The Evaluation Edition

Clarion for Windows.

No coding. No compromise. Powerful compiled applications for a broad range of database users: novices, power users, and professional developers.

You'll be able to create a 32-bit, multi-threaded application for your existing database in just one hour by following this guide through exercise one.

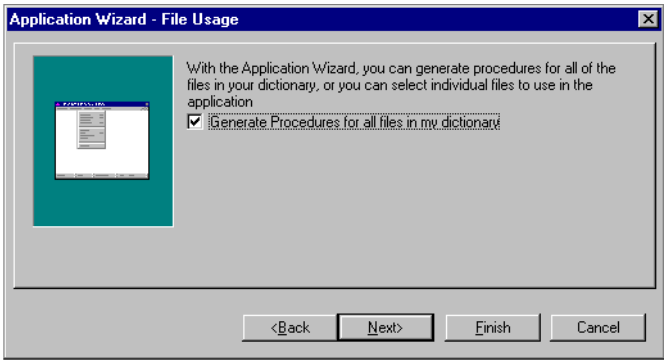
A Brief Preface

Clarion for Windows rises to "the next level" of Rapid Application Development—a new tier of programmer-friendly tools that maximize productivity, power, and performance.

The level of automation in creating applications with Clarion for Windows surpasses other RAD tools. With other tools, you create the user interface visually, but you still have to hand code the behavior of your application. Clarion produces a complete business solution for you: immediately and automatically.

Start Where Others Leave Off

The Clarion for Windows Application Wizard builds complex applications with no intervention necessary. Your starting point is a full-featured application.



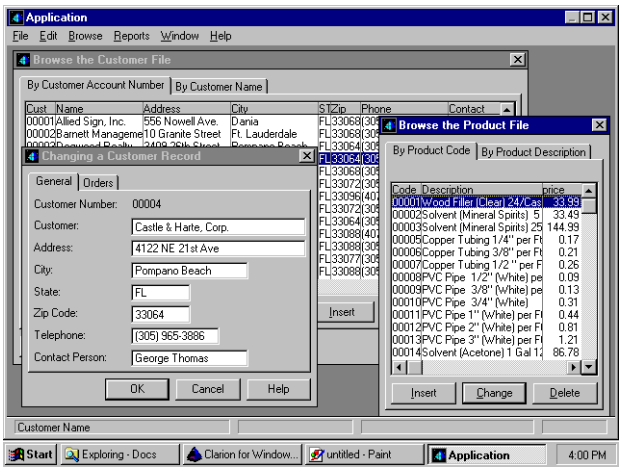
Create Applications From Data

From your database dictionary, which stores your data model plus optional pre-formatting options, the Wizard creates multi-threaded, multiple-order browse lists, with tab-selectable sort orders. You can take a database dictionary with dozens of files or tables defined, then generate and compile an multi-threaded application with literally hundreds of procedures—cleanly.

You just run the Application Wizard—it requires just a few clicks—then press the Make button.

Create Applications From Data

Your app contains synchronized parent-child dialogs with an update form for the parent on one tab, and a listbox displaying related child records on another. Reports for every file.



First 32-Bit RAD Compiler

Clarion, which was the first 16-bit RAD compiler, is the first RAD tool to compile applications in the new Win 32 Portable Executable format. It actually contains two compilers—a 16-bit compiler, and a 32-bit compiler. Clarion for Windows 1.5 is an evolutionary tool for database programmers who expect to support both 16 and 32-bit Windows for a transitional period.

No matter what version of Windows you develop for—Windows 3.x, Windows 95, or Windows NT—Clarion for Windows increases your productivity, allows you to deliver the same app to each platform while maintaining a single project, and helps you produce small, fast, royalty-free, native executables.

(Windows) Platform Independence

As a developer, you don't have to worry about what version of Windows either your end users—or you—run. You can build apps for any platform, from any platform, simply by selecting an option.

Clarion clones features across 16 and 32-bit Windows. Windows 3.x end users can be provided a Windows 95 look and feel via 16-bit versions of controls such as property sheets, tree, and progress controls. Windows 95 and Windows NT applications can safely and reliably incorporate 16-bit .VBX controls.

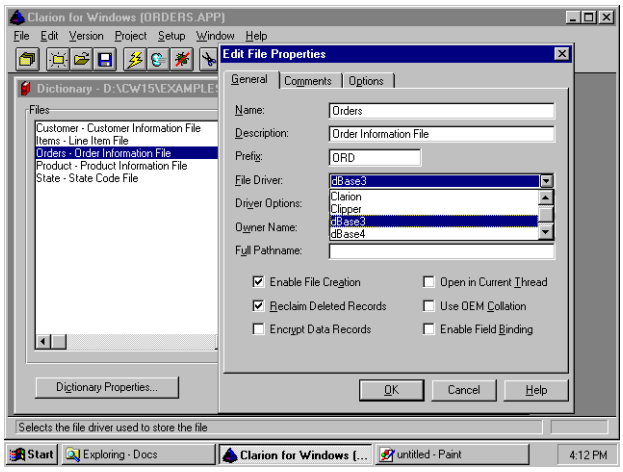
Database Independence

Clarion's replaceable database drivers allow you to connect to data in virtually any DBMS or file format. You can use direct drivers, ODBC, or optional SQL drivers. The Clarion database access grammar produces database independence with native mode performance. It accomplishes this with a compact and expressive syntax.

Choosing a file driver is as simple as specifying a choice in a file structure. When generating applications, you choose a driver from a drop down list in the Data Dictionary. You can even import a file definition from an existing data file, just by selecting a file in an Open File dialog.

Database Independence

There's no huge, external database engine. You either ship the file driver as an external .DLL (between 75 and 250Kb), or compile it directly into your executable.



The Evaluation Edition

Thank you for investing your time to evaluate the Evaluation Edition of Clarion for Windows. This section has three goals:

- ❏ To help you identify how Clarion for Windows' key benefits—productivity, performance, and easy project maintenance—fit into your application development needs.

- ❏ To briefly introduce the parts of the development environment.

- ❏ To outline the contents of this guide and the additional materials on the CD. This will help you quickly identify what you need to know, and provide a guide to setting up the Evaluation edition of CW.

Better, Faster, Smarter

Clarion for Windows can do three things for you: help you work better; help you work faster; help you work smarter.

- ■ **Working Better**: are you an independent software developer who wants to write more powerful programs? Programs with improved functionality whose impact on your end users can be measured? Programs that deliver the business solution the spec requires?

The template system is an intelligent code generator. Its design-time user interface helps you quickly select properties and functionality. The Clarion language allows you to custom code a solution unique to your business problem—and execute it at any point within a template procedure.

Better, Faster, Smarter

- **Working Faster**: are you a corporate programmer who wants to increase productivity by delivering programs faster? Do you need a tool to break up that application backlog? Do you have to solve a business problem now?

"Push-button programming" frees you to concentrate on the data model and business rules. By emphasizing pre-planning and design within the database dictionary, it's easy to generate robust business database apps with a common look and feel. The dictionary can store pre-formatting options, such as specifying a specific type of window control for a particular field. You set a common look for as many applications as will be made to maintain the same set of tables, since you can create many applications from a single dictionary.

Better, Faster, Smarter

- **Working Smarter**: is long term program maintenance a big headache? Is your project flexible enough to grow with end user demands, or does it hit the wall when the app changes from spec to delivery? What are the net results—return on investment, time to market, savings, user/customer satisfaction, quality—over the project's life cycle?

Clarion achieves a level of maintainability that other RAD tools can't match. Rather than using one-way wizards to define a database, as other tools do, changes made in the data dictionary migrate via live links to the application file. If you have to, for example, add a field to the database, you just regenerate the code and recompile. Using other RAD tools, it's often best to start over from scratch.

The Evaluation Edition

The CD contains a special version of Clarion for Windows.

Its limits are splash screen disclaimers on the applications you develop, and a limit on the data files you can edit. If you open a file or table, from any data source, with more than 100 records, you'll automatically open it in Read Only mode. Any less, and you can edit records normally.

The only other limit is that the applications you develop require that CWRUN16.DLL or CWRUN32.DLL be available in the path. The full version of Clarion for Windows allows you to link the functions from these libraries directly into your executables, allowing you to create even smaller applications.

The Documentation

The Evaluation Edition CD also contains all the documentation, in Adobe Acrobat Viewer documents. There is additional literature—product reviews, newsletters, and other materials—to help you learn more about

Clarion for Windows, and present third party views of it. There are also multimedia presentations that show you the development environment, plus demos of third party tools and of finished Clarion applications.

The “Hub” is the “center” of this document. It helps you jump to other documents. The “nodes” divide each section. They help you jump to important topics. To “jump” to another section, click on one of the icons on the side. To switch between full window and window viewing (the later includes more navigational controls), press ESC, and CTRL-SHIFT-L.

We Want to Show You Clarion

We want to show you as much as we possibly can with this CD. We think Clarion for Windows is a more efficient way to develop Windows applications. Many of our longtime developers tell us they regard Clarion as their secret weapon that out-produces the competition every time. Now we’d like to offer this secret weapon to you.

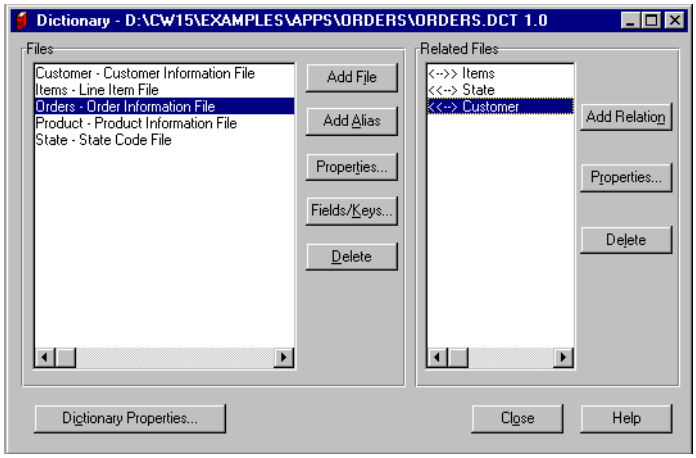
The printed insert you received with the Evaluation Edition includes a special offer to purchase Clarion for Windows. It’s an investment that gives you the power to be a better, faster, smarter programmer—and the Evaluation Edition will prove it!

What is Clarion for Windows? -1-

Clarion for Windows 1.0 was released in the fall of 1994. It was the first Rapid Application Development system for Windows featuring an optimizing compiler. Its replaceable database drivers and full-featured data dictionary support powerful database applications for either local or Client/Server environments. The Clarion language, the only true Fourth Generation Language in a Windows RAD tool, is easy to master, and supports easy code maintenance.

What is Clarion for Windows? -2-

The Evaluation Edition features Clarion for Windows 1.5, released in the Summer of 1995. It adds a 32-bit compiler, and many new productivity features, including the Application Wizard. The Application Wizard builds full-featured applications based the database description stored in the data dictionary.



What is Clarion for Windows? -3-

The Clarion for Windows Application Generator creates powerful, fully customizable applications quickly, through the use of templates. Clarion Templates provide functionality equivalent to the base object classes in object oriented programming environments; they contain pre-written data and code. Clarion templates, in addition, include another design-time layer consisting of a user interface, enabling the Clarion developer to customize their properties and functionality without knowing anything about underlying code. Clarion programmers can easily create their own templates. Clarion developers enjoy the benefit of high code reusability, usually associated with OOP development tools, yet also enjoy the gentler learning curve of a traditional structured programming language with a seamless, built-in database grammar.

What is Clarion for Windows? -4-

As a development environment, Clarion Templates distinguish Clarion for Windows from other RAD tools by the depth of functionality contained within the templates. Other RAD tools feature visual design tools which allow the developer to place a user interface control, such as a button. Yet the developer must write code to implement the behavior required when the end user, for example, presses the button. Clarion templates contain both the user interface control and the executable code to make that control a complete business solution, such as a file lookup or a record locator. In effect, Clarion for Windows is “the next level” in Rapid Application Development. It requires significantly less coding than other RAD tools when used to create business-oriented applications.

What is Clarion for Windows? -5-

Clarion also distinguishes itself by its dynamic links between the data dictionary and application generator. Other tools that rely on a database or app wizard work only one way—the wizard sets up the application, but cannot be called a second time to change an option. With Clarion, you can change an option in the data dictionary any time. For its entire life cycle, a Clarion app is automatically updated whenever its data dictionary changes. Besides pre-formatting controls, you can also modify Referential Integrity Constraints, Field Validity rules, and other options. If you need to change the database design, (sometimes a nightmare with other RAD tools), with Clarion you simply add a field to the dictionary, open the window you want to display the new field in, then populate it. Clarion even generates an executable to convert your existing data file to the new structure.

What is Clarion for Windows? -6-

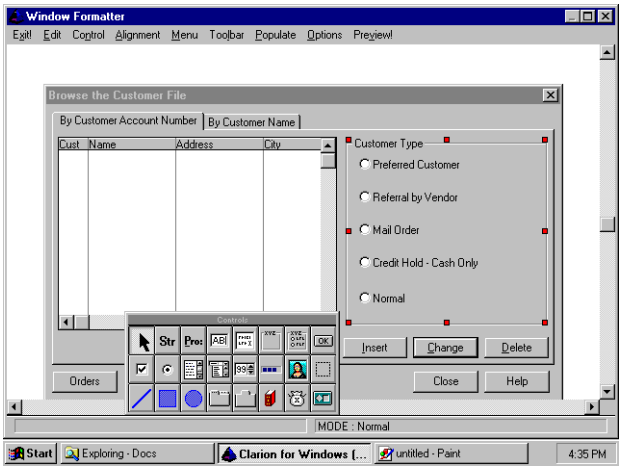
Clarion for Windows also supports the use of various Windows 95 controls, as well as .VBX controls—no matter whether the end user is running Windows 3.x, Windows 95, or Windows NT. For Windows 3.x, Clarion clones the resources necessary to support controls such as tree controls, tool tips, progress bars and others. For Windows 95 and Windows NT, it provides a special .VBX server, which runs a 16-bit .VBX safely and reliably in the 32-bit Windows environment.

Applications developed with Clarion for Windows are small, fast, stable, and royalty-free.

That’s just the short description. Here’s how you can apply the Clarion solution to your development needs—how you can work better, faster, smarter.

Built-in Productivity

- **Visual Design Tools** help you create your user interface quicker. Many RAD tools let you design windows and place controls just as easily. Where Clarion for Windows surpasses the competition is the amount of functionality that comes built into the templates.



