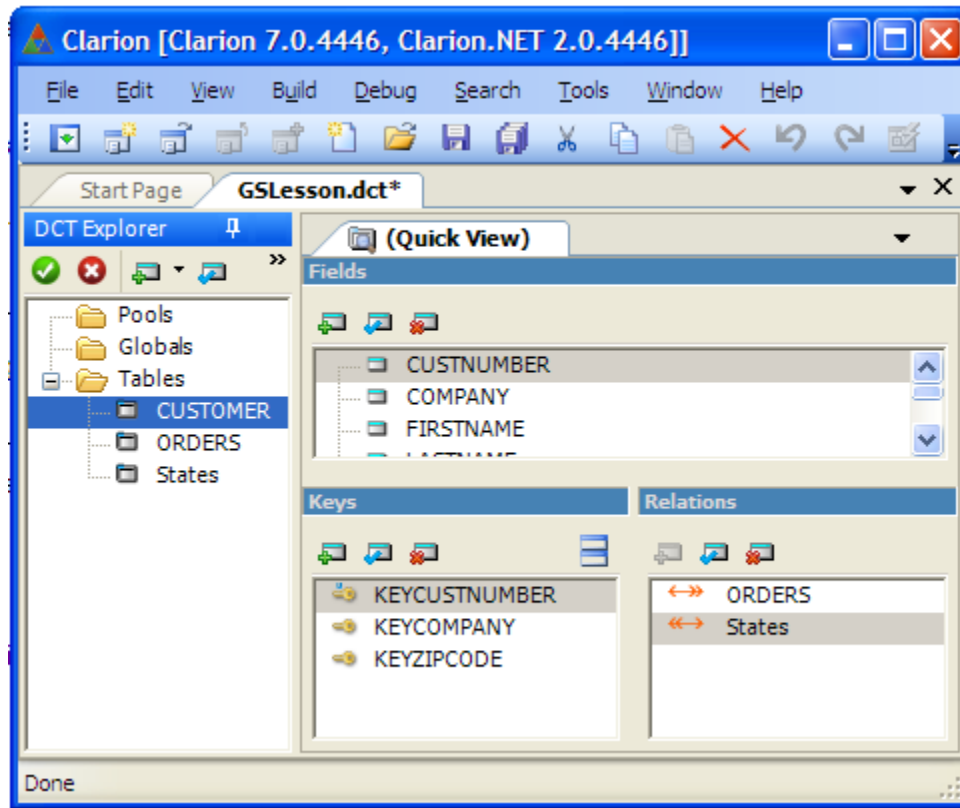
The linking columns between the two tables appear in the two Column Mapping list boxes. This is one reason to name linking columns the same.

Note:

If you did not have a matching column name, no problem! Simply double-click on the **Columns Mapping** entry, and select the linking field from the popup window provided.

Since the table relationship defined is only in one direction, **Referential Integrity Constraints** are not needed.

7. Press **OK** to close the *New Relationship Properties* dialog. Press **Cancel** to exit the *Relationship Properties* dialog.




Exercise Six - Set the Lookup Validity Check

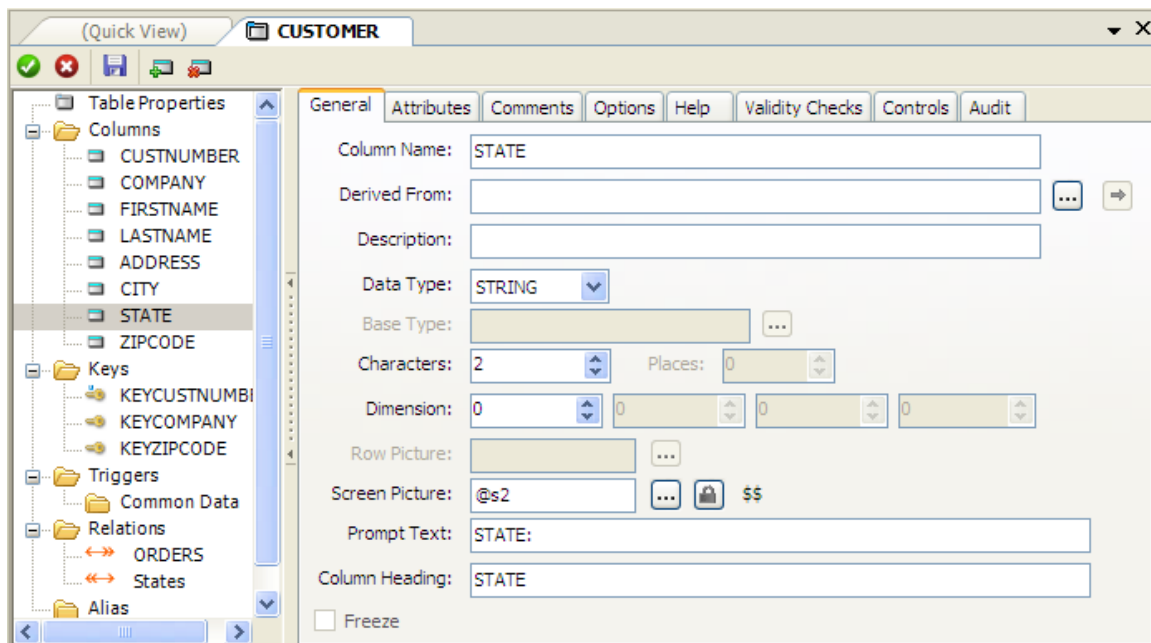
Now that the table relationship is defined, we can set the Validity Checks for the column that we expect to use on update forms.

