Advanced Programming Resources

SoftVelocity

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

Printed in the United States of America (1003)

Contents:

Customizing the Development Environment

An Overview

The Clarion Development Environment was set up in an optimal manner for developing applications. However, it is also extensible to allow you to modify it to meet your specific needs. You can add menu items to call built-in applets (a small application that performs a single task) or any external tools you wish to use. Each portion of the Clarion Development Environment is an applet that can be called to perform a task. For example, you can call the Source Editor (CWedt) to open and edit a template file.

The configuration settings file —C60PE.INI or C60EE.INI—controls this extensibility. This is a standard Windows .INI file that the Clarion GETINI and PUTINI statements can modify.

As is the case with most Windows applications, the Clarion Development Environment reads the .INI file upon opening and writes to it upon closing. Keep this in mind when editing it. If you make any changes to the .INI file while Clarion is open, these changes will be overwritten when you close Clarion. For this reason, you should use Notepad or any other text editor to edit the .INI file.

Note: We strongly recommend making a backup copy before modifying your .INI file.

Command Line Parameters

There are several command line parameters you can use when you call C60EE.EXE to invoke specified initial actions. All of them will automatically disable the Pick list and prevent it from appearing as the opening dialog.

-P*name*	Sets the initial Project to the passed name.
-M*name*	Makes the Project name.
-R*name*	Makes and runs the Project name.
-**D*name***	Makes and debugs the Project name.
-**E*name***	Opens the name file.

Example:

```
C60EE.EXE -Rtutorial.app
C60EE.EXE -Ehello.clw
```

Non-Modifiable Clarion INI File Sections

User Information

The Clarion Development Environment stores your user information (Name, Company, Serial Number, and Key Number) in the [user] section of the .INI file. You should not modify this section.

Paths

The last opened project or Application is stored in the [paths] section of the .INI file. This is updated every time the environment is closed.

You can modify this section of the INI file, but it is always overwritten when Clarion closes.

Example:

```
[paths]
prjfile=C:\C60\APPS\MYAPP.APP
```

Environment Options

The settings specified in the [Environment] section control the appearance of the Clarion development environment. Two settings are specified in the [Environment] section which are always overwritten when Clarion closes:

maximized=on|off

Specifies whether the environment opens maximized.

quickstart=on|off

Specifies whether Quick Start is on by default.

Other settings appear in this section which may be modified:

windows95dlgs=on|off

Specifies whether to use Windows 95-style common dialogs.

wallpaper=bmpfilename

Specifies the .BMP file the environment displays for wallpaper.

wallpapermode=tiled|centered|full

Specifies how to display the wallpaper BMP file: tiled, centered, or stretched to fill the environment client area.

autopick=on|off

Specifies whether the Pick dialog appears when the environment first opens.

Example:

```
[Environment]
maximized=off
quickstart=off
windows95dlgs=off
wallpaper=c:\C60\images\C60wall.bmp
wallpapermode=tiled
autopick=on
```

The Clarion Applets

The Clarion Development Environment has several built-in applets that you can call from a User Defined Menu or a User Defined Tab on one of the File Dialogs (New, Open, or Pick). These applets are “registered” through the [Clarion Applets] section of the INI file.

You should not modify this section. The applets are documented here so you know which ones you can call from menus or tabs.

Applets marked with an asterisk (*) are for internal use only and cannot be called separately. They are listed here for informational purposes only.

C60edt	The Source Code Editor
C60cif	The Compiler Interface
C60scr	The Window Formatter*
C60prj	The Project Editor
C60rpt	The Report Formatter*
C60adm	The Data Dictionary Editor
C60gen	The Application Generator
C60asv	Application Generator Services*
C60frm	The Formula Editor*
C60brw	The Database Manager
C60qck	The QuickStart Wizard*
C60tty	The Search Files Utility
C60rpc	IDE 32-bit interface*

These are listed in the [Clarion Applets] section of the .INI file.

For example:

```
[Clarion Applets]
_version=44
C60edt=on
C60cif=on
C60scr=on
C60prj=on
C60rpt=on
C60adm=on
C60gen=on
C60asv=on
C60frm=on
C60brw=on
C60qck=on
C60tty=on
C60rpc=on
```

Modifying the Clarion Environment

Specifying User Defined Applications

To call an application or applet form the Clarion development environment, you must first define the application in the [User Applications] section of C60EE.INI. This “registers” the application with the environment. To add your own, modify the [User Applications] section of C60EE.INI in the following manner:

AppReference=CommandLine

AppReference
The name by which the application is referenced.

CommandLine
The text of the command to be launched. This text may contain either the %f or %a expansion macros.

Example:

```
[User Applications]
notepad=notepad %f
wordpad=wordpad %f
notepad.2=notepad c:\windows\win.ini
notepad.3=notepad %a.txt
```

Note: If your application is in a directory that is not in your PATH and is using Long Filenames (Windows 95) you must use the DOS equivalent of the directory and filenames. For example, if Wordpad is in the C:\Program Files\Accessories directory, you must refer to it as: C:\Progra~1\ACCESS~1\wordpad.

Notice the last two lines of the example reference notepad.2 and notepad.3. This defines notepad.2 to open Notepad to edit the WIN.INI file. Notepad.3 is defined to open (or create) a file using the name of the current application with a .TXT extension. You can add an extension User Applications to define multiple occurrences of the same application to behave in different fashions.

The %f expansion macro shown in the example denotes the current filename. This is passed as a command line parameter to the application. Valid expansion macros you can use as command line parameters are:

%f Filename and Path

%-f Filename only

%a Current Application (*.APP) or Project (*.PRJ)

Additionally, you can add an extension to an expansion macro to change the extension of the filename passed to it. For example, calling notepad with the parameter %a.txt would open a file using the current application's filename and a .TXT extension. This enables you to define a User Application that you can call to open (or create) a text file matching the current application name.

Adding choices to the Clarion Menu

To call an external application from the Clarion development environment, you must first define the application in the [User Applications] section of C60EE.INI as described above. This "registers" the application with the environment.

Next, you add Menu(s) and define the menu selections in the [User Menus] section in the following manner:

n=MenuDisplayText/MenuDisplayText|Applet

n The number of the entry

MenuDisplayText

The text to display in the action bar.

ItemDisplayText

The text to display in the Drop-Down List.

Applet

The applet to call.

Example:

```
[User Menus]
_version=1
1=&Editors/&Wordpad|wordpad
2=&Editors/&Notepad|notepad
3=&Utilities/&Calculator|Calculator
4=&Utilities/&Paint|Paint
```

Note: The first line (_version=1) denotes a version number used internally. You should not modify that line.

Adding choices to the Clarion Setup Menu

You can also add choices to the Setup Menu in the Clarion Development Environment. To call an external application from the Clarion development environment, you must first define the application in the **[User Applications]** section of C60EE.INI as described above. This “registers” the application with the environment.

Next, you add menu choices and define the menu selections in the **[Setup Menu]** section as shown in numbers 10 and 11 below. Number 11 uses a User Defined Application (See Specifying User Defined Applications). Numbers 4, 8, and 10 create separators by using the n=- syntax.

n=Text|Application

n The number of the entry

Text
The text to display on the menu.

Application
The Clarion applet or application to open. Any external applications must be defined in the **[User Applications]** section.

Example:

```
[Setup Menu]
_version=10
1=&Editor Options|CWedt
2=&Dictionary Options|CWadm
3=&Application Options|CWgen.1
4=-
5=&Template Registry|CWgen.103
6=Data&base Driver Registry|CWasl.38
7=&VBX Custom Control Registry|CWasl.37
8=-
9=Edit Redirection &File|CWasl.36
10=-
11=Edit WIN.INI|notepad.2
```

Note: The first line (_version=10) denotes a version number used internally. You should not modify that line.

Adding File Masks to File Types Drop Down Lists

The **[File Types]** section defines the masks used by the drop-down list in any of the Clarion file dialogs (New, Open, or Pick) and the applet or application that opens or creates the file. One additional entry should follow, in the format of ALL=*n*. This defines which should be used for ALL files, allowing more than one entry to use *.* as the file mask.

Note: The first line (_version=41) denotes a version number used internally. You should not modify that line.

The defaults are :

```
[File Types]
_version=41
1=Application (*.app)=*.app|CWgen
2=Dictionary (*.dct)=*.dct|CWadm
3=Project (*.prj)=*.prj|CWprj
4=Clarion source (*.clw,*.inc,*.trn)=*.clw;*.inc;*.trn|CWedt
5=Report (*.txr)=*.txr|CWRW
6=Text (*.txt)=*.txt|CWedt
7=Database tables=*.tps;*.dbf;*.dat;*.db|CWbrw
8=All Database tables (*.*)=*.*|CWbrw
9=All files (*.*)=*.*|CWedt
all=9
```

To add your own, modify the **[File Types]** section of C60EE.INI in the following manner:

n=DisplayText=Filemask|Application

n	The number of the entry
DisplayText	The text to display in the Drop-Down List.
Filemask	The mask to use when listing files.
Application	The Clarion applet or external application to use to open the file. External applications must be defined in the **[User Applications]** section.

Example:

```
[File Types]
1=Application (*.app)=*.app|CWgen
2=Clarion source (*.clw)=*.clw|CWedt
3=Dictionary (*.dct)=*.dct|CWadm
4=Project (*.prj)=*.prj|CWprj
5=Text (*.txt)=*.txt|notepad
6=Doc (*.doc)=*.doc|wordpad
7=Database files=*.tps;*.dbf;*.dat;*.db|CWbrw
8=All Database files (*.*)=*.*|CWbrw
9=All files (*.*)=*.*|CWedt
all=9
```

Adding Tabs to New, Open, or Pick File Dialogs

Each of the Clarion File dialogs can be modified to include TABs of your choice.

To add your own, modify the [Pick Dialog], [New Dialog], or ,[Open Dialog] sections of C60EE.INI in the following manner:

n=TabText=FileMask|Application

n The number of the entry

TabTex
The text to display on the Tab.

FileMask
The extension of the files displayed.

Application
The Clarion applet or application to use to open the file. Any external applications must be defined in the [User Applications] section.

Note: The first line (_version=41) denotes a version number used internally. You should not modify that line.

Example:

```
[Pick Dialog]
_version=41
1=&Application=app|CWgen
2=&Dictionary=dct|CWadm
3=&Project=prj|CWprj
4=Data&base=*|CWbrw
5=&Source=clw|CWedt
6=&Report=txr|CWRW
7=A&ll=*|CWasl
xpos=10
ypos=10
height=160
width=320
default=-2

[Open Dialog]
_version=41
1=&Application=app|CWgen
2=&Dictionary=dct|CWadm
3=&Project=prj|CWprj
4=Data&base=*|CWbrw
5=&Source=clw|CWedt
6=&Report=txr|CWRW
7=A&ll=*|CWasl

[New Dialog]
_version=41
```

```
1=&Application=app|CWgen
2=&Dictionary=dct|CWadm
3=&Project=prj|CWprj
4=&Source=clw|CWedt
5=&Report=txr|CWRW
```

The last two lines of each of these sections affect the TABs on each of the File Dialogs. The lines=2 specifies that enough space should be allotted for two rows of TABs. This does not cause TABs to split onto two lines, that is controlled automatically by the space needed.

The default= line controls which TAB appears on top when the dialog is called. If the number is positive, the TAB equated to the number will always open on top. If the number is negative, the environment will save the last open tab's number to that position causing the dialog to reopen with the TAB which was selected last on top.

Specifying Make File Types

The Compiler Interface uses Project files or Application files when compiling and linking. If you like, you can define alternate extensions for these types of files. By defining your own, for example, you can compile a project file with an extension of .MAK. The file must be in the same format as other project files (see Chapter 13—The Project System).

To add your own, modify the [Make File Types] section of C60EE.INI in the following manner:

n=DisplayText=Filemask|Applet

n The number of the entry

DisplayText
The text to display in the Drop-Down List.

Filemask
The mask to use when listing files.

Applet
The Clarion applet to use to open the file.

Note: The first line (_version=41) denotes a version number used internally. You should not modify that line.

Example:

```
[Make File Types]
_version=41
1=Application (*.app)=*.app|CWgen
2=Project File (*.prj)=*.prj|CWcif
3=Text Project File (*.pr)=*.pr|CWcif
```

Print Specifications

By default, all files printed from the environment have a header listing the filename, date, and page number. This is controlled by the [printing] section of the INI file. You can modify this section to print a different header if desired. This uses any of the following expansion macros:

%F Filename

%D Current System Date

%T Current System Time

%P Current Page Number

Note: The first line (_version=1) denotes a version number used internally. You should not modify that line.

```
[printing]
_version=1
header=File: %F  Date: %D  Time: %T   Page: %P
```

Environment Option Settings

Auto Populate Options

The **[Auto Populate]** section specifies where controls are placed when auto populating (by double-clicking) with the Fields Box in the formatters.

startX=5
Specifies the starting position on the X axis (horizontal axis) for the first control placed.

startY=5
Specifies the starting position on the Y axis (vertical axis) for the first control placed.

incrementX=50
Specifies the number of dialog units to increment the horizontal offset for controls from the associated prompt.

incrementY=16
Specifies the number of dialog units to increment the vertical space between auto-populated controls.

alignment=top
Specifies whether to vertically align controls to the top, bottom, center, or default (0) of its associated prompt.

Example:

```
[Auto Populate]
startX=5
startY=5
incrementX=50
incrementY=16
alignment=0
```

Dictionary Options

Data Dictionary Options are controlled by the **[Dictionary Editor]** section of the INI file. These options are all modifiable through the development environment, but you can also change them in the INI file.

Default driver=DriverName
The default database driver for new files in a dictionary

Set threaded=on|off
Specifies whether or not new file definitions default to adding the THREAD attribute.

Assign message=on|off
Specifies whether to use the field descriptions you specify when defining a field as the text for the MSG attribute.

Order files=on|off
Specifies whether to display files in alphabetical order or in the order in which they were created.

Display field type=on|off
Specifies whether to display the field's data type in the Fields list.

Display field prefix=on|off
Specifies whether to display the prefix in the Fields list.

Display field picture=on|off
Specifies whether to display the screen display picture in the Fields list.

Display field description=on|off
Specifies whether to display the description in the Fields list.

Display key type=on|off
Specifies whether to display the key type in the Key list.

Display key prefix=on|off
Specifies whether to display the prefix in the Key list.

Display key description=on|off
Specifies whether to display the description in the Key list.

Display key primary=on|off
Specifies whether to display "Primary" if the Key is the Primary Key.

Display key unique=on|off
Specifies whether to display "Unique" if the Key is unique.

Display key attributes=on|off
Specifies whether to display the other key attributes in the Key list.

Display file driver=on|off
Specifies whether to display the file driver in the Files list.

Display file prefix=on|off
Specifies whether to display the file prefix in the Files list.

Display file description=on|off
Specifies whether to display the file description in the Files list.

```
Example:
[Dictionary Editor]
Default driver="TOPSPEED"
Set threaded=on
Set OEM=off
Assign message=on
Assign tooltip=on
Order tables=on
Display column type=off
Display column prefix=off
Display column deriviation=off
Display column picture=off
Display column description=on
Display key type=off
Display key prefix=off
Display key description=on
Display key primary=off
Display key unique=off
Display key attributes=off
Display database driver=off
Display database prefix=off
Display table description=on
PromptQuickLoad=on
UseQuickLoad=off
```

Project System Options

Project System Options are controlled by the [Project System] section of the INI file. These options are all modifiable through the development environment, but you can also change them in the INI file.

automake=on|off

Specifies whether an application should be automatically compiled and linked (if necessary) before running.

autosave=on|off

Specifies whether an application should be automatically saved before it is compiled and linked.

runmin=on|off

Specifies whether the development environment should be minimized during execution of the application.

runwait=on|off

Specifies whether the Project System should suspend Clarion until after you terminate the application upon executing it with the Run command.

newtype=<extension>

Specifies any new project file types you have defined in the [Make File Types] section of the .INI file. This line is added automatically by the environment when a new Make File Type is added.

Default32bit=on|off

Specifies any new project is created as 32-bit by default (always on).

Debugresume=on|off

Specifies whether the debugger starts in “resume previous debug session” mode.

Example:

```
[Project System]
automake=on
autosave=on
runmin=off
runwait=off
default32bit=on
defmax=off
defx=0
defy=0
defw=200
defh=140
mdefx=0
mdefy=0
mdefw=225
mdefh=140
```

Application Generator Options

Application Generator Options are controlled by the **[Application]** section of the INI file. These options are all modifiable through the development environment, but you can also change them in the INI file.

CondGeneration=on|off

Specify ON to ensure only source code modules changed since the last make should be compiled.

DebugGeneration=on|off

Specifies whether or not to write to a text file to log events for the Application Generator, and turns logging on and off. In case of a fatal error by the Application Generator, this log provides a trace to identify where the problem occurred. You specify the file name in the Debug Filename entry. This is particularly useful when designing templates.