Built-in Reusability

- **Reusable Components**—Clarion Templates—mean less code for you to write. A recent Infoworld column (5/8/95) described the broken promises of some other RAD tools, specifically, just how much code you have to write to create an app that actually does something: “but once the controls are laid out, Visual Basic turns into Manual Basic, and the oracle at Delphi starts predicting a long coding session in your future.” (Nicholas Petreley, p. 103).

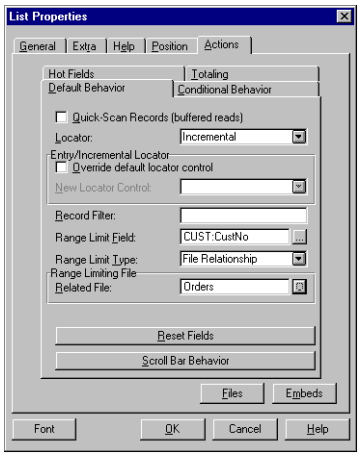
Be More Productive

Clarion templates provide pre-packaged, easy-to-use, easy-to-customize functionality. Common aids for business applications, such as lookup tables and record locators, require zero coding. Clarion requires significantly less coding than other RAD tools when used to create business-oriented applications.

The high degree of template functionality helps you hit the ground running. The tutorial application, for example, loads existing data files into a data dictionary. You simply define the relationship. Then you use the Application Wizard. With one button press, you create a full featured application, including browses, update forms, and reports. Developers who have never seen Clarion before can create this app from scratch within one hour after tearing the shrink wrap off the package.

Write Less Code

You set template properties by right-clicking an associated control, then checking a box, or choosing from a list; not by looking up obscure syntax in a language reference manual. You get complete access to template properties in a “come-and-get-you” interface.



Never Write Code Twice

The application generator evaluates the template options you specify, then generates Clarion language code, which is compiled to create your executable. With a reasonable investment of time, you can add to the standard template set by writing your own custom templates to provide solutions to your very specific business needs. You never have to write the same code twice. Maximum code reuse means higher productivity—for you, and for other members of your programming team.

Apps That Perform Better

- **Optimizing Compiler**: it's a seamless path from prototype to production; from database design to deployment. No more having to use one tool to produce a prototype, and another for the actual application. Clarion's optimizing compiler produces applications as fast as any 3GL—in a fraction of the time. We use the same "back-end" compiler for our C, C++ and Modula-2 products—the .OBJ files created by Clarion are indistinguishable from those created by our other language products. You get conditional compilation and smart linking, helping you create "lean, mean" executables or dynamic link libraries, royalty-free, and callable from other Windows applications.

Apps That Perform Better

- **The Clarion Language**: there's a rich, powerful language underneath—and you have complete access to it. Clarion is a structured, compact and expressive fourth generation language. It's easy to learn. If you are fluent in any conventional programming language from COBOL to Xbase, you can read Clarion.

Your Code Works With the Template

Each template contains many points at which you can embed a Clarion language statement (or a long block of code, for that matter). If a template contains a window, you can add your custom code "before opening the window." Your statement is placed into the generated source code, in-between lines generated from the Clarion template. You'll never "hit the wall" because you have complete control—you work with the code; not around it.

The Clarion language provides the ease of development and gentle learning curve of a true fourth generation language. Yet it's extensible and flexible enough to support any call to the Windows API, or to any dynamic link library. Then the Clarion for Windows optimizing compiler takes your code and creates Intel machine code.

Maintain Projects More Efficiently

- A professional strength data dictionary gives you power, flexibility, and connectivity. The data dictionary and the application generator work together—make a change in the dictionary, and the application generator automatically updates your application.

The templates support referential integrity as many levels deep as you specify in the database dictionary. Choose a replaceable database driver to connect to standard PC data file format, Btrieve Client/Server, or ODBC for other Client/Server DBMS's (AS/400, Oracle, and other C/S drivers are available separately). You can import file definitions from existing tables or files.

Maintain Projects More Efficiently

- Template options are always available. Unlike one-way wizards, you can always change a template option, regenerate code, and recompile. You never have to go back to square one when the client tells you to make a change!

- The "readability" of the Clarion language makes code maintenance easier. Whether you need to revise code you wrote six months ago, or you need to look over another programmer's code, the expressiveness of the Clarion language saves time.

Development Environment

When working with the Evaluation Edition , you'll be running the development environment from your hard drive (it requires 8-22MB, depending on the options you select). All the documentation will remain on the CD. After you've finished the evaluation, you can simply delete one directory, and three lines in your WIN.INI to clean up completely (details in the section on Setting Up). These are the development environment components you'll expand and copy to your hard drive:

System Requirements

You can run the Development Environment on any system meeting the minimum system requirements for Microsoft Windows 3.x, Microsoft Windows 95, or Microsoft Windows NT 3.51. For better performance, we suggest your system have at least 8MB RAM, 12 MB RAM, or 16 MB RAM for these respective operating systems.

You'll also need between 8 and 22MB free hard disk space, depending on the Setup options you select.

The applications you develop will run comfortably on even those machines meeting only the minimum requirements for these operating systems.

Visual Design Tools

- ✓ The **Window Formatter** supports all standard window controls, plus .VBX controls, with direct connections to variables or database fields defined in the data dictionary. It supports two-way development—you can flip between visually editing the controls, or source code editing.

- ✓ The **Report Formatter** allows you to visually create and customize reports. It supports preview, multiple grouping, and multiple details. No additional runtime modules or external report writers are necessary. Reports are compiled into your executable, and it all appears seamless to the end user

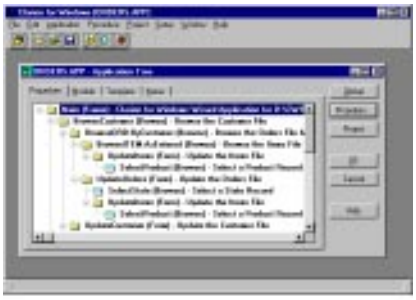
Application Generator/Templates

- ✓ Tried and true **pre-written code**, an intelligent interface for customizing it, automatic links to the database dictionary, and a smart code generator that generates only the code needed to support the functionality you select.

- ✓ The **standard templates** include a browse (listbox page-loaded from database records, using filters or range limits); update form (updates field data upon insert, change or delete record); frame (an MDI frame, with automatic window, menu, and toolbar management, plus thread capability to support multiple record buffers); and report (a default report with automatic support for collecting and formatting data from a database, including print preview.

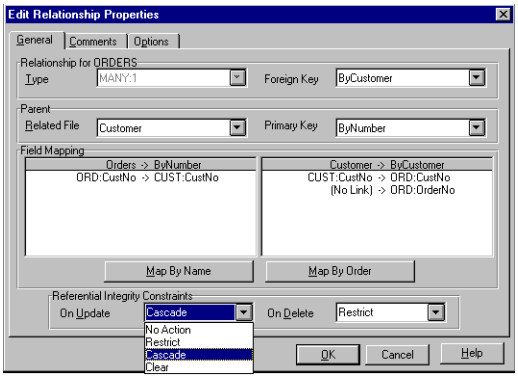
Application Generator/Templates

- ✓ The **Application Tree**—a logical procedure call tree—is the center of the Clarion for Windows development environment. It provides a hierarchical list of the procedures that make up your program. As you add functionality, for example by defining a new menu item, a new procedure is added and marked as “To Do.” The Application Tree both describes and organizes your project.



Database Dictionary

✓ The Database Dictionary organizes your application's data files, their relationships, referential integrity constraints, validity checking rules, even pre-formatting controls referencing database fields. Changes made to the dictionary update the application.

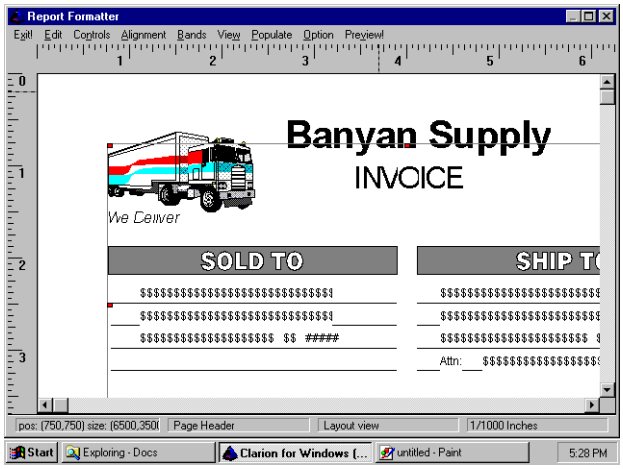


Database Drivers

✓ The drivers are compact dynamic link libraries; for a given file format, you ship the .DLL (typically 100K) with the application. The drivers that come with the product include ASCII, BASIC (comma or tab-delimited), Btrieve, Clarion, Clipper, dBase III and IV, FoxPro, DOS (binary), ODBC, Paradox, and TopSpeed. SQL drivers are available separately.

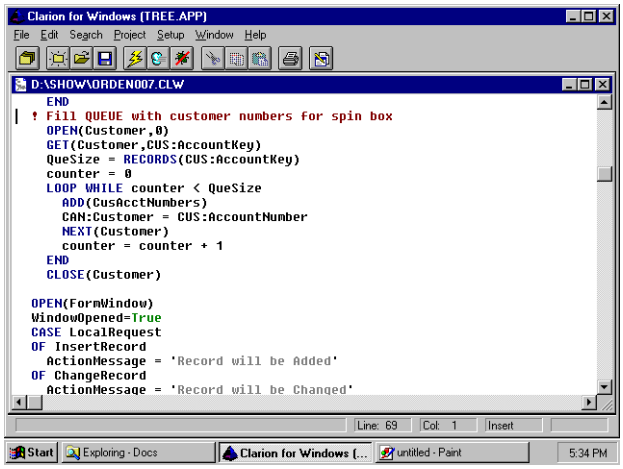
Reports

✓ Built in programmable report writer creates reports which are compiled into the executable. Works with Report Formatter and report template. Supports complete control over the Windows printer device settings.



Clarion Language

✓ Clarion is a structured, compact, expressive fourth generation language. It's easy to read, easy to write, and easy to learn.



Clarion Template Language

✓ The Clarion Template Language is a meta-set of the Clarion language. It controls the development environment, gathers design specifications, and generates source code. In addition to using the standard templates, you can write your own templates, or purchase additional templates from third party vendors.

Compiler/Project System

✓ The compiler produces small, fast, royalty-free executables. Most Clarion applications fit on a single high density disk (database excluded). They run comfortably on a 4MB machine. The project system features smart linking (deleting unused data and code) and smart method linking (deleting unused virtual methods—the Clarion library is object-oriented). Clarion applications load faster than competitive applications, which may require megabytes of runtime support for database access or reporting, even if the executable is compiled. And as a compiled executable, Clarion for Windows apps execute literally tens of times faster than interpreted applications. And, of course, you distribute your applications royalty-free.

Debugger

✓ Two fully Windows-hosted, full featured debuggers (one for 16-bit, one for 32-bit) include support for conditional breakpoints and watchpoints, and a spy window for Windows messages.

Some features that are extra in other RAD systems are standard in the Clarion for Windows debugger, including assembly language listings, machine register views, debugging support for dynamic link libraries, and breakpoint support for the debug version of Windows.

Documentation

✓ Over one thousand pages including a comprehensive User's Guide, Language Reference, Template Language Reference, and Getting Started tutorial. For the Evaluation Edition, we've provided all this to you with the Adobe Acrobat Reader. Additionally, you'll find comprehensive on-line help.

How Should You Evaluate?

The CD contains a special version of Clarion for Windows. Its only limits are splash screen disclaimers on the applications you develop, and a limit on the data files you can edit. If you open a file or table, from any data source, with more than 100 records, you'll automatically open it in Read Only mode. Any less, and you can edit records normally.

So go ahead, judge us on the basis of what you see here.

How Should You Evaluate?

Here are a few questions to bear in mind as you evaluate:

❏ Can you create an application faster than with your present development tools?
❏ Which tool creates the faster app?
❏ Which tool creates the smaller app?
❏ Which app costs less to distribute?
❏ Which app is more stable?
❏ Which development environment is more stable?
❏ Which app requires less resources?
❏ Which source code is easier for you to read?
❏ Which source code is easier for others to read?

—more—

How Should You Evaluate?

Here are a few more questions to bear in mind as you evaluate:

- ❏ Which development environment supports greater code reusability?
- ❏ Which development environment allows you to organize and control your project best?
- ❏ Which development environment best connects to your data?
- ❏ Which development environment has the strongest database dictionary?
- ❏ Which development environment gives you the overall speed and flexibility you need to create the best solutions to a wide array of your end users' needs?

This Guide

- This section introduces Clarion for Windows, summarizes its features, discusses your evaluation process, and provides instructions for setting up the Evaluation Edition.
- Section two, the tutorial, helps you build and customize a simple application.
- The third section provides more information about Clarion for Windows. This includes the development environment architecture, the general design of Clarion for Windows applications, plus discussions of the Clarion language and Clarion template language.
- The fourth section discusses how Clarion for Windows compares to the competition.

Setting Up

The Setup program, in the root directory of the CD, decompresses and copies the Clarion for Windows Evaluation Edition files to your hard drive:

- It provides options for installing various components such as example files.
- It asks before updating the PATH statement in your AUTOEXEC.BAT file to include the Clarion for Windows directory.
- It installs icons for the development environment and Acrobat Reader documents (the latter remain on the CD).

Starting Setup

To start the Clarion for Windows Setup program:

1. Insert the CD into your CD-ROM drive.

2. From the Start menu, choose Run, or from Program Manager, File Manager, or other shell program capable of launching a program, choose File ➤ Run.

3. Type D:\SETUP (where D: is the letter of your CD-ROM drive) in the Run dialog, and press the OK button.

The Setup program provides an introductory screen and other text information.

Setup Options -1-

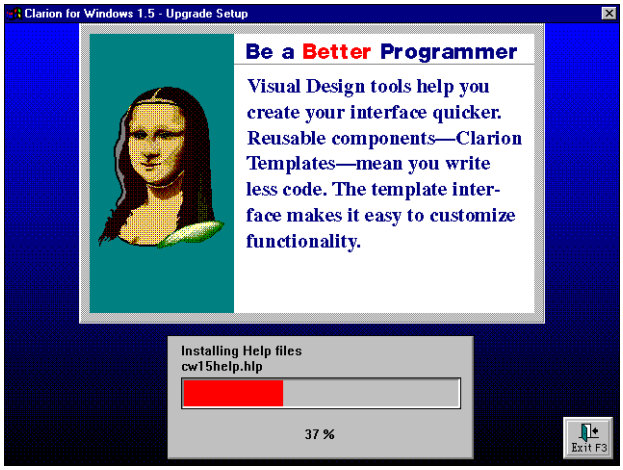
1. Choose the Setup options by checking on the boxes you want to install, then press the OK button.