When entering a new Customer, we can specify that the State column must match an existing record in the States table.

Define the Validity Check for the Customer State column

1. Highlight the *Customer* table in the **DCT Explorer** list.
2. In the **Fields Quick View** list, highlight the *State* column entry and press the **Change** button  located in the **Fields** list toolbar.

This opens the Entity Browser and Properties for the Customer table.



The screenshot shows the 'Entity Browser and Properties' window for the 'CUSTOMER' table. The 'General' tab is active. In the left-hand 'Table Properties' tree, the 'Columns' folder is expanded, and 'STATE' is selected. The right-hand pane displays the properties for the 'STATE' column:

- Column Name: STATE
- Derived From: (empty)
- Description: (empty)
- Data Type: STRING
- Base Type: (empty)
- Characters: 2
- Places: 0
- Dimension: 0
- Row Picture: (empty)
- Screen Picture: @s2
- Prompt Text: STATE:
- Column Heading: STATE
- Freeze: ☐

3. Select the **Validity Checks** tab.
4. Select the **Must Be In Table** radio button.
5. Choose *States* from the **Table Label** dropdown list.

This requires that the column can only contain values verified by getting a matching row from the States table. This is validated using the table relationship information, which is why this Validity Check cannot be set until the relationships have been defined.

6. Press the **Save and Close** button  to close the *Customer* table.

Exercise Seven - Fine Tuning the Customer and Orders Tables

Back in Exercise Two (2), we imported raw table information into the dictionary. Although this is the fastest and most accurate way of defining your tables in the data dictionary, the file format information that is also imported may not be the best display format for your applications.

For example, a table may store a zip code as a 4-byte integer on disk. A zip code of 33024, using the default picture for that data type, will be displayed as 33,024. If the zip code stores an extension, it may be worse. For example, 330241159 stored in the table would appear as 330,241,159 by default.

Also, some data types should not be accessible to the user, like an order number or customer number. By default, all columns imported into the dictionary are designated as entry data types.

This exercise demonstrates the art of fine tuning your dictionary elements defined.

Tip

The more time spent fine tuning your dictionary, the more time you will save generating a near perfect application using the various wizard tools!

Auto number your numeric Unique Keys

1. Highlight the *Customer* table in the **DCT Explorer** list.
2. In the *Keys View*, RIGHT-CLICK on *KeyCustNumber*, and select **Properties** from the popup menu.
3. In the **Attributes** group, check the **Auto Number** check box.
This will effectively cause any new customer that we add to automatically be assigned with the next sequential customer number (by default).
4. Press **Save and Close** to close the *Customer* window.
5. Highlight the *Orders* table in the **DCT Explorer** list.
6. In the *Keys View*, RIGHT-CLICK on *KeyOrderNumber*, and select **Properties** from the popup menu.
7. In the **Attributes** group, check the **Auto Number** check box.
This will effectively cause any new order that we add to automatically be assigned with the next sequential order number (by default).
8. Press **Save and Close** to close the *ORDERS* window.

Set the Autonumber Unique Columns as Display Only

After adding the auto numbering capability to our numeric unique keys, we can also change the display characteristics of the auto number column(s). The reason for this is that we do not want the user to be able to modify a number that is automatically assigned. In some applications, developers sometimes like to hide this information all together from the user by not populating it. In our application, we will choose to display this information, but not allow the user to modify it.

1. Highlight the *Customer* table in the **DCT Explorer** list.
2. RIGHT-CLICK on the *CustNumber* column in the **Fields** list, and select **Properties** from the popup menu.
3. Select the **Controls** tab. Change the **Control Type:** from *Entry* to *String*, and press the **Save and Close button** to close the *Customer* window.
4. Highlight the *Orders* table in the **Tables** list.
5. RIGHT-CLICK on the *OrderNumber* column in the **Fields** list, and select **Properties** from the popup menu.
6. Select the **Controls** tab. Change the control type from *Entry* to *String*, and press the **Save and Close button** to close the *Orders* window.

Examine this section carefully before proceeding to the next section. By choosing the properties for a control at this time, you can save time later. Every application you generate from the dictionary, and every procedure in the application will automatically format the control the way you want it. If you don't format it here, and if the control requires custom formatting, you will have to custom format it for each procedure and application later. Let's continue this thought to our next section.

Column Display Formatting

Several columns contained in our imported tables will make better sense if we can present that data in a more readable format. For example, a date may be stored as a long integer in our table, but what does 75543 really mean? This could be 75543 days that has occurred from a base date, but is that useful to the user?

1. In the *Orders* table, RIGHT-CLICK on the *OrderDate* column, and select **Properties** from the popup menu.
2. In the **Screen Picture** entry, type @D2. Press the **Save and Close button** to close the *Orders* window.

Clarion provides a number of different date and time picture formats that are used with date and time data types. Click [here](#) for a brief discussion of the date formats.

