Migrating from Prior Versions

Clarion 6

SoftVelocity

SoftVelocity Incorporated
2769 East Atlantic Blvd.
Pompano Beach, Florida 33062
(954) 785-4555
www.softvelocity.com

Printed in the United States of America (1103)

Contents:

How to Migrate to Clarion 6

Tips on an efficient migration process, proven methods and spectacular results. Find out how it's done.

This online document summarizes a wide variety of migration issues that you may encounter (or need to be aware of) when migrating applications from prior versions to Clarion 6. Please refer to the Contents page for a complete list of all topics.

For most applications there will be little or no required changes. In this document, we'll point out the changes that can impact migration and we'll make recommendations for the changes you can make to ensure an easy migration cycle.

New fully pre-emptive threading model

Clarion 6 introduces a new, and more powerful, thread support in the templates and runtime library.

The new thread model now uses preemptive threads. Typical Clarion programs won't require more than a "compile and link" to get the benefits of the new thread model.

For more detailed information, see the *Multi-Threaded Programming* PDF

General Recommendations

1. Use the THREAD attribute on global Data (file, class, group, queue or simple types).
2. Use the THREAD attribute on module Data (file, class, group, queue or simple types).
3. Avoid the use of static variables when they can be accessed from multiple threads simultaneously, or make them thread safe. Refer to the *Multi Threaded Programming* PDF for detailed information on this process.
4. Don't pass the address of anything within a START command - this was a common trick used by people to communicate between threads.
5. When the THREAD attribute is used with a FILE structure, it is important to remember that the record buffer and other attributes using variables *outside* of the record buffer (OWNER, NAME, DRIVER) are all threaded. It is permissible to use non-threaded variables for certain attributes of threaded files, but *never* use a threaded variable for attributes of files that are declared as non-threaded (no THREAD attribute).

Launching a thread - behind the scenes

With the advent of two new language statements supporting thread management in Clarion 6 (SUSPEND and RESUME), it is important to understand that there are a few things that are initialized and executed behind the scenes by the runtime library each time a thread is STARTed.

Here is the sequence of actions performed by the launching thread and the runtime library(RTL) each time a thread is STARTed:

1. Launching Thread executes START(ThreadProc)
2. RTL creates the physical thread in suspended state.
3. RTL resumes the launched thread created in step 2.
4. RTL sets an internal semaphore to a non-signaled state.
5. Launching Thread waits for the semaphore from the RTL.
6. RTL creates instances of threaded variables and calls initialization routines for them.
7. RTL sets the semaphore to signaled state.
8. RTL suspends the launched thread creates in step 2.
9. Launching Thread continues program execution.

   The launching thread will continue until it encounters the ACCEPT statement. Upon execution of the ACCEPT statement:

10. RTL resumes the launched thread.
11. RTL calls the entry point of the ThreadProc.

Therefore, a launched thread will remain suspended until the next call to ACCEPT from the launching thread. Only initialization and constructors for threaded variables are executed.

The use of RESUME with the START statement immediately executes Step 10 above without waiting for the call to ACCEPT. In other words, use of RESUME with START does not depend on the ACCEPT statement for resuming thread execution. This allows a new thread to be started from windowless threads.

The same can be said by using the SUSPEND statement immediately after START, e.g., SUSPEND immediately stops thread execution and does not wait for the ACCEPT loop.

Moving from Clarion DOS versions

Tips and Techniques

The following techniques have been used to maximize the amount of work performed in prior DOS versions, while minimizing the amount of time required when converting to a Windows version of your application. **Always make an archive copy of your project files and data before attempting a conversion**.

Dictionary / Data tables

- Use existing data files to reconstruct a dictionary by importing them and establishing the keys / relationships as necessary.

Applications

- Create a new application from your reconstructed dictionary file utilizing the application wizard.
- Open your Clarion DOS IDE and application, cutting and pasting the embed code to the appropriate locations.
- A REPORT structure is never the default target for graphic primitive procedures. Therefore, if your report is using these procedures to draw graphics to a specific band of a report, use the SETTARGET statement and specify the band in the second parameter.

Example: SETTARGET(Report,?PageHeader)

See the SETTARGET statement in the Language Reference Manual for more information.

Projects from prior Clarion for Windows versions

All projects compiled in Clarion 6 are 32-bit. Prior to loading older project files (.PR or .PRJ) into the Clarion 6 environment, load them into your favorite text editor and make sure that the following pragma entry is set properly:

```
#system win32
```

You can also load the project into the project editor. When you press OK to close the project, the #system pragma will be automatically updated.

Dictionaries from prior Clarion for Windows versions

Migrating Large Dictionaries and Data Paths

One of the nice new features of this release is a new system property: SYSTEM{PROP:DataPath}.