Repeat Procedures=on|off

Specifies whether or not the Application Generator displays the names of all procedures, as it encounters them in the source code modules, in the progress box displayed during code generation.

PopulateMain=on|off

Specifies whether or not the Application Generator writes procedures to the main source code module. When this option is off, the main module only contains the MAIN procedure, program global code, internally generated procedures standard for every application. All other procedures reside in other file(s).

RequireDictionary=on|off

Specifies whether each new application must have a data dictionary.

DefaultDictionary=dictionaryname

Specifies the name of the default data dictionary.

MultiUser=on|off

Specifies whether to use file management options for multiple developer projects. See the Multi-Programmer Development appendix in the User's Guide for more information.

ApplicationWizard=on|off

Specifies whether to use the Application Wizard by default when creating a new application. You can override this choice when creating an application by checking or unchecking the Application Wizard box in the Application Properties dialog.

ProcedureWizard=on|off

Specifies whether to use a the appropriate Procedure Wizard by default when creating a new procedure. You can override this choice when creating a procedure by checking or unchecking the Procedure Wizard box in the Select Procedure Type dialog.

NameClash=n

Specifies how the Application Generator handles procedure names from an imported application file which clash with procedure names already resident.

The choices are:

0 - Query on First Clash

1 - Ask for Alternative

2 - Auto Rename

3 - Replace Previous

ModuleProcs=n

Specifies the number of procedures that the Application Generator writes to each source code module. This can affect compile time when used with Conditional Generation turned on. Specifying one procedure per module, for example, means that each successive compile rebuilds only those procedures changed since the last one, and no more. The down side to this is that it requires more disk space. Generally, a smaller number is faster.

Debug File=filename

Specifies the file to which template debug information is written.

DisableField=on|off

Specifies whether or not template-generated field-specific (#FIELD) prompts will not display. This does not disable prompts created by Control Templates.

Cw21ProcedureCall=on|off

When on, specifies the Procedures button on a Procedure Properties dialog calls the Clarion version 2 Procedures dialog.

LocalMap=on|off

When on, specifies local MAP structures are generated in each source module.

TranslateControl=on|off

When on, specifies a dialog appears when controls are populated asking whether to populate the control itself or a related Control Template.

AskOnOK=on|off

When on, specifies an action confirmation dialog appears when exiting with the OK button.

AskOnCancel=on|off

When on, specifies an action confirmation dialog appears when exiting with the Cancel button.

AskOnClose=on|off

When on, specifies an action confirmation dialog appears when exiting.

LegacyAction=0|1|2

Specifies the appearance of Legacy embeds in the Embeditor and Embeds dialog: 0 = Show All and Generate All, 1 = Show Filled and Generate All, 2 = Ignore All.

ShowPriority=on|off

When on, specifies priority embed numbers appear as comment in the Embeditor.

AlphaSortEmbeds=on|off

When on, specifies embed points appear alphabetically in the Embeds dialog.

CommentEmbeds=on|off

When on, specifies embed points appear with comments in the generated source and the Embeditor.

StrictRuntimeError=on|off

IgnoreFamily=on|off

TranslateErrors=on|off

CacheGlobalEmbed=on|off

CacheProcedureEmbed=on|off

RefreshEmbedAfterSource=on|off

MaxCachedEmbeds=32

RecursiveText=" (Recursive)"
Specifies the description text to display for recursive procedures detected by the Application Generator.

ExpandAboveText=" (Expanded Above)"
Specifies the description text to display for repeated functions marked by the Application Generator.

LastSelectTab=1

UseLongFilenames=on|off

ComplexEmbedTree=on|off

When ON, the filled embeds for a procedure show the expanded embed labels on the right pane. When OFF the embeds display without the descriptive labels.

Example:

```
[Application]
CondGeneration=on
DebugGeneration=off
Repeat Procedures=on
PopulateMain=off
RequireDictionary=on
MultiUser=off
ApplicationWizard=on
ProcedureWizard=on
NameClash=0
ModuleProcs=1
Debug File="c:\tpldebug.txt"
DefaultDictionary=""
DisableField=off
TranslateControl=on
LocalMap=on
AskOnOK=off
AskOnClose=on
AskOnCancel=on
LegacyAction=1
ShowPriority=on
AlphaSortEmbeds=on
CommentEmbeds=off
StrictRuntimeError=on
IgnoreFamily=off
Cw21ProcedureCall=off
TranslateErrors=on
CacheGlobalEmbed=off
CacheProcedureEmbed=off
RefreshEmbedAfterSource=on
MaxCachedEmbeds=32
RecursiveText=" (Recursive)"
ExpandAboveText=" (Expanded Above)"
LastSelectTab=1
UseLongFilenames=off
ComplexEmbedTree=on
```

Dictionary Synchronization Options

Dictionary Synchronization Options are controlled by the [Control Synchronization] section of the INI file. These options are all modifiable through the development environment, but you can also change them in the INI file.

ApplicationLoad=on|off

Specify ON to re-synchonize the application with the dictionary every time the application is opened.

Windows=on|off

Specify ON to re-synchonize windows with the dictionary when the application is re-synchonized.

Reports=on|off

Specify ON to re-synchonize reports with the dictionary when the application is re-synchonized.

SyncVariables=on|off

Specify ON to re-synchonize controls for memory variables when the application is re-synchonized.

PrimaryAttrOnly=on|off

Specify ON to re-synchonize only the “primary” attributes.

ChangeControlTypes=on|off

Specify ON to allow control types to change.

SyncDropNonDrop=on|off

Specify ON to allow a LIST with the DROP attribute to change to a LIST without the DROP attribute.

ClearHelpEtc=on|off

Specify ON to clear the Help tab attributes for a control when omitted in the dictionary.

OverrideSize=on|off

Specify ON to allow the dictionary to override the size of controls.

Refreeze=on|off

Specify ON to refreeze a frozen control after synchronization.

IgnoreFreeze=on|off

Specify ON to synchronize all controls, despite their freeze settings.

SyncListbox=on|off

Specify ON to synchronize LIST box FORMAT strings.

SyncHeading=0|1|2|3

Specify heading sychronization: 0 for Always, 1 for In Dictionary, 2 for Dictionary and Window, and 3 for Never.

WarnDialog=on|off

Specify ON to display a warning dialog if problem occur during synchronization.

WarnOnResize=on|off

Specify ON to add a warning to the log file when a control changes size.

WarnFile=textfile.txt

Specify the name of the .TXT file to receive the synchronization report.

ClearAlrtEtc=on|off

Specify ON to clear all miscellaneous attributes if blank in the dictionary.

ClearFont=on|off

Specify ON to clear the FONT attribute if blank in the dictionary.

ClearCursor=on|off

Specify ON to clear the CURSOR attribute if blank in the dictionary.

ClearIcon=on|off

Specify ON to clear the ICON attribute if blank in the dictionary.

ClearAlert=on|off

Specify ON to clear the ALRT attribute if blank in the dictionary.

ClearKey=on|off

Specify ON to clear the KEY attribute if blank in the dictionary.

ClearColor=on|off

Specify ON to clear the COLOR attribute if blank in the dictionary.

ClearTally=on|off

Specify ON to clear the TALLY attribute if blank in the dictionary.

Example:

```
[Control Synchronization]
ApplicationLoad=off
Windows=on
Reports=on
SyncVariables=off
PrimaryAttrOnly=off
ChangeControlTypes=on
SyncDropNonDrop=off
ClearHelpEtc=off
ClearAlrtEtc=off
OverrideSize=on
Refreeze=on
```

```
IgnoreFreeze=off
SyncListbox=on
SyncHeading=2
WarnDialog=on
WarnOnResize=off
WarnFile="sync.txt"
ClearFont=off
ClearCursor=off
ClearIcon=off
ClearAlert=off
ClearKey=off
ClearColor=off
ClearTally=off
```

Template Registry Options

Template Language code can be logically split among many files. Clarion uses the files to produce one logical template set for creating applications. The Registry Options are mainly for programmers who produce their own template files or make modifications to the default templates. These options are all modifiable through the development environment, but you can also change them in the INI file. Template Registry Options are controlled by the **[Registry]** section of the INI file.

Reregister if changed=on|off

Specify ON to automatically re-register your templates when the Application Generator detects a change.

Update Template Chain=on|off

Specify ON to automatically update the Template files when making a change in the Template Registry.

Recreate deleted templates=on|off

Specify ON to specify the Application Generator should re-generate the .TPL and .TPW files from REGISTRY.TRF, should the files be deleted.

MessageLines=3

Specifies what displays during generation. The choices are:

0-No Messages

1-Module Names

2-Module and Procedure Names

3-All Messages

Default Width

Specifies the default width of the template registry window.

Default Height

Specifies the default height of the template registry window.

Enable Assert

When set to ON, enables #ASSERT checking.

Example:

```
[Registry]
Reregister if changed=on
Update Template Chain=on
Recreate deleted templates=off
MessageLines=2
Default Width=300
Default Height=200
EnableAssert=on
```

Window Formatter Options

Window Formatter Options are controlled by the [Window Formatter] section of the INI file. These options are all modifiable through the development environment, but you can also change them in the INI file.

grid=on|off

Specifies whether or not the Snap to Grid function is enabled on the sample window.

GridX=n

Specify the horizontal distance between the grid dots (x axis).

GridY=n

Specify the vertical distance between the grid dots (x axis).

GridOriginX=n

Specify the horizontal coordinate at which to begin placing the grid dots.

GridOriginY=n

Specify the vertical coordinate at which to begin placing the grid dots.

ScreenExtents=on|off

Specifies whether or not the screen resolution boundaries are displayed on the sample window.

SnapResize=on|off

Specifies whether or not to snap to the nearest grid point grid when resizing from the right or bottom edges.

showtoolbox=on|off

Toggles displaying the Controls tool box.

showalignbox=on|off

Toggles displaying the Alignment Box.

showpropertybox=on|off

Toggles displaying the Property Box.

showfieldbox=on|off

Toggles displaying the Fields Box

commandbox_app_docked =on|off

Designates whether or not the Command Toolbox is docked.

commandbox_app_xpos =n

The horizontal position of the Command Toolbox.

commandbox_app_ypos =n

The vertical position of the Command Toolbox.

commandbox_app_width =n

The width of the Command Toolbox.

commandbox_app_height=n

The height of the Command Toolbox.

controlbox_app_docked =on|off

Designates whether or not the Controls Toolbox is docked.

controlbox_app_xpos =n

The horizontal position of the Controls Toolbox.

controlbox_app_ypos =n

The vertical position of the Controls Toolbox.

controlbox_app_width =n

The width of the Controls Toolbox.

controlbox_app_height=n

The height of the Controls Toolbox.

controlbox_app_horiz=n

The number of buttons to display on each row.

propertybox_app_docked =on|off

Designates whether or not the Property Box is docked.

propertybox_app_xpos =n

The horizontal position of the Property Box.

propertybox_app_ypos =n

The vertical position of the Property Box.

propertybox_app_width=n

The width of the Property Box.

propertybox_app_height=n

The height of the Property Box.

propertybox_app_horiz=n

The number of buttons to display on each row.

alignbox_app_docked =on|off

Designates whether or not the Alignment Toolbox is docked.

alignbox_app_xpos =n

The horizontal position of the Alignment Box.

alignbox_app_ypos=n

The vertical position of the Alignment Box.

alignbox_app_width=n

The width of the Alignment Box.

alignbox_app_height=n

The height of the Alignment Box.

alignbox_app_horiz=n

The number of buttons to display on each row.

Example:

```
[Window Formatter]
_version=51
Grid=off
GridX=4
GridY=4
GridOriginX=0
GridOriginY=0
ScreenExtents=on
SnapResize=off
commandbox_app_docked=2
commandbox_app_xpos=0
commandbox_app_ypos=0
commandbox_app_width=215
commandbox_app_height=18
propertybox_app_docked=16
propertybox_app_xpos=20
propertybox_app_ypos=0
propertybox_app_width=130
propertybox_app_height=59
propertybox_app_horiz=2
controlbox_app_docked=16
controlbox_app_xpos=20
controlbox_app_ypos=20
controlbox_app_width=92
controlbox_app_height=104
controlbox_app_horiz=5
alignbox_app_docked=16
alignbox_app_xpos=80
alignbox_app_ypos=20
alignbox_app_width=74
alignbox_app_height=88
alignbox_app_horiz=4
```

Control Default Size Options

Control Default Size Options are controlled by the **[Control Defaults]** section of the INI file. These options are all modifiable through the development environment under the Window Formatter Options, but you can also change them in the INI file.

All these options receive the default value for the size (width or height) of the specified control. If the value assigned is zero (0), the size defaults to the default values assigned by the runtime library (no value appears in the control's AT attribute). Positive values indicate the default value for the control if no other control's of that type have been populated (otherwise, the size of the most common control of that type is used). Negative values indicate the (absolute) value for the control, whether other control's of that type have been populated or not.

entry_h=*n*

Specifies the default height of an ENTRY control.

entry_w=*n*

Specifies the default width of an ENTRY control.

button_h=*n*

Specifies the default height of a BUTTON control.

button_w=*n*

Specifies the default width of a BUTTON control.

spin_h=*n*

Specifies the default height of a SPIN control.

spin_w=*n*

Specifies the default width of a SPIN control.

text_h=*n*

Specifies the default height of a TEXT control.

text_w=*n*

Specifies the default width of a TEXT control.

list_h=*n*

Specifies the default height of a LIST control.

list_w=*n*

Specifies the default width of a LIST control.

combo_h=*n*

Specifies the default height of a COMBO control.

combo_w=*n*

Specifies the default width of a COMBO control.

Example:

```
[control defaults]
entry_h=-10
entry_w=0
button_h=-14
button_w=48
spin_h=-10
spin_w=0
text_h=50
text_w=50
list_h=100
list_w=0
combo_h=-10
combo_w=0
```

Report Formatter Options

Report Formatter Options are controlled by the **[Report Formatter]** section of the INI file. These options are all modifiable through the development environment, but you can also change them in the INI file.

grid=on|off

Specifies whether or not the Snap to Grid function is enabled on the sample window.

GridX=*n*

Specify the horizontal distance between the grid dots (x axis).

GridY=*n*

Specify the vertical distance between the grid dots (x axis).

showtoolbox=on|off

Toggles displaying the Controls tool box.

showalignbox=on|off

Toggles displaying the Alignment Box.

showpropertybox=on|off

Toggles displaying the Property Box.

showfieldbox=on|off

Toggles displaying the Fields Box

commandbox_app_docked =on|off

Designates whether or not the Command Toolbox is docked.

commandbox_app_xpos =*n*

The horizontal position of the Command Toolbox.

commandbox_app_ypos =*n*

The vertical position of the Command Toolbox.

commandbox_app_width =*n*

The width of the Command Toolbox.

commandbox_app_height=*n*

The height of the Command Toolbox.

controlbox_app_docked =on|off

Designates whether or not the Controls Toolbox is docked.

controlbox_app_xpos =*n*

The horizontal position of the Controls Toolbox.

controlbox_app_ypos =*n*

The vertical position of the Controls Toolbox.

controlbox_app_width =*n*

The width of the Controls Toolbox.

controlbox_app_height=*n*

The height of the Controls Toolbox.

controlbox_app_horiz=*n*

The number of buttons to display on each row.

propertybox_app_docked =on|off

Designates whether or not the Property Box is docked.

propertybox_app_xpos =*n*

The horizontal position of the Property Box.

propertybox_app_ypos =*n*

The vertical position of the Property Box.

propertybox_app_width=*n*

The width of the Property Box.

propertybox_app_height=*n*

The height of the Property Box.

propertybox_app_horiz=*n*

The number of buttons to display on each row.

alignbox_app_docked =on|off

Designates whether or not the Alignment Toolbox is docked.

alignbox_app_xpos =*n*

The horizontal position of the Alignment Box.

alignbox_app_ypos=*n*

The vertical position of the Alignment Box.

alignbox_app_width=*n*

The width of the Alignment Box.

alignbox_app_height=*n*

The height of the Alignment Box.

alignbox_app_horiz=*n*

The number of buttons to display on each row.

pagewidth=*n*

The default paper size width

pageheight=*n*

The default paper size height.

Example:

```
[report formatter]
_version=51
grid=off
showfieldbox=off
commandbox_app_docked=2
commandbox_app_xpos=0
commandbox_app_ypos=0
commandbox_app_width=177
commandbox_app_height=18
propertybox_app_docked=16
propertybox_app_xpos=20
propertybox_app_ypos=0
propertybox_app_width=130
propertybox_app_height=59
propertybox_app_horiz=2
controlbox_app_docked=16
controlbox_app_xpos=20
controlbox_app_ypos=20
controlbox_app_width=80
controlbox_app_height=66
controlbox_app_horiz=4
alignbox_app_docked=16
alignbox_app_xpos=80
alignbox_app_ypos=20
alignbox_app_width=56
alignbox_app_height=72
alignbox_app_horiz=3
gridx=100
gridy=100
pagewidth=8500
pageheight=11000
```

Editor Options

The Source Editor's Options are controlled by the **[editor]** section of the INI file. These options are all modifiable through the development environment, but you can also change them in the INI file.