3. Highlight the *Customer* table in the **DCT Explorer** list.
4. RIGHT-CLICK on the *ZipCode* column in the **Columns** list, and select **Properties** from the popup menu.
5. In the **Screen Picture** entry, type @P#####P. Press the **Save and Close** button to close the *CUSTOMER* window.

The zip codes stored in our Customer table are 5-digit zip codes. However, in the raw picture format, you would see 33,334 for a zip code of 33334, and 6,792 for a zip code of 06792. The picture entered above is called a *pattern* picture. For a quick look at example pattern pictures, click [here](#).

6. Press **Save and Close** from the DCT Explorer toolbar. If you are prompted to save changes to *GSLesson.DCT*, press the **Yes** button to save and close the Dictionary Editor.

Exercise Eight - Generate your Program using the Wizards

Now it is time to unleash a very powerful part of Clarion and the IDE: The Application Wizard.

In this exercise, we will use the dictionary that we created in Exercises 1 – 7, and generate a full-featured application, all in just a few minutes with no hand coding needed!

Note:

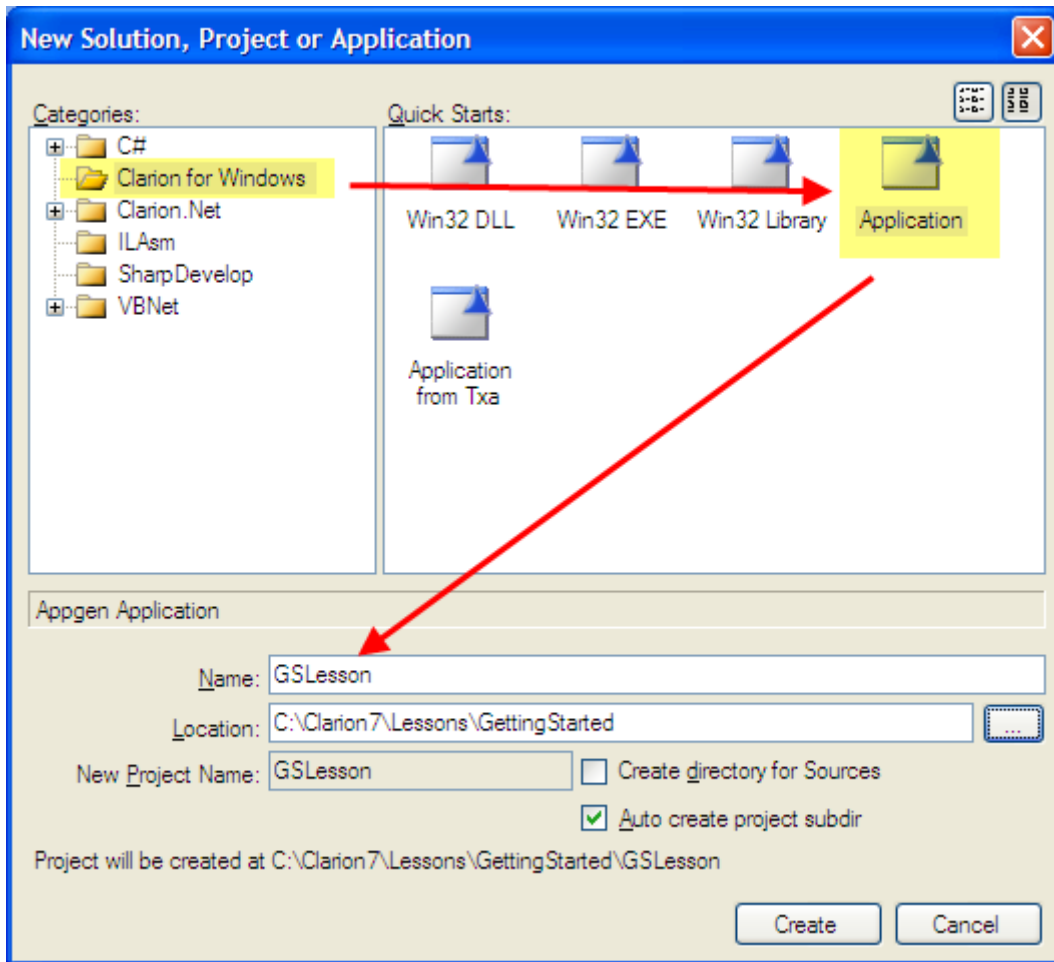
Make sure that the Clarion IDE is loaded. Close any windows that may be opened at this time. You can leave this help file opened, and task switch back and forth with the IDE windows. In this lesson, the install folder is identified as *Clarion7*. If you decided to install to a different root name, please substitute accordingly.

The Application Wizard generates entire applications based on the minimal information that you provide in response to a series of Wizard dialogs that ask for one piece of information at a time.