With this, you can set your data file names in the dictionary to have no path in them, and then set the data path once in the program start up. From there, each file will inherit the common data path.

With that in mind, dictionaries created in a prior version will continue to work. The only issue is where file names and structures are stored exclusively in a DLL and referenced from the EXE. In prior versions, you had to define these objects in the EXE as EXTERNAL, and did not care if the files in this object were threaded or not. In this release, any objects that contain threaded data must add the THREAD attribute to the object definition.

Mixing threaded and non-threaded data in an object is dangerous and likely to cause problems.

Applications from prior Clarion for Windows versions

Dictionary Class

The new Clarion threading model dictates that the existing File and Relation Managers use threaded objects (i.e. a new instance on every thread).

One of the effects of this is that the traditional ABC code that initializes both File and Relation Managers (contained in the *DctInit* generated procedure) now has to be executed whenever a new thread is started. Likewise, the Managers' kill code (traditionally contained in *DctKill*) must be called whenever a thread is terminated.

> To facilitate this, a small globally defined class called **Dictionary** will be generated into every ABC template based application that does not have its global data defined external to the application. (i.e. the File and Relation managers compiled locally). The Dictionary object contains only construct and destruct methods but, more important, it is a *threaded object*.

Example:

```
Dictionary            CLASS,THREAD
Construct               PROCEDURE
Destruct                PROCEDURE
                      END

Dictionary.Construct PROCEDURE
  CODE
  DctInit()

Dictionary.Destruct PROCEDURE
  CODE
  DctKill()
```

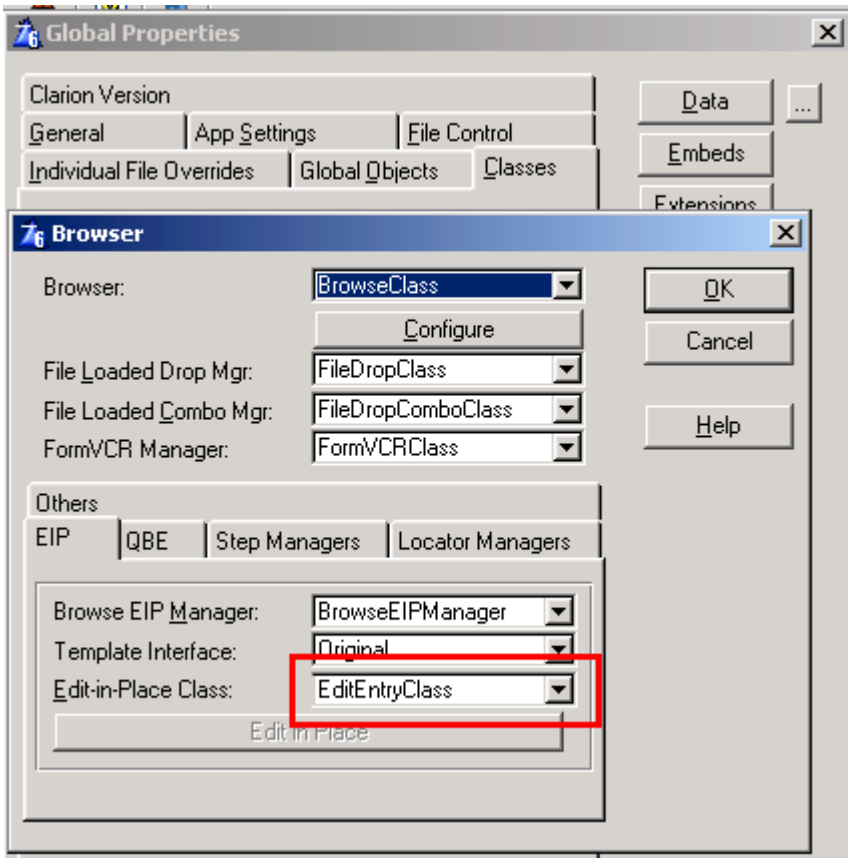
This means that the Construct method will be called whenever a new thread comes into existence and the Destruct method will be called whenever a thread is terminated. The constructor calls *DctInit* and the destructor calls *DctKill*. Therefore, *DctInit* is called whenever a thread is started and *DctKill* is now called whenever a thread is terminated; thus ensuring that threaded File and Relation managers are created and destroyed correctly.

Dockable toolbar window with the MDI attribute

In Clarion 6, the MDI attribute is no longer permitted on any toolbar window that is dockable (windows with the DOCK and TOOLBOX attributes).