Note: The first line (_version=4) denotes a version number used internally. You should not modify that line.

font=system

Specifies the font to use in the editor. The default is the system font—the default monospaced font on your system.

fontsize=*n*

Specifies the font size to use in the editor. The default size is 8.

maximized=off

Specifies whether the Source Editor window is maximized.

insert=on

Specifies whether the Source Editor is in Insert mode.

Example:

```
[editor]
_version=4
font=system
fontsize=8
insert=on
maximized=off
zonedinsert=off
autoindent=on
wordwrap=on
entersplit=on
splitfirstbelow=off
autodelete=on
markline=off
markcol=off
hometobol=off
showfieldbox=on
copyremblock=on
syntaxcolor=on
autoreload=off
nobackup=off
autobackup=0
tabsize=4
sc_extensions=clw;cla;equ;inc
```

Additional Editor options are found in an associated INI file, C60EDT.INI. These options are described in detail in the *User's Guide - Chapter 9 - Text Editor INI File*. In that section, Key and Color Mapping are discussed. Although these options are generally set up in the environment, there are some custom key and color mappings only available by modifying the INI file.

Editor Tabs

These options are all modifiable and stored internally through the development environment, but you can also change them externally in the INI file using the following syntax.

Extension=TabSpacing

Extension	The file extension for which the specified tabs are used. Any valid DOS file extension can be used including wildcards.
TabSpacing	The spacing of tabs to use.

Note: The first line (_version=1) denotes a version number used internally. You should not modify that line.

Example:

```
[Editor Tabs]

_version=1

CLW=T...T...T...T...T...T...T...T...T...T...T...T...T...T...T...T...T...T
TXT=T...T...T...T...T...T...T...T...T...T...T...T...T...T...T...T...T...T
???=T...T...T...T...T...T...T...T...T...T...T...T...T...T...T...T...T...T
```

Using Clarion as a DDE Server

Overview

Dynamic Data Exchange (DDE) is a very powerful Windows tool that allows a user to send or access data from another separately executing Windows application. The Clarion development environment is a DDE server which you can call from a client application. Using Clarion as a DDE server allows you to write programs which call the Clarion development environment to perform a task or a set of tasks.

For example, you can write a program to register a new template class during the template set's installation program. Another example is a program that manages all of your development projects and will generate, compile, and link multiple applications.

DDE is based upon establishing "conversations" (links) between two concurrently executing Windows applications. One of the applications acts as the DDE server to provide the data, and the other is the DDE client that receives the data. A single application may be both a DDE client and server, getting data from other applications and providing data to other applications. Multiple DDE "conversations" can occur concurrently between any given DDE server and multiple clients.

When Clarion opens, it establishes itself as a DDE server. To accomplish this it must:

- Register with Windows as a DDE server, using the DDESERVER procedure.
- Provide services that are available to client applications. The available services are detailed in this chapter.
- When DDE is no longer required, it terminates the link by using the DDECLOSE statement. It will terminate the DDE connection automatically when the user closes Clarion.

To be a DDE client, a Clarion application must:

- Open at least one window, since all DDE services must be associated with a window. Additionally, the application should set the SYSTEM's DDETimeout property (SYSTEM{Prop:DDETimeOut} = nn) to an appropriate amount of time (depending on the service the application will request).
- Open a link to a DDE server as its client, using the DDECLIENT procedure.

  The DDE process posts DDE-specific field-independent events to the ACCEPT loop of the window that opened the link between applications as a server or client.

- Ask the server for data, using the DDEREAD statement, or ask the server for a service using the DDEEXECUTE statement.
- When DDE is no longer required, terminate the link by using the DDECLOSE statement.

The DDE procedures are prototyped in the DDE.CLW file, which you must INCLUDE in your program's MAP structure.

To INCLUDE the file in the Application Generator:

1. Press the Global button.
2. Press the Embeds button.
3. Highlight Inside the Global Map, then press the Insert button.
4. Highlight Source, then press the Select button.
5. In the Source Editor, type the following:

```
INCLUDE('dde.clw')
```

6. Save and Exit the Source Editor.
7. Press the Close button, then press the Ok button.

Connect to Clarion as DDE Server

Server=**DDECLIENT**('ClarionWin')

Server A LONG variable to hold the DDE client channel number.

DDECLIENT This procedure returns a new DDE client channel number for the application and topic. If the application is not currently executing, DDECLIENT returns zero (0).

ClarionWin The identifier for Clarion.

This connects your client application to the Clarion environment to use it as a server. A connection must be made to the server before any other commands can be issued.

The example below demonstrates how you can determine if C60EE is running and RUN it if it isn't.

Example:

```
!Embedded Source in Window Event Handling- after generated code, Timer


Server = DDECLIENT('ClarionWin')

IF Server                   ! If the server is running
  DO ProcedureReturn        ! Return to caller
ELSIF NOT Flag#             ! Check flag to see if RUN has been issued
  RUN('C60EE')              ! Start it
  Flag# =1                  ! Set flag to show run has been issued
END
```

Disconnect from the Clarion DDE Server

DDECLOSE(*Server*)

DDECLOSE The procedure which terminates the DDE server link.

Server A LONG variable to hold the DDE client channel number.

This terminates the DDE channel between your client application and the Clarion environment server. The connection must be made to the server before any other commands can be issued.

Example:

```
Server = DDECLIENT('ClarionWin')

!...some executable code

DDECLOSE(Server)
```

Export a Dictionary to Text (TXD) format

DDEEXECUTE(*Server*,'[**ExportDct**(*DctName,TextFile,ScreenOutput*)]')

DDEEXECUTE	Send a command string to a DDE server.
Server	A LONG containing the DDE client channel number.
ExportDct	The service of Exporting a Dictionary to Text format
DctName	Clarion Dictionary from which the TXD will export.
TextFile	The file to which the dictionary is exported.
ScreenOutput	If one (1) or TRUE, screen output is displayed, if zero (0) or FALSE, screen display is suppressed. If omitted, screen output is suppressed.

This service allows you to export a dictionary to text format (.TXD). A connection must be made to the server before this (or any other) command can be issued. You can export a TXD file from an open Dictionary.

Example:

```
Server      LONG
DDEErrorMsg CSTRING(300)
DDEErrorNum USHORT

 CODE

 SYSTEM{PROP:DDETimeOut} = 12000       ! Time out after two minutes
 DDEEXECUTE(Server,'[ExportDct(c:\C60\APPS\Myapp1.DCT,MYTXD.TXD,0)]')
 DO CheckDDEError

CheckDDEError   ROUTINE
 DDEErrorMsg = ''
 err# = ERRORCODE()
 ! Check for DDE error
 IF err# > 600       !ERRORCODE is DDE related
  IF err# = 603      !! DDEExecute Failed
   DDEREAD(Server, DDE:manual, 'GetErrorNum', DDEErrorNum)
   DDEREAD(Server, DDE:manual, 'GetErrorMsg', DDEErrorMsg)
   MESSAGE('Error ' & DDEErrorNum & ' : ' & CLIP(DDEErrorMsg))
   ELSIF err# = 605 ! Timeout Error
    MESSAGE('DDE timeout')
  ELSE
   MESSAGE(err#)
  END
 END
```

Import a Dictionary from Text (TXD) format

DDEEXECUTE(*Server*,'[**ImportDct**(*TextFile,DctName,ScreenOutput*)]')

DDEEXECUTE	Send a command string to a DDE server.
Server	A LONG containing the DDE client channel number.
ImportDct	The service of Importing a Dictionary from Text format
TextFile	The file from which the dictionary is imported.
DctName	Clarion Dictionary to which the TXD will import.
ScreenOutput	If one (1) or TRUE, screen output is displayed, if zero (0) or FALSE, screen display is suppressed. If omitted, screen output is suppressed.

This service allows you to import a dictionary from text format (.TXD). A connection must be made to the server before this (or any other) command can be issued. You cannot import a TXD file to an open Dictionary.

Example:

```
Server      LONG
DDEErrorMsg CSTRING(300)
DDEErrorNum USHORT

 CODE

 SYSTEM{PROP:DDETimeOut} = 12000       ! Time out after two minutes
 DDEEXECUTE(Server,'[ImportDct(MYTXD.TXD,c:\C60\APPS\Myapp1.DCT,0)]')
 DO CheckDDEError

CheckDDEError   ROUTINE

 DDEErrorMsg = ''
 err# = ERRORCODE()
 ! Check for DDE error
 IF err# > 600       ! ERRORCODE is DDE related
  IF err# = 603      !! DDEExecute Failed
   DDEREAD(Server, DDE:manual, 'GetErrorNum', DDEErrorNum)
   DDEREAD(Server, DDE:manual, 'GetErrorMsg', DDEErrorMsg)
   MESSAGE('Error ' & DDEErrorNum & ' : ' & CLIP(DDEErrorMsg))
   ELSIF err# = 605  ! Timeout Error
    MESSAGE('DDE timeout')
  ELSE
   MESSAGE(err#)
  END
 END
```

Export an Application to Text (TXA) format

DDEEXECUTE(Server,'[**ExportApp**(AppName,TextFile,ScreenOutput)]')

DDEEXECUTE	Send a command string to a DDE server.
Server	A LONG containing the DDE client channel number.
ExportApp	The service of Exporting an Application to Text format
AppName	Clarion Application from which the TXA will export.
TextFile	The file to which the Application is exported.
ScreenOutput	If one (1) or TRUE, screen output is displayed, if zero (0) or FALSE, screen display is suppressed. If omitted, screen output is suppressed.

This service allows you to export an Application to text format (.TXA). A connection must be made to the server before this (or any other) command can be issued. The service will operate on the specified application regardless of any .APP file that is currently open in the Clarion development environment.

Example:

```
Server      LONG
DDEErrorMsg CSTRING(300)
DDEErrorNum USHORT

 CODE
 SYSTEM{PROP:DDETimeOut} = 12000       ! Time out after two minutes
 DDEEXECUTE(Server,'[ExportApp(c:\C60\APPS\Myapp1.APP,MyTXA.TXA,0)]')
 DO CheckDDEError
CheckDDEError   ROUTINE
 DDEErrorMsg = ''
 err# = ERRORCODE()
 ! Check for DDE error
 IF err# > 600       ! ERRORCODE is DDE related
  IF err# = 603      !! DDEExecute Failed
   DDEREAD(Server, DDE:manual, 'GetErrorNum', DDEErrorNum)
   DDEREAD(Server, DDE:manual, 'GetErrorMsg', DDEErrorMsg)
   MESSAGE('Error ' & DDEErrorNum & ' : ' & CLIP(DDEErrorMsg))
   ELSIF err# = 605  ! Timeout Error
    MESSAGE('DDE timeout')
  ELSE
   MESSAGE(err#)
  END
 END
```

Import an Application from Text (TXA) format

DDEEXECUTE(*Server*,'[**ImportApp**(*TextFile,AppName,ScreenOutput,NameClash*)]')

DDEEXECUTE	Send a command string to a DDE server.
Server	A LONG containing the DDE client channel number.
ImportApp	The service of Importing an Application from Text format
TextFile	The file from which the dictionary is imported.
AppName	Clarion Application to which the TXA will import.
ScreenOutput	If one (1) or TRUE, screen output is displayed, if zero (0) or FALSE, screen display is suppressed. If omitted, screen output is suppressed.
NameClash	Specifies how the Application Generator handles procedure name clashes. Specify “Rename” to automatically rename any procedures which clash with existing procedures. Specify “Replace” to automatically replace existing procedures with procedures of the same name being imported. If omitted, the user is prompted to specify how to handle name clashes.

This service allows you to import an Application from text format (.TXA). A connection must be made to the server before this (or any other) command can be issued. The service will operate on the specified application regardless of any .APP file that is currently open in the Clarion development environment.

Example:

```
Server      LONG
DDEErrorMsg CSTRING(300)
DDEErrorNum USHORT
  CODE
 SYSTEM{PROP:DDETimeOut} = 12000       ! Time out after two minutes
 DDEEXECUTE(Server,|
'[ImportApp(MYTXA.TXA,c:\C60\APPS\Myapp1.APP,0,Replace)]')
 DO CheckDDEError
CheckDDEError   ROUTINE
 DDEErrorMsg = ''
 err# = ERRORCODE()
 ! Check for DDE error
 IF err# > 600       ! ERRORCODE is DDE related
  IF err# = 603      !! DDEExecute Failed
   DDEREAD(Server, DDE:manual, 'GetErrorNum', DDEErrorNum)
   DDEREAD(Server, DDE:manual, 'GetErrorMsg', DDEErrorMsg)
   MESSAGE('Error ' & DDEErrorNum & ' : ' & CLIP(DDEErrorMsg))
   ELSIF err# = 605  ! Timeout Error
    MESSAGE('DDE timeout')
  ELSE
   MESSAGE(err#)
  END
 END
```

Load an Application

DDEEXECUTE(*Server*,'[**LoadApplication**(*AppName*)]')

DDEEXECUTE	Send a command string to a DDE server.
Server	A LONG containing the DDE client channel number.
LoadApplication	The service of Loading an Application.
AppName	The name of the Clarion Application to load.

This service allows you to load an Application into the Application Generator. A connection must be made to the server before this (or any other) command can be issued.

Example:

```
Server      LONG
DDEErrorMsg CSTRING(300)
DDEErrorNum USHORT

 CODE

 SYSTEM{PROP:DDETimeOut} = 12000       ! Time out after two minutes
 DDEEXECUTE(Server,'[LoadApplication(c:\C60\APPS\Myapp.app,0)]')
 DO CheckDDEError

CheckDDEError   ROUTINE

 DDEErrorMsg = ''
 err# = ERRORCODE()
 ! Check for DDE error
 IF err# > 600       ! ERRORCODE is DDE related
  IF err# = 603      !! DDEExecute Failed
   DDEREAD(Server, DDE:manual, 'GetErrorNum', DDEErrorNum)
   DDEREAD(Server, DDE:manual, 'GetErrorMsg', DDEErrorMsg)
   MESSAGE('Error ' & DDEErrorNum & ' : ' & CLIP(DDEErrorMsg))
   ELSIF err# = 605  ! Timeout Error
    MESSAGE('DDE timeout')
  ELSE
   MESSAGE(err#)
  END
 END
```

Generate an Application

DDEEXECUTE(*Server*,'[**GenerateApp**(*AppName,ScreenOutput*)]')

DDEEXECUTE	Send a command string to a DDE server.
Server	A LONG containing the DDE client channel number.
GenerateApp	The service of Generating an Application.
AppName	Clarion Application to generate.
ScreenOutput	If one (1) or TRUE, screen output is displayed, if zero (0) or FALSE, screen display is suppressed. If omitted, screen output is suppressed.

This service allows you to generate the source code for an Application. A connection must be made to the server before this (or any other) command can be issued. The service will operate on the specified application regardless of any .APP file that is currently open in the Clarion development environment.

Example:

```
Server      LONG
DDEErrorMsg CSTRING(300)
DDEErrorNum USHORT

 CODE

 SYSTEM{PROP:DDETimeOut} = 12000       ! Time out after two minutes
 DDEEXECUTE(Server,'[GenerateAPP(c:\C60\APPS\Myapp.app,0)]')
 DO CheckDDEError

CheckDDEError   ROUTINE

 DDEErrorMsg = ''
 err# = ERRORCODE()
 ! Check for DDE error
 IF err# > 600      ! ERRORCODE is DDE related
  IF err# = 603     !! DDEExecute Failed
   DDEREAD(Server, DDE:manual, 'GetErrorNum', DDEErrorNum)
   DDEREAD(Server, DDE:manual, 'GetErrorMsg', DDEErrorMsg)
   MESSAGE('Error ' & DDEErrorNum & ' : ' & CLIP(DDEErrorMsg))
   ELSIF err# = 605 ! Timeout Error
    MESSAGE('DDE timeout')
  ELSE
   MESSAGE(err#)
  END
 END
```

Execute a Project or Application

DDEEXECUTE(*Server*,'[**ExecuteProject**(*ProjectName*)]')

DDEEXECUTE	Send a command string to a DDE server.
Server	A LONG containing the DDE client channel number.
ExecuteProject	The service of compiling and linking a Project (.PRJ) or Application (.APP).
ProjectName	The Clarion Project or Application to execute.

This service allows you to compile and link a Project or Application. A connection must be made to the server before this (or any other) command can be issued. The service will operate on the specified application regardless of any .APP file that is currently open in the Clarion development environment. The screen output is displayed, but the window will close upon completion without any interaction.