Create a New Application

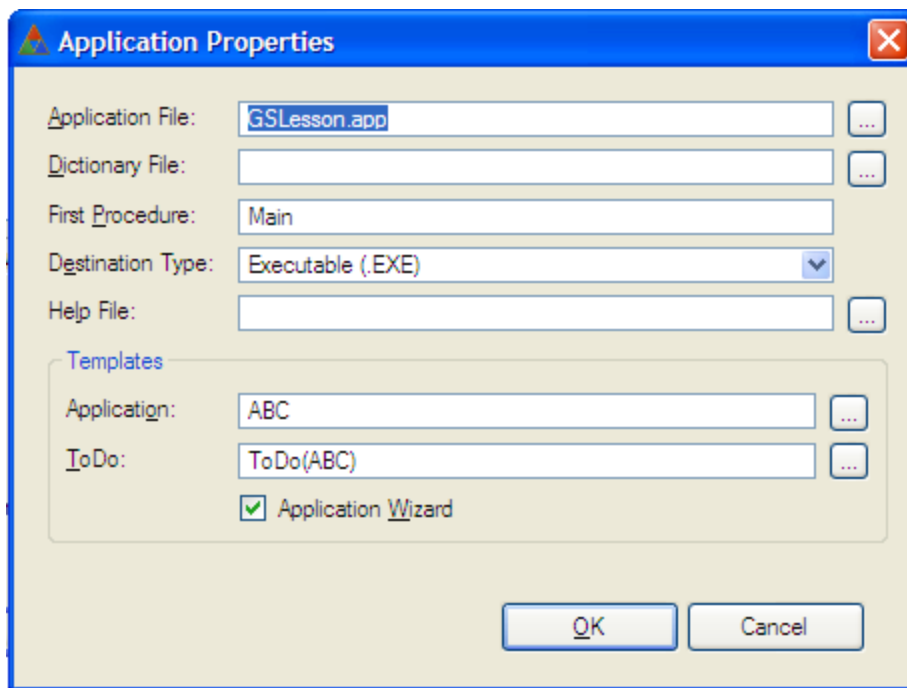
1. From the IDE Menu, choose **File ► New ► Solution, Project or Application**.

The *New Solution, Project or Application* dialog appears.



2. Select the *Clarion for Windows* item in the left **Categories** pane, and navigate to the *Application* Quick Start as shown above.
3. Verify that the **Location** is in the target ...*\Lessons\GettingStarted* directory.
4. Type *GSLesson* in the **Name** field.
5. Uncheck the **Auto create project subdir** checkbox.
6. Press the **Create** button.

The *Application Properties* dialog appears.



7. Press the ellipsis (...) button to the right of the **Dictionary File** entry box.

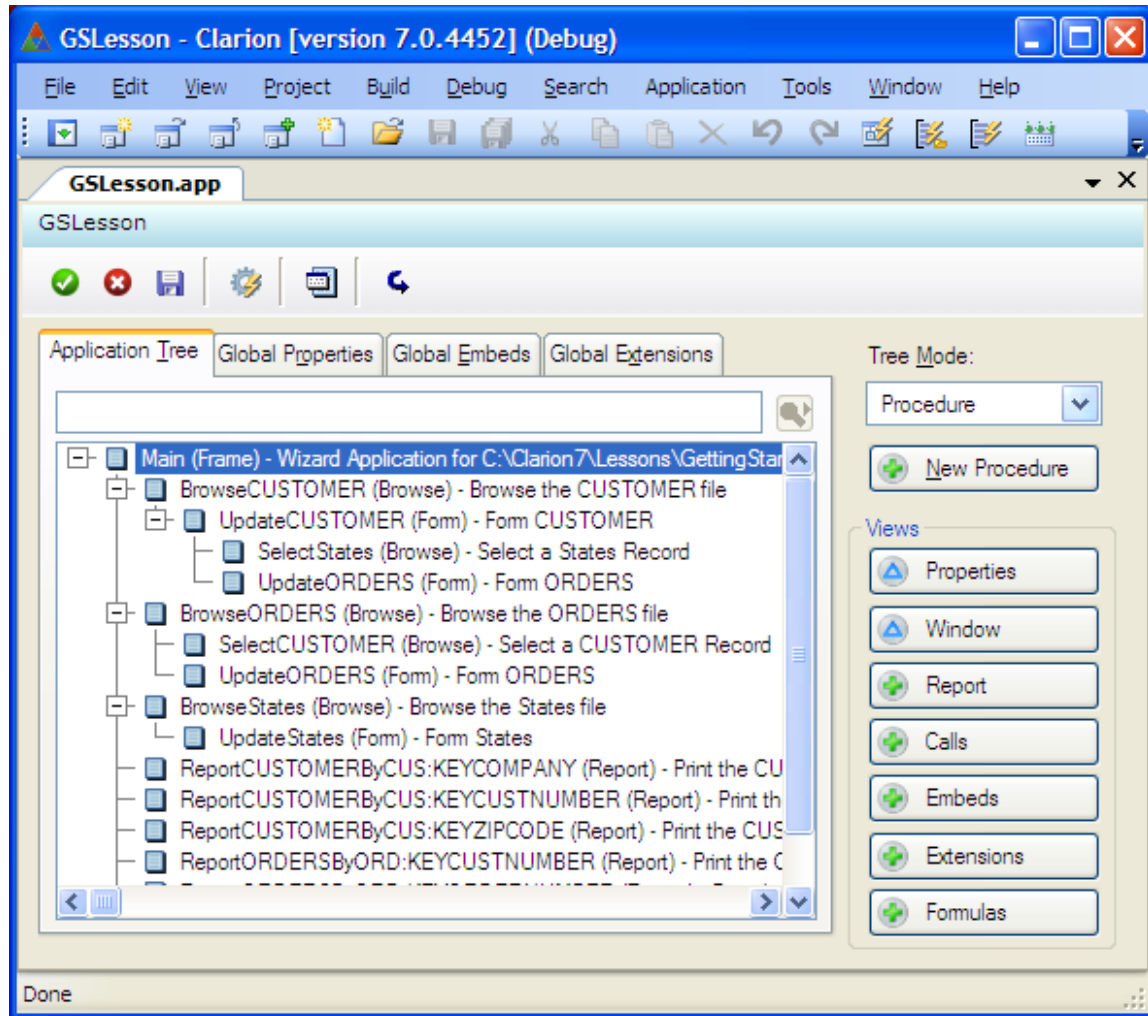
The *Select Dictionary* dialog appears.