Global EIP Classes

Some applications that use edit-in-place in prior versions will migrate to Clarion 6 with the *EditClass* defined as the default edit-in-place class. In Clarion 6, this class needs to be changed to the *EditEntryClass*. To do this, access the *Global Properties* in the Application Generator. In the **Classes** tab, press the **Browser** button and change the default edit in place class from *EditClass* to *EditEntryClass* (see below).



Global Pools and GROUPs

In Clarion versions prior to 5.5H, if a GROUP were defined in the dictionary's Global Pool without using a prefix, the prefix of the Global Pool (which is required) would be used in its place.

For example:

```
GLOB:GROUPONE GROUP,PRE()   !`GLOB' is the prefix of the Global Pool
GLOB:STRING1   STRING(20)
GLOB:STRING2   STRING(20)
              END
```

With Clarion 6, this same group will be generated without the Global prefix, and would appear as follows:

```
GLOB:GROUPONE GROUP,PRE()   !`GLOB' is the prefix of the Global Pool
STRING1        STRING(20)
STRING2        STRING(20)
              END
```

Since it is possible to define multiple Global Pools in your dictionary, a second Global Pool could be created as follows:

```
GLO2:GROUPTWO GROUP,PRE()   !`GLO2' is the prefix of the 2nd Global Pool
GLO2:STRING1   STRING(20)
GLO2:STRING2   STRING(20)
              END
```

Which in Clarion 6 will be generated as follows:

```
GLO2:GROUPTWO GROUP,PRE()   !`GLO2' is the prefix of the 2nd Global Pool
STRING1       STRING(20)
STRING2       STRING(20)
             END
```

This will cause the compiler to generate "Duplicate Label" warnings when migrating your applications.

The solution to this is to add a prefix to each of your GROUPs that share common label names across Global Pools.

For example:

```
GLOB:GROUPONE GROUP,PRE(GLOB)   !`GLOB' is the prefix of the 1st GROUP
STRING1        STRING(20)
STRING2        STRING(20)
              END

GLO2:GROUPTWO GROUP,PRE(GLO2)   !`GLO2' is the prefix of the 2nd GROUP
STRING1        STRING(20)
STRING2        STRING(20)
              END
```

Heap Overflow Error when migrating applications

During the early testing phase of Clarion 6, it was noted that some applications would post a “Heap Overflow” error when attempting to load applications of prior Clarion versions into the Clarion 6 IDE.

In nearly all cases, the solution is to first export the application to a text file (TXA), and then import it as text into the Clarion 6 environment.

INI Files – Migration Tip

If an application you are migrating to Clarion 6 is dependent on one or more INI files (either by template or custom control), make sure to check your INI entries after converting for possible references to obsolete version information and/or search paths.

Internet Connect / Web Builder / ClarioNET

The Internet Connect Application Server and “linked-in broker” are no longer supported. The executable Internet Connect Application Broker has been enhanced to support the Clarion based thin-client TCP/IP solutions.

Large WINDOW structures

In each control that is populated in a WINDOW structure in Clarion 6, there is now extra information for each control that takes about 10 extra bytes per control. This may cause some large and complex windows to not import properly from prior versions.

You may need to shorten some use variables or remove controls and create/destroy them at runtime or redesign the window to make it a bit more efficient.

PVCS version control

The PVCS Version Control interface has been replaced by a much more flexible version control interface. The new interface permits the use of *any* command-line driven system of your choice.

- Previous versions of Clarion Enterprise shipped with a DLL version of PVCS and are not supported via the new interface. If you own a PVCS retail license you can use the command line version and your existing projects will be accessible from the new interface.
- RCS/CVS - http://www.componentsoftware.com/
    - You can locate information on migrating your PVCS source files at this location: http://www.componentsoftware.com/Products/RCS/faq/index.html
- How-to's
    - http://tldp.org/HOWTO/CVS-RCS-HOWTO.html
    - http://tldp.org/HOWTO/RCS.html
    - http://cvsbook.red-bean.com/

RTF Control (Rich Text)

The RTFControl has been replaced by the RTFTextControl, supporting templates and classes. The RTFTextControl template works with the standard Clarion TEXT control which has a new RTF attribute. All existing applications will be required to have the previous template version removed and the new one populated. Additionally any code that specifically refers to the declared object, methods or code templates will need to be reviewed to determine necessary changes. Please refer to help system Templates by Topic, RTFTextControl.

When using the new RTF control for existing plain text, loading the plain text into the RTF control, and making any change, will cause the plain text to be converted and saved as Rich Text.

TXA Comparison Techniques

If you are having troubles with applications converted to Clarion 6 using DLLs, there is a possibility that the DLL that was converted contained hidden information (like a third party library) that was not detected by the conversion process.