2. Specify the target directory, then press the OK button.

Setup will install the main components of the IDE to a BIN subdirectory one level below the target directory you specify in the dialog. The Clarion for Windows Setup program installs all files to the target directory, and subdirectories beneath it. It installs no files to any other directory.

Setup Options -2-

During the installation, progress bars will display as Setup copies the files.



Setup Options -3-

3. Choose Yes or No when Setup asks whether to modify the PATH for you.

The Clarion for Windows IDE requires that the BIN subdirectory be listed in the PATH environment variable. If you choose No, you must edit the AUTOEXEC.BAT file manually. The only other change to any of your system files is that Clarion for Windows appends its own section to WIN.INI when you run it for the first time; this section is only a few lines long.

Setup Options -4-

4. Choose Yes or No when Setup asks whether to display the ReadMe file.

If you don't wish to read it right away, you'll find an icon for it in the Program Manager group which Setup creates for you. We recommend reading it as soon as Setup has copied all the files.

Setup Options -5-

5. Follow the directions for installing Adobe Acrobat Reader and/or Video for Windows.

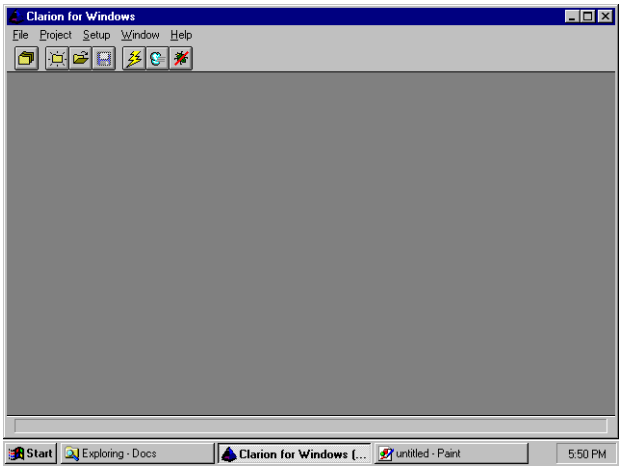
These are separate setup programs provided by Adobe and Microsoft, respectively. They provide support for viewing documentation and demos on the CD. They were the most up-to-date versions available of both these products, as of October, 1995. The Microsoft Video for Windows setup is only necessary for Windows 3.1 users.

6. Press the OK button when Setup is done.

Starting Clarion for Windows

To run Clarion for Windows, locate the Clarion for Windows icon in the Clarion for Windows program group, and double-click it:

The Clarion for Windows IDE appears, ready for you to begin work.

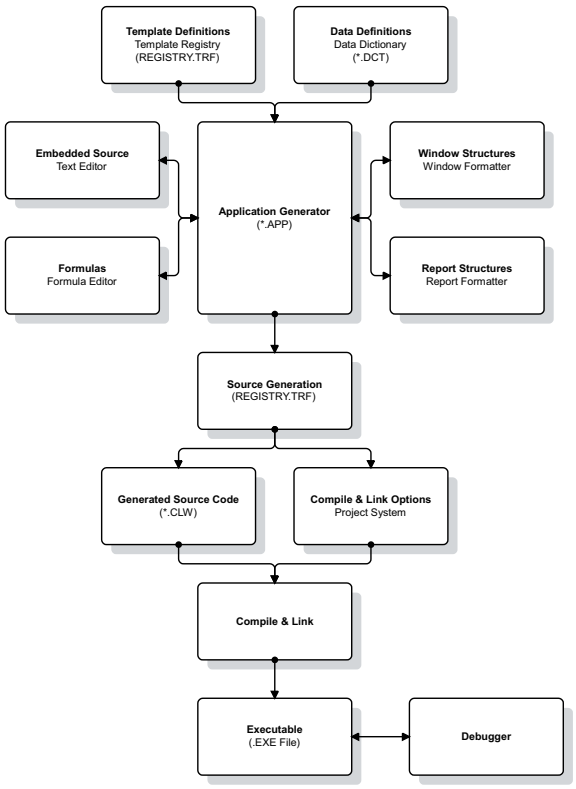


Architecture

This section describes the parts of the development environment and how they work together. The development flowchart on the next “page” visualizes it all.

After you view the flowchart, we’ll explain how the Clarion for Windows architecture benefits you and your development projects.

Development Flowchart



High Performance Engine

With over ten years of evolution and experience both as a 4GL language, and as a developer of highly regarded compilers, the Clarion language and backend TopSpeed compiler provide the foundation for the Clarion for Windows development environment. Most of the interface is written in Clarion. The compiler, debugger and other basic components are written in TopSpeed languages (C, C++ and Modula-2).

TopSpeed CORPORATION

Vision

The vision behind the TopSpeed compiler line is a generic back-end compiler. Each language exists as a front-end or “surface.” The project system processes the source code modules for each “surface” language, then generates intermediate symbols and tokens, and forwards them to a common backend compiler. The .OBJ files are the same, no matter which language provided the “surface.” This provides a seamless integration for multi-language projects.

4GL With 3GL Performance

More importantly, it enables a developer working in a fourth generation language to create applications which run as fast as those written in a third generation language such as C. The advantage, of course, is in the fact that 4GL development is more suited to business applications which typically require shortened development schedules and “canned” support for special needs such as binary coded (fixed point) decimal math (typically required for accurate multiplication and division of currency values).

Code Reuse Made Easy -1-

Over the last few years, objects have been increasingly portrayed as the modern road to code reusability. Yet every object oriented language is also admitted to have a steep learning curve, without exception. Programmers and businesses have had to decide whether the investment of a year’s time per programmer to learn an OOP language (this is the time period commonly allotted by C programmers to learn C++) is worth the return in “reusability.”

Code Reuse Made Easy -2-

But there are discrepancies between theory and execution. PC Week (12/26/94), in an article entitled “The picks and pans of 1994,” listed “the C++ Programming Language” as number one in the “Biggest Letdowns of the Year:”

“With syntax so chaotic that even compilers have to guess at it, C++ code had better be reusable, because no one will ever want to reverse-engineer it. The programming language’s “feature”—being a superset of C—is a fundamental bug. With numerous large projects being written in already obsolete dialects, C++ is arguably an “instant legacy” language.”

Bottom line: what good is code re-use if you can’t figure out what the code is doing in the first place?

Code Reuse Made Easy -3-

If the new class created by one programmer needs to be debugged or adapted, how does one programmer pick up another programmer’s work when it takes an inordinate amount of time to read the first programmer’s work? Even the original programmer may have difficulty picking up a project he or she put down six months ago. Additionally, when an OOP programmer creates a new object based upon a

previously created class, the object includes all the functions and other baggage from that previous class. This is compiled into a “code-bloated” executable that is then liable to be called “fatware.”

Code Reuse Made Easy -4-

This is where Clarion, its Application Generator, and Clarion templates shine. Anyone can use your code again and again when you create a template—because your template has its own user interface. You specify the data or options necessary to make the code work. The template collects them for you, and the application generator writes source code specific to the options chosen. The compiler creates an executable which supports only the functionality the developer wishes, no more.

Clarion Code Reusability -1-

Clarion’s approach to code reusability is built on the Clarion language, and the Clarion template language. The statements in a template include template symbols which link to the dictionary and resolve at code generation time, for example, to a database field name. Template statements declare the controls in the template design-time interface. These then capture developer input and store it in template symbols which are variables maintained by the application generator. Template control statements determine what code to generate based on the value of the symbols. The template language itself, being a meta-set of the Clarion language, is only slightly less expressive than Clarion, and quite readable. You’ll find more on the Clarion template language in the language section.

Clarion Code Reusability -2-

Therefore, Clarion’s overall approach to code reusability is this: an intelligent, flexible set of instructions that gathers input from the developer, reads the database dictionary, then writes Clarion language source code to implement the tasks that the developer wants the application to do. We compile only what the developer wants to do; there are no extraneous functions in the executable, as is typical with OOP compilers.

❏ Which development environment supports greater code reusability?
✔ Clarion. Clarion templates are intelligent and flexible.

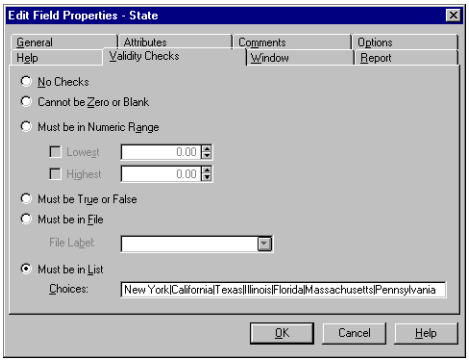
Database Dictionary: Pre-Planning

It’s become popular for RAD tools to offer database wizards instead of full-fledged dictionaries. Yet wizards are typically one-way tools which don’t evolve with your application: wizards generally set up a collection of object properties which you then have to maintain. The Clarion Database Dictionary allows you to add or edit files, fields, relationships, Referential Integrity constraints, and many other properties as you go.

With a well-planned dictionary, an application practically writes itself. Clarion templates make full use of the Application Generator hooks to the dictionary. Your RI constraints, field validity rules, and pre-formatting for controls—all the way down to what color, font size and style a prompt for an entry field should be—can be set in the dictionary.

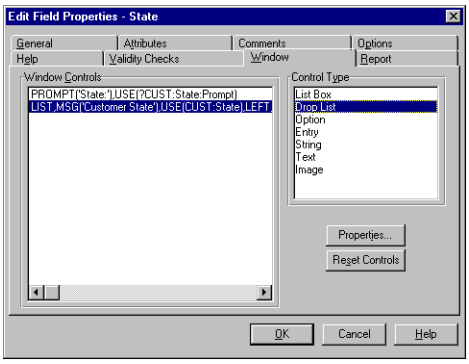
Dictionary Pre-Formatting -1-

Clarion for Windows goes far beyond the “standard” support for validation rules, such as requiring a value entered by the end user fall within a numeric range. Developers, for example, can specify a value must exist in a related file, or be an item in a list. When reading the database dictionary, the Application Generator even knows to create a list box control to display the choices if the developer checks a box.



Dictionary Pre-Formatting -2-

Dictionary options can be used in multiple procedures, or multiple applications. Because the same database dictionary can be used for multiple applications, the developer gets “more bang for the buck,” formatting the controls for many applications in a single step. And, the formatting options are “live;” if you change an option in the dictionary, it updates the control in your application.



Dictionary Pre-Formatting -3-

All the fields you define in the database dictionary are accessible to the window and report formatter. You can simply place an entry control (edit box) in a window, and select a data file and field from the file schemata window. The template generates the code to access the value from the current record, or write a change to it. Similarly, using the report formatter, you can select a field whose contents will fill the string control you place, cycling through the data file and printing the value in the current record for as many records that meet the criteria you set using the template interface.

International Custom Sorts

Clarion for Windows also supports user defined sort sequences. This provides extremely strong support for shared international applications utilizing different alphabets or accented letters. Clarion may be the only Windows development environment in the world with which you can support two users, working side by side accessing the same database, the same tables and keys, in different languages, with each seeing alpha sorting in the precise order expected in each language.

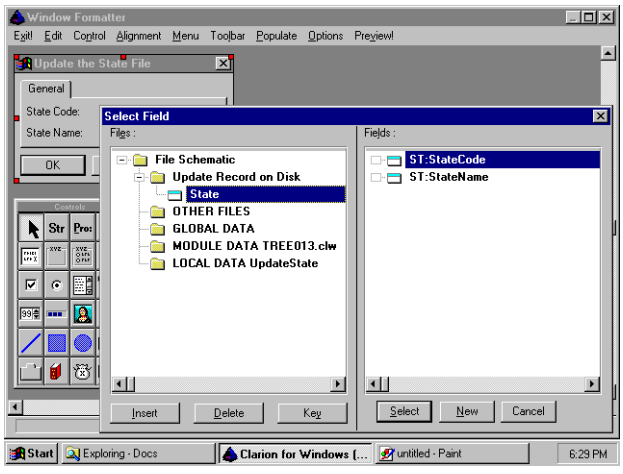
❑ Which development environment has the strongest database dictionary?
✔ Clarion. The active link between the database dictionary and application generator provides strength, flexibility and connectivity.

Visual Design Tools: Fast Development

Window and Report Formatters cut the time required for building your user interface and reports by generating the data structures necessary for the compiler to create them. As you edit the controls graphically, re-sizing or otherwise changing their appearance, the attributes of the data structures change accordingly. You can also edit them at the source code level, and when you next open the Window Formatter, it reflects the changes.

Placing Database Fields

The visual design tools are tightly integrated into the Application Generator. When you place a control, you can reference a database field or memory variable. The reference is stored in the control's data structure, with an attribute called its USE variable.



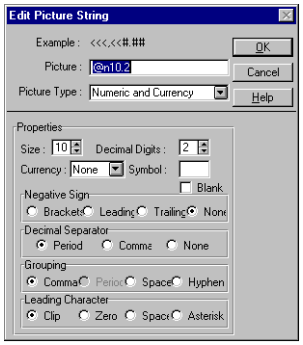
Visual Design Tools -2-

The control loads the correct value into itself, when the window opens, or whenever the DISPLAY statement is encountered. If the end user changes the value in the control, another statement updates the database field or memory variable.

Additionally, Clarion supports PICTURE strings to format the values in the controls, so that they appear in the way the end user expects them. If, for example, a database field called ExtendedPrice contains a value of 10736.77, the picture can automatically format its appearance in an edit box as $10,736.77. You can specify a date or currency picture to automatically take the same format as the default specified in the end user's WIN.INI file.

Visual Design Tools -3-

Additional control attributes can specify read-only, password (typing appears like ****), upper case, and others. These can all be stored for you in the database dictionary, by field, or custom placed from within the Window Formatter.



How They All Work Together -1-

The template symbols, which you never see unless you want to, provide the glue that holds it all together. They not only provide the field names from your database, but additionally provide the "extras" the database dictionary can set for you, like default column headings for a report.