8. Highlight the ... \Lessons\GettingStarted\GSLesson.DCT file and press the **Open** button.
9. Verify that the **Application Wizard** check box is checked, and then press the **OK** button.

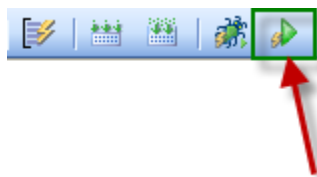
The *Application Wizard* dialog appears.



10. On the opening wizard window, press the **Next** button.
11. Accept the *Default Theme* and *Layout1 Report Layout*, and press the **Next** button.
12. Make sure that the **Generate Procedures for all files in my dictionary** drop list choice is selected, and press the **Next** button four (4) times.
13. You should now be in the *Procedures & Reports* dialog window. Make sure that the **Generate Reports for each file** checkbox is checked.
14. Press the **Finish** button. The Application Wizard now creates the application.



15. From the IDE Toolbar, **Generate Source**, and **Build** and **Run** your application by pressing the **Start without Debugger** button (shown below):



Congratulations! Your complete application is now running.

Explore this new application. Try viewing and running reports on the default data that is supplied.

In the running application, we don't need to add or modify and data at this time, but just want to see the part of the application that our changes affected.

In the Customer's update form, entering an invalid state that does not exist in the States file will cause a pick list to pop up when the user moves away from the field.

16. Exit the *GSLesson* program, which returns you to the Clarion IDE. In the Application toolbar, press the **Save and Close** button to close the application, and then press the **Close Solution** button to close the application's solution.

Summary

Let's recap what was accomplished:

- ✓ You used the Dictionary Editor to create a data dictionary.
- ✓ In your new dictionary, we defined a Customer and Orders table using the dictionary Editor's Table Import feature.
- ✓ We set up table relationships between the tables that we imported.
- ✓ You used the Data Dictionary to quickly add another table definition.
- ✓ You defined the relationship between the two tables and specified the Referential Integrity rules governing the relationship.
- ✓ You used the Application Wizard to create your first application, with added validation to the Customer's State column.

In less than thirty minutes—without any "coding" on your part—you've created complete application for a database containing three related tables. That's a real accomplishment!

If you can do this much with Clarion's "shortcuts," what can you expect to accomplish with all the other tools Clarion provides?

In this first lesson, you should notice the overall design flow: Dictionary changes, Application changes, back to the Dictionary, back to the Application, and so on.

You will find that the more time that you spend on your dictionary design at the start, the less modifications you will need on the application side.

In this Getting Started lessons, we have only scratched the surface into the power of the Clarion development environment.

The Getting Started and Learning Clarion lesson applications

The completed application and dictionary files are located in the ...\\Lessons\\GettingStarted\\Solution and ...\\Lessons\\LearningClarion\\Solution directory. These are the files created by following the steps in the Getting Started and Learning Clarion documents.

There are two other lessons contained in this folder. The files used for the Online User's Guide's Debugger lesson can be found in the ...\\Lessons\\Debugger folder.

A lesson targeting the use of the Dictionary Synchronizer The files used for the Online User's Guide's Synchronizer lesson can be found in the ...\\Lessons\\SQL folder.

Clarion Programming Concepts

At the foundation of Clarion (the development tool) is the Clarion programming language—a fourth-generation, general-purpose (fully object-oriented) programming language, which has been highly optimized since its inception for business (database) application development. The Clarion language is the real tool that you now hold in your hands. But Clarion is much more than just the Clarion language—it is also a complete *toolset* designed specifically for business database application creation and maintenance.

On top of the Clarion programming language is the Clarion development environment—a completely integrated set of tools that are specially designed for Rapid Application Development (RAD), Rapid Application Maintenance (RAM), and Rapid Application Enhancement (RAE). These tools are all geared towards generating Clarion language source code for you, so you don't have to write it yourself. That's what makes Clarion so powerful—although you can write your own Clarion language code, the toolset can generate (and re-generate) most "standard" code for you, leaving you free to concentrate on your application's design issues.

Levels of Abstraction

There is a principle that, the further you can get away from the "bare metal" of the computer (pure binary), the more productive you can be.

A Bit of History

The very first digital computers were programmed in binary, one machine language instruction at a time, and the programmer's job was very difficult. Then Assembly language came along to make the programmer's job easier by making the program code readable by a human being. Each Assembly language statement generated exactly one machine instruction.