To confirm this, try the following:

1. Export the old DLL application to TXA format (Export Text)
2. Export a new Clarion 6 DLL application to TXA format (Export Text)
3. Next, compare the TXA's up through the first procedure (i.e., the program/global area). This might give you some ideas regarding information converted from an old application that may not be compatible, or does not exist in Clarion 6.

Use of ABC Classes with Clarion template based applications

This release of Clarion includes many new template features that are now supported in the Clarion template chain, but rely on the use of the ABC Library Classes. Examples of this are the support for RTF Text Controls, Pop-up Calendars, and the new enhanced edit-in-place.

The linking of the ABC class support requires the following default pragmas settings to be included in the project system:

```
_ABCDllMode_=>0
_ABCLinkMode_=>1
```

Template support to automatically include these pragmas can be found in the *Global Properties* **Classes** tab by checking the **Enable the use of ABC Classes** check box.

Version specific image references

Loading a C55 application into Clarion 6 that uses the default splash screen will produce a link error that says 'C55.GIF' not found. The best workaround is to simply change the icon, or just delete the old splash procedure and add a new one using the Clarion 6 default. This may occur for other image resources as well, we will however attempt to resolve this wherever possible.

Advanced Topics

DLL Initialization

Enforcement of threaded variables in multi-DLL applications is critical in Clarion 6. In older versions, if your file definitions are set to "open in current thread" in the dictionary (the THREAD attribute is set in the FILE definition), your file definitions in your DLLs must match that definition. To ensure this, examine each application's global file control section, and make sure that all of your files are set to 'ALL THREADED' in the Threaded drop list.

You CANNOT mix thread and non-threaded attributes on files in a multi-dll application. Although this programming style was permitted in earlier versions of Clarion, the initialization of preemptive threads will not allow this in Clarion 6.

You can use *either* setting, thread or non-threaded, as long as it's consistent across all DLLs and your executable.

With the advent of the new threading model in this release, it is important to understand how threaded and non-threaded data elements are initialized, and in what specific order.

Initialization Schema

Prior to this release, data elements in respective modules were initialized in the following schema.

- non-threaded data of module1
- non-threaded data of module2
- ...
- non-threaded data of module*n*

- threaded data of module1
- threaded data of module2
- ...
- threaded data of module*n*

With the current release, the data initialization schema is as follows:

- non-threaded data of module1
- threaded data of module1
- non-threaded data of module2
- threaded data of module2
- ...
- non-threaded data of module*n*
- threaded data of module*n*

This means that a problem could arise if you attempt to access a threaded file from the constructor of a non-threaded class module. The initialization sequence for objects has changed so that non-threaded objects within a module are initialized before threaded ones.

There are two solutions to overcome this issue.

1. Make the threaded file non-threaded, or:

2. Move the definition for the class into a different module and set its initialization priority (init_priority) to a number less than the default initialization of 5.

```
PRAGMA ('define(init_priority=>4)') !add to the top of the module
```

Compiler Initialization

The compiler initialization sequence of data, and calls to constructors is as follows:

1) Patching for threaded data in module-1

2) Patching of non-threaded data in module-1

3) Constructors of non-threaded data in module-1

4) Constructors of threaded data in module-1

.... repeat (1)-(4) for additional modules.

Steps (1) and (2) can be required if FILEs or VIEWs use strings that are imported from external DLLs.

Special Considerations for a one-piece (single) executable

A one-piece executable is defined as a project that has been linked into a single, stand-alone executable. The Clarion runtime library and all of the application's procedure calls and libraries are linked into a single file.

Callback functions are a standard part of Windows programming in most programming languages. A callback function is a PROCEDURE that you (the programmer) write to handle specific situations that the operating system deems the programmer may need to deal with. A callback function is called by the operating system whenever it needs to pass on these situations. Therefore, a callback function does not appear to be part of the logic flow, but instead appears to be separate and "magic" without any logical connection to other procedures in your program.

Callbacks are valid when used in one-piece executables (EXEs), but there is a special case which must be handled in a different manner.

Here is the case:
If the EXE makes some call to the Operating System, the Operating System starts a new thread inside this call, and then calls to a passed callback function. Using this program design, the one-piece EXE must be converted to a DLL linked in local mode, and a starter EXE must be created, using an External link to the DLL entry point that is used to load and run the one-piece DLL.

The following approach demonstrates how this is done.