Example:

```
Server     LONG
DDEErrorMsg CSTRING(300)
DDEErrorNum USHORT

 CODE

SYSTEM{PROP:DDETimeOut} = 12000       ! Time out after two minutes
 DDEEXECUTE(Server,'[ExecuteProject(c:\C60\APPS\Myapp.app)]')
 DO CheckDDEError

CheckDDEError   ROUTINE

 DDEErrorMsg = ''
 err# = ERRORCODE()
 ! Check for DDE error
 IF err# > 600      ! ERRORCODE is DDE related
  IF err# = 603     !! DDEExecute Failed
   DDEREAD(Server, DDE:manual, 'GetErrorNum', DDEErrorNum)
   DDEREAD(Server, DDE:manual, 'GetErrorMsg', DDEErrorMsg)
   MESSAGE('Error ' & DDEErrorNum & ' : ' & CLIP(DDEErrorMsg))
   ELSIF err# = 605 ! Timeout Error
    MESSAGE('DDE timeout')
  ELSE
   MESSAGE(err#)
  END
 END
```

Running a Utility Template

DDEEXECUTE(*Server*,'[**GenerateUtilityTemplate**(*UtilTemplateName,AppName,ScreenOutput*)]')

DDEEXECUTE	Send a command string to a DDE server.
Server	A LONG containing the DDE client channel number.
GenerateUtilityTemplate	The service of Generating a Utility Template on the specified Application.
UtilTemplateName	The Utility template to generate.
AppName	The Clarion Application to which the Utility Template is generated.
ScreenOutput	If one (1) or TRUE, screen output is displayed, if zero (0) or FALSE, screen display is suppressed. If omitted, screen output is suppressed.

This service allows you to run a Utility template on an Application. A connection must be made to the server before this (or any other) command can be issued. The service will operate on the specified application regardless of any .APP file that is currently open in the Clarion development environment.

Example:

```
Server      LONG
DDEErrorMsg CSTRING(300)
DDEErrorNum USHORT

 CODE
 SYSTEM{PROP:DDETimeOut} = 12000       ! Time out after two minutes
 DDEEXECUTE(Server,|
'[GenerateUtilityTemplate(MyUtil,c:\C60\APPS\Myapp.APP,0)]')
 DO CheckDDEError

CheckDDEError   ROUTINE

 DDEErrorMsg = ''
 err# = ERRORCODE()
 ! Check for DDE error
 IF err# > 600       ! ERRORCODE is DDE related
  IF err# = 603      !! DDEExecute Failed
   DDEREAD(Server, DDE:manual, 'GetErrorNum', DDEErrorNum)
   DDEREAD(Server, DDE:manual, 'GetErrorMsg', DDEErrorMsg)
   MESSAGE('Error ' & DDEErrorNum & ' : ' & CLIP(DDEErrorMsg))
   ELSIF err# = 605 ! Timeout Error
    MESSAGE('DDE timeout')
  ELSE
   MESSAGE(err#)
  END
 END
```

Registering a Template Class

DDEEXECUTE(*Server*,'[**RegisterTemplateChain**(*TemplateName,ScreenOutput*)]')

DDEEXECUTE	Send a command string to a DDE server.
Server	A LONG containing the DDE client channel number.
RegisterTemplateChain	The service of registering a specified template set.
TemplateName	The name of the template set to register.
ScreenOutput	If one (1) or TRUE, screen output is displayed, if zero (0) or FALSE, screen display is suppressed. If omitted, screen output is suppressed.

This feature is primarily intended for add-on products for Clarion. By allowing a template set to be registered by a client application, template developers can add this functionality to the installation procedure. This eliminates the need for end users to register add-on template sets.

Example:

```
Server     LONG
DDEErrorMsg CSTRING(300)
DDEErrorNum USHORT

 CODE

 SYSTEM{PROP:DDETimeOut} = 12000       ! Time out after two minutes
 DDEEXECUTE(Server,'[RegisterTemplateChain(MYTPL.TPL,0)]')
 DO CheckDDEError

CheckDDEError   ROUTINE

 DDEErrorMsg = ''
 err# = ERRORCODE()
 ! Check for DDE error
 IF err# > 600      ! ERRORCODE is DDE related
  IF err# = 603      !! DDEExecute Failed
   DDEREAD(Server, DDE:manual, 'GetErrorNum', DDEErrorNum)
   DDEREAD(Server, DDE:manual, 'GetErrorMsg', DDEErrorMsg)
   MESSAGE('Error ' & DDEErrorNum & ' : ' & CLIP(DDEErrorMsg))
   ELSIF err# = 605 ! Timeout Error
    MESSAGE('DDE timeout')
  ELSE
   MESSAGE(err#)
  END
 END
```

Unregistering a Template Class

DDEEXECUTE(*Server*,'[**UnRegisterTemplateChain**(*TemplateName,ScreenOutput*)]')

DDEEXECUTE	Send a command string to a DDE server.
Server	A LONG containing the DDE client channel number.
UnRegisterTemplateChain	The service of unregistering a specified template set.
TemplateName	The name of the template set to unregister.
ScreenOutput	If one (1) or TRUE, screen output is displayed, if zero (0) or FALSE, screen display is suppressed. If omitted, screen output is suppressed.

This feature is primarily intended for add-on products for Clarion. By allowing a template set to be unregistered by a client application, template developers can add this functionality to the installation procedure.

Example:

```
Server     LONG
DDEErrorMsg CSTRING(300)
DDEErrorNum USHORT

 CODE

 SYSTEM{PROP:DDETimeOut} = 12000       ! Time out after two minutes
 DDEEXECUTE(Server,'[UnRegisterTemplateChain(MYTPL.TPL,0)]')
 DO CheckDDEError

CheckDDEError   ROUTINE

 DDEErrorMsg = ''
 err# = ERRORCODE()
 ! Check for DDE error
 IF err# > 600      ! ERRORCODE is DDE related
  IF err# = 603      !! DDEExecute Failed
   DDEREAD(Server, DDE:manual, 'GetErrorNum', DDEErrorNum)
   DDEREAD(Server, DDE:manual, 'GetErrorMsg', DDEErrorMsg)
   MESSAGE('Error ' & DDEErrorNum & ' : ' & CLIP(DDEErrorMsg))
   ELSIF err# = 605 ! Timeout Error
    MESSAGE('DDE timeout')
  ELSE
   MESSAGE(err#)
  END
 END
```

Getting DDE Error Messages

DDEREAD(*Server*,DDE:Manual,'**GetErrorMsg**',*ErrorMessageText*)

DDEREAD(*Server*,DDE:Manual,'**GetErrorNum**',*DDEErrorNum*)

DDEREAD	Get data from the DDE server.
Server	A LONG containing the DDE client channel number.
DDE:Manual	An EQUATE defining the type of data link as manual (defined in EQUATES.CLW).
'GetErrorMsg'	Request for the error message data item from the server. This must be provided exactly as shown.
'GetErrorNum'	Request for the error number data item from the server. This must be provided exactly as shown.
ErrorMessageText	A STRING(255) variable to which the error message text is assigned.
DDEErrorNum	A USHORT variable to which the error number is assigned.

This feature is provided to allow error checking in your DDE client application. You can check the Clarion ERRORCODE to determine if the error posted is DDE-related, then query the DDE server and evaluate the DDE error returned.

Example:

```
Server      LONG
DDEErrorMsg CSTRING(300)
DDEErrorNum USHORT

 CODE
 DDEEXECUTE(Server,'[GenerateAPP(c:\C60\APPS\Myapp.app,0)]')
 DO CheckDDEError

CheckDDEError   ROUTINE

 DDEErrorMsg = ''
 err# = ERRORCODE()
 ! Check for DDE error
 IF err# > 600      ! ERRORCODE is DDE related
  IF err# = 603     !! DDEExecute Failed
   DDEREAD(Server, DDE:manual, 'GetErrorNum', DDEErrorNum)
   DDEREAD(Server, DDE:manual, 'GetErrorMsg', DDEErrorMsg)
   MESSAGE('Error ' & DDEErrorNum & ' : ' & CLIP(DDEErrorMsg))
   ELSIF err# = 605 ! Timeout Error
    MESSAGE('DDE timeout')
  ELSE
   MESSAGE(err#)
  END
 END
```

Clarion DDE Errors

601 Invalid DDE Channel

602 DDE Channel Not Open

603 DDEEXECUTE Failed

604 DDEPOKE Failed

605 Time Out

DDE Service Errors and associated messages

1 "Screen output must be either 'TRUE' or 'FALSE'"
2 "Cannot open specified dictionary"
3 "Cannot create specified text file"
4 "Cannot open specified text file"
5 "Cannot create specified dictionary"
6 "Cannot open specified application"
7 "Cannot create specified application"
8 "Cannot open specified template file"
9 "Cannot open specified project file"
10 "Wrong number of parameters"
11 "Requested DDE service is not supported"

The following errors return a dynamic message generated by the requested service:

12 Export dictionary failed
13 Import dictionary failed
14 Export application failed
15 Import application failed
16 Register templated chained failed
17 Generate utility template failed
18 Generate application failed
19 Error in Make

.TXD File Format

.TXD Files: Clarion Dictionaries in ASCII Format

.TXD files are simply ASCII text versions of Clarion .DCT files. .DCT files are where the Clarion environment stores all the reusable information about a database. A .TXD file contains all the information stored in an .DCT file (except for prior versions), but in a text format that is readable (and writable) by most text editors.

You can easily create a .TXD file for any .DCT file by loading the dictionary into the Clarion development environment, then choosing **File > Export Text** from the menu.

Why would you need a .TXD file? For at least three reasons:

Backup	Because a .TXD file can be imported just as easily as it is exported, it may serve as a backup of the current version of your .DCT file. Previous versions are not available in the .TXD.
Mass Changes	Because a .TXD file may be manipulated in your favorite text editor, you can use the power of the text editor to make mass changes to your dictionary, or for that matter, any changes that would be easier with a text editor.
First Aid	Occasionally, a .DCT file may exhibit some strange behavior in the development environment. Exporting to a .TXD file, then importing from that same file can, highlight problems which will enable you to correct any problems in the dictionary.

.TXD File Organization

There are four major sections in the .TXD file: [DICTIONARY], [FILES], [ALIASES], and [RELATIONS]. These sections correspond to the main Clarion language database structures, and they contain subsections and keywords that fully describe these structures, for example, the [FILES] section contains, among other things, the field and key definitions for the file.

.TXD Skeleton

The following is an ordered list of .TXD sections, subsections, and keywords. This list is designed to give you a feel for the overall structure and organization of the .TXD file.

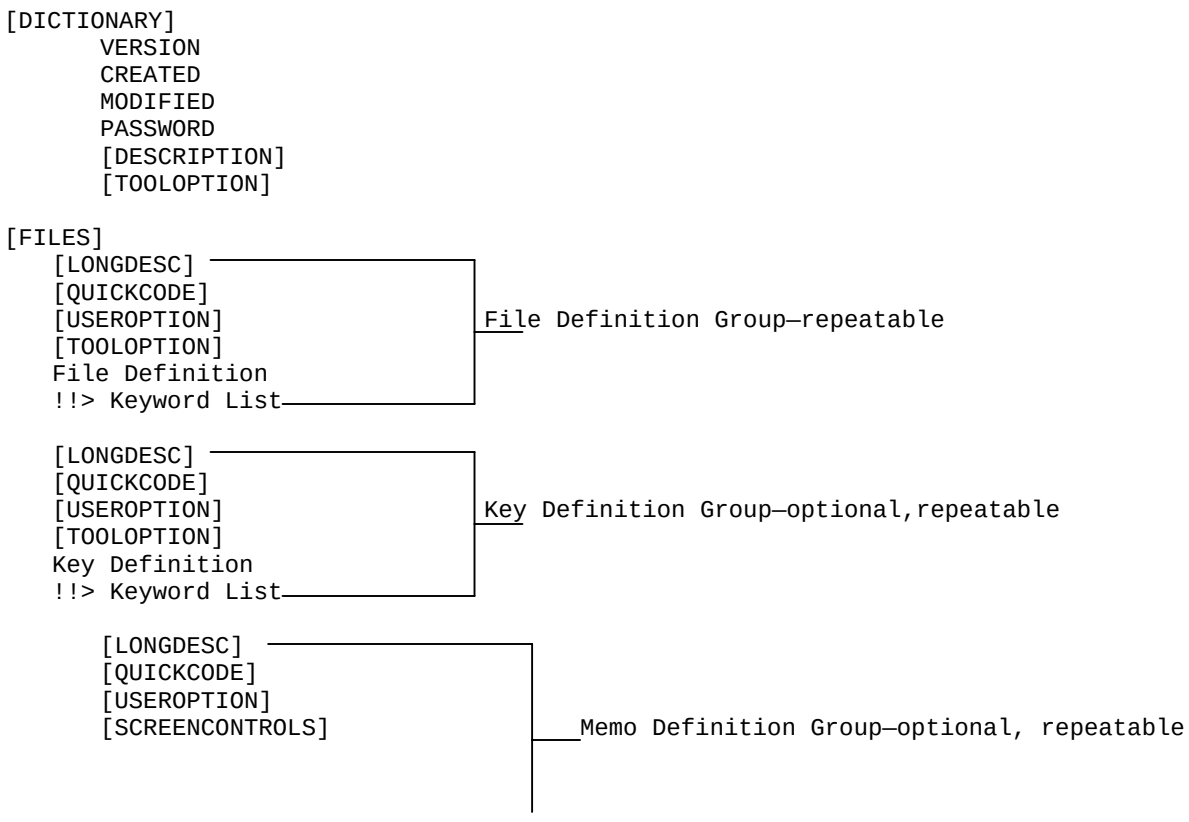
Each section begins with its title enclosed in square brackets and ends with the beginning of another section. Each section may contain subsections and keywords.

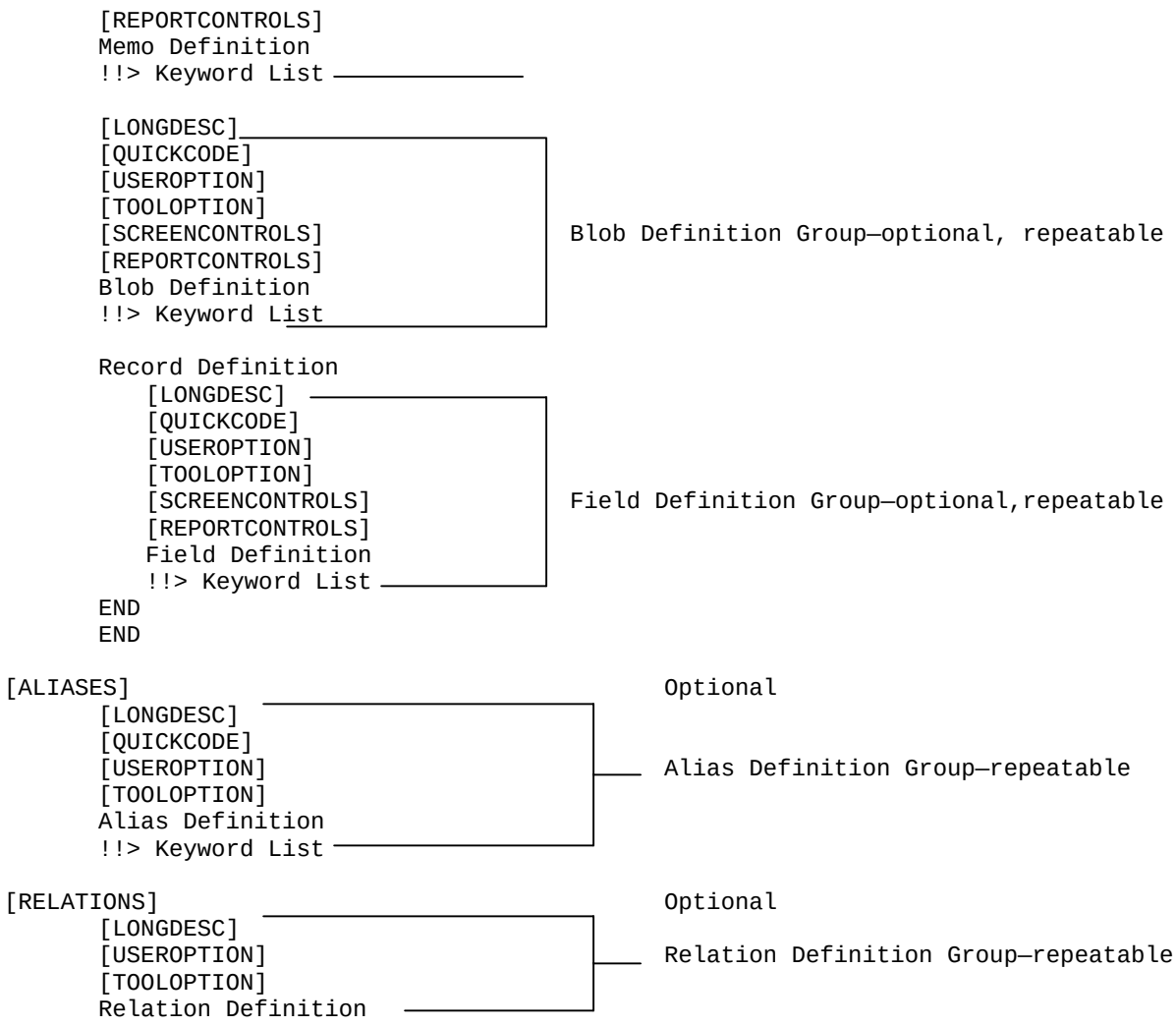
Subsections begin just like the major sections—with its title surrounded by square brackets.