Early Assembly programmers noticed that they were consistently writing the same sequences of instructions to perform standard tasks. Therefore, they created Third Generation languages (3GL—like Fortran, C, PL/1, etc.) so that each 3GL statement would generate multiple machine instructions for these standard tasks and make programmers more productive. By doing this, they started operating at a higher level of abstraction away from the "bare metal" of the computer. The drawback was that only "standard" sequences of machine code were generated, so some of the total flexibility that's possible when you work directly in Assembly language was lost.

Present Day—4GL

This same basic process has occurred to bring about the advent of Fourth Generation languages (4GL) like Clarion. For example, the Clarion language automatically handles all the standard Windows chores (such as re-painting the screen) that programmers are required to write for themselves in 3GL languages. In Clarion, a "Hello World" program is just 9 lines of source code, while in C/C++ you must write over 80!

The Secret of Productivity

The Secret of Productivity (you've all heard this before) is working smarter, not harder. That is exactly what Clarion is designed to allow you to do with its Template-driven programming paradigm—Clarion writes the standard code for you so you don't have to write it yourself, and then it is still flexible enough to let you add the "bells and whistles" (the fun part).

The secret to real programmer productivity without loss of flexibility is this: always operate at the highest level of abstraction that can get the job done the way it needs to be done. The "problem" with operating at higher abstraction levels is the loss of flexibility to always do things exactly the way you want. With Clarion, this problem is solved.

The Clarion toolset contains multiple abstraction levels—allowing you to always operate at the best abstraction level to do exactly what you need to do. You're never "stuck" at a high level or forced to always "muck around" at a low level. The toolset generates object-oriented Clarion source code for you using Clarion's Application Builder Class (ABC) Library (the high level), and when you need to, you can also add your own code (lower level) to augment the standard template-generated functionality.

Generally speaking, the higher the level of abstraction a programmer works at, the more productive they can be. At each higher abstraction level, the programmer becomes more productive by letting the computer do more of the work for them—generating standard code for standard tasks. This is the fundamental principle behind Clarion—and we implement this principle through Template-driven programming.

Template Driven Programming

Clarion's Application Generator is "template driven." This means that it is a tool that changes based on the requirements of the template you are using to generate your code. Clarion's Procedure, Control, Extension, and Code Templates all write Clarion language source code for you, giving you a tremendous productivity boost. Clarion's templates provide many of the benefits of object-oriented programming, especially reusability, and the default template set generates truly object-oriented Clarion code for you using the Application Builder Class (ABC) Library. This makes Templates the real key in Clarion to Rapid Application Development.

What is a Template?

In Clarion, a template is not just a "one-time" tool generating code that you must then maintain yourself (like the things that some other languages call "templates"), but is a continually interactive tool that asks specifically for the information it needs to generate your source code. Changing the information you provide to the template causes different source code to generate the next time you make your application. This interactivity makes Clarion's Templates the key to Rapid Application Maintenance.

Clarion templates allow you to insert your own Clarion source code into their generated source at any of the many Embed Points available to you within each template. This makes it unnecessary for you to maintain all the generated code in order to customize the procedure's behavior. It also guarantees that your special customizations aren't overwritten the next time you generate source—the template simply generates your code inside its standard code. This easy customization makes Clarion's Templates the key to Rapid Application Enhancement.

All the templates are stored in the registry file (TemplateRegistry.TRF). This file contains the pre-written executable code and data structures which you customize and reuse. You can use the Template Registry to modify the design of each template's default window or report to the way you want them to appear when you first create a procedure.

You can modify the Clarion templates. You may also add third party templates, or write your own, and use them in addition to, and along with, the Clarion templates. This makes the Application Generator an infinitely extensible tool.

How do you use Templates?

When you create a new procedure, you identify the Procedure template that generates code that does the task you need to perform, then you customize it with the development tools. These Procedure templates include elements such as "browse windows" for viewing groups of rows, and "form windows" for editing one row at a time. If the procedure is for a window with a menu, the menu actions are automatically added to the application's procedure tree, and marked "ToDo," as any other procedure call would be.

The usual way to customize a procedure is to call one of the formatters. The Window and Report Formatters are visual design tools that allow you to "point and click" to design windows and reports. You just pick a control from a toolbox, click to place the control, then right-click to modify its properties.

Once you have created a procedure you can also use the Control, Extension, and Code templates to add even more functionality to the procedure. Control templates contain controls and generate all the standard executable code needed to use and maintain them. All the pre-populated controls that appear in the default window designs for the Procedure templates are actually Control templates that perform all the functions of the procedure.

Extension templates add executable code that increases the functionality of the procedure. Each typically provides you with on-screen instructions on what information to "fill in" to incorporate its functionality into the application.

Another way you may customize a procedure is to add your own embedded source code. The Application Generator displays a tree displaying all of the Embed Points that you have: before, during, and after the procedure's main logic, and for every event the window or controls in the procedure can generate. You can also directly edit your embedded source within the context of all the generated source code. This allows you to pick the precise logical point at which to execute the code, then "hand code" it, or tell a Code template to write code for you. The Application Generator generates all your application's source code from the templates and all your customizations (including embedded source code).