1. The one-piece EXE must be converted to a DLL linked in local mode.

2. The Local mode DLL must export the name of the entry point's procedure and the following names from the RTL:

```
__checkversion
__sysstart
__sysinit
_exit
Cla$code
Cla$init
Wsl$Closedown
```

Here is an example of the export file (EntryPoint is the procedure entry into the DLL)

```
------------------------------------
EXPORTS
EntryPoint@F   @?
__checkversion @?
__sysstart     @?
__sysinit      @?
_exit          @?
Cla$code       @?
Cla$init       @?
Wsl$Closedown  @?
```

In this example, the entry point procedure name in the Local DLL is: "EntryPoint"

Use the Inside the Export List Global Embed to add to your export list within the application.

3. The starter EXE must use External link mode. The source is written so that it just calls the DLL's entry point procedure.

Example starter EXE code:

```
  --------------------------
        PROGRAM

        MAP
          MODULE('')
            EntryPoint()
          END
        END

  CODE
  EntryPoint
```

Considerations when working with OLE objects

There are several potential problems if you CREATE an OLE control in the window for one thread and attempt to work with it from a different thread. This is because the Windows Operating System can load additional DLLs into process memory on creation of an OLE object. This creates several potential problems:

The initialization code for the OLE control can be executed in the context of the current thread and the initialized data would not be available for the control when it is running in the thread of its host window.

Extra DLLs can be unloaded from the process memory after closing the thread that created the OLE control, so the Virtual Memory Tables of interfaces used by the OLE control will point to deallocated memory. As a result, it's impossible to guarantee correct operation of OLE controls if they are created in a window other than the current thread.

Example:

```
WinOne WINDOW,AT(0,0,200,200)
        OLE,AT(10,10,160,100),USE(?OLEObject),CREATE('Excel.Sheet.5')
        END
       END
```

Use of Error Managers during DLL Initialization

A change has also been made in the DLL initialization of ABC-based applications. During initialization, the DLL uses a LocalErrors Class rather than the Global executable's GlobalErrors Class.

For example, in a multi DLL application and during initialization of the DLL containing Global data, if errors need to be posted to the error manager, they will be posted to the DLL's local error manager (LocalErrors) instead of the application’s global error manager. The reason for this is that the DLL's error manager is not set to use the application’s error manager until after initialization of the DLL. During initialization, the DLL uses the LocalErrors Manager rather than the executable’s GlobalErrors Manager. Inside the DLL Init procedure, extra code is generated to assign GlobalErrors, and also assign the passed error manager to the already initialized file managers and relation managers.

Developers who modified the global error manager in their applications using DLLs will now need to be aware of the new local error managers that are applied.

Embedding code when closing a Process procedure

The Process procedure used with the ABC templates calls ThisProcess.Close() after ThisWindow.Kill has fully completed. Consequently, any object created in the scope of the process window which is called inside ThisProcess.Close() will cause a GPF since the destructor for that object will already have completed during ThisWindow.Kill.

In Clarion 6, the ViewManager class destructor used with the ProcessClass is now calling ThisProcess.Close() to make sure that the VIEW is closed. This was not needed in previous versions of Clarion because local VIEWs were automatically closed when a procedure exited. With the new threading model, local VIEWs are now not automatically closed until the thread is destroyed.

There is a distinct chance that any call to a local object inside ThisWindow.Close() will cause a GPF when exiting the process procedure, because it has already been disposed by the time the final ThisProcess.Close() call happens.

Anyone embedding source code in the ThisWindow.Close() method needs to add some kind of condition surrounding any call to a local object that stops it happening after ThisWindow.Kill() has occurred.

Language / Runtime Library

POINTER(File) and POSITION(File)

The behavior of POINTER(File) is different for different file systems. For example, the first record in a TopSpeed file doesn't have a pointer value of 1.

It may still be safe to use for certain file systems, but for code portability, POSITION(File) is the way to go.

ISAM File Access Performance

Some users have reported that if there is any experience of slow file accesses when using ISAM files, switching off the Defer opening files until accessed in the application's Global Properties - File Control will improve the performance.

Change of EVALUATE Error Codes

The error codes posted by the EVALUATE statement have been modified in Clarion 6:

1010 - formerly 800: Bad expression syntax
1011 - formerly 801: Unknown identifier in the expression
1012 - formerly 802: Mismatched POPBIND

LoadImage()

The LoadImage function has been introduced to support icon sizes greater than 32x32. An algorithm is used to determine the best resource given the size of the control, if the icon file contains more than a single resource. This may not produce the desired effect. To ensure that the desired resource is used limiting your icon files to a single image resource size should be considered.

Deprecated

- WIZATRONS
- VBX support
- All 16bit support
- IN
- OUT
- ERRORCODE
- LENGTH
- PROP:SQLFilter (superceded by SQL())
- EVENT:Suspend
- EVENT:Resume

Clarion 6 Template Changes from Prior Clarion for Windows Versions

ABC

The ABC template chain and classes continue to expand in functionality. Please refer to the feature matrix for some of the highlights.