The Application Generator is the linchpin. It reads the template registry, presents the template interface to you, opens the door to the Window and Report Formatters and the Formula Editor, maintains the Application Tree, presents database and pre-formatted options you've set in the dictionary, and finally, gathers all your work together and generates the Clarion language source code modules, inserting any custom code you've written into the appropriate places.

How They All Work Together -2-

The Application Generator then forwards these neatly wrapped bundles to the project system, which calls the compiler and linker. From there, you debug it or just run it. When you're ready to send it off to your end user, the application generator, via the default templates, has even left you a text file (appname.SHP) which provides a list of files you need to include—the .EXE file and any required .DLL's.

How They All Work Together -3-

It's organized—not hidden behind "a thousand doors." It's logical—your project is organized according to the procedures you define—not a gordian knot of user interface objects. It's compact—why do you need a

2.5MB database engine when a 100K driver .DLL will do? It's tight—by all means, please examine a Clarion for Windows compiled application with Bounds Checker or a similar tool—your Clarion applications deliver absolute reliability, without the GP faults often found in script apps created by competitive RAD tools.

How They All Work Together -4-

Clarion for Windows is a business tool. Unfortunately, over the years developers have come to expect slow, lackluster performance from 4GL's and business-oriented development tools. Clarion for Windows presents a RADical departure from your previous expectations.

How You Make It Work

This section presents a simplified overview of the steps for creating a typical Clarion for Windows Application. The Development environment contains seven main functional parts, all of which are accessible from the others. This section provides a description of each, in the order that a typical programmer might encounter them.

Each part contains dialog boxes which the programmer fills out to "describe" the Application's functionality to the Application Generator. Programming Clarion for Windows is in many ways a "walk through" a series of dialog boxes. There's no mandatory sequence in which you must "fill in" the dialogs, though you do need to create some files before others. If you know which dialogs do what, it makes building your application that much quicker.

The Dictionary Editor -1-

The Data Dictionary (a .DCT file), maintained by the Dictionary Editor, holds a description of the database, including its files, driver(s), fields, relations, field validation rules, and referential integrity constraints. It's the first file you create when you design your application. You can create the file definitions "from scratch" (using Quick Load or not), or import definitions from existing data files. The other parts of the IDE look up the options you set in the dictionary to let you, for example, easily place data fields in a dialog box you design for the end user. The Application Generator creates code for all the statements that access the data files based on how you construct the Data Dictionary.

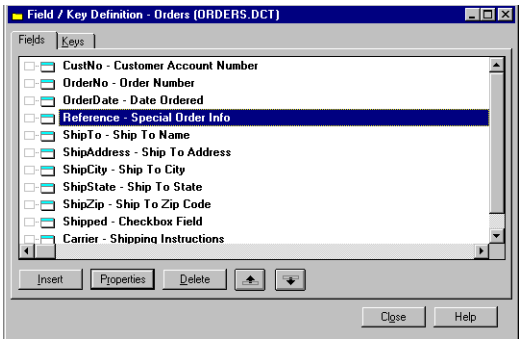
The Dictionary Editor -2-

Start a new dictionary with the File ä New command, then select Dictionary. This leads you to the Dictionary dialog. Define your application's data files, aliases, and views in this dialog. It also shows the relationships between files.

Buttons lead to the New File Properties, the New File Alias, the New File View, and the New Relationship dialogs. Specify the name and file driver for each data file, one by one, in the New File Properties dialog. It also allows you to set options such as Threaded, which specifies that each execution thread accessing the file gets its own record buffer. This is useful for MDI applications.

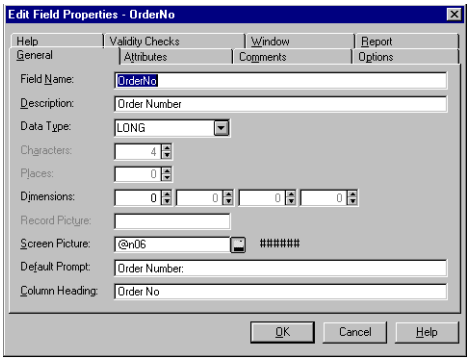
The Dictionary Editor -3-

From the Field/Keys Definition dialog, press the Insert button to specify fields, keys, and index files. All the information is arranged hierarchically.



The Dictionary Editor -4-

Define fields, their data type and size in the New Field Properties dialog. You can pre-define control properties, such as text justification. Two buttons provide shortcuts to all the properties dialogs in the Window and Report Formatters. You can also “back up” to the previous dialog to define keys and relationships.



The Dictionary Editor -5-

Specify the key components in the Key Components dialog. Clarion for Windows automatically builds the key correctly even if you specify mixed field types, such as a string and a decimal. From here, you can “back up” to the Dictionary dialog to define relationships.

Define relationships in the New Relationship Properties dialog. You can also specify Referential Integrity constraints from controls in this dialog.

With the major parts of the dictionary defined, you save the dictionary and move on to the .APP file.

The Application Generator -1-

The Application Generator generates your application's code, based on the predefined templates you pick from the template registry. It allows you to add global and local memory variables, and customize the procedures with embedded source code.

The Application Generator also provides access to other parts of the IDE to customize the look and functionality of the windows, menus, reports and other user interface elements.

The Application Generator -2-

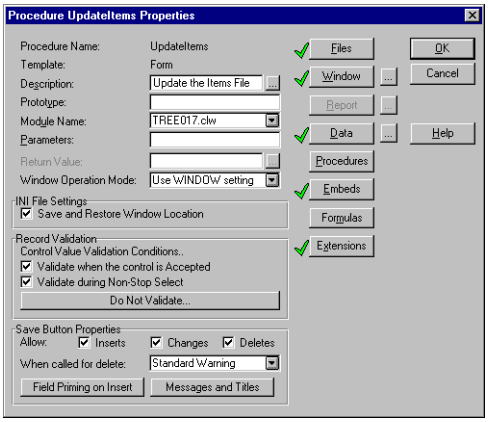
Start a new application with the File ➤ New command, then select Application. You set the basics—application name, data dictionary name, help file and the application template—in the Application Properties dialog. This creates the .APP file and displays the Application Tree.

View and maintain the parts of your application in the Application Tree dialog. It hierarchically displays your application's procedures, and marks the ones still to be defined as "ToDo." A button press accesses the Select Procedure Type dialog.

Select functionality for a "ToDo" procedure in the Select Procedure Type dialog. Procedure templates such as Browse and Form appear in a list. The Select button brings up the Procedure Properties dialog.

The Application Generator -3-

The Procedure Properties dialog is the hub for all the other dialogs that let you customize the procedure so that your application does the job the way you want. Press the Global button to define global memory variables, or press the Data button to define local memory variables.



The Application Generator -4-

Define and set the order the program initializes local memory variables in the Data dialogs. Press the Insert button to define variable name, type, size etc., in a dialog box identical to the New Field Properties dialog.

Press the Embed button to display the Embedded Source dialog. This allows you to insert custom executable code at points before, during, and after the procedure, or on window and field-specific events. Select the point and press the Add button.

After you've customized the procedure template using the Window Formatter, Report Formatter and/or Text Editor, you can return to the Application Tree and generate the code!

The Window Formatter -1-

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

When using the Application Generator, you'll call the Window Formatter by pressing the Window button in a Procedure Properties dialog, to customize the window or dialog box.

The Window Formatter -2-

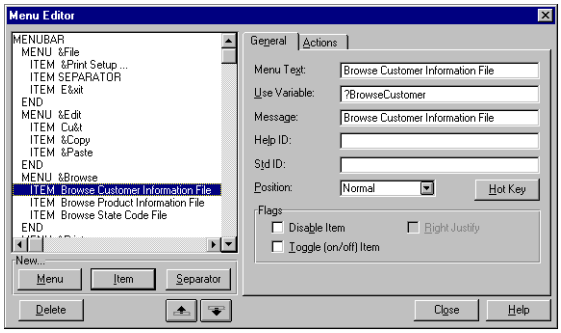
The Window Formatter provides a view of the window under development. Click in the toolbox, then click in the window to place a new control. You can then press the Test button to see exactly how the window appears to the end user.

Each window and each control in the window has an associated property dialog that controls its appearance, and if an entry field, its contents. You select the window or anything in it, then press the Properties button. The Window Properties dialog sets basic elements such as system menus, caption, etc.

The Window Formatter -3-

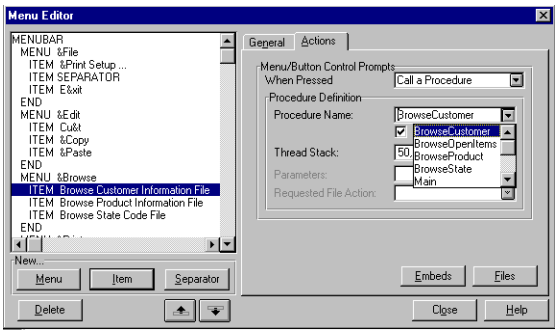
A typical control property dialog sets options such as a label, its field equate label to reference it in executable code, or if an entry box, a field or variable name to reference its contents.

If the window has a menu, choose Menu ➤ New Menu to call the Menu Editor. Edit the menu text, add new items with the Add Item button, and specify menu item functionality by pressing the Actions button.



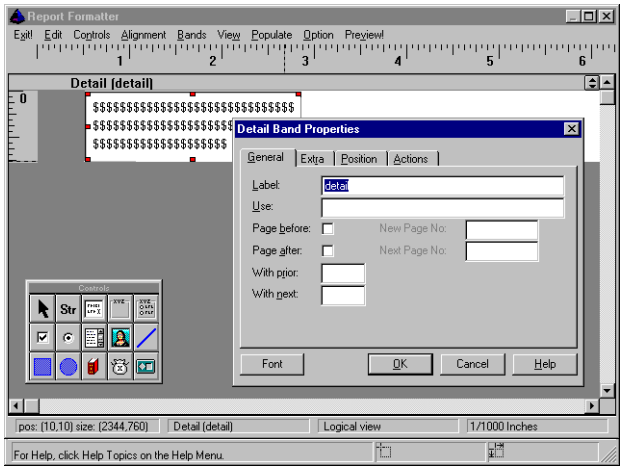
The Window Formatter -4-

Using the Actions sheet for a menu item, you can associate a procedure call, so that when the user selects the menu command, it executes the procedure you name.



The Report Formatter

Using the Report Formatter, you place controls in bands. At run time, the print engine processes the records you specify, printing the fields you specify in the Detail, or other bands. The Report template automatically processes your files, filling the controls according to the key you specify.



The Text Editor

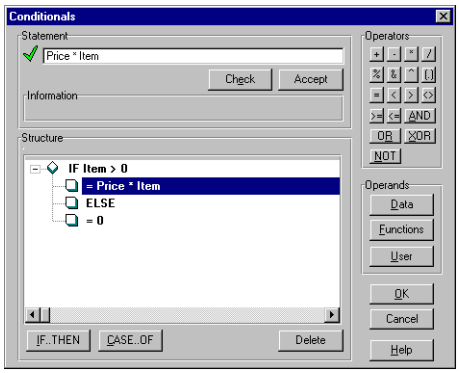
The Text Editor is a full-function programmer's editor in which you can write source code. Most likely, when using the Application Generator, you'll call the Text Editor to create embedded executable source code to further customize the way a procedure operates.

The 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.

You can create a window or report structure with hand code, or press a button to call the window or report formatter to edit it graphically. You can flip between graphical and source code editing.

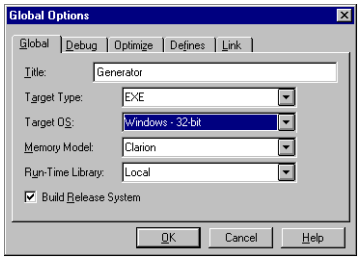
The Formula Editor

The Formula Editor helps you to quickly generate a statement resulting in a value. You can use the Formula Editor to assign values to computed fields, conditional fields, and record filters.



The Project System

The Application Generator automatically creates the project file for the application. The project file contains compiler and link options, such as whether to include debug code, optimization choices, and so on. You set whether your project is for 16 or 32-bit Windows by pressing the Project button, pressing the Properties button, then choosing the target OS.



The Project System

The Project Tree displays the source code files, libraries and other external files for the project. Press the Properties button to set specific compiler and link options.

More likely, when creating an application using the Application Generator, you'll just press the Make button to compile and link the application. The only Project system dialog you'll encounter is the Compile Results dialog, which is one way to access the Debugger.

The Debugger -1-

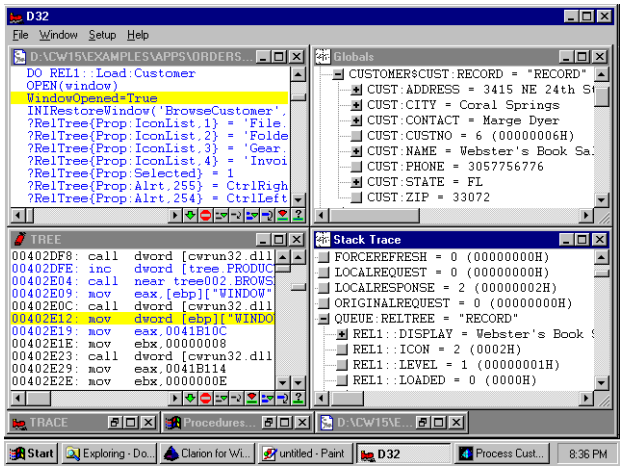
Debugging a program usually requires running the program and repeatedly stopping it to examine the value of different variables. The Debugger consists of a number of windows which display source code, variable contents, active procedures and more.

To view a source code window, you'll tell the project system to include debug information in the .EXE file (this is the default setting), then start the Debugger by pressing the Debug button in the Compile Results dialog, or by choosing Project ä Debug.

The simplest way to debug your application is to identify the part of the program that you think is producing the bug, and set a breakpoint, using the Breakpoint dialog, at that part of the code.

The Debugger -2-

You can then run the program, and the Debugger will suspend it at that point so that you can examine the value of the variables. This will (hopefully!) help you pinpoint the error so that your application is perfect!



Speed and Flexibility

❑ Which development environment gives you the overall speed and flexibility you need to create the best solutions to a wide array of your end users' needs?

✔ Clarion. You write the least amount of code necessary to do the job, and when you have to write code, it's easier and quicker to write, easier to read when you go back to maintain it six months later. It compiles to a small, fast, tight app which you distribute royalty-free.

A Typical Clarion Application

This section discusses the default paradigm of a typical Clarion for Windows application—specifically the program flow. Clarion for Windows gives you the power to substitute any scheme you wish, so this discussion is only a starting point.