Keywords appear as a keyword name in all capitals followed by the keyword value. The keyword values appear in various formats described below on a case by case basis.

The spacing and indentations in the list are for readability and do not appear in the actual .TXD files.

Following the skeleton is a detailed discussion of each .TXD section, subsection, and keyword.




[REPORTCONTROLS]
Memo Definition
!!> Keyword List
[LONGDESC]
[QUICKCODE]
[USEROPTION]
[TOOLOPTION]
[SCREENCONTROLS]
[REPORTCONTROLS]
Blob Definition
!!> Keyword List
Blob Definition Group—optional, repeatable
Record Definition
[LONGDESC]
[QUICKCODE]
[USEROPTION]
[TOOLOPTION]
[SCREENCONTROLS]
[REPORTCONTROLS]
Field Definition
!!> Keyword List
Field Definition Group—optional,repeatable
END
END
[ALIASES]
Optional
[LONGDESC]
[QUICKCODE]
[USEROPTION]
[TOOLOPTION]
Alias Definition
!!> Keyword List
Alias Definition Group—repeatable
[RELATIONS]
Optional
[LONGDESC]
[USEROPTION]
[TOOLOPTION]
Relation Definition
Relation Definition Group—repeatable

.TXD File Sections

[DICTIONARY]

The dictionary section is required, although all of its individual keywords and subsections are optional. The dictionary section contains the following keywords and subsections:

```
[DICTIONARY]
      VERSION
      CREATED
      MODIFIED
      PASSWORD
      [DESCRIPTION]
[TOOLOPTION]
```

VERSION — Lists the version number of the dictionary (optional). Version is ignored on import of .TXD. For example:

```
VERSION '1.0'
```

CREATED — Lists the date and time the dictionary was originally created (optional). For example:

```
CREATED '19 AUG 95' '11:02:33AM'
```

MODIFIED — Lists the date and time the dictionary was last saved (optional). For example:

```
MODIFIED ' 7 NOV 95' ' 9:44:18AM'
```

PASSWORD — Lists the password needed to access the dictionary (optional). Use the Password button in the Dictionary Properties dialog to establish the password for your dictionary.

The password is not case sensitive. The password is encrypted in the .DCT file, but is not encrypted when exported to the .TXD file. For example:

```
PASSWORD cablecar
```

[DESCRIPTION] — Lists up to 1000 characters of descriptive text on multiple lines (optional). Each line of text begins with an exclamation point (!) and contains up to 75 characters. The text comes from the Comments tab in the Dictionary Properties dialog.

A [DESCRIPTION] will be split into lines of 75 characters each in the .TXD. If the text is separated by a carriage return (<CR>) then it will write out an extra empty line. This is true for any description ([DESCRIPTION] and [LONGDESC]) within the .TXD and for variables in the .TXA. For example:

Original text in the dictionary:

This is a description : <CR>
and it continues on the next line whereby the text from now on is written without carriage returns.

Text in the .TXD:

```
!This is a description :
!
!and it continues on the next line whereby the text from now on is !written
without carriage returns.
```

[TOOLOPTION] See Common Subsections below (optional).

Example—[DICTIONARY]

```
[DICTIONARY]
VERSION ‘1.0’
CREATED ’19 AUG 95' ’11:02:33AM’
MODIFIED ‘ 7 NOV 95’ ‘ 9:44:18AM’
PASSWORD cablecar
[DESCRIPTION]
!Up to 1000 characters of text describing this dictionary.
!Each line of up to 75 characters begins with an exclamation point.
```

[FILES]

The files section appears only once in the .TXD file. All files in the dictionary are defined in this section. Fields and keys are also defined in this section as an integral part of the definition of each file.

File Definition Group

The file definition group is a series of .TXD subsections and keywords that fully describes a single FILE within the data dictionary. The group is repeated for every FILE in the dictionary, so that all are fully documented. The file definition group contains the following keywords and subsections:

```
[FILES]
       [TRIGGERDATA]
       [TRIGGERS]
              [TRIGGER]
              [SOURCE]
       [LONGDESC]
       [QUICKCODE]
       [USEROPTION]
       [TOOLOPTION]
File Definition Group—repeatable
       File Definition
       !!> Keyword List
```

[TRIGGERS] Begins the triggers group for the associated file.

[TRIGGER]

Identifies the location of the trigger source. Valid values are:

BEFORE_INSERT
BEFORE_UPDATE
BEFORE_DELETE
AFTER_INSERT
AFTER_UPDATE
AFTER_DELETE

[SOURCE]

Identifies the source code that will be inserted in the {TRIGGER} location directly preceding it.

[LONGDESC] See *Common Subsections* (optional).

[QUICKCODE] Information used by the Clarion Wizards to configure your Wizard generated applications and procedures (optional). Use the Options tab of the File Properties dialog to input the [QUICKCODE] information.

The only Wizard/QUICKCODE option available for files is NOPOPULATE. NOPOPULATE means the Wizards will not generate a form, a browse, or a report procedure for this file.

[USEROPTION] See *Common Subsections* (optional).

[TOOLOPTION] See *Common Subsections* (optional).

File Definition Begins with the first line of the Clarion language FILE declaration statements for this file (required). Optionally includes a comment up to 40 characters long. The file definition also includes the keyword list, plus the key, memo, blob, record, and field definition groups. The entire structure ends with END. For example:

```
Customers FILE,DRIVER('TOPSPEED'),PRE(CUS),CREATE
 Key Definition Group
 Memo Definition Group
 Blob Definition Group
 Record Definition Group
 Field Definition Group
          END
```

Keyword List A list of internal keywords that describe options not specified on the FILE statement (optional). The list begins with !!>.

IDENT() The internal reference number the development environment uses to identify the FILE (optional).

```
!!> IDENT(1)
```

USAGE() Contains either FILE, GLOBAL, or POOL to identify whether the FILE is global data or a field pool (optional).

```
!!> IDENT(1),USAGE(POOL)
```

LAST Specifies USAGE(GLOBAL) data which should generate last (optional).

```
!!> IDENT(1),USAGE(GLOBAL),LAST
```

Example—File Definition Group

```
[FILES]

[TRIGGERS]
[TRIGGER]
!!> BEFORE_UPDATE

[SOURCE]
! ! Add date and time changed to Pr_info
!  PBI:Pr_info = CLIP(PBI:Pr_info) &'<13,10>Date changed
'&FORMAT(TODAY(),@d1)&'.<13,10>'

[LONGDESC]
!This is the main customer file. Contains names addresses and
!phone numbers. One record per customer. Each customer has a unique
!key that is auto numbered...
[QUICKCODE]
!NOPOPULATE
[USEROPTION]
!ThirdPartyTemplateSwitch(on)
!ThirdPartyPreProcessLevel(release)
Customers FILE,DRIVER('TOPSPEED'),PRE(CUS),CREATE,THREAD
!!> IDENT(1)
     .
     .
     .
     END
```

Key Definition Group

The key definition group is optional. It is a series of .TXD subsections and keywords that fully describe a single KEY or INDEX for a file. The group is repeated for every KEY or INDEX to the file, so that all are fully documented. The key definition group contains the following keywords and subsections:

[LONGDESC]
[QUICKCODE]
[USEROPTION]
[TOOLOPTION]
Key Definition Group—optional, repeatable
Key Definition
!!> Keyword List

[LONGDESC]	See Common Subsections below (optional).
[QUICKCODE]	Information used by the Clarion Wizards to configure your Wizard generated applications and procedures (optional). Use the Options tab of the Key Properties dialog to input the [QUICKCODE] information.

	[QUICKCODE] information may be specified on multiple lines, each line beginning with an exclamation point. However, multiple lines are concatenated to one string by the development environment. Therefore keywords should be separated by a comma, even when on different lines in the .TXD. The Wizard/QUICKCODE options available for keys are NOPOPULATE and ORDER:
NOPOPULATE	The Wizards do not generate reports or browses sorted on this key (optional).
ORDER()	The order in which browses and reports sorted on this key appear within Wizard generated menus and browse boxes (optional). The order may be specified as Normal, First, or Last. Normal displays items in the order in which they are found in the dictionary. First forces the item to appear before all Normal and Last items. Conversely, Last forces the item to appear after all First and Normal items. For example:

```
[QUICKCODE]
!ORDER(First)
```

[USEROPTION]	See *Common Subsections* (optional).
[TOOLOPTION]	See *Common Subsections* (optional).
KEY Definition	Lists the Clarion language KEY or INDEX declaration (required). Optionally includes a comment up to 40 characters long. For example: KeyCust KEY(CUS:CustNumber),NOCASE !Auto-number
Keyword List	A list of internal keywords that describe options not specified on the KEY statement (optional). The list begins with !!>. The two valid keywords for keys are IDENT and AUTO.
IDENT()	The internal reference number the development environment uses to identify the KEY (optional).
AUTO	The key is auto-numbered (optional). This means the application generator will supply code to automatically increment the value of the key. Use the Attributes tab of the Key Properties dialog to set the auto-number keyword. For example: !!> IDENT(1),AUTO

Example—Key Definition Group

```
KeyCustNumber KEY(CUS:CustNumber),NOCASE,OPT !Auto-numbered Cust Key
!!> IDENT(1),AUTO
[QUICKCODE]
```

```
!ORDER(First)
```

Memo Definition Group

The memo definition group is optional. It is a series of .TXD subsections and keywords that fully describes a single MEMO field in a file. The group is repeated for each MEMO field in the file, so that all are fully documented. The memo definition group contains the following keywords and subsections:

```
[LONGDESC]
[QUICKCODE]
[USEROPTION]
[TOOLOPTION]
[SCREENCONTROLS]     Memo Definition Group—optional, repeatable
[REPORTCONTROLS]
MEMO Definition
!!> Keyword List
```

[LONGDESC]	See *Common Subsections* (optional).
[QUICKCODE]	Information used by the Clarion Wizards to configure your Wizard generated applications and procedures (optional). Use the Options tab of the Field Properties dialog to input the [QUICKCODE] information. [QUICKCODE] information may be specified on multiple lines, each line beginning with an exclamation point. However, multiple lines are concatenated to one string by the development environment. Therefore keywords should be separated by a comma, even when on different lines in the .TXD. Wizard/QUICKCODE options available for MEMOs are NOPOPULATE, ORDER(), VERTICALSPACE, and TAB():
NOPOPULATE	This field is not included on Wizard generated reports, forms, and browses (optional).
ORDER()	Not meaningful for MEMOs. MEMOs always appear alone on a separate tab, in the order in which they appear in the dictionary.
VERTICALSPACE	Not meaningful for MEMOs. MEMOs always appear alone on a separate tab.
TAB()	Not meaningful for MEMOs. MEMOs always appear alone on a separate tab.

```
[QUICKCODE]
!NOPOPULATE
```

[USEROPTION]	See *Common Subsections* (optional).
[TOOLOPTION]	See *Common Subsections* (optional).

[SCREENCONTROLS]	MEMOs always appear in TEXT controls. See Common Subsections below (optional).
[REPORTCONTROLS]	MEMOs always appear in TEXT controls. See Common Subsections below (optional).
MEMO Definition	The Clarion language MEMO declaration (required). Optionally includes a comment up to 40 characters long. For example:

```
CustomerMemo  MEMO(500) !500 character text field
```

Keyword List—MEMOs

A list of internal keywords that describe options not specified by the Clarion language MEMO statement. The list begins with !!> and is optional. Within the .TXD, the keyword list appears on a single line, with keywords separated by commas.

The keywords in this list are set using the Field Properties dialog. Many of the keywords correspond directly to Clarion language keywords. See the Language Reference for more information on these keywords.

Following is a complete list of keywords available for memos:

```
IDENT()
VALID()
INITIAL()
PROMPT()
HEADER()
HELP()
MESSAGE()
TOOLTIP()
PICTURE()
CASE()
TYPEMODE()
PASSWORD
READONLY
```

Notice that the keywords in the list correspond closely to the tabs and prompts in the Field Properties dialog. This is because the Field Properties dialog is where these values are set (see the User's Guide, the Language Reference, and the on-line help for more information on these fields).

IDENT() — The internal ID number that the development environment uses to reference this item (optional). For example:

```
IDENT(3)
```

VALID() — Specifies the validity checking code that is generated for this field (optional). The VALID keyword has a significant parameter list which is diagrammed as follows, where the vertical list represents alternative parameters and the curly braces represent optional parameters:

```
VALID( NONZERO       )
       INRANGE({minimum}{,maximum})
       BOOLEAN
       INFILE(filename,parentfile:childfile)
       INLIST('item1|item2|...|itemn')
```

For example, a required field has:

```
VALID(NONZERO)
```

A non-negative numeric field has:

```
VALID(INRANGE(0))
```

A binary logical field (yes/no, true/false) has:

```
VALID(BOOLEAN)
```

A field validated against another file has:

```
VALID(INFILE(States,Customers:States))
```

A field validated against a fixed, finite list has:

```
VALID(INLIST('Left|Center|Right'))
```

	The **Validity Checks** tab in the **Field Properties** dialog specifies these values. See *Using the Dictionary Editor* in the *User's Guide* for more information on validity checks.
INITIAL()	The initial value of the field (optional). The **Attributes** tab in the **Field Properties** dialog sets this value. For example: `INITIAL(0)`
PROMPT()	The text string used as the default prompt for this field on a screen (optional). The **General** tab in the **Field Properties** dialog sets this value. For example: `PROMPT('&Product Number')`
HEADER()	The text string used as the default column title for reports and list boxes (optional). This value is set on the **General** tab of the **Field Properties** dialog. For example: `HEADER('PRODUCT NUMBER')`
HELP()	The help topic for this field. This value is set on the **Help** tab of the **Field Properties** dialog. For example: `HELP('PRODNUMBER')`
MESSAGE()	Up to 75 characters of default message text for this field (optional). This value is set on the **Help** tab of the **Field Properties** dialog. For example: `MESSAGE('Enter the 6 digit product number')`
TOOLTIP()	Up to 75 characters of default tool tip (balloon help) text for this field (optional). This value is set on the **Help** tab of the **Field Properties** dialog. For example: `TOOLTIP('Enter the 6 digit product number')`
PICTURE()	The default screen picture token for this field (optional). See the Language Reference for a complete list of picture tokens and their uses. This value is set on the **General** tab of the **Field Properties** dialog. For example: `PICTURE('@n6')`
CASE()	The default case translation for this field (optional). The choices are UPPER and CAPS. UPPER converts all text to uppercase. CAPS converts all text to mixed case word capitalization. This value is set on the **Attributes** tab of the **Field Properties** dialog. For example: `CASE('UPPER')`

TYPEMODE()	The default typing mode for this field (optional). The choices are INS and OVR. INS preserves existing text by inserting new characters. OVR discards existing text by overwriting it with new text. This value is set on the **Attributes** tab of the **Field Properties** dialog. For example: `TYPEMODE('INS')`
PASSWORD	Text typed in this field is displayed as asterisks (optional). This value is set on the Attributes tab of the Field Properties dialog. For example: `PASSWORD`
READONLY	The field is read only and therefore will not accept input (optional). This value is set on the Attributes tab of the Field Properties dialog. For example: `READONLY`

For example:

```
!!> IDENT(47),PROMPT('Customer Memo:'),PICTURE(@s255)
```

Example—Memo Definition Group

```
[LONGDESC]
!Long description of Memo field
[QUICKCODE]
!TAB('TabName'),ORDER(Last),VERTICALSPACE
[SCREENCONTROLS]
! PROMPT('Customer Memo:'),USE(?CUS:CustomerMemo:Prompt)
! TEXT,USE(CUS:CustomerMemo)
[REPORTCONTROLS]
! TEXT,USE(CUS:CustomerMemo)
CustomerMemo               MEMO(500)
!!> IDENT(47),PROMPT('Customer Memo:'),PICTURE(@s255)
```

Blob Definition Group

The blob definition group is optional. It is a series of .TXD subsections and keywords that fully describes a single BLOB field in a file. The group is repeated for each BLOB field in the file, so that all are fully documented. The blob definition group contains the following keywords and subsections:

```
[LONGDESC]
[QUICKCODE]
[USEROPTION]
[TOOLOPTION]
[SCREENCONTROLS]     Blob Definition Group—optional, repeatable
[REPORTCONTROLS]
BLOB Definition
!!> Keyword List
```

[LONGDESC]	See *Common Subsections* (optional).
[QUICKCODE]	Not meaningful for BLOBs.
[USEROPTION]	See *Common Subsections* (optional).
[TOOLOPTION]	See *Common Subsections* (optional).
[SCREENCONTROLS]	See *Common Subsections* (optional).
[REPORTCONTROLS]	See *Common Subsections* (optional).
BLOB Definition	Lists the Clarion language BLOB declaration (required). Optionally includes a comment up to 40 characters long. For example: `CustomerPhoto BLOB,BINARY !Customer Image/Logo`
Keyword List	A list of internal keywords that describe options not specified on the BLOB statement (optional). The list begins with !!>. The only valid keyword for a BLOBs is IDENT. IDENT supplies the internal reference number by which the development environment identifies the FILE. For example: `!!> IDENT(48)`

Example—Blob Definition Group

```
[LONGDESC]
!This digitized image may contain customer logo or photograph
[QUICKCODE]
!TAB('Personal'),ORDER(First)
[SCREENCONTROLS]
! IMAGE,USE(?CUS:CustomerPhoto)
[REPORTCONTROLS]
! IMAGE,USE(?CUS:CustomerPhoto)
CustomerPhoto              BLOB,BINARY !Customer Image/Logo
!!> IDENT(48)
```

Record Definition

The record definition is required. It begins with the Clarion language RECORD declaration statements for this file. The RECORD definition then includes all the field definition groups for the file. The entire structure ends with END. For example:

```
CustomerRecord  RECORD
```

field definition groups

```
                END
```

Field Definition Group

The field definition group is optional. It is a series of .TXD subsections and keywords that fully describes a single field in a file. The group is repeated for each field in the file, so that all are fully documented. The field definition group contains the following keywords and subsections:

```
[LONGDESC]
[QUICKCODE]
[USEROPTION]
[TOOLOPTION]
[SCREENCONTROLS]     Field Definition Group—optional, repeatable
[REPORTCONTROLS]
Field Definition
!!> Keyword List
```

[LONGDESC]	See *Common Subsections* (optional).
[QUICKCODE]	Information used by the Clarion Wizards to configure your Wizard generated applications and procedures (optional). Use the Options tab of the Field Properties dialog to input the [QUICKCODE] information.