Clarion

The Clarion template chain is now a supported technology. A number of enhancements and additions have been introduced, including the use of some ABC classes. Please refer to the feature matrix.

3rd party template/library considerations

Each existing application that uses a 3rd party library or tool will require one of the two following in order to successfully migrate to Clarion 6.

- Secure and install the Clarion 6 version of the 3rd party product from the vendor
- Remove the 3rd party product from your application from within the previous Clarion for Windows version prior to opening the application in Clarion6.

Enhancing Your Applications with New Clarion 6 Features

Global Application options (in appearance order)

A number of Application options have been added to help you enhance your applications features and functionality.

- Set the Default program icon
- Enhanced options for persisting application settings to INI or the Registry
- Include a default XP manifest file
- Display a Visual Indicator on window controls to assist users in locating the “current” control
- Set the field navigation option to use ENTER instead of the TAB key
- Enable runtime auto-size for your browse box columns
- Enable runtime list box formatting that can be modified and saved by the end user
- Enable “re-basing” of DLL projects to specific memory addresses

Business Rules Manager

The Business Rules Manager Class and Templates provide a mechanism for you to define Clarion logic “rules”, which are applied at runtime and can be evaluated. Each rule is associated with a data column and any event that references the data column will cause the rule to be evaluated. When a rule is evaluated it can pass or fail. Based upon your template prompt choices a failure can display an indictor next to the control that displays the referenced data column. Additionally rules can be defined and located in a resource DLL and utilized in many applications, thus reducing maintenance. For more detailed information, refer to the *Business Rules* topic in the online help and Template User’s Guide PDF.

Client side triggers

The Client Side Triggers feature is maintained in the Clarion dictionary. This provides you with a method to consolidate Clarion logic code in a single location which would normally be placed throughout an Application at embed points. There are “trigger-points” before and after the file operations for Insert, Update, and Delete. This means that every time a data row operation occurs you can enforce specific Clarion logic to be evaluated regardless of when or by which procedure it is called. For more detailed information, refer to the *Dictionary Triggers* topic in the online help and *IDE Guide* PDF.

Business Graphing

The Clarion Business Graph Classes and Templates provide a Clarion source solution for standard business graphing needs. The templates are very detailed in construction and easy to use with logical prompts. The Graphing solution can display data from a file or queue and has the ability to be placed on a Report structure. There are a large number of 2d and 3d graph types available that can be programmer or user selected at runtime. For more detailed information, refer to the *Graphing* topic in the online help and *Template User's Guide* PDF.

Data exchange using XML

The XML classes and Templates have been created to utilize the highly regarded Open Source Centerpoint XML C++ library. The Clarion XML classes are native and the templates are easy to use code templates providing Import and Export features using Tag or Attribute based XML data, optionally formatted to ADO.Net or ADO 2.6. We even support using a runtime Map for Source/Target column matching. Clarion Queues, Files and Views are the source and target. For more detailed information, refer to the online help and *Clarion XML Support* PDF.

Style Management

The definition of list box Styles been simplified and enhanced with a procedure dialog that allows easy selection of fonts, properties and colors which are saved to be referenced as a style number in your source. For more detailed information, refer to the online help and *IDE Guide* PDF.

Report Output Generators

The Clarion 6 Report Output Generators are comprised of an architecture which performs parsing of your Clarion Reports and using a "plug-in" generator can be transformed into another format. Currently generators are available to create PDF, HTML, XML and TEXT. In order to use the generators a simple Global extension is populated which results in that Generators options being available for each report procedure. For more detailed information, refer to the Advanced Report Generation online help and *Template User's Guide* PDF.

New and Changed Features Matrix

	Enterprise	Professional
Fully pre-emptive Threading Model	X	X
Thread Synchronization Interfaces for Critical Sections, Semaphores, Mutexes and ReaderWriter	X	X
Easily "rebase" the target memory load area for your DLL's with the integrated global template support for memory address selection	X	X
Application Themes which can be saved and reused to further accelerate your development	X	X
Theme Maintenance Wizard provides an easy to use interface for creating and changing Themes	X	X
Application Wizard with additional options and Theme Support	X	X
Procedure Wizard with additional options and Theme Support	X	X
Label Wizard with 70 pre-defined Avery label sizes and layouts	X	X
XP Manifest File Support	X	X
Smart Locators throughout which provide efficient navigation in lists and can be repeated via Ctrl-Enter key combination	X	X
Zoom Alt-F2 window provides an expanded scrollable edit area for entries and prompts	X	X
Popup table / variable window now available in more locations	X	X
The Expression Editor consolidates all Tables, Global, Module and Local data variables and an extensive list of RTL functions for constructing expressions in select template prompts.	X	X
Create a Project (PRJ) from your Application (APP) automatically	X	X
Generate source code for a single Module	X	X
The Dictionary View Toolbox is now available in the Application Generator	X	X
The Formula Editor has been enhanced to provide additional Class support	X	X
Global and procedure support to "swap" the Enter key for the Tab key for field navigation	X	X
Global and procedure support for a Visual Indicator that places a box around the current control, changes the background color, or adds an indicator character next to the control. These options can be mixed and matched, overridden at the procedure level and set for specific control types. There's even an option to use alternate colors for required fields.	X	X