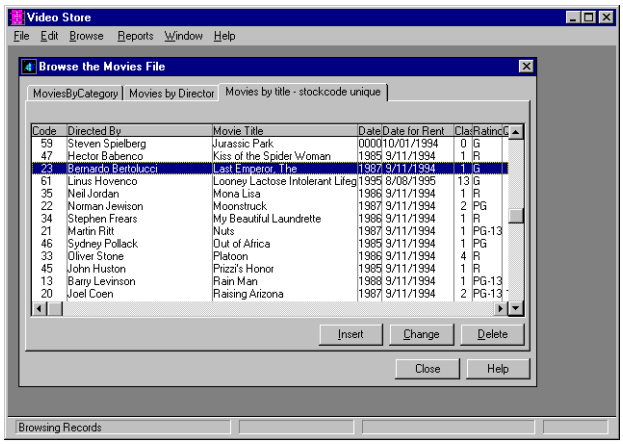
Start With the Database

Good database design helps make an application more coherent. With proper normalization, and well-chosen keys, your application practically writes itself. That's the idea behind the Clarion for Windows Application Wizard. Lay out a file definition, defining keys; CW then generates browse windows, update forms, and reports (one of each), for each of the keys you define. And it provides visual representations of the relationships by providing synchronized list boxes showing child records when the end user clicks on a tab on an update form for the parent record.

That's also the default program flow embodied in the default templates. When viewing and managing data, the typical application presents records to the end user in the following way:

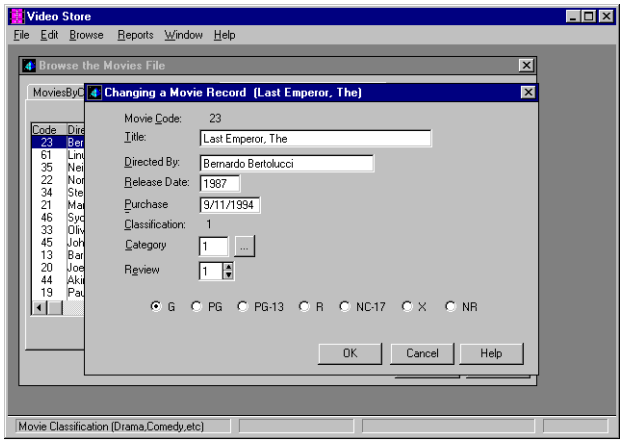
Program Flow -1-

- The end user navigates the database—all or part of it, one data file or multiple related data files—by scrolling through a listbox, within which each item represents one record. The window in which this takes place is called a "browse."



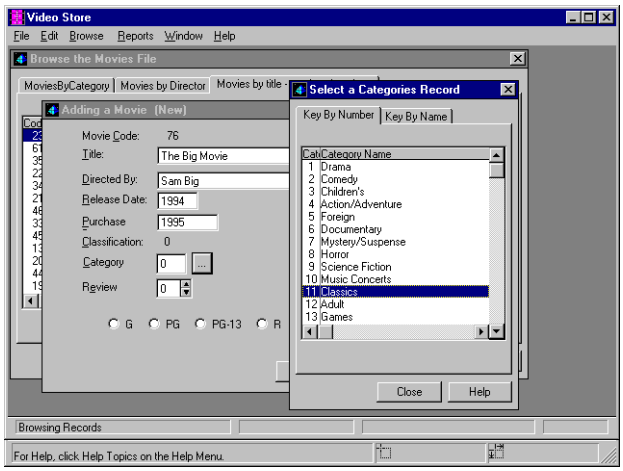
Program Flow -2-

- 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.



Program Flow -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.”



Browse-Form-Browse Paradigm -1-

We call this overall program flow the “browse-form-browse” paradigm. This is the starting point, after which you can add all the reports, customized screens, and specialized procedures that make up your application.

The underlying structure, which we used in the design for the default template set, can be described as a Browse-Form-Browse approach.

Browse-Form-Browse Paradigm -2-

- By Browse, we refer to a simple scrollable list of records maintained in a data grid or listbox. The records can reside in a single data file, include fields from related tables, or even include fields derived from

formulae. As the end user scrolls through the list box, the underlying code navigates the database, usually following the order set by a key.

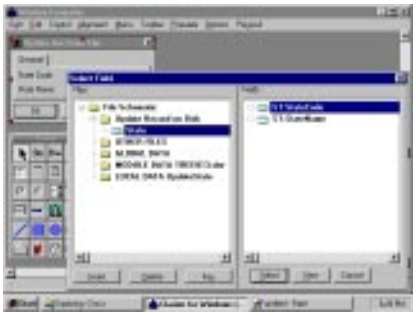
The Browse Procedure

The Clarion for Windows default browse template maintains the browse via a page loaded queue. The code loads as many records into the queue as will fit into the current dimensions of the listbox. A queue is a memory structure akin to an intelligent linked-list, and can be sorted, added to, and otherwise edited.

The Clarion for Windows default browse template also contains four buttons, labeled “Insert,” “Change,” “Delete,” and “Select.” The first three allow the end user to call an update form to add and edit a new record, edit the field values for the record currently selected in the listbox, or delete the record currently selected in the list box, respectively. The “Select” button is for use in look-up procedures, which are discussed below.

The Form Procedure -1-

- The default Form procedure is a simple update form. By default, unless you’ve preformatted fields in the dictionary, the fields appear as edit boxes (ENTRY controls in Clarion syntax). When creating a form with the Window Formatter, you select a field previously defined in the data dictionary, then click to place both the edit box and a prompt control into the window under construction.



The Form Procedure -2-

The “Insert,” “Change,” and “Delete” buttons all lead to the same form procedure. An action message, in the form of a string control appearing inside the window, advises the end user whether the data entered into the update form will update an existing record, provide the data for a new record, or ask the end user to confirm that the record containing the data in the current form should be deleted.

In the first two cases, the end user can type in the data for the fields referenced by the edit boxes, then press OK. The code then writes the changes to the database.

The Record Buffers -1-

In a typical MDI application, the template code automatically creates a new record buffer each time a browse is opened by the end user (even if it’s two child window copies of the same browse), so that if two

separate forms display data from the same data file, any record updates will be safer and more quickly implemented.

The Record Buffers -2-

Additionally, the template code automatically handles concurrency checking to guard against deadlock in cases where multiple users may edit the same record. It uses an optimistic concurrency algorithm which provides good protection no matter what file driver is in use. The code reads the data from a record, loads the data into memory variables, and accepts editing by the end user of the variables. After the end user presses the OK button to close the form, the code checks the record to verify no other station has changed it since last read. Only then does it write data to the file, updating the field values from the memory variables.

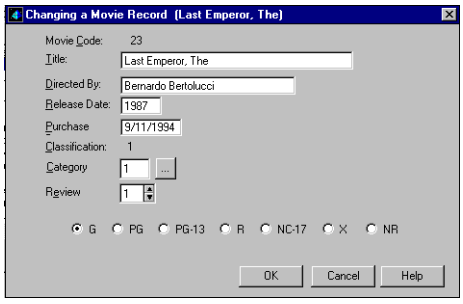
The Lookup Browse -1-

- The Lookup is a generalized term for the process of opening a browse onto a related file, from inside a form, then selecting a record, returning the data from a field in the selected record to an edit box within the form. When the second browse (the lookup; the first browse was the one the form was called from) is opened, the “Select” button is enabled, which allows the end user to choose the currently highlighted item in the list box. The code then fills the edit box from the form that called the lookup with the item.

A browse lookup, therefore, serves the purpose of reducing the typing required by an end user, by presenting choices from a related file. Additionally, it can optionally limit end user choices to those in the related file.

The Lookup Browse -2-

The general convention for calling a lookup from a form has been to place an ellipsis button to the immediate right of the edit box which should accept the lookup value. In various GUI environments, the ellipsis (...) is often used to signify more choices will be presented to the end user in a separate window or dialog, upon the selection of a menu item or press of a button.



Threads and Global Action Variables

Clarion MDI applications automatically support threading. The default templates start a new thread each time the end user opens a browse window. Subsequent forms or lookups called from the browse are

included in the same thread started by the browse. A second call to the same browse (i.e., choosing for a second time the menu procedure that calls a browse) opens a second thread.

Under 16-bit Windows 3.x, thread management is cooperative. A Clarion for Windows application automatically splits its application thread between the child MDI window threads. Under 32-bit Windows, Clarion apps automatically start a separate preemptively tasked thread for each browse opened, when creating a 32-bit application for Win 95 or Windows NT.

Request and Response

Global variables are made available to each thread. In this way, a Clarion for Windows application can signal across threads. The template system incorporates such a system; we refer to it as a Request and Response system. The actual variable names for the “signals” are GlobalRequest and GlobalResponse. Typical standard “values,” which the templates act upon when the “request” is made, are “InsertRecord,” “ChangeRecord,” and so forth.

Within this system, each thread gets its own copy of the global variables, which can safely be changed within the thread without interrupting a process occurring in another thread. To learn more about the system, please read the topic entitled “Request and Response,” in the main Clarion for Windows on-line help file.

The ACCEPT Structure -1-

Once the browses and forms are in place, and the threads started, what actually manages the process of end user interaction with your application? The ACCEPT loop processes all system messages and “accepts” end user input for a given window. There are two general types of messages the ACCEPT loop handles:

The ACCEPT Structure -2-

- When Windows sends a message to the window: for example, if the end user initiates a close-down from the app’s system menu, it’s processed via the EVENT() function. You can optionally control your app’s response, or you can let it be handled automatically.

- If the end user selects a control by clicking in it, or if the end user types a value in a control, that’s processed by the FIELD() function; and again, you have total control.

The ACCEPT Structure -3-

You have no need to worry about the Windows messaging flow, or the handles and pointers that manage its dynamic memory management of your application. The templates automatically generate the ACCEPT structure, plus all the code that implements updating the database with any new data entered by the end user.

How It Works

So how does it all work? OPEN(MyWindowLabel) creates and displays the window. The ACCEPT loop enables mouse and keyboard processing by the end user. If the end user tabs or selects a control with the mouse, types characters then tabs to the next control, or presses a hot key, the ACCEPT gives the program control. It's a simple yet thoroughly effective approach in an event-driven system such as Windows. The end user can interact with the program however he or she likes, yet the programmer, who after all is trying to run a business, maintains control to verify that all the information required to take an order, for example, is entered. Finally, CLOSE(MyWindowLabel) restores the state of the application prior to opening the window.

The ACCEPT Structure -3-

```
OPEN(MyWindowLabel)
ACCEPT
          CASE EVENT()
          OF EVENT:CloseWindow
                    !(user closed window - call routine or procedure to do something)
          END
          OF EVENT:CloseDown
                    !(user exiting Windows - call routine or procedure to do something)
          END
          CASE FIELD()
          OF ?MyButton
                    !(user pressed my button - call routine or procedure to do something)
          END
          OF ?OK
                    !(user pressed OK button - call routine or procedure to do something)
          END
          OF ?Cancel
                    !(user pressed Cancel button - call routine or procedure to do something)
          END
END
```

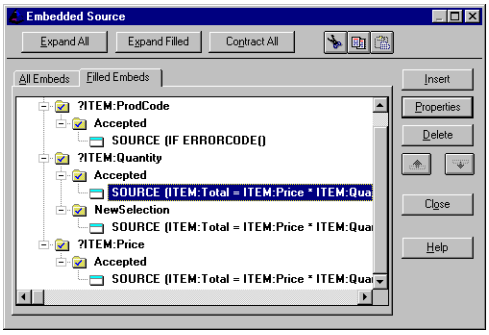
Embed Points -1-

You can optionally embed your own Clarion language code to execute upon any specific event, in response to the end user selecting a control, or when the end user types data in an edit box and tabs to the next control.

Access to these "embed points" is via the Embedded Source dialog. Each control in a window appears in a tree list. Controls generally have two nodes: selected, to execute as soon as the control obtains focus; accepted, to execute immediately after the end user provides input and moves to the next control.

Embed Points -2-

To write your own custom code to execute at one of these embed points, you select the node for control you want, press the Add button, and type your custom code into the text editor. Your custom code is inserted into the ACCEPT loop, in-between source code lines generated by the template, and from there it's compiled into the application. This provides for seamless integration between your code and the template code.



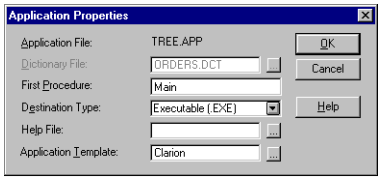
Development Methodology -1-

In the previous section, we listed a “walk through” the dialogs you encounter as you create a Clarion for Windows application. Here’s a more “conceptual” view:

- Create a new database dictionary. This includes importing file structures from existing data files or tables, or defining them from scratch. You can optionally pre-format window and report controls by field. You then optionally set the relationships and Referential Integrity constraints.

Development Methodology -2-

- Create a new application file. You specify the name of the database dictionary, and set project options, such as whether you wish to create an executable (.EXE) or dynamic link library (.DLL).



Development Methodology -3-

At this point, you can either run the Application Wizard, or define one procedure at a time. In this section, we’ll describe the latter method.

- Fill in the first procedure. For most database applications, we find an MDI application frame is the template of choice. Generally, the “starting point” is customizing the menu and toolbar, filling in the menu commands your application requires. For each menu command, you set an action, which in turn provides template code to support a new thread for each child window you expect to open from the app frame.

Development Methodology -4-

- Define global variables. You define variables, and optionally, their initial contents, in a dialog similar to the one with which you define database fields in the database dictionary.

- Customize the appearance and contents of subsequent windows and reports. You can choose procedure templates such as the browse template, with its predefined controls, then use the listbox formatter to specify and format the database fields or variables that appear in the listbox. You can also use a generic window template, then populate it with control templates. This step also includes placing edit boxes or other user entry controls to edit database fields, or implementing other functionality, such as placing a .VBX control .

Development Methodology -5-

- Optionally add handwritten source code to embed points. Generally, the templates provide all the code necessary for maintaining the database. You might add code to manage other aspects of the program. For example, you may wish to store end user preferences in an application's .INI file. In this case, the templates can create the .INI file, even automatically storing window positions. To store additional options, you might add code to write to the .INI file at an embed point such as "before closing the window," for the application frame, and use the Clarion PUTINI function to do the actual writing.

Development Methodology -6-

- Compile, Run and Debug. You can still go back to either the application or database dictionary to change application and procedure options. The Application Generator maintains your flexibility to change template options, unlike one-way-wizards found in other RAD tools.

