Contents

Vision

Four generations of software tools failed to provide the productivity that developers require and deserve. In the same way that new machine tools revolutionized 19th century industrial manufacturing, TopSpeed's *vision* is to provide the "toolware" for a new "industrial revolution" for software development.

Software Manufacturing

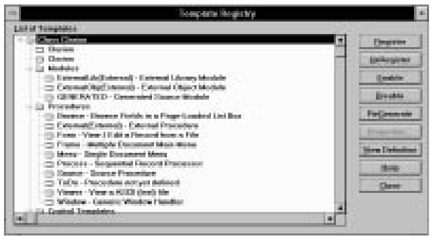
At the time of the industrial revolution, machine tools enabled craftsmen to adopt a non-specialized approach to producing goods from interchangeable parts, changing them into *manufacturers*. The mass production process allowed ordinary workers to do the labor which formerly required a specialist—the craftsman. Better tools produced better products cheaper and faster.

Today, Object Oriented Programming languages (OOP) support "interchangeable parts" in somewhat the same manner as the machine tools of the 19th century. Yet OOP has failed to fuel a mass production revolution, because it hasn't allowed ordinary workers to take over the craftsman's former role in production. OOP programmers might just as well be goldsmiths, apprenticing for years before doing significant work—and costing just as much, once past their apprenticeship.

OOP has failed to provide an enabling technology for "manufacturing" software. The OOP languages available today are object frameworks grafted on top of 3rd generation languages. Without "GUI-Awareness" built into the language, developers must spend time learning proprietary object libraries. Once learned, each program is written with an obscure syntax, making it all but impossible for one programmer to easily pick up another's work.

In practical terms, 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 queue. Level two is when a single programmer creates an object class, and it gets reused in another program by the same programmer. If another programmer needs the same functionality, even if both programmers are in the same shop, the second programmer usually ends up coding from scratch rather than deciphering the first programmer's work—as a direct result of the difficulty in reading OOP code.

Clarion for Windows represents the first "GUI-aware language." The Clarion SCREEN structure has evolved into the graceful WINDOW data structure; Clarion possesses an elegant and versatile user interface grammar for accomplishing the window creation and maintenance functions that OOP handles via object frameworks. Programmers can easily read each others' code and share it.



Template Registry

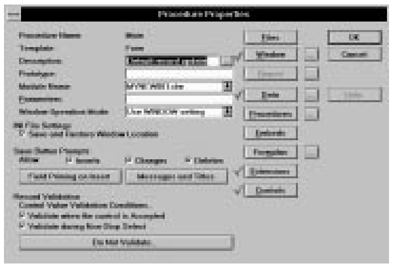
List of Templates

- Modules
  - ExternalLib(External) - External Library Module
  - ExternalObj(External) - External Object Module
  - GENERATED - Generated Source Module
- Procedures
  - Browse - Browse Fields in a Page-Loaded List Box
  - External(External) - External Procedure
  - Form - View / Edit a Record from a File
  - Frame - Multiple Document Main Menu
  - Menu - Single Document Menu
  - Process - Sequential Record Processor
  - Source - Source Procedure
  - ToDo - Procedure not yet defined
  - Viewer - View a ASCII (text) file
  - Window - Generic Window Handler
- Control Templates

The Clarion for Windows Template Registry

Clarion for Windows' enabling technology, which makes "software manufacturing" possible, is its powerful template language. Like objects, Clarion templates store executable code and data. Moreover, templates have something important that objects lack—a user interface for maintaining the template properties and customizations during the development process.

The Clarion template user interface takes the software reusability monopoly away from the OOP craftsmen, and allows ordinary "workers" to use the tool. Instead of managing mumbo-jumbo object oriented syntax, the Clarion developer fills in a simple series of edit boxes and chooses options from drop down lists. These options tell the template exactly what code to generate.



Procedure Properties

Procedure Name: Main
Template: Form
Description:
Prototype:
Module Name:
Parameters:
Window Operation Mode: Use WINDOW setting

INI File Settings
Save and Restore Window Location

Save Button Prompts
Allow [✓ Inserts] [✓ Changes] [✓ Deletes]

Record Validation
Control Value Validation Conditions:
[✓ Validate when the control is Accepted]
[✓ Validate during Non-Stop Select]

The Form Procedure Template user interface

Unlike base classes of C++ object frameworks, the templates can be easily modified and shared between programmers. The templates thus represent true interchangeable parts which don't require a specialist to assemble. An enabling tool for "software manufacturing" has arrived.

The Clarion template user interface also allows Clarion developers to produce leaner applications. The templates generate only the code necessary to implement the functionality requested by the developer. Compare this to an object class, some of whose methods might barely be documented, yet which always compile unless specifically overridden by the OOP developer.

Our "toolware" produces *better* applications—*fast*. These are the beliefs that guide the development of our compiler products:

- **We believe that new development tools will produce significantly better software with significantly lower cost and risk.**
- **We believe in high-performance compilers.**
- **We believe in template-driven Rapid Application Development.**
- **We believe in the Clarion language—the first “GUI-aware” language.**
- **We believe in client-server architecture.**
- **We believe in reusing software.**