	[QUICKCODE] information may be specified on multiple lines, each line beginning with an exclamation point. However, multiple lines are concatenated to one string by the development environment. Therefore keywords should be separated by a comma, even when on different lines in the .TXD. The Wizard/QUICKCODE options available for fields are NOPOPULATE, ORDER, VERTICALSPACE, and TAB:
NOPOPULATE	The Wizards do not include this field on browses, forms, and reports (optional).
ORDER()	The order in which this field is populated on Wizard generated browses, forms, and reports (optional). The order may be specified as **Normal**, **First**, or **Last**. Normal displays items in the order in which they are found in the dictionary. First forces the item to appear before all Normal and Last items. Conversely, Last forces the item to appear after all First and Normal items.
VERTICALSPACE	Extra space is inserted between this field and the preceding field on Wizard generated forms.

TAB()	The name of the TAB on which this field appears on Wizard generated forms.
	
	For example:
	
	`[QUICKCODE]`
	`!ORDER(Last),VERTICALSPACE,TAB('Personal')`

[USEROPTION]	See *Common Subsections* (optional).
[TOOLOPTION]	See *Common Subsections* (optional).
[SCREENCONTROLS]	See *Common Subsections* (optional).
[REPORTCONTROLS]	See *Common Subsections* (optional).
Field Definition	The Clarion language field declaration statement (required). Optionally includes a comment up to 40 long. For example: `CustNumber DECIMAL(7,2) !unique key`

Keyword List—Fields

A list of internal keywords that describe options not specified by the Clarion language field declaration statement. The list begins with !!> and is optional. Within the .TXD, the keyword list appears on a single line, with keywords separated by commas.

The keywords in this list are set using the **Field Properties** dialog. Many of the keywords correspond directly to Clarion language keywords. See the *Language Reference* for more information on these keywords.

Following is a complete list of keywords available for data dictionary fields.:

```
IDENT()
VALID()
VALUES()
INITIAL()
PROMPT()
HEADER()
HELP()
MESSAGE()
TOOLTIP()
PICTURE()
CASE()
TYPEMODE()
PASSWORD
READONLY
```

Notice that the keywords in the list correspond closely to the tabs and prompts in the **Field Properties** dialog. This is because the **Field Properties** dialog is where these values are set (see the *User's Guide*, the *Language Reference*, and the on-line help for more information these fields).

IDENT() — The internal ID number that the development environment uses to reference this item (optional). For example:

IDENT(3)

VALID() — Specifies the validity checking code that is generated for this field (optional). The VALID keyword has a significant parameter list which is diagrammed as follows, where the vertical list represents alternative parameters and the curly braces represent optional parameters:

```
VALID( NONZERO       )
       INRANGE({minimum}{,maximum])
       BOOLEAN
       INFILE(filename,parentfile:childfile)
              INLIST('item1|item2|...|itemn')
              NOCHECKS('item1|item2|...|itemn')
```

	For example, a required field has:
	`VALID(NONZERO)`
	A non-negative numeric field has:
	`VALID(INRANGE(0))`
	A binary logical field (yes/no, true/false) has:
	`VALID(BOOLEAN)`
	A field validated against another file has:
	`VALID(INFILE(States,Customers:States))`
	A field validated against a fixed, finite list has:
	`VALID(INLIST('Left|Center|Right'))`
	The **Validity Checks** tab in the **Field Properties** dialog specifies these values. See *Using the Dictionary Editor* in the *User's Guide* for more information on validity checks.

VALUES()	Specifies the displayed list of selections for validity checking code that is generated for this field (optional). This is related to the VALID keyword and is active for NOCHECKS, INLIST, and BOOLEAN. The **Validity Checks** tab in the **Field Properties** dialog specifies these values. See Using the Dictionary Editor in the User's Guide for more information on validity checks.
INITIAL()	The initial value of the field (optional). The **Attributes** tab in the **Field Properties** dialog sets this value. For example: `INITIAL(0)`
PROMPT()	The text string used as the default prompt for this field on a screen (optional). The **General** tab in the **Field Properties** dialog sets this value. For example: `PROMPT('&Product Number')`
HEADER()	The text string used as the default column title for reports and list boxes (optional). This value is set on the **General** tab of the **Field Properties** dialog. For example: `HEADER('PRODUCT NUMBER')`
HELP()	The help topic for this field. This value is set on the **Help** tab of the **Field Properties** dialog. For example: `HELP('PRODNUMBER')`
MESSAGE()	Up to 75 characters of default message text for this field (optional). This value is set on the **Help** tab of the **Field Properties** dialog. For example: `MESSAGE('Enter the 6 digit product number')`

TOOLTIP()	Up to 75 characters of default tool tip (balloon help) text for this field (optional). This value is set on the **Help** tab of the **Field Properties** dialog. For example: `TOOLTIP('Enter the 6 digit product number')`
PICTURE()	The default screen picture token for this field (optional). See the *Language Reference* for a complete list of picture tokens and their uses. This value is set on the **General** tab of the **Field Properties** dialog. For example: `PICTURE('@n6')`
CASE()	The default case translation for this field (optional). The choices are UPPER and CAPS. UPPER converts all text to uppercase. CAPS converts all text to mixed case word capitalization. This value is set on the **Attributes** tab of the **Field Properties** dialog. For example: `CASE('UPPER')`
TYPEMODE()	The default typing mode for this field (optional). The choices are INS and OVR. INS preserves existing text by inserting new characters. OVR discards existing text by overwriting it with new text. This value is set on the **Attributes** tab of the **Field Properties** dialog. For example: `TYPEMODE('INS')`
PASSWORD	Text typed in this field is displayed as asterisks (optional). This value is set on the **Attributes** tab of the **Field Properties** dialog. For example: `PASSWORD`
READONLY	The field is read only and therefore will not accept input (optional). This value is set on the **Attributes** tab of the **Field Properties** dialog. For example: `READONLY`

For example:

```
!!> IDENT(5),PROMPT('&Cust Number:'),PICTURE(@n4)
```

Example—Field Definition Group

```
[QUICKCODE]
!ORDER(First)
[SCREENCONTROLS]
! PROMPT('&Cust Number:'),USE(?CUS:CustNumber:Prompt)
! ENTRY(@n4),USE(CUS:CustNumber)
```

```
[REPORTCONTROLS]
! STRING(@n4),USE(CUS:CustNumber)
CustNumber                  DECIMAL(7,2)
!!> IDENT(5),PROMPT('&Cust Number:'),HEADER('Cust Number'),PICTURE(@n4)
```

[ALIASES]

The aliases section appears once in the .TXD file. This section is optional, and all aliases in the dictionary are defined here.

Alias Definition Group

The alias definition group is a series of .TXD subsections and keywords that fully describes a single alias within the data dictionary. The group is repeated for every alias in the dictionary. The alias definition group contains the following keywords and subsections:

```
[ALIASES]
       [LONGDESC]
       [QUICKCODE]
       [USEROPTION]
       [TOOLOPTION]
Alias Definition Group—repeatable
       Alias Definition
       !!> Keyword List
```

[LONGDESC]	See *Common Subsections* (optional).
[QUICKCODE]	Information used by the Clarion Wizards to configure your Wizard generated applications and procedures (optional). Use the Options tab of the Alias Properties dialog to input the [QUICKCODE] information. The only Wizard/QUICKCODE option available for aliases is NOPOPULATE. NOPOPULATE means the Wizards will not generate a form, a browse, or a report procedure for this alias.
[USEROPTION]	See *Common Subsections* (optional).
[TOOLOPTION]	See *Common Subsections* (optional).
Alias Definition	Specifies the file the alias is redefining, and the new prefix for the alias (required). Optionally includes a comment up to 40 characters long. For example: `Sales  ALIAS(Orders),PRE(SAL) !Alias for Order`
Keyword List	A list of internal keywords that describe options not specified in the alias definition (optional). The list begins with !!>.

The only valid keyword for an alias is IDENT. IDENT supplies the internal reference number by which the development environment identifies the alias. For example:

```
!!> IDENT(6)
```

Example—Alias Definition Group

```
[ALIASES]

[QUICKCODE]
!NOPOPULATE
Sales                ALIAS(Orders),PRE(SAL)  !Alias for Order
!!> IDENT(6)
```

[RELATIONS]

The relations section appears once in the .TXD file. This section is optional, and all file relationships in the dictionary are defined here. Referential Integrity constraints are also defined here as an integral part of each relationship.

Relation Definition Group

The relation definition group is a series of .TXD subsections, keywords, and a special relation definition that fully describes a relationship between two files in the data dictionary. The group is repeated for each relationship in the dictionary. The relation definition group contains the following keywords and subsections:

```
[RELATIONS]
 [LONGDESC]
 [USEROPTION]
 [TOOLOPTION]
 Relation Definition Group—repeatable   Relation Definition
```

[LONGDESC]	See *Common Subsections* (optional).
[USEROPTION]	See *Common Subsections* (optional).
[TOOLOPTION]	See *Common Subsections* (optional).
Relation Definition	A relationship defined in the **Edit Relationship Properties** dialog (required). The .TXD relation definition is diagrammed as follows, where the vertical lists represent alternative parameters and the curly braces enclose optional parameters:

```
RELATION, ONE:MANY, UPDATE( RESTRICT ),DELETE( RESTRICT )
          MANY:ONE          CASCADE            CASCADE
                            CLEAR              CLEAR
filename FILE( {key} )
```

```
filename RELATED_FILE( {key} )
{FILE_TO_RELATED_KEY
 FIELD( filefieldname,relatedfilefieldname )    repeatable
     NOLINK,    NOLINK
 END}
{RELATED_FILE_TO_KEY
 FIELD( relatedfilefieldname,filefieldname )    repeatable
        NOLINK,             NOLINK
 END}
END
```

RELATION	The beginning of the relationship's definition.
ONE:MANY | MANY:ONE	The type of relationship (required).
UPDATE	The action taken to maintain database referential integrity when the parent file's linking key field values are changed (optional). The action is either RESTRICT, CASCADE, or CLEAR.
RESTRICT	The update is not allowed if there are related child records.
CASCADE	The update is cascaded to the linking key fields of all related child records.
CLEAR	The update is cascaded in the form of blanks, zeros, or nulls to the linking key fields of all related child records.
RESTRICT_SERVER	The back-end server will prevent the user from deleting or changing an entry if the value is used in a foreign key.
CASCADE_SERVER	The back-end server will update or delete the foreign key record.
CLEAR_SERVER	The back-end server will set the foreign key to null or zero.
DELETE	The action taken to maintain database referential integrity when the parent file's record is deleted (optional). The action is either RESTRICT, CASCADE, or CLEAR.
RESTRICT	The delete is not allowed if there are related child records.
CASCADE	The delete is allowed, and the related child records are deleted too.
CLEAR	The delete is cascaded in the form of blanks, zeros, or nulls to the linking key fields of all related child records.
RESTRICT_SERVER	

The back-end server will prevent the user from deleting or changing an entry if the value is used in a foreign key.

CASCADE_SERVER

The back-end server will update or delete the foreign key record.

CLEAR_SERVER

The back-end server will set the foreign key to null or zero.

filename FILE({key})

The key in the first file in the relationship that contains the field(s) common to both files (optional). This file is either the parent in a 1:MANY relationship, or the child in a MANY:1 relationship. key is optional for a “one-way” or “lookup” relationship.

filename RELATED_FILE({key})

The key in the second file in the relationship that contains the field(s) common to both files (optional). This file is either the child in a 1:MANY relationship, or the parent in a MANY:1 relationship. key is optional for a “one-way” or “lookup” relationship.

```
RELATION, ONE:MANY, UPDATE( RESTRICT ),DELETE( RESTRICT )
          MANY:ONE          CASCADE            CASCADE
                            CLEAR              CLEAR
filename FILE( {key} )
filename RELATED_FILE( {key} )
{FILE_TO_RELATED_KEY
 FIELD( filefieldname,relatedfilefieldname )    repeatable
     NOLINK,    NOLINK
 END}
{RELATED_FILE_TO_KEY
 FIELD( relatedfilefieldname,filefieldname )    repeatable
     NOLINK,       NOLINK
 END}
END
```

FILE_TO_RELATED_KEY

Defines the links between the two files (optional—for a “one-way” or “lookup” relationship). Linked fields are compared, and those records with matching values are deemed related.

FIELD — Identifies a pair of linked fields between the related files (required).

filefieldname | NOLINK

filefieldname is the label of the field in the first file that is linked (compared) to a field in the related file. NOLINK indicates no comparison is made to the field in the related file.

relatedfilefieldname | NOLINK

relatedfilefieldname is the label of the field in the related file that is linked (compared) to a field in the first file. NOLINK indicates no comparison is made to the field in the first file.

RELATED_FILE_TO_KEY

Defines the links between the two files (optional—for a “one-way” or “lookup” relationship). Linked fields are compared, and those records with matching values are deemed related.

FIELD — Identifies a pair of linked fields between the related files (required).

relatedfilefieldname | NOLINK

relatedfilefieldname is the label of the field in the related file that is linked (compared) to a field in the first file. NOLINK indicates no comparison is made to the field in the first file.

filefieldname | NOLINK

filefieldname is the label of the field in the first file that is linked (compared) to a field in the related file. NOLINK indicates no comparison is made to the field in the related file.

Example—Relation Definition Group

```
[RELATIONS]

                        RELATION,MANY:ONE,UPDATE(RESTRICT),DELETE(RESTRICT)
OrderDetails              FILE(DTL:KeyProductNumber)
Products                  RELATED_FILE(PRO:KeyProductNumber)
                          FILE_TO_RELATED_KEY
                            FIELD(DTL:ProductNumber,PRO:ProductNumber)
                          END
                          RELATED_FILE_TO_KEY
                            FIELD(PRO:ProductNumber,DTL:ProductNumber)
                          END
                        END
                        RELATION,MANY:ONE
Customers                 FILE()
States                    RELATED_FILE(STA:KeyStateCode)
                          FILE_TO_RELATED_KEY
                            FIELD(CUS:State,STA:StateCode)
                          END
                        END
                        RELATION,ONE:MANY,UPDATE(CASCADE),DELETE(RESTRICT)
Customer                  FILE(CUS:KeyCustNumber)
Orders                    RELATED_FILE(ORD:KeyCustDate)
                          FILE_TO_RELATED_KEY
                            FIELD(CUS:CustNumber,ORD:CustNumber)
                            FIELD(NOLINK,ORD:Date)
                          END
                          RELATED_FILE_TO_KEY
                            FIELD(ORD:CustNumber,CUS:CustNumber)
                          END
                        END
```

Common Subsections

The following subsections appear in many instances within the .TXD file. Their syntax and structure is consistent wherever they are found within the file.