The Clarion Language

The Clarion language is an elegant, practical language especially designed for business applications. It's compact, so you can write less to do more. It's expressive, so you can read your code, and others can read yours more easily. It contains unique data structures especially designed for Windows programming. It has a straightforward database grammar. It's flexible, extensible, and whether you use it a little, just to supplement the functionality of the templates, or a lot (many Clarion developers like to code from scratch, using only the text editor and the visual design tools), after a short time we think you'll agree it's a perfect language for business oriented programmers.

Expressive, Compact Code

Back at the very beginning, in 1985, the introduction to the language reference manual for version 1.0 of Clarion (for DOS) included these statements:

"Clarion has been on my mind since a FORTRAN compiler kicked me off for missing an "H" count in a FORMAT statement. The author of the compiler must have thought I could count better than the computer." "A programming language ought to help a programmer, requiring the fewest statements necessary to clearly define a process. Sequences that are always required should never be required. Why treat a memory-mapped display like a teletype? Why don't compilers generate data type conversions? Why so much punctuation? And why are compilers so fussy?

Expressive, Readable Code

In other words, why don't the language and the compiler work together to make the programmer's job easier, leveraging the strengths of the computer to handle all the repetitive aspects of the job, freeing the programmer to concentrate on the solution to the business problem at hand?

The more expressive a programming language is, the easier it is to read code that someone may have written, or that you yourself wrote, say, six months ago. The more readable a programming language, the easier it is to learn the language, and improve your own skill at it by reading example code written by someone else.

Compact Code = Productivity

A compact language requires less time to complete a task. One of the more common traits of fourth generation languages is that they compress the grammar for processing databases. The many steps required to open a file, read it, and place the field contents into record structures, typically requiring dozens of lines in third generation languages, merely require a few statements in Clarion. As a 4GL, Clarion leverages your current programming skills; it's a smooth learning curve. Hand coding is extremely efficient, and probably faster than any compiled language we know of.

“Hello World”

Consider that typical example program—Hello World—as illustrated in example code for Clarion and C++ (Microsoft Visual C++).

The Clarion for Windows “Hello World” is precisely eleven lines—no include files. In Clarion, the Window itself is a data structure—and the compiler knows exactly what to do with it. User interface objects—buttons and strings—reside inside the window structure, making the code very readable.

```
PROGRAM
Window    WINDOW('Clarion for Windows'), AT(,,160,60), SYSTEM
                        STRING('Hello World'), AT(30,15,90,12), CENTER
                        BUTTON('OK'), AT(60,35,,), USE(?OK), DEFAULT
                  END
CODE
          OPEN(Window)
ACCEPT
          IF ACCEPTED() = ?OK THEN BREAK.
END
RETURN
```

Freeing You From Grunt Work

The Clarion ACCEPT loop, described in the previous chapter, elegantly supports the Windows messaging model. It also makes the programmer’s task easier by freeing the Clarion developer from the “grunt” work normally associated with Windows programming, such as manually specifying that the window should repaint if it’s temporarily covered, then uncovered by another application’s window.

To the Clarion developer, specialized Windows data structures are just common sense. If the compiler understands the operating system, why shouldn’t it do the work for you?

C++ “Hello World” -1-

Now consider a “bare bones” Microsoft Visual C++ “Hello World:”

```
//Contents of HELLO.H
#ifndef _——HELLO—H——__
#define _——HELLO—H——_
class CMainWindow : public CDialog
{
public:
    CMainWindow();
    afx_msg void OnOkButton();
    DECLARE_MESSAGE_MAP()
};
class CTheApp : public CWinApp
{
public:
    BOOL InitInstance();
};
#endif _
```

C++ “Hello World” -2-

```
//Contents of HELLO.CPP
#include “stdafx.h”
#include “resource.h”
#include “hello.h”
CMainWindow::CMainWindow()
{
Create( “HelloBox”, NULL );
}
void CMainWindow::OnOkButton()
{
CloseWindow();
}
```

```
BEGIN_MESSAGE_MAP( CMainWindow, CDialog )
          ON_COMMAND( IDM_OKBUTTON, OnOkButton )
END_MESSAGE_MAP()
BOOL CTheApp::InitInstance()
{
TRACE( “HELLO WORLD\n” );
SetDialogBkColor();
m_pMainWnd = new CMainWindow();
m_pMainWnd->ShowWindow( m_nCmdShow );
m_pMainWnd->UpdateWindow();
return TRUE;
}
//Contents of RESOURCE.H
#define IDM_OKBUTTON                    100
//Contents of HELLO.RC
#include “resource.h”
#include “afxres.h”
HELLOBOX DIALOG DISCARDABLE 34,22,144,75
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION “Visual C++”
FONT 8, “Helv”
BEGIN
          CTEXT “Hello World”,IDC_STATIC,0,23,144,8
          DEFPUSHBUTTON “OK”,IDM_OKBUTTON, 56,53,32,14,WS_GROUP
END
```

Teaching Windows to the Compiler

Because C++ doesn’t innately understand the Windows operating system, the programmer has to “teach” it to the compiler—by including a header file, which in turn includes yet more files with the defines and class definitions necessary for the Microsoft Foundation Classes. In all, it takes about two megabytes of header files before the compiler can even address your code! And then your code isn’t nearly as readable as Clarion. To look at the actual code, you have to examine a constructor function, message maps and resource files to understand what the application does. No wonder C++ programmers can’t read each others’ code!

Code Reusability -1-

In practical usage, OOP code reusability exists on only two levels. The first is the base object framework, which usually handles only creation and simple maintenance functions, such as opening a window and checking the message queues. Frameworks rarely provide more than an app skeleton, for which your hand coding provides the “flesh.” The second level is when a single programmer creates an object class, and it’s reused in another program by the same programmer. If another programmer in the same shop needs the same functionality, more often than not, that second programmer codes from scratch rather than deciphering the first programmer’s work—as a direct result of the difficulty of reading OOP code.

Code Reusability -2-

We don’t contend that Clarion is superior to OOP languages—but we do think Clarion is an alternative that often delivers higher productivity. By combining the structured design of a traditional programming language, with the maximum code reusability often associated with OOP, Clarion gives the developer the best of both worlds.

Windows Data Structures

Another Rapid Application Development tool’s ads endlessly proclaim its compiler can compile 350,000 lines of code per minute. Who cares? If it requires that much code to teach your compiler what a window is, it had better compile fast!

Frankly, these days, when every professional strength tool (including Clarion, of course) can easily do an incremental compile, it's how many lines of code you have to write that matters. And Clarion's compactness, its specialized data structures, and even its expressiveness (shorter comments to write!) mean you write less.

Windows Data Structures

Because Clarion supports special data structures such as a WINDOW, we like to say it's the only Windows-aware compiler in the world. In fact, you could say that Clarion condenses the grammar for processing graphical user interfaces in the same way that Fourth Generation Languages compress the grammar for processing databases.

There are other important structures to make your job easier:

Other Specialized Structures

- FILE/RECORD

Like COBOL, Clarion provides support for the database file structures at the language level. Frankly, we don't see how any compiler product can claim to be a business-oriented language without this degree of support.

Moreover, Clarion implements a driver-neutral database grammar. You can access your data with the same compact file access statement, no matter what the database or file format is. The OPEN statement opens a data file referenced by a FILE structure. The SET statement positions the record pointer to match a key value. The GET and PUT statements read a record into the record buffer, or write an updated record, respectively.

File/Record -1-

Whether ISAM, SQL, or other DBMS, the driver creates an optimized instruction specific to its target, implementing your intention. The Clarion database grammar assumes every database possesses features that include relational join, filter and project operations, as well as scrollable cursors. The Clarion database drivers exploit the functionality available from the DBMS, where available, and automatically fill in whatever is missing. Consequently, Clarion applications are automatically optimized for any chosen database.

File/Record -2-

The following is an example of a file structure for a Clipper database file. Notice that the data structure is flexible enough to support options for defining expressions for the index files, as well as support for converting the strings with which the Xbase file stores numeric fields into real numeric values. This definition was created from an existing file using the Import command, so no coding was required; the only steps necessary were to allocate the length of the record buffer for the memo field (in this case, 2048), and to name the pointer to it (ptrMEMO) which is actually stored in the .DBF file.

File/Record -3-

```
NAMEFILE  FILE,DRIVER('Clipper'),NAME('NAMEFILE.DBF'),PRE(NAM),BINDABLE,CREATE,THREAD
LastName            INDEX(NAM:LNAME),NAME('lname.ntx'),OPT
Persons             INDEX(NAM:P_ID,NAM:DOB),NAME('persons.ntx=C[str(P_ID)+dtos(ctod(DOB))]'),NOCASE,OPT
Spouses             INDEX(NAM:S_ID,NAM:DOB),NAME('spouses.ntx=C[str(S_ID)+dtos(ctod(DOB))]'),NOCASE,OPT
Fathers             INDEX(NAM:F_ID,NAM:DOB),NAME('fathers.ntx=C[str(F_ID)+dtos(ctod(DOB))]'),NOCASE,OPT
Mothers             INDEX(NAM:M_ID,NAM:DOB),NAME('mothers.ntx=C[str(M_ID)+dtos(ctod(DOB))]'),NOCASE,OPT
Memo                        MEMO(2048),NAME('memo')
Record              RECORD
LNAME                       STRING(30)
FNAME                       STRING(30)
STREET                      STRING(30)
CITY                        STRING(30)
STATE                       STRING(4)
ZIP                         STRING(10)
COUNTRY                     STRING(30)
NOTES                       STRING(30)
DOB                         STRING(10)
DOM                         STRING(10)
DOD                         STRING(10)
POB                         STRING(30)
POM                         STRING(30)
POD                         STRING(30)
F_ID                        REAL,NAME('F_ID=N(6,0)')
M_ID                        REAL,NAME('M_ID=N(6,0)')
S_ID                        REAL,NAME('S_ID=N(6,0)')
P_ID                        REAL,NAME('P_ID=N(6,0)')
ptrMEMO                     LONG,NAME('MEMO')
SEX                         STRING(1)
                    END
          END
```

Report Structure

- REPORT

Unlike other tools which require separate, runtime reporting tools, Clarion for Windows includes a programmer's report writer, to create Clarion REPORT structures which are compiled into your executable. Some stand-alone report writers require you to learn another query language entirely to customize your reports, and the reporting tools can require as much as five megabytes for the runtime version alone. Just the difference in the time it takes to load the report into memory may provide a performance advantage for Clarion.

General Ledger -1-

The following is a complete program, illustrating a REPORT structure and the statements necessary to print a complete General Ledger. Note also the use of a Clarion VIEW structure, which provides support for virtual files consisting of selected fields from one or more (related) data files.

General Ledger -2-

```
!
!         Clarion for Windows Example Program
!
!         Print a General Ledger
!


Ledger    PROGRAM   !Print the General Ledger

          INCLUDE('EQUATES.CLW')

Account   FILE,PRE(ACT),DRIVER('TOPSPEED'),NAME('ACCOUNT.TPS')
ByAccount           KEY(ACT:AcctNo)
Record              RECORD
AcctNo                      LONG                !Account Number
Desc                        STRING(32)          !Account Description
Type                        STRING(1) !Account Type
```

```
                    END
          END

Ledger    FILE,PRE(LGR),DRIVER('TOPSPEED'),NAME('LEDGER.TPS')
ByAccount           KEY(LGR:AcctNo,LGR:Period,LGR:Date),DUP,OPT
ByBatch             KEY(LGR:Period,LGR:Batch,LGR:Sequence),DUP,OPT
Record              RECORD
Year                          SHORT               !Fiscal Year
Period                        LONG                !Accounting Period
Batch                         STRING(8) !Batch ID
Sequence                      SHORT               !Sequence Number
AcctNo                        LONG                !Account Number
Date                          LONG                !Transaction Date
Desc                          STRING(32)          !Description
Debit                         DECIMAL(11,2)       !Debit Amount
Credit                        DECIMAL(11,2)       !Credit Amount
                    END                           !End RECORD
          END                                     !End FILE

View      VIEW(Ledger)
                    PROJECT(LGR:Year,LGR:Period,LGR:Batch,LGR:AcctNo,    |
                              LGR:Date,LGR:Desc,LGR:Debit,LGR:Credit)
                    JOIN(ACT:ByAccount,LGR:AcctNo)
                              PROJECT(ACT:Desc)
                    END
          END
```

General Ledger -3-

```
Report    REPORT,AT(1000,2000,6000,7000),PRE(RPT),FONT('Arial',10,,),THOUS
                    HEADER,AT(1000,1000,6000,1000)
                              STRING('Consolidated Industries'),AT(1510,104,,),FONT(,18,,FONT:Bold)
                              STRING('General Ledger'),AT(2104,469,,),FONT(,16,,FONT:Bold)
                              STRING('Account'),AT(42,781,,)
                              STRING('Batch'),AT(708,770,,)
                              STRING('Description'),AT(1542,770,,)
                              STRING('Credit'),AT(5177,770,,)
                              STRING('Debit'),AT(4281,770,,)
                    END
AcctBreak           BREAK(LGR:Date)
DayBreak                    BREAK(LGR:Date)
Detail                                DETAIL
                                                STRING(@D1),AT(10,10,,),USE(LGR:Date)
                                                STRING(@S10),AT(615,10,,),USE(LGR:Batch)
                                                STRING(@S30),AT(1396,10,,),USE(LGR:Desc)
                                                STRING(@N(14.2)B),AT(3823,10,,),USE(LGR:Debit)
                                                STRING(@N(14.2)B),AT(4875,10,,),USE(LGR:Credit)
                                      END
                                      FOOTER,AT(,,,167)
                                      END
                              END
                              FOOTER,AT(,,,219)
                                      STRING(@N(14.2)B),AT(3823,10,,),USE(LGR:Debit),SUM
                                      STRING(@N(14.2)B),AT(4875,10,,),USE(LGR:Credit),SUM
                              END
                    END
          END
```

General Ledger -4-

```
          CODE
          OPEN(Account)                                 !Connect to the Account table
          OPEN(Ledger)                                  !Connect to the Ledger table
          SET(LGR:ByAccount)            !Use the account number key
          OPEN(View)                                              !Open the Ledger/Account view
          OPEN(Report)                                  !Open General Ledger report
          LOOP                                                    !Loop thru each row
                    NEXT(View)                                    !  Fetch a row
                    IF ERROR() THEN BREAK.        !  If no more then quit
                    PRINT(Detail)                               !  Print a line
          END                                                     !End the LOOP
          CLOSE(View)                                             !Close the view
          CLOSE(Report)                                 !Close the report
          CLOSE(Account)                                !Disconnect from the Account table
          CLOSE(Ledger)                                 !Disconnect from the Ledger table
          RETURN                                                  !End the program
```

Smart Typing

Clarion supports a very wide selection of data types—and we make it easy for you to work with them. Most importantly, you never have to worry about casting or incompatible data types—it's the compiler's job to automatically convert them for you. You can concatenate a string and a decimal variable, without having to use string functions to convert the decimal:

```
'The grand total is: ' & GrandTotal
```

Some of the specific data types, such as TIME, are designed to match specific databases such as Btrieve. Other special types are designed to make working in Windows easier.