Clarion provides a rich assortment of standard templates with which you can rapidly develop applications. Just as the Quick Start lesson introduced you to the Wizards, the Application Generator lesson in the Learning Clarion document introduces you to using Clarion's Templates to produce any Windows application you need.

Clarion's Levels of Abstraction

At this point in your introduction to Clarion, you've already worked with Clarion's highest abstraction levels: the Wizards. You'll work with most of the following features later in the Learning Clarion lessons.

Dictionary Editor and Dictionary Diagrammer

Your Data Dictionary is a single repository for all the table definitions and their default data entry control settings. For existing data, defining your tables can be as simple as importing the table's definition into the Data Dictionary directly from the table itself!

Application Wizard

Generates a complete "standard" application for you in just minutes, based on the table definitions and settings in your Data Dictionary.

Procedure Wizards

Generate "standard" procedures for you (like a Browse list, or data entry Form) based on the settings in your Data Dictionary.

Procedure Templates

Generate "standard" procedures for you (such as the application Frame menu, or a Report), based on the window or report design you create in Clarion's Window Formatter and Report Formatter, and the options you select for the Template's prompts for information about your procedure.

Control Templates

Add "standard" controls to your window or report design (such as a related tables Tree control) and generate all the code needed to "drive" the behavior of the control.

Extension Templates

Generate additional code into a procedure to provide functionality unrelated to any specific window control (such as a Date/Time display in the application Frame's status bar).

Code Templates

Generate code into a procedure that usually performs a single task related to a specific window control (such as Data Entry validation).

Embedded Source Code

You can write Clarion language source code of your own in any of the very numerous Embed Points in each procedure to customize and/or modify the behavior of the Template-generated code.

Source Template

A Template which allows you to completely write your own Clarion language procedures while still maintaining the application in the Application Generator.

Windows API and C Standard Library

In any Clarion language code that you write (whole procedures or into Embed Points), you can directly call Windows API functions or C standard library functions, if you really need to. A utility program will generate the Clarion language Windows API function prototypes, and C Standard Library prototypes are documented in the Programmer's Guide.

C/C++ or Modula-2 Code

Write any SoftVelocity C/C++ or Modula-2 functions that you need to and link them into your Clarion application. These compilers are included and are invoked by simply adding a .C, .CPP, or .MOD source code module to your Clarion project (also sold separately).

As you can see, the levels of abstraction available to you in Clarion span every level from the Application Wizard right down to writing C code (or even Assembly, if you want to work that hard).

The Clarion Key to Maximum Productivity

The key to making Clarion work for you is to always work at the highest abstraction level possible:

- ✓ Let the Application Wizard write a "starting point" application for you.
- ✓ Use the Procedure Wizards to add new procedures, as needed.
- ✓ Modify what you need to in each Procedure Template using Clarion's two-way interactive tools (like the Window and Report formatters).
- ✓ Add Control, Extension, and Code Templates as you find you need their functionality.
- ✓ Customize it all with some Embedded Source Code.
- ✓ When you find you need to do something that no template does for you, use the Source Template and write it in the Clarion language (or get a third-party Template that does what you need).
- ✓ Dive all the way down to the Windows API and C/C++ levels only when you absolutely must.

This is the Clarion philosophy of working smarter, not harder, by letting the toolset do the "drudge" work. Do this, and you'll find that your productivity soars compared to any other tool you've ever used.

Clarion's Development Environment

The development environment contains several main tools, all of which are accessible from the others. When using the Application Generator, buttons and menus in the various dialogs lead to the other tools.

The following Application Development Flow chart shows how all the tools common to both Professional and Enterprise Editions of Clarion interact with each other and the template registry, with the Application Generator at the center of the whole process:

This section provides a description of each tool, in the order that a typical programmer using the Application Generator might encounter them. Each tool contains dialog boxes which the programmer fills out to describe the Application's functionality to the Application Generator. On your command, the Application Generator generates the specified source code, and the Project System compiles and links it to make an executable program.

Common Tools

Programming in Clarion is, in many ways, a personal journey through a series of dialog boxes. There is no mandatory sequence to follow through the dialogs, though some are prerequisites for others. The following are brief descriptions of the tools common to both Professional and Enterprise editions of Clarion.

The Dictionary Editor

The Data Dictionary (a .DCT file), maintained by the Dictionary Editor, holds a description of the database, including its tables, keys, indexes, database drivers, file relations, columns, column validation rules, referential integrity constraints, and more. This is the first file you create when you design your application.

You can create the table definitions from scratch, or you can import definitions from existing tables. The other tools of the development environment use the information in the Data Dictionary to let you, for example, easily place columns in a dialog box you design for the end user. The Application Generator creates code for all the statements that access the tables based on how you construct the Data Dictionary. In fact, the Application Wizard can generate a fully functional application based solely on your Data Dictionary!

The Application Generator

The Application Generator generates your application's source code, one procedure at a time, based on the templates you pick from the template registry. It lets you add global and local memory variables, and customize the procedures with visual design tools and embedded source code.

The Application Generator provides access to the other tools of the development environment so you can customize the look and functionality of the windows, menus, reports and other user interface elements of your application. The templates provide their own controls (which appear within the Application Generator) allowing you to provide the information with which the template customizes the requested functionality.