[LONGDESC]

Lists up to 1000 characters of descriptive text on multiple lines (optional). Each line of text begins with an exclamation point (!) and contains up to 75 characters. The text comes from the **Comments** tab of the respective **Properties** dialog for the Dictionary, File, Alias, Relation, Field, or Key. For example:

```
[LONGDESC]
!Up to 1000 characters of text describing this item.
!Each line of up to 75 characters begins with an exclamation point.
```

[USEROPTION]

Lists up to 1000 characters of text that is available to templates (optional). Each line of text begins with an exclamation point (!) and contains up to 75 characters.

The text is available to any templates that process the file, alias, field, key, etc. See the EXTRACT procedure documented in the *Template Language Reference* for more information.

The text comes from the **Options** tab of the respective **Properties** dialog for the File, Alias, Relation, Field, or Key. For example:

```
[USEROPTION]
!ThirdPartyTemplateAttribute(on)
!DefaultWindowSize=Max
```

[TOOLOPTION]

Lists up to 1000 characters of text that is available to templates (optional). Each line of text begins with an exclamation point (!) and contains up to 75 characters.

The text is available to any templates that process the file, alias, field, key, etc. See the EXTRACT procedure documented in the Template Language Reference for more information.

The text comes from third-party add-ons and does not appear on any dialog in the Dictionary Editor. For example:

```
[TOOLOPTION]
!MyTool('Global option = A')
!HisTool('True')
```

[SCREENCONTROLS]

Marks the beginning of the subsection that describes the default window controls used to manage a data dictionary field, MEMO, or BLOB (optional).

Following [SCREENCONTROLS], an exclamation point (!) plus a space marks the beginning of a control declaration. The control declaration is the Clarion language statement that defines a window control used to manage the data item. There may be several controls associated with the data item, so there may be several control declarations, beginning immediately after [SCREENCONTROLS] and continuing until the next subsection, usually [REPORTCONTROLS], begins. For example:

```
[SCREENCONTROLS]
! PROMPT('CurrentTab:'),USE(?CurrentTab:Prompt)
! ENTRY(@s80),USE(CurrentTab)
```

[REPORTCONTROLS]

Marks the beginning of the subsection that describes the default report controls used to manage a data dictionary field, MEMO, or BLOB (optional).

Following [REPORTCONTROLS], an exclamation point (!) plus a space marks the beginning of a control declaration. The control declaration is the Clarion language statement that defines a report control used to manage the data item. There may be several controls associated with the data item, so there may be several control declarations, beginning immediately after [REPORTCONTROLS] and continuing until the field definition begins. For example:

```
[REPORTCONTROLS]
! STRING(@s80),USE(CurrentTab)
```

.TXA File Format

.TXA Files: Clarion Applications in ASCII Format

.TXA files are simply ASCII text versions of Clarion .APP files. .APP files are where the Clarion for Windows environment stores all the application specific information necessary to generate and make an application. A .TXA file contains all the information stored in an .APP file (except for project system information), but in a text format that is readable (and writable) by most text editors.

You can easily create a .TXA file for any .APP file by loading the application into the Clarion for Windows development environment, then choosing **File > Export Text** from the menu.

Why would you need a .TXA file? For at least three reasons:

Backup	Because a .TXA file can be imported just as easily as it is exported, it may serve as a backup of your .APP file.
Mass Changes	Because a .TXA file may be manipulated in your favorite text editor, you can use the power of the text editor to make mass changes to your application, or for that matter, any changes that would be easier with a text editor.
First Aid	Occasionally, an .APP file may exhibit some strange behavior in the development environment. Exporting to a .TXA file, then importing from that same file can, in some cases, create a clean .APP file.

.TXA File Organization

The organization and syntax of the .TXA file reflects the Application-Program-Module-Procedure paradigm on which Clarion for Windows generated applications are based. The Class Clarion templates define this paradigm.

There are five major sections in the .TXA file that roughly correspond to the files produced by the Application Generator.

- [APPLICATION] section corresponds to the *appname*.APP file.
- [PROJECT] section contains the project system settings within the *appname*.APP file.
- [PROGRAM] section corresponds to the *appname*.CLW file.
- [MODULE] sections correspond to the *appna00n*.CLW files.
- [PROCEDURE] sections correspond to the procedures defined in the Application Tree and stored in the *appna00n*.CLW files.

.TXA Skeleton

On the following page is an ordered list of .TXA sections, subsections, and keywords. This list is designed to give you a feel for the overall structure and organization of the .TXA file.

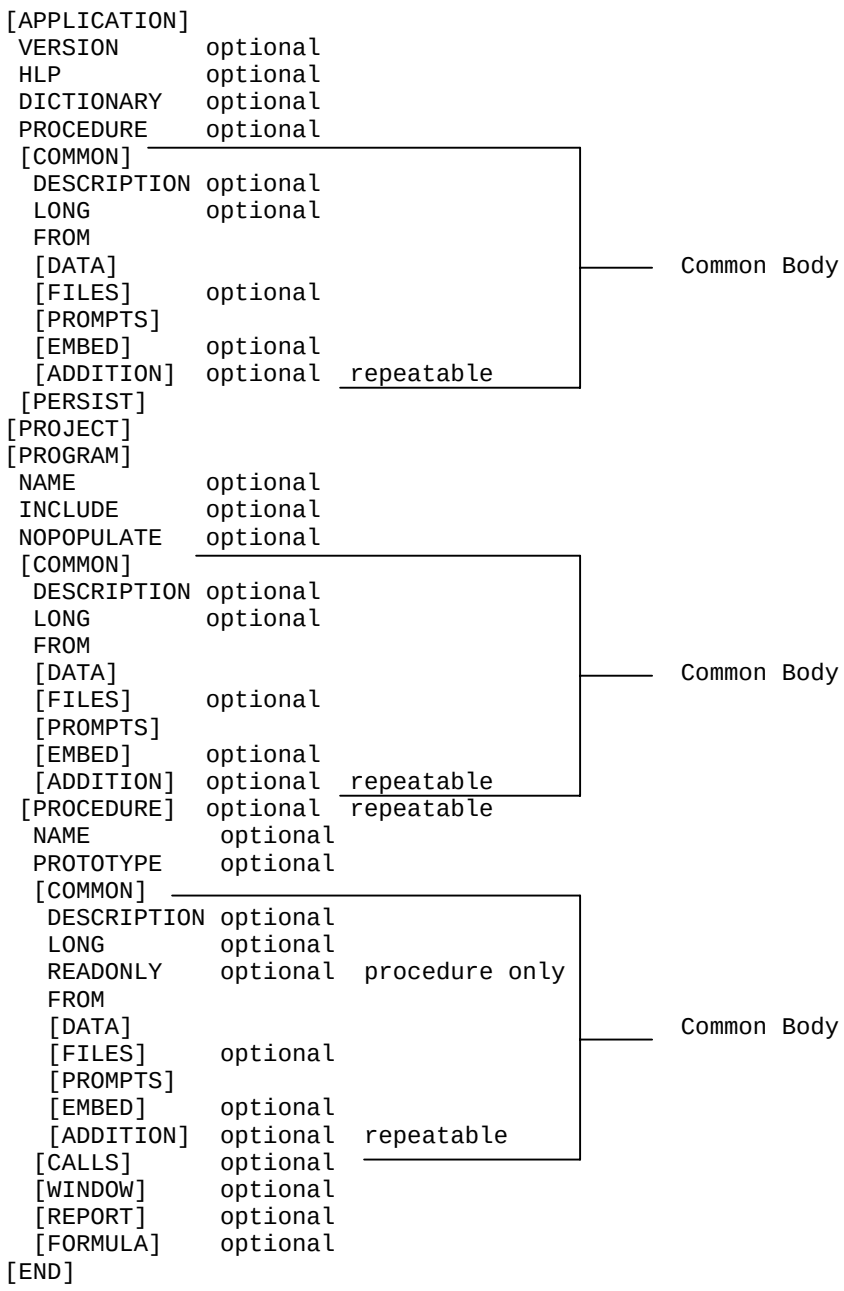
Each section begins with its title enclosed in square brackets and ends with [END], or with the beginning of another section. Each section may contain subsections and keywords.

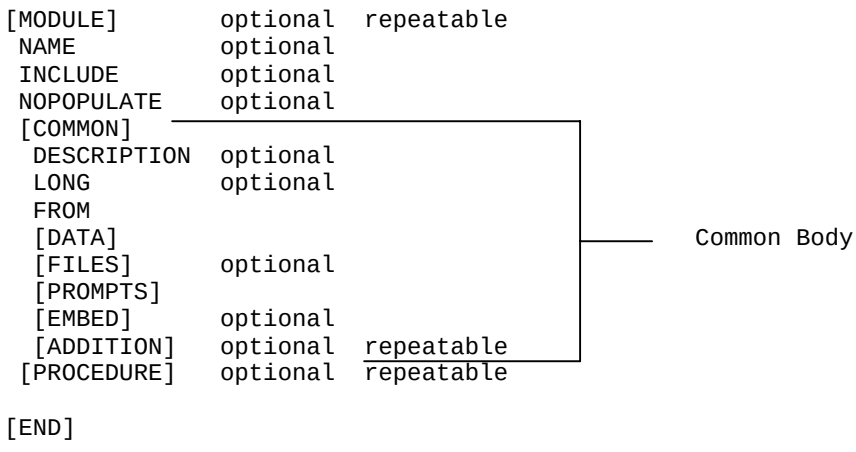
Subsections begin just like the major sections—with a title surrounded by square brackets.

Keywords appear in all capitals followed by the keyword value. The keyword values appear in various formats described below on a case by case basis.

The indentations in the list are provided here for readability and do not appear in the actual .TXA file.

Following the skeleton is a detailed discussion of each .TXA section, subsection, and keyword.


[APPLICATION]
VERSION optional
HLP optional
DICTIONARY optional
PROCEDURE optional
[COMMON]
DESCRIPTION optional
LONG optional
FROM
[DATA]
[FILES] optional
[PROMPTS]
[EMBED] optional
[ADDITION] optional repeatable
Common Body
[PERSIST]
[PROJECT]
[PROGRAM]
NAME optional
INCLUDE optional
NOPOPULATE optional
[COMMON]
DESCRIPTION optional
LONG optional
FROM
[DATA]
[FILES] optional
[PROMPTS]
[EMBED] optional
[ADDITION] optional repeatable
Common Body
[PROCEDURE] optional repeatable
NAME optional
PROTOTYPE optional
[COMMON]
DESCRIPTION optional
LONG optional
READONLY optional procedure only
FROM
[DATA]
[FILES] optional
[PROMPTS]
[EMBED] optional
[ADDITION] optional repeatable
Common Body
[CALLS] optional
[WINDOW] optional
[REPORT] optional
[FORMULA] optional
[END]


[MODULE] optional repeatable
NAME optional
INCLUDE optional
NOPOPULATE optional
[COMMON]
DESCRIPTION optional
LONG optional
FROM
[DATA]
[FILES] optional
[PROMPTS]
[EMBED] optional
[ADDITION] optional repeatable
[PROCEDURE] optional repeatable
[END]
Common Body

.TXA File Sections

[APPLICATION]

The application section is required and appears only once at the top of each .TXA file (unless only a portion of the application is exported, e.g. a .TXA file may contain a single module or procedure, in which case the file begins with [MODULE] or [PROCEDURE] respectively). This section begins with [APPLICATION] and ends with the beginning of the [PROGRAM] section. The application section contains the following keywords and subsections that pertain to the entire application.

```
[APPLICATION]
 VERSION      optional
 HLP          optional
 DICTIONARY   optional
 PROCEDURE    optional
 [COMMON]
  DESCRIPTION optional
  LONG        optional
  FROM
  [DATA]
  [FILES]     optional
  [PROMPTS]
  [EMBED]     optional
  [ADDITION]  optional  repeatable
 [PERSIST]
```

VERSION — The version number of the application (optional). This value reflects the version of generator, and changes when the TXA format changes. For example:

```
VERSION 10.
```

HLP — The Windows help file called by the application (optional). The file name may be fully qualified or not. If not, Clarion searches in the current directory, the system path, then in paths specified by the redirection file (C:\C60\BIN\C60EE.RED). For example:

```
HLP 'C:\C60\.APPS\MY.APP.HLP'
```

DICTIONARY — The data dictionary file used by the application (optional). The file name may be fully qualified or not. If not, Clarion searches the paths specified by the redirection file (C:\C60\BIN\C60EE.RED). For example:

```
DICTIONARY 'TUTORIAL.DCT'
```

PROCEDURE The name of the first procedure in the application (optional—- no meaning for a .LIB or .DLL). This procedure calls all other procedures in the application, either directly, or indirectly. For example:

```
PROCEDURE Main
```

[COMMON] The common subsection appears in the [APPLICATION], [PROGRAM], [MODULE], and [PROCEDURE] sections (required). This subsection begins with [COMMON] and ends with the beginning of the next subsection. This subsection can vary substantially in length and appearance, depending on the section in which it resides and on the subsections it contains or omits. See the [COMMON] section below for a full discussion.

[PERSIST] Information about the application that is “remembered” across sessions (required). Although these items appear in .TXA [PROMPT] format, they do not appear to the developer as prompts, rather, they are #DECLAREd in the application template with the SAVE attribute, which causes them to be saved in the .APP file so they are available for each new session.

See [PROMPT] below for more information on the syntax of these application keywords.

Example—[APPLICATION]

```
[APPLICATION]
VERSION 10
HLP ‘C:\C60\.APPS\MY.APP.HLP’
DICTIONARY ‘TUTORIAL.DCT’
PROCEDURE Main
[COMMON]
       .
       .
       .
```

[PROJECT]

The project section is always present and appears only once, after the [APPLICATION] section of each .TXA file. The project section begins with [PROJECT] and ends with [PROGRAM]. It is generated (exported) automatically for each application.

This section contains the project system settings specified for this application. It is provided so that project system settings are preserved when you export an application to a .TXA file, then import the .TXA back into an .APP file.

```
[PROJECT]
-- Generator
#noedit
#system win
#model clarion dll
#pragma debug(vid=>full)
#compile BIG_RD.CLW /define(GENERATED=>on)-- GENERATED
#compile BIG_RU.CLW /define(GENERATED=>on)-- GENERATED
#compile BIG_SF.CLW /define(GENERATED=>on)-- GENERATED
#compile ResCode.Clw /define(GENERATED=>on)-- GENERATED
#compile BIG.clw /define(GENERATED=>on)-- GENERATED
#compile BIG001.clw /define(GENERATED=>on)-- GENERATED
#compile BIG002.clw /define(GENERATED=>on)-- GENERATED
#pragma link(C%L%2AS4%S%.LIB)-- GENERATED
#link BIG.EXE
[PROGRAM]
```

See *Project System* for complete information on project system syntax and pragmas.

[PROGRAM]—[END]

The program section is required and appears only once, after the [APPLICATION] section of each .TXA file. The program section begins with [PROGRAM] and ends with [END]. This section contains the following keywords and subsections that pertain to the source file that contains the PROGRAM statement.

```
[PROGRAM]
 NAME           optional
 INCLUDE        optional
 NOPOPULATE     optional
 [COMMON]
  DESCRIPTION   optional
  LONG          optional
  FROM
  [DATA]
  [FILES]       optional
  [PROMPTS]
  [EMBED]       optional
  [ADDITION]    optional  repeatable
 [PROCEDURE]    optional  repeatable
[END]
```

NAME — The name of the source file that contains the PROGRAM statement for this application (optional).). If omitted it defaults to the application name. For example:

```
NAME 'TUTORIAL.CLW'
```

INCLUDE — The name of a source file that is included in the data declaration section of the program source file (optional). For example:

```
INCLUDE 'EQUATES.CLW'
```

NOPOPULATE — The Application Generator may not store procedures in this source file (optional). This is typically present for external modules. For example:

```
NOPOPULATE
```

[COMMON] — The common subsection appears in the [APPLICATION], [PROGRAM], [MODULE], and [PROCEDURE] sections (required). This subsection begins with [COMMON] and ends with the beginning of the next subsection. This subsection can vary substantially in length and appearance, depending on the section in which it resides and on the subsections it contains or omits. See the [COMMON] section below for a full discussion.

[PROCEDURE]

Information that fully defines a procedure (optional). The procedure subsection may be repeated for each procedure in the module. The information stored in this subsection applies only to the procedure identified by the NAME keyword.

See the [PROCEDURE] section below for more information on the structure and syntax of this subsection.

Example—[PROGRAM]-[END]

```
[PROGRAM]
NAME ‘ TUTORIAL.CLW‘
INCLUDE ‘EQUATES.CLW’
NOPOPULATE
[COMMON]
       .
       .
       .
[END]
```

[MODULE]—[END]

The structure and syntax of the module section is identical to that of the program section. The module section is optional and may be repeated as many times as necessary. The section begins with [MODULE] and ends with [END].