A Rich Selection of Data Types -1-

- The GROUP type allows you to declare a custom structure to match a C struct. This is important if you wish to use many Windows API calls. You can prototype any function in an external .DLL (such as the Windows API libraries) and use it in your applications. This provides full access to Windows features even as Microsoft adds new ones. It allows you to fully integrate your applications into the Windows environment.

A Rich Selection of Data Types -2-

- The REFERENCE type allows you to use the equivalent of C pointers, but safely. The most common cause for General Protection Faults in Windows are errant pointers—when an application writes to an illegal memory address referenced by a pointer which no longer points to where it was supposed to. Clarion automatically dereferences REFERENCE variables after their use, so you can't make this mistake.

A Rich Selection of Data Types -3-

- The DECIMAL type allows you to use our Binary Coded Decimal math libraries. Does your business program perform multiplication or division operations on currency? Consider the following operation:

  ```
  AmtInYen = AmtInDollars * 100.375
  ```

  If you utilize floating point variables for this, you may receive an incorrect answer. If you convert to integer variables, you're more likely to arrive at the correct answer. (This has nothing to do with the famous Pentium flaw, by the way; this is universal for all PCs). The Binary Coded Decimal math support in Clarion, by performing integer operations only, automatically insures your currency calculations are accurate.

A Rich Selection of Data Types -4-

- The BIND and EVALUATE statements provide an additional level of flexibility beyond most compiled languages. You can take a variable whose contents are known only at runtime (generally to hold some end user input), and execute a statement dependent upon it. This provides the freedom programmers coming from interpreted development environments are used to, even while taking advantage of the far greater performance offered by a compiled application.

A Rich Selection of Data Types -4-

- The PICTURE string automatically formats or deformats the appearance of values in window or report controls. Consider all the casts and lines of hand-coded string functions necessary for formatting output in any other language—the dictionary can store PICTURE strings by field.

Origins of the Clarion Language...

Though too long to print here, we've included an article called "Origins of the Clarion Language" in the Adobe Acrobat documents on the Evaluation Edition CD. It documents some of the thinking that went into the creation of the Clarion language syntax and structure. Another article, called "Software Manufacturing" reprinted from the February TopSpeed Developer Newsletter, discusses TopSpeed's vision of the future of programming. It discusses how the Clarion language and template development system fits into component oriented software development.

Clarion Template Language

Previously, we've compared templates to base object classes: they contain code and data; but they provide better code reusability because templates have an interface that draws you into the process of setting properties, encouraging developers to use them more often. Developers can assemble an application from prefabricated parts—templates—in a fraction of the time it would take to hand-code. Whether from the box, from a third party vendor, or one written by the developer, TopSpeed wants developers to have a template for every functionality their applications will ever need.

Learning to Use Templates -1-

Like objects, Clarion templates store executable code and data. Learning how to use templates, however, doesn't require the huge investment of time that learning an object oriented language usually requires. The template user interface makes setting template properties interactive. When using templates, Clarion developers have full access to visual design tools for customizing windows and reports. The templates include symbolic, live links to the data dictionary which holds the plan for the application. It's like having an object that's intimately acquainted to your database. When placing data controls, the developer simply chooses a database field from a hierarchical list.

Learning to Use Templates -2-

Learning to use the templates, or learning to write your own, gives you the software reusability previously claimed exclusively by OOP programmers. Better still, the template interface makes it far easier to extend reusability from application to application, or from developer to developer. Instead of managing dense object oriented syntax, the Clarion developer, after designing the application's windows, fills in a series of edit boxes and chooses options from drop down lists. These options tell the template exactly what functionality the developer wants to implement. The Application Generator then generates only that code—without "object bloat."

Intelligent Code Generators

Templates, therefore, are intelligent code-generators, over which you have complete control. The actual "physical" template is a text file which contains Clarion language and Clarion Template language statements. These include:

Template Symbols

- Predefined template symbols (variables) which store developer input and/or database dictionary options.

```
%Application
        %DictionaryFile
                %File
                        %Field
                        %Key
                        %Relation
        %Program
        %GlobalData
        %Module
                %ModuleProcedure
                %MapItem
                %ModuleData
        %Procedure
                %Report
                        %ReportControl
                                %ReportControlField
                %Window
                        %WindowEvent
                        %Control
                                %ControlEvent
                %ProcedureCalled
                %LocalData
                %ActiveTemplate
                        %ActiveTemplateInstance
                %Formula
                        %Formula Expression
```

Template User Interface Controls

- Template user interface controls which appear in the Procedure Properties and Actions dialogs for each template. These include edit boxes, strings, check boxes, drop down lists, etc. The Application Generator stores the developer input gathered by these controls in template symbols. When writing your own templates, you use the #PROMPT statement to create a control and provide options for the developer using your template. Further statements validate the developer's input, so that you can be sure of generating the correct code. For example, the following template language statement asks for a

file name and stores it. The REQ (required) attribute specifies it must be filled out before the Actions dialog can be completed:

```
#PROMPT('Ask for File Name', FILE), %InputFile, REQ
```

Clarion Language Within Templates

- Clarion language code, mixed with template symbols where necessary. Assume, for example, the template makes a list of files from the database dictionary available to the developer, then stores the developer's choice in a template symbol called %FileToOpen. To specify that the generated code should use that choice as the parameter of the OPEN statement, the template includes this statement:

```
OPEN(%FileToOpen)
```

If the developer chose a data file called "ShipTo" from the list, the actual generated code would then be:

```
OPEN(ShipTo)
```

Template Control Structures

- Control structures that branch source code generation based on the developer's input, or options defined in the database dictionary. In this way, the templates generate only the source code that should be generated, based on the developer's input, and the database dictionary. Additional control statements specify exactly where to "place" the generated Clarion language source code. An include statement, for example, can be directed to the MAP, which contains all the prototypes and external source code modules.

Template Types

These, taken all together, provide the Application Generator with the ability to gather input from the developer and from the database dictionary, then generate Clarion language source code files with precisely the statements needed to support the functionality requested, in the sequence they should appear. There are several types of Clarion templates, each appropriate for a different job. These include:

Procedure Templates

- Procedure templates generate an entire procedure or function. They typically include a window or report structure and several predefined controls. To customize a procedure template, you generally populate the window or report with database fields. The Application Generator places the controls appropriate to the field. Because they incorporate the most functionality in "one fell swoop," procedure templates help you develop apps very quickly.

Control Templates

- Control templates place one or more controls in a window or report. Templates are granular in nature; you can drill down from big parts (procedure templates) to smaller parts (control templates). In fact the default browse template is built out of smaller parts—the control templates for the list box and buttons you see when you choose the browse template. A single window can contain many control templates. One of the example applications displays a genealogical tree—made up of seven listbox control templates—synchronized to display the ancestors for a single selected member. All the code necessary for synchronizing is generated by the control templates. The developer simply selects fields and other options in the Actions dialog boxes for the control templates.

Code Templates

- Code templates generate executable code into an embed point. Perhaps more than any other template type, these are the most useful to the developer who wishes to cut his or her repetitive coding tasks.

Extension Templates

- Extension templates add executable code to a procedure template, without tying the code to a specific control. If you find yourself habitually adding the same code to customize the default templates, you should consider writing an extension template. Some developers, for example, have written extension templates to change the behavior of the default form template. Their end users prefer the “old” DOS method of using the Enter key to accept user entry and move to the next field, rather than the Windows convention of using the Tab key (some refer to this as heads-down entry). Rather than adding code to support this for every form, the template extension appears as an extra checkbox in the form template interface. One check and the developer adds the code.

Competitive Summaries

This section identifies some competitive development systems, and presents capsule summaries of their strengths and weaknesses.

We'll discuss three competitors. Two produce interpreted or scripted applications: Microsoft® Visual Basic™ 3.0, and Powersoft™ PowerBuilder DeskTop™ 4.0. The third produces compiled applications: Borland Delphi™ 1.0. At the end of the section, you'll find a competitive feature grid.

Frank, Honest Discussion

Frankly, we thought we'd be a little different than your usual marketing document in this section: after we discuss the other products' strengths and weaknesses, we'll honestly discuss our own!

We expect you've already explored Clarion for Windows a bit, and have already formed some opinion. There's a wealth of Windows development tools out there; and there's even more business problems awaiting the programmer with the right tool for the job. We want to help you identify the right tool—because we think, in many cases, it's Clarion for Windows!

32-Bit Competitors

At the time we went to press, Microsoft Visual Basic had gone gold, but had not yet shipped. Delphi was expected to ship a 32-bit compiler before the end of the year. Powersoft was expected to ship a 32-bit compiler sometime in 1996.

These comparisons, therefore, are against the 16-bit competitors available at the time we created the Evaluation Edition. We look forward to adding comparisons to their updated 32-bit compilers as they become available.

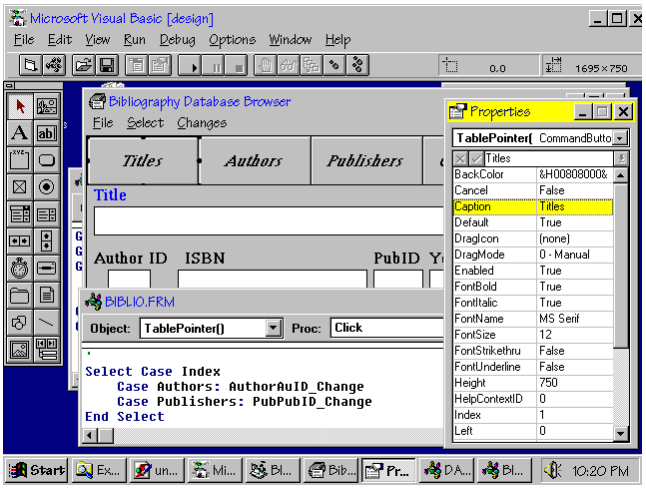
The Gauge Application

The Evaluation Edition CD includes an application which measures the competitive performance of 16-bit applications created by the products described in this section.

It tests and compares CPU intensive performance (how many prime numbers can be computed in the same amount of time), and display performance (repeatedly painting a set of controls).

To install the application, please run SETUP.EXE in the \GAUGE subdirectory on the Evaluation Edition CD.

Microsoft Visual Basic Professional



Microsoft Visual Basic Professional

Visual Basic produces executables that call a run-time interpreter that executes pseudo-code, rather than machine code. The additional overhead of reading the p-code, then producing the machine instructions to run it generally results in significantly slower performance than that provided by a compiled application. The Professional Edition of VB includes the Jet database engine, database controls, and is priced for the mass market. It's the market leader in unit sales for all Rapid Application Development tools.

Visual Basic's Strengths Include:

- VB's biggest strength is as a visual designer of applications. This has helped it become the best selling prototyping tool in the world. You can place user interface controls quickly.
- Visual Basic's dialect of BASIC is easy to learn.
- Significant extensibility is available via third party .VBX's, and now, .OCX's.
- A visually attractive development environment, based on the forms-based development paradigm. Virtually all code is attached to user interface controls.
- It's aggressively priced.

Visual Basic's Drawbacks Include: -1-

- Slow performance of finished applications.
- The most common complaint among VB developers is referred to as "hitting the wall." The BASIC dialect lacks the flexibility to customize; typically, the developer needs to call the Windows API to go further. This increases development time because it must be hand coded.

- When using the Jet database engine, the finished application can be resource hungry. End users may sometimes complain about performance on RAM-hungry systems, especially when running other applications at the same time. Both the Jet engine and the VB Runtime modules require a larger footprint with VB 4.0.

Visual Basic's Drawbacks Include: -2-

- The Jet database engine, when called by data aware controls in the default manner, can be slow. Many developers find they must hand code direct calls to Jet, increasing development time.

- The Jet database engine displays significant weaknesses in a multi-user environment, both in stability and speed. It simply does not reliably safeguard against deadlock.

- Data controls, by default, do not support transaction processing. This must be hand-coded, increasing development time. Referential Integrity is only supported when using transaction processing, meaning that in its default state, VB does not support RI.

Visual Basic's Drawbacks Include: -3-

- Visual Basic 4.0 requires separate development environments **and** separate projects for 16 and 32-bit applications.

- The dropping of support for .VBX's in VB 4.0 obsoletes significant investments for many VB programmers. Programmers often note that although VB is aggressively priced, real programming jobs nearly always require .VBX's, raising the price of the total system.

- It has no database dictionary.

- It has no built-in reporting tools, relying on a limited edition of Crystal Reports, instead.

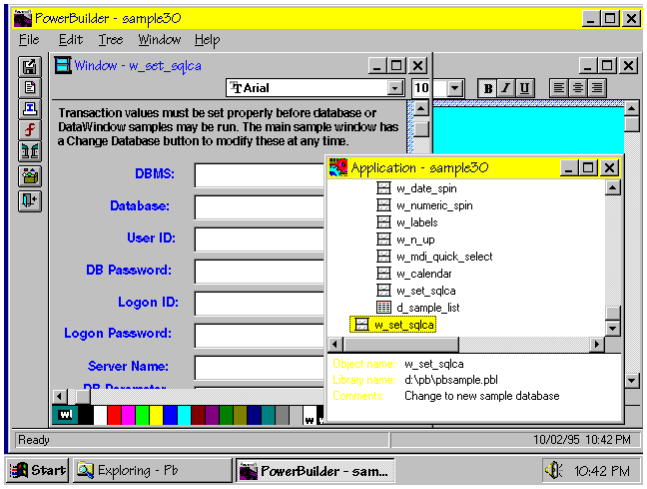
Visual Basic's Drawbacks Include: -3-

- With virtually all code attached to user interface controls, organization can be a sore point in a large project. Many have called the code "hidden behind a thousand doors." When looking for a specific piece of code to edit, the programmer has to remember what control it's attached to!

- Every control requires you to write some code.