Database Drivers

ADO Data Layer Support providing additional database connectivity options	X	X
OleDB Connection Builder available at design time and runtime	X	X
ODBC driver updated to support multi-table import	X	
BLOB column support for all native SQL drivers	X	X
BLOB column size for TopSpeed (TPS) files is 640 MB	X	X
Oracle Call Interface (OCI) 8.x supported *	X	
Oracle BLOB column types (4) supported *	X	
Functions for Reading and Writing to BLOB columns	X	X

* Oracle Accelerator available as an option for the Professional version

Language / Runtime Library Enhancements

New Inline PRAGMA statement control of the project system from within your source code	X	X
Support of the LAYOUT attribute for the window and report formatters	X	X
New FLAT attribute now supported on list controls	X	X
ICONs of sizes other than 32x32 and 16x16 now supported	X	X
DESTROY(FILE) now supported	X	X
Progress bar support for colors, transparency and range(MIN(LONG) to MAX(LONG)	X	X
Vertical and smooth Progress bar options now supported	X	X
BOXED attribute for TEXT controls supported	X	X
RTF attribute for TEXT controls supported on windows and reports	X	X
MESSAGE() dialog text can be copied to the clipboard	X	X
New Runtime Properties: PROP:Datapath, PROP:NextTabStop, PROP:PreviousTabStop, PROP:Vscroll, PROP:WheelScroll and PROP:WindowsVersion	X	X
Results of ?, *? and *STRING types from functions prototyped with the RAW attribute are now treated as untyped references	X	X
New QUOTE and UNQUOTE functions added	X	X
New BINDEXPRESSION(name, expr) added	X	X
New INSTANCE(variable, threadno) returns the address of a variable or entity's thread instance	X	X

New function SUSPEND, suspends a thread's execution	X	X
New function RESUME, resumes a thread's execution	X	X
New SQL driver string BINDCOLORDER (Bind column order) added	X	X
FILEDIALOG can now accept any sized string for the extension list	X	X

ABC Library Changes

SaveBuffer and RestoreBuffer made virtual to enable users to do BLOB comparisons in a derived version of EqualBuffer if required.	X	X
The ABC INIClass has been enhanced to support using the Windows registry for storage.	X	X
New methods have been added to the ABC Popup Class; DeleteMenu and GetLastNumberSelection methods.	X	X
Add new Dictionary class to make ABC classes safe for a fully preemptive environment.	X	X

Database Dictionary Editor

Reorganized layout maximizes the available on screen information so that you can more easily access property dialogs	X	X
Client side Triggers provide similar functionality to server-side triggers, i.e. code is executed when a table is accessed for any or all of (Add,Update,Delete). The code can be called either before the operation or after, or both. Provides an opportunity for additional validation, computed field values, etc. that is stored in the Dictionary and then incorporated into all applications that share that dictionary. Triggers are valid for both ISAM and SQL tables. When used for SQL tables, client-side triggers provide independence from any particular backend.	X	X
Create a single file conversion program for a single table	X	X
Create a single file conversion program for select tables	X	X
Each database driver now has a custom driver string properties dialog editor providing for "quick and easy" changes	X	X