The Window Formatter

Visually design your application's windows and controls—everything the end user sees—in the Window Formatter. It automatically generates the source code for the elements you visually design on screen.

The Report Formatter

The Report Formatter works with the Application Generator in much the same way as the Window Formatter. You place controls in a sample report page. At run time, the print engine processes the rows, handling page breaks, group breaks, headers, and footers as specified.

The Text Editor

The Text Editor is a fully functional programmer's editor which you can use to write any source code your application needs. Most likely, when using the Application Generator, you'll use the Text Editor to create embedded source code to customize the way a procedure operates. Or, you can write entire applications from scratch (if you really want to work that hard).

The Text Editor features color coded syntax highlighting, making it easier to identify the different parts of the Clarion language statements for editing purposes. It also has full text search and replace capabilities, along with all the standard editing tools.

The Formula Editor

The Formula Editor helps you quickly generate and manage simple or complex assignment statements based on any kind of mathematical or string expression. The Formula Editor provides syntax checking, plus instant access to all the variables, functions, and operators, so that the formulas generated are always syntactically correct.

The Project System

The Application Generator automatically creates the project file (.PRJ) for the application for you. The project file contains compile and link options, such as whether to include debug code, optimization choices, external files, the source code files, libraries and other external files included in the compile and link process.

The Debuggers

Debugging a program usually requires running the program and repeatedly stopping it at various points during execution to examine the value of different variables to determine the cause of logic errors in the program. The Clarion Debuggers (both 16-bit and 32-bit) have a number of windows which display source code, variable contents, active procedures, and much more.

There are three lessons on using the Debuggers contained in the User's Guide. We recommend that you go through them after finishing Getting Started and Learning Clarion lessons.

The On-Line Help

Clarion provides extensive on-line context sensitive help from almost every screen. Click on the Help button, or press f1.

The Common Questions section of the on-line help provides instructions and examples for performing common application tasks. You can cut and paste example code directly from the help system into your program. Choose Help Contents, then press the FAQ button.

The Clarion on-line help system includes a main help file, plus auxiliary files. The on-line help files are searchable by keyword and by index. To perform keyword or index searches on an auxiliary file, open the file directly with the Windows Explorer or Program Manager.

Enterprise Edition Tools

The following are brief descriptions of the tools which are in the Enterprise Edition of Clarion and are not available in Professional Edition, except as add-ons (in certain cases). These tools are all fully documented in the core help—including lessons for each.

Dictionary Synchronizer

The Dictionary Synchronizer allows you to synchronize one data dictionary to another (available only in Enterprise Edition). You can synchronize two Clarion data dictionaries (.DCT files), or synchronize a Clarion data dictionary with an SQL back end's data dictionary (either direction). This allows you to get all the table definitions and relationships at once from the SQL backend database and eliminate a lot of front-end work and keep the Clarion data dictionary in synch with the SQL RDBMS at all times, no matter what changes the SQL Database Administrator makes.

Dictionary Diagrammer

The Data Dictionary (a .DCT file), can also be maintained by the Dictionary Diagrammer (also available as an add-on to Professional Edition). This is a visual data modeling tool, allowing you to view your database design from an entity-relationship diagram perspective.

Business Math Library

The Business Math Library is a set of standard business and statistical functions which you may use (also available as an add-on to Professional Edition). There are also a full set of templates which make implementing these functions in your programs very easy.

What's Next?

There are several directions that we would like to recommend:

1. The *GSLesson* application that we started here is greatly expanded with many of Clarion's powerful templates and other features in the Learning Clarion section of this online help file. These lessons are designed to introduce you to all the common development tools Clarion offers in both Professional and Enterprise Editions.
2. Dive in and get started with your own projects. The Online User's Guide, accompanied with the Clarion Core Help, provide comprehensive information to help you through your particular projects.
3. SoftVelocity offers educational seminars at various locations. Call Customer Service at (954) 785-4555 to enroll.
4. Join (or form) a local Clarion User's Group and participate in joint study projects with other Clarion developers.
5. Participate in SoftVelocity's online newsgroups. SoftVelocity's internal news server offers newsgroups for all SoftVelocity products. On this news server you can exchange ideas with other Clarion programmers. Log into news.softvelocity.com and subscribe to the groups for the products you want to discuss (Strongly recommended!).
6. In addition, there is a Usenet newsgroup—comp.lang.clarion— where you can network with other Clarion programmers from all around the world. If English is not your native language, there is alt.lang.clarion.
7. Good luck and keep going—the programming power that Clarion puts at your fingertips just keeps on growing as you learn more!

Index

Clarion Programming Concepts	33	Ex. 6 - Set the Lookup Validity Check	21
Getting Started.....	1	Ex. 7 - Fine Tuning the Customer and Orders Tables	22
Ex. 1 - Create the Dictionary	5	Ex. 8 - Create your Program using the Wizard.....	25
Ex. 2 - Import Your Data.....	8	Summary.....	31
Ex. 3 - Relate Your Tables	11	Getting Started Lessons	3
Ex. 4 - Add a Lookup File	13		
Ex. 5 - Create the Lookup Relationship.....	19		