- Debugging is weak.

Powersoft PowerBuilder DeskTop



Powersoft PowerBuilder DeskTop

PowerBuilder produces executables that call a run-time interpreter that executes scripts, rather than machine code. The additional overhead of reading the script, then producing the machine instructions to run it generally results in significantly slower performance than that provided by a compiled application. PowerBuilder includes a limited version of WatCom database engine. It's the market leader in middleware tools for connecting to SQL databases.

PowerBuilder 5.0, due in the first half of 1996, will include a compiler.

PowerBuilder's Strengths Include: -1-

- Many consider its higher priced Enterprise "sibling" the tool of choice for connecting to industry leading SQL databases.
- When scalability is paramount, it offers an excellent migration path to its higher priced versions.
- When portability to other platforms is important, it may offer the best choice of any RAD tool.
- The scripting language offers many of the features of OOP languages, including inheritance and encapsulation.

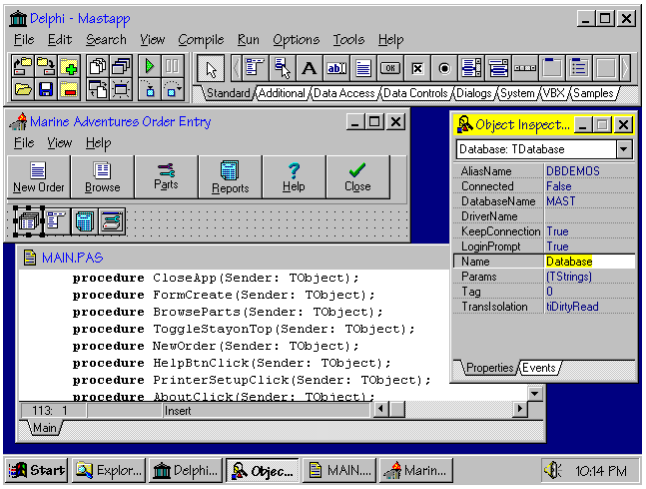
PowerBuilder's Strengths Include: -2-

- It's the only other product in this group besides Clarion that has a professional database dictionary.
- Reporting tools are good.

PowerBuilder's Weaknesses Include: -1-

- According to many users, the development environment and applications are notoriously unstable, leading to frequent GPF's.
- Slow performance of finished applications: PowerBuilder's DataWindows, which are its main means of displaying controls containing data from the database, have come in for much criticism in particular.
- Large runtime requirements for applications, both for disk space and memory.
- The scripting language has a steep learning curve.
- It provides the weakest .VBX support of these Borland Delphi of these four products.

Borland Delphi



Borland Delphi

Borland Delphi produces true executables, using the Borland Object Pascal compiler. It uses the Borland Database Engine as its database engine. Delphi is aggressively priced (though the Client/Server edition is significantly more expensive).

Delphi32 is expected to ship in late 1995. It's expected that the product will require separate compilers and separate projects for 16 and 32-bit applications.

Delphi's Strengths Include: -1-

- Compiled applications run fast, and are small if you don't use the Borland Database Engine or ReportSmith.
- Borland, as the publisher of both Paradox and dBase, offers the most comprehensive support for these file formats.
- It's a superb tool for general purpose programming projects, or personal database applications where the flexibility and strengths of a database dictionary are considered unnecessary by the programmer.

Delphi's Strengths Include: -2-

- It features a visually attractive interface along the forms-based programming paradigm pioneered by Visual Basic. (While this can lead to the "behind 1000 doors" syndrome that VB suffers from, it's somewhat more organized).
- It features a good assortment of data controls.
- Object Pascal offers many of the features of OOP languages, including inheritance and encapsulation.
- Good support for OLE 2.
- The development environment is very stable.

Delphi's Weaknesses Include -1-

- No database dictionary: Delphi's database wizards are one-way tools. If, for example, you realize you need to add a field in the middle of a project, you're on your own.
- There is no built-in reporting tool. The add-in, Borland ReportSmith, looks like a million dollars, but is unwieldy. It's a resource glutton, and appears to your end user as a separate application.
- Huge run-times for the add-ins, ReportSmith and the Borland Database Engine: when distributing a database app with reports, allow for six distribution disks at a minimum. Load times for the end user correspond.

Delphi's Weaknesses Include -2-

- Despite the VCL's (Visual Component Libraries, Delphi's answer to .VBX controls), database controls still require hand coding. Additionally, you cannot utilize VCL's when building a .DLL.

- For best performance, you **must** hand code your SQL statements.

- Records returned from an SQL SELECT are kept in memory, and not automatically refreshed by a new SELECT statement. This may lead to concurrency problems in a multi-user environment.

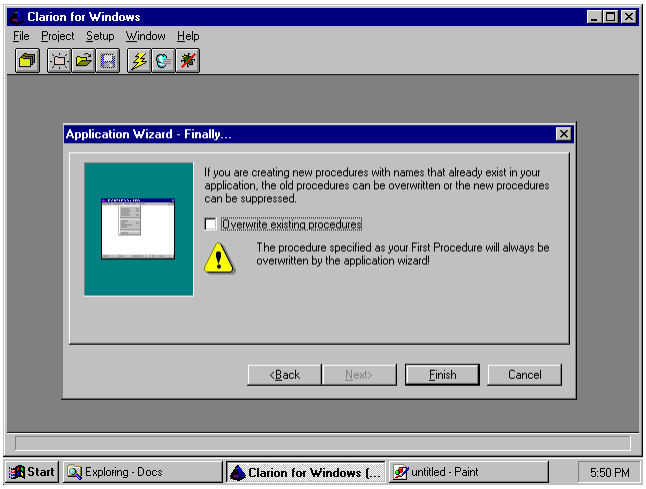
Delphi's Weaknesses Include -3-

- Poor documentation: the language reference manual does not ship with the product and costs extra. Object Pascal requires the steep learning curve that most object oriented languages require.

- Using the configuration tools for the Borland Database Engine, an end user can delete the references to your database.

- MDI database applications are very difficult to program.

TopSpeed Clarion for Windows



TopSpeed Clarion for Windows

Clarion for Windows produces true executables, using the TopSpeed backend compiler. Templates provide more functionality with less hand coding than other RAD tools. The Clarion language is a true 4GL which is easy to learn, and promotes code maintainability. Clarion for Windows uses replaceable database drivers for direct access; it ships with Btrieve, and provides ODBC support for Client/Server.

Clarion for Windows currently includes (you have them in your hands) both 16 and 32-bit compilers, and allows you to target either type of executable while maintaining a single project file.

Clarion's Strengths Include: -1-

- Small, fast, compiled executables.
- Templates support greater productivity, with most database applications requiring less coding than other RAD tools.
- It has a professional strength database dictionary.
- It's undoubtedly the RAD tool of choice for Client/Server Btrieve, which we believe to be the most economical and reliable Client/Server environment for small and moderate sized businesses. We use BTI's drivers for direct access. No other RAD tool can make that claim.

Clarion's Strengths Include: -2-

- The Clarion language seems like a throwback to the days when you could actually read the code you wrote; yet as the only Windows-aware language, it's brought 4GL's into the GUI age. It's a programming language that shows it was designed for business applications (as shown, for example, in the decimal data type designed for accurate currency calculations).
- Applications are remarkably stable. End users always comment on this.
- Support for Referential Integrity constraints, multi-user concurrency checking are enabled via options in the database dictionary, no matter what the database driver. You never have to write code for this.

Clarion's Strengths Include: -3-

- It has a built in programmer's report designer. Reports are compiled into the application.
- Clarion has a small (compared to Visual Basic) but vocal group of loyal developers, who magazine reviewers often describe as zealots. The great advantage to the novice Clarion developer is that they like to share their knowledge. Our CompuServe forum probably has more give and take than any other support forum. You can post a programming question, and receive three replies in five hours, often with sample code. This is not an exaggeration.
- It has some really nice extras you wouldn't expect in a database compiler—like support for JPEG (16.7 million color) images, for example.

Clarion's Weaknesses Include -1-

- No OLE. We will address this in a future release.

- It's expensive when compared to some other products. But we think it offers greater productivity than the other products, and the time you save and the money you make by writing more and better programs will more than pay for the difference in price. Frankly, we consider our product to be value-priced; not a loss-leader.

- It's user interface is fairly modal for a Windows application. While the procedure-centric design of the Application Generator is great for organizing a programmer who likes to be organized, the development environment's single-minded-modality has driven others crazy.

Clarion's Weaknesses Include -2-

- .VBX compatibility is a nuisance, though we're probably a little stronger than PowerBuilder in this category. Between using the huge memory model and supporting cooperative multi-tasking within the application thread (16-bit), we stress a .VBX.

  Even commercially available .VBX's often have bugs; any significant bugs surface and make the .VBX unusable. Fortunately, you can tell right away if a .VBX will be a problem—if you can register it in the .VBX registry, then you should be able to use it without a problem.

Competitive Feature Grid

The following charts provide you with comparative feature references.

Legend: ❍ means the feature is available only by purchasing a higher priced edition or third party add-on product.

Competitive: Performance

Performance	Microsoft Visual Basic Pro Edition	Powersoft Power Builder Desktop	Borland Delphi	TopSpeed Clarion for Windows
Compiler			✓	✓
Creates 32 bit Applications	✓			✓
One Project for 16/32 bit Apps				✓
Creates .DLL's			✓	✓
Typical Distribution Footprint	2MB	4MB	6MB	1MB
Linker eliminates unused data			✓	✓
Linker eliminates unused code			✓	✓
Linker eliminates unused methods				✓

Legend: ❍ means the feature is available only by purchasing a higher priced edition or third party add-on product.

Competitive: Database

Database Maintenance	Microsoft Visual Basic Pro Edition	Powersoft Power Builder Desktop	Borland Delphi	TopSpeed Clarion for Windows
Database Dictionary		✓		✓
Live Dictionary Link		✓		✓
Automatic concurrency checking		✓		✓
Automatic RI constraints		✓		✓
Filters without coding		✓	✓	✓
Range Limits without coding		✓	✓	✓
Stores Field Validity Rules		✓		✓
Preformats controls by field				✓
Preformats reports by field				✓
User Defined Sort Sequences				✓
Auto-increment fields/keys				✓
SQL Queries	✓	✓	✓	❍
Queries database for table names	✓	✓	✓	✓
Multiple sources from one query	✓		✓	✓
Join tables from multiple sources			✓	✓

Legend: ❍ means the feature is available only by purchasing a higher priced edition or third party add-on product.

Competitive: Connectivity -1-

Database Connectivity	Microsoft Visual Basic Pro Edition	Powersoft Power Builder Desktop	Borland Delphi	TopSpeed Clarion for Windows
AS/400				❍
ASCII text	✓			✓
Basic (delimited)	✓	❍		✓
Btrieve local		❍		✓
Btrieve Client/Server		❍		✓
Clarion				✓
Clipper				✓
Dataease		❍		
DRDA		❍		
dBase III	✓	❍	✓	✓
dBase IV			✓	✓
DOS Binary			✓	✓

Legend: ❍ means the feature is available only by purchasing a higher priced edition or third party add-on product.

Competitive: Connectivity -2-

Database Connectivity	Microsoft Visual Basic Pro Edition	Powersoft Power Builder Desktop	Borland Delphi	TopSpeed Clarion for Windows
Informix		❍	❍	
InterBase Local			✓	
InterBase C/S			❍	
MS SQL Server		❍	❍	
ODBC	✓	✓	✓	✓
Oracle		❍	❍	❍
Paradox			✓	✓
Sybase		❍	❍	
TopSpeed				✓
Watcom		✓		
XDB		❍		

Legend: ❍ means the feature is available only by purchasing a higher priced edition or third party add-on product.

Competitive: Productivity

Productivity	Microsoft Visual Basic Pro Edition	Powersoft Power Builder Desktop	Borland Delphi	TopSpeed Clarion for Windows
Reusable Components	✓	✓	✓	✓
Visual tools for UI controls	✓	✓	✓	✓
Data controls	✓	✓	✓	✓
Masked edit controls			✓	✓
.VBX Support (highest level)	3	1	1	1
Report Writer	bundled	✓	bundled	✓
Visual tools	✓	✓	✓	✓
Multiple detail bands	✓		✓	✓
Charts	✓	✓	bundled	
Reports Compiled				✓
MDI support for data controls		✓		✓
Professional Debugger			✓	✓
Hooks to Version Control Systems	❍	✓	❍	❍

Legend: ❍ means the feature is available only by purchasing a higher priced edition or third party add-on product.

Competitive: Language

Language	Microsoft Visual Basic Pro Edition	Powersoft Power Builder Desktop	Borland Delphi	TopSpeed Clarion for Windows
Structured		✓	✓	✓
4GL				✓
Expressive/Readable			✓	✓
Compact				✓
Easy-to-learn	✓			✓
OOP	✓	✓	✓	
Inheritance	✓	*(one level)*	✓	
Encapsulation		✓	✓	
Polymorphism		✓	✓	
User Defined Types			✓	✓
Windows API Accessible	✓	✓	✓	✓
Pointers/References		✓	✓	✓
PICTURE auto-string formatting				✓
Fixed Point Math (BCD)				✓
Packed Decimal types	✓			✓
Maximum string size	255 bytes	255 bytes	64K	64K
Runtime Expression Evaluation	✓	✓		✓

Competitive: Business Solutions

Business Solutions	Microsoft Visual Basic Pro Edition	Powersoft Power Builder Desktop	Borland Delphi	TopSpeed Clarion for Windows
Application Wizard Creates Full Applications				✓
Lookup from Related File			✓	✓
Record Locator				✓
Field Validate				✓
Listbox Column Totals				✓
Edit Masks		✓	✓	✓
Synchronized Parent-Child Lists				✓
Combo Box Parent-Child Adds				✓
Bundled .VBX Controls	14	1	4	

Legend: ❍ means the feature is available only by purchasing a higher priced edition or third party add-on product.

Competitive: Debugger

Debugger	Microsoft Visual Basic Pro Edition	Powersoft Power Builder Desktop	Borland Delphi	TopSpeed Clarion for Windows
Breakpoints	✓	✓	✓	✓
Watchpoints			✓	✓
Watch variables	✓	✓	✓	✓
Tracing	✓	✓	✓	
Message spy			*bundled*	✓
Registers			❍	✓
Assembler Debug			❍	✓
Debug .DLL's			❍	✓
PostMortem Debugging			*bundled*	✓

Legend: ❍ means the feature is available only by purchasing a higher priced edition or third party add-on product.