Template Additions and Enhancements

The **Business Rules Manager** class and templates provide an easy to use method to implement validation logic and apply it throughout your application. You can define a Rule Globally and it is automatically implemented to each procedure that contains the data variable associated with the Rule. You have the option to override and ignore the Rule at the local Procedure level as well as define additional Rules Locally specific to a single Procedure. The runtime characteristic of an implemented Rule is that it can Dis/Enable Un/Hide any number of screen controls and has a mechanism to notify the User as to its' state.	X	X
A comprehensive **XML Class** has been created with code templates which support serializing a Clarion Queue into XML, de-serializing an XML string into a Clarion Queue, mapping headers where possible. Code templates are also provided to perform the same functions for any File or View structure defined. The class contains additional features that permit arbitrary rendering of any XML in a Queue structure.	X	
The **Listbox Format Manager** allows the User to re-order list columns and save the formats for reuse	X	X
VCR Form Navigation permits navigation, inserts, updates and deletes from a form procedure. Advanced features included are Locator and multiple insert support.	X	X
A comprehensive **Business Graph** Class and templates have been created which provide a method to implement standard business graphics on screen and printed reports. An extensive set of classes, methods, properties and templates allow detailed control over presentation and data. Many styles are available to choose from in 2 and 3 dimensional mode.	X	
Advanced QBE Browse list control template adds intuitive drag and drop query capability to your application. This control will create an ISAM filter or SQL select statement depending on the data source. End users have the ability to save their queries and reuse them later. A query may also be applied to Report or Process procedures.	X	X
Report Output Generators are exciting features that allow your printed reports to be saved as *PDF*, *HTML*, *TEXT* or *XML* files. You have complete control over the data that is available for export and how. Each output format is populated as a Global Extension template and procedure specific properties are set locally. Specific features include: PDF - bookmarks, hyperlinks, notes, thumbnails, anchor links and more. HTML - Hyperlinks and a page navigation bar automatically built. XML - Export using Tags, Attributes or a combination. *	X	
* Output generators individually available as an option for the Professional version		
The **ADO Class and template** set are designed to use the new ADO driver layer to construct applications using the Microsoft OleDB/ADO layer. Browse, Form, Report and Process templates are optimized to generate and access data from your SQL data store. The Advanced QBE control is also designed to work with this template set as well as an Export to XML code template.	X	X
A calendar class and template are available date for lookups	X	X

New Browse sort selection options are available. You can use a drop down control, pop-up "menu button" or the traditional tab to select the sort order.	X	X
Speed up your development and reduce your application size with the new runtime key / sort selection options for Reports. This feature is supported in the application and procedure wizards and the choice can be persisted in your theme as the default choice.	X	X
A new control template (with a funny name) BrowseNoRecordsButton helps to automate the enable/disable logic on a browse control when there are no records available on the list. This permits conditional procedure calls when inserting new records.	X	X
Browse column resizing supported with a simple right click on the column boundary	X	X
Filtered locator support has been added to the Clarion chain	X	X
Edit in place EIP has been added to the Clarion chain using the ABC EIP Class	X	X
The ABC EIP templates have been enhanced to make setting the options easier and nearly code free.	X	X
Higher Key component filtering support has been added for the Browse, Drop, Combo, Report, Process and LookupNonRelatedrecord templates in the ABC and Clarion chains. This permits filtering at a given level of multi-component keys.	X	X
Enhanced conditional browse totaling	X	X
Enhanced Print Preview control in the Clarion chain	X	X
Reports now have a data source option; print from File, Queue or Memory!	X	X
A completely new interface for BIND support is now available for all templates. Expressions and Procedures are now supported	X	X
List box style support has been added for each window in the ABC and Clarion chains	X	X
Styles and Tips tab added to the Drop list and Combo controls in the Clarion chain	X	X
Easy to implement color rows and columns for a Browse box, Drop and Combo list box controls in the ABC and Clarion chains.	X	X
Greenbar coloring effect is supported on Browse, Drop and Combo controls in the ABC and Clarion chains.	X	X
A report definition is now supported on any procedure in the ABC and Clarion chains	X	X
Added a %LocalDataDescription symbol as a comment for %LocalData definitions	X	X
Lookups now support SPIN controls in the ABC and Clarion chains	X	X

Some ABC classes are now directly supported in the Clarion chain	X	X
New default WINDOW types have been added	X	X
Parameter passing now supported from the Browse Update Buttons on the Clarion chain.	X	X
Parameter passing from a Lookup field prompt to the lookup procedure in the ABC and Clarion chains	X	X
Added the Additional Field Assignments options for Lookups in the Clarion chain	X	X
"!variableName" format is now supported in the Report, Process, Browse, Drop and Combo filter entry prompts in the ABC Clarion chains.	X	X
Significantly enhanced Sort Order features for the Browse, Process, and Reports procedures	X	X
A new "Assisted" option has been added for building Browse, Report and Process sort orders in the ABC and Clarion chains.	X	X
A List Line Height property has been added	X	X
A new "Conditional Assisted" option has been added for building Report and Process sort orders in the ABC and Clarion chains.	X	X
A new "Dynamic Sorting" option has been added for runtime Report and Process sort orders selection in the ABC and Clarion chains.	X	X
A new "Dynamic Named" option has been added for enhanced runtime Report and Process sort order selection in the ABC and Clarion chains.	X	X

The feature matrix displays highlights contained in Clarion 6. For a more detailed description of new features, enhancements and changes refer to the help system and release notes.