Although identical in syntax and structure, the module subsection differs in the scope of it’s applicability. The module section is repeated once for each module in the application, and, the information it contains applies only to the source file identified by its NAME keyword, that is, only to those procedures that reside within this module. Data defined in its [DATA] section is “module” data, and is available to all procedures in the module.

[PROCEDURE]

The procedure subsection is optional and is repeated for each procedure in the program or module. The subsection begins with [PROCEDURE] and ends with the next [PROCEDURE], or the [END] of the module or program. This subsection contains information that pertains only to the procedure identified by the NAME keyword.

```
[PROCEDURE]    optional  repeatable
 NAME          optional
 PROTOTYPE     optional
 [COMMON]
  DESCRIPTION  optional
  LONG         optional
  READONLY     optional
  FROM
  [DATA]
  [FILES]      optional
  [PROMPTS]
  [EMBED]      optional
  [ADDITION]   optional  repeatable
 [CALLS]       optional
 [WINDOW]      optional
 [REPORT]      optional
 [FORMULA]     optional
```

NAME — The name of the procedure the keywords in this section apply to (optional). For example:

```
NAME BrowseCustomers
```

PROTOTYPE — The prototype for the procedure (optional). See the Language Reference for more information on prototyping your procedures. For example:

```
PROTOTYPE ‘LONG Count, REAL Sum’
```

[COMMON] — The common subsection appears in the [APPLICATION], [PROGRAM], [MODULE], and [PROCEDURE] sections (required). This subsection begins with [COMMON] and ends with the beginning of the next subsection. This subsection can vary substantially in length and appearance, depending on the section in which it resides and on the subsections it contains or omits. See the [COMMON] section below for a full discussion.

[CALLS] — Procedures called by this procedure (optional). For example:

```
[CALLS]
UpdateCustomers
BrowseOrders
```

[WINDOW] — Clarion Language statements that define the window managed by this procedure (optional).

In addition to the Clarion Language statements defining the WINDOW structure, the window subsection may contain the following four keywords which are used internally by the development environment:

#SEQ(instance number)

All controls populated from a control template have this keyword which gives the instance number of the control template of which they are a member.

#ORIG(original name of control)

The original name of the control as given in the control template.

#FIELDS(list of fields in a list box)

The list of fields to display in a LIST control.

#LINK(field equate label of linked fields)

If several controls are populated from the same control template, they are linked together in a cycle, each being linked to the next. If a field is populated from the dictionary the prompts are also linked to the entry fields.

For example:

```
[WINDOW]
QuickWindow WINDOW('Browse the Customers File'),AT(,,358,188),SYSTEM,GRAY,MDI
 LIST,AT(8,20,342,124),USE(?Browse:1),IMM,HVSCROLL,FORMAT('16L|M~Cust' &|   'Number~@n4@80L|M~Comp
Name~@S20@80L|M~Address~@S20@80L|M~City~@S20' &|
 '@8L|M~State~@S2@20L|M~Zip~@S5@'),FROM(Queue:Browse:1),#SEQ(1),#ORIG(?List), |
 #FIELDS(CUS:CustNumber,CUS:CompanyName,CUS:Address,CUS:City,CUS:State,CUS:Zip)
  BUTTON('&Insert'),AT(207,148,45,14),USE(?Insert:2),#SEQ(2),#ORIG(?Insert),#LINK(?Change:2)
  BUTTON('&Change'),AT(256,148,45,14),USE(?Change:2),DEFAULT,#SEQ(2),#ORIG(?Change),#LINK(?Delete:
  BUTTON('&Delete'),AT(305,148,45,14),USE(?Delete:2),#SEQ(2),#ORIG(?Delete),#LINK(?Insert:2)
 END
```

[REPORT] Clarion Language statements that define the REPORT managed by this procedure (optional). In addition to the Clarion Language statements, the report subsection may contain the four keywords above.

[FORMULA] Describes each formula defined for this procedure by the **Formula Editor** (optional). Notice that the keywords and their values correspond exactly to the **Formula Editor** dialog. See the *User's Guide* and the on-line help for more information on these fields. For example:

```
[FORMULA]
DEFINE OrderTax
ASSIGN OrderTax
CLASS After Lookups
DESCRIPTION Calculate State Tax
= OrderTotal * StateTaxRate
[END]
```

ple—[PROCEDURE]

```
JRE]
)wseCustomers
'E 'LONG Count, REAL Sum'[CALLS]

.
.
.

ıstomers
·ders

ıdow WINDOW('Browse Customers'),AT(,,358,188),SYSTEM,GRAY,MDI
·(8,20,342,124),USE(?Browse:1),IMM,FORMAT('16L|M~Cust' &|
·@n4@80L|M~Name~@S20@80L|M~Address~@S20@80L|M~City~@S20' &|
·St~@S2@20L|M~Zip~@S5@'),FROM(Queue:Browse:1),#SEQ(1),#ORIG(?List),|
;(CUS:CustNumber,CUS:CompanyName,CUS:Address,CUS:City,CUS:State,CUS:Zip)
I('&Insert'),AT(207,148,45,14),USE(?Insert:2),#SEQ(2),#ORIG(?Insert),#LINK(?Change:2)
I('&Change'),AT(256,148,45,14),USE(?Change:2),DEFAULT,#SEQ(2),#ORIG(?Change),#LINK(?Delete:2)
I('&Delete'),AT(305,148,45,14),USE(?Delete:2),#SEQ(2),#ORIG(?Delete),#LINK(?Insert:2)

 ]
)rderTax
)rderTax
'ter Lookups
'ION Calculate State Tax
'otal * StateTaxRate
```

Common Subsections

[COMMON]

The common subsection appears in the [APPLICATION], [PROGRAM], [MODULE], and [PROCEDURE] sections. This subsection begins with [COMMON] and ends with the beginning of the next subsection. This subsection is required, although many of its keywords and subsections are optional. This subsection can vary substantially in length and appearance, depending on the section in which it resides and on the subsections it includes or omits. [COMMON] contains the following keywords and subsections.

```
[COMMON]
      DESCRIPTION         optional
      LONG                optional
      FROM
      MODIFIED
      [DATA]
      [FILES]             optional
      [PROMPTS]
      [EMBED]             optional
      [ADDITION]          optional            repeatable
```

DESCRIPTION — Up to 40 characters of text describing the application, program, module, or procedure (optional). For example:

```
DESCRIPTION 'Print dynamic label report'
```

LONG — Up to 1000 characters of text describing the application, program, module, or procedure (optional). For example: The text is actually split into several lines that are concatenated together as they are read. For example:

```
LONG 'Print dynamic label report. At runtime, the '
LONG 'user selects (or adds a new description of) '
LONG 'a label paper from the LAB file. This proced'
LONG 'ure then makes property assignments to adjus'
LONG 't the size and location of the label text.'
```

READONLY — The procedure may be viewed, but not modified from the Clarion for Windows environment (optional). Only allowed in a [PROCEDURE] subsection. READONLY cannot currently be added to a procedure by the environment, but is provided for future use so that SoftVelocity's developers can implement multi-developer environments that allow a procedure to be "checked out" and "checked in" in order to preserve code integrity. For example:

```
READONLY
```

FROM — The name of the template class for an application, or the name of the template class and the specific template from which the program, module, or procedure is generated (optional - can only be omitted for a ToDo procedure). For example:

```
[APPLICATION]
.
.
.
FROM Clarion
      or
[PROCEDURE]
.
.
.
FROM Clarion Report
```

MODIFIED — The date and time the procedure was last modified. For example:

```
[PROCEDURE]
MODIFIED '1998/07/02' '10:43:32'
```

Example—[COMMON]

```
[COMMON]
DESCRIPTION 'Print dynamic label report'
LONG 'Print dynamic label report. At runtime, the user selects (or adds a new
description of) a label paper from the LAB file. This procedure then makes
property assignments to adjust the size and location of the label text to fit
the selected label paper.'
READONLY
FROM Clarion Report
```

[DATA]

The data subsection is an optional part of the [COMMON] subsection. It may contain several subsections and keywords that describe each memory variable defined for this procedure, module, program, or application. See the *.TXD File Format* chapter of this book for a discussion of how this same syntax applies to data dictionary fields. Also see *Defining Procedure Data* in the *User's Guide*.

For each memory variable defined, there is a series of optional subsections and keywords that fully describe the variable, as well as any default formatting conventions the Application Generator is expected to follow. The complete list of possible subsections and keywords is:

```
[DATA]
 [LONGDESC]          optional
 [USEROPTION]        optional
 [SCREENCONTROLS]    optional
 [REPORTCONTROLS]    optional
 field definition
 keyword list        optional
```

[LONGDESC] Up to thirteen (13) lines of text, up to seventy-five (75) characters in length each (optional). Each line of text begins with an exclamation point (!). Comes from the **Comments** tab of the **Field Properties** dialog. For example:

```
[LONGDESC]
!CurrentTab is used internally by the template
!generated code to store the number/id of the !TAB
control that has focus.
```

[USEROPTION] Up to thirteen (13) lines of text up to seventy-five (75) characters in length each (optional). Each line of text begins with an exclamation point (!). The text is available to templates. See the **EXTRACT** procedure in the *Template Language Reference*. Comes from the **Options** tab of the **Field Properties** dialog. For example:

```
[USEROPTION]
!ThirdPartyTemplateAttribute:Details(on)
!ThirdPartyTemplateAttribute:WizardHelp(off)
```

[SCREENCONTROLS] The beginning of the subsection that describes the default window controls used to manage the memory variable (optional). Use the **Window** tab of the **Field Properties** dialog to set the default controls.

Following [SCREENCONTROLS], an exclamation point (!) marks the beginning of a control declaration. The control declaration is the Clarion language statement that defines a window control. There may be several controls associated with the memory variable, so there may be several control declarations, beginning immediately after [SCREENCONTROLS] and continuing until the next subsection, usually [REPORTCONTROLS]. For example:

```
[SCREENCONTROLS]
! PROMPT('CurrentTab:'),USE(?CurrentTab:Prompt)
! ENTRY(@s80),USE(CurrentTab)
```

[REPORTCONTROLS] The beginning of the subsection that describes the default report controls used to manage the memory variable (optional). Use the **Report** tab of the **Field Properties** dialog to set the default controls.

Following [REPORTCONTROLS], an exclamation point (!) marks the beginning of a control declaration. The control declaration is the Clarion language statement that defines a report control. There may be several controls associated with the memory variable, so there may be several control declarations, beginning immediately after [REPORTCONTROLS] and continuing until the field definition begins. For example:

```
[REPORTCONTROLS]
! STRING(@s80),USE(CurrentTab)
```

Field Definition — Lists the Clarion language field declaration (required). Optionally includes a text description of up to 40 characters. The text description begins with an exclamation point (!). For example:

```
CurrentTab   STRING(80) !user selected tab
```

Keyword List — The keywords that specify various attributes of the memory variable (optional). The list begins with !!>.

The keywords in this list are set using the **Field Properties** dialog. Many of the keywords correspond directly to Clarion language keywords. See the *Language Reference* for more information on these keywords.

See the *.TXD File Format* chapter of this book for a complete discussion of the keywords and their effects. Also see the *User's Guide*, the *Language Reference*, and the on-line help for more information on particular Clarion language keywords.

Notice that the keywords in the [DATA] subsection correspond closely to the tabs and prompts in the **Field Properties** dialog. This is because the **Field Properties** dialog is where these values are set

Example—[DATA]

For any given variable, you will usually see only a fraction of the possible subsections and keywords, because, some are mutually exclusive, and many others are simply not required. The one item that is *required* is the Clarion Language field definition.

Let's examine each line of the following typical example to illustrate how this subsection works:

```
[DATA]
[SCREENCONTROLS]
! PROMPT('CurrentTab:'),USE(?CurrentTab:Prompt)
! ENTRY(@s80),USE(CurrentTab)
[REPORTCONTROLS]
! STRING(@s80),USE(CurrentTab)
CurrentTab              STRING(80) ! Tab selected by user
!!> IDENT(4294967206),PROMPT('CurrentTab:'),HEADER('CurrentTab'),
PICTURE(@s80)
[SCREENCONTROLS]
       .
       .
```

[DATA] marks the beginning of this subsection which describes all the memory variables for this application, program, module, or procedure.

[SCREENCONTROLS] marks the beginning of the subsection that describes the default window controls used to manage the first memory variable.

Following [SCREENCONTROLS], the exclamation point (!) marks the beginning of a control declaration. The control declaration is the Clarion language statement that defines a window control used to manage the memory variable. There may be several controls associated with the variable, so there may be several control declarations beginning immediately after [SCREENCONTROLS] and continuing until a new subsection, usually [REPORTCONTROLS], begins.

[REPORTCONTROLS] marks the end of the [SCREENCONTROLS] subsection, and the beginning of the subsection that describes the default report controls used to display the memory variable. The control declaration is the Clarion language statement that defines a report control used to manage the memory variable. There may be several controls associated with the variable, so there may be several control declarations beginning immediately after [REPORTCONTROLS] and continuing until the field definition begins.

Following the report control declaration, is the Clarion Language field definition for the variable: "CurrentTab String(80)." *This is the only required keyword* for each memory variable described in the .TXA file. It may optionally be followed by an exclamation point (!) and the short description for the variable.

Finally, "!!>" marks the beginning of the keyword list associated with the memory variable. The keywords are separated by commas, and the list continues, wrapping onto multiple lines if necessary, until the next subsection begins. See the *.TXD File Format* chapter of this book for a complete discussion of the keywords and their effects.

[FILES]

The files subsection is an optional part of the [COMMON] subsection. It may contain several subsections and keywords that describe the files used by this procedure, module, program, or application. In Class Clarion generated applications, the [FILES] subsection is most commonly

seen in the [PROCEDURE] subsection, since the procedures do most of the file access, while applications, programs, and modules are more concerned with managing the user's environment.

For each file used by the procedure, module, program, or application, there is a series of subsections and keywords that identify the file, the key, and any related files that will also be used. The complete list of possible subsections and keywords is:

```
[FILES]        optional
 [PRIMARY]     optional, repeatable
  [INSTANCE]
  [KEY]        optional
  [SECONDARY]  optional, repeatable
 [OTHERS]      optional
  .
  .
  .
```

[FILES] The beginning of this subsection which identifies the essential information about the files used by this procedure (optional).

[PRIMARY] The beginning of the subsection which describes a single primary file tree for a control template in this procedure (required). There may be a [PRIMARY] subsection for each control template in the procedure.

Primary simply means that this is the main file processed by the associated control template. Any other files processed by the control template are dependent files.

The line immediately after [PRIMARY] lists the filename. For example:

```
[PRIMARY]
Customers
```

[INSTANCE] Introduces the instance number of the control template for which this file is primary (required). See Common Subsections—[ADDITION] for more information on instance numbers. For example:

```
[INSTANCE]
6
```

[KEY] Introduces the key used to access this file (optional). For example:

```
[KEY]
CUS:KeyCustNumber
```

[SECONDARY] Introduces the child and the parent file accessed by this control template (optional). This subsection is repeated for each related file.

Both files are named in order to avoid any ambiguity when there is more than one related file. The first file listed is always the "child" file, and the second file listed is always the "parent" file.

The [SECONDARY] file identities are part of the [PRIMARY] subsection, that is, a [SECONDARY] does not mark the end of the [PRIMARY] subsection but is a continuation of it. For example:

```
[SECONDARY]
Phones Customers
```

[OTHERS] Introduces other files accessed by the procedure, but not by a control template (optional). It also marks the end of the prior [PRIMARY] subsection. The only code generated for an [OTHERS] file is just that code necessary to open the file at the beginning of the procedure, then close the file at the end of the procedure. For example:

```
[OTHERS]
Labels
```

Example—[FILES]

```
[FILES]
[PRIMARY]
Customers
[INSTANCE]
6
[KEY]
CUS:KeyCustNumber
[SECONDARY]
Phones Customers
[SECONDARY]
Orders Customers
[OTHERS]
Labels
       .
       .
       .
```

Let's examine each line of the following comprehensive example to illustrate how this subsection works:

[FILES] marks the beginning of this subsection, which identifies the essential information about the files that are used by this procedure.

[PRIMARY] marks the beginning of the subsection, which describes a single primary file tree for a control template. Primary simply means that this is the main file processed by the associated control template. Any other files processed by the control template are dependent files.

The line immediately after [PRIMARY] shows the filename, that is, Customers.

[INSTANCE] means the following line shows the instance number of the control template for which this file is primary. The development environment uses the instance number to link the appropriate file to the appropriate control template.

[KEY] means the following line shows the key used to access the primary, that is, CUS:KeyCustNumber.

[SECONDARY] means the following line shows a related file that is also accessed by this control template: Phones. Notice that the line naming the secondary file also names the parent file:

Customers. This is to avoid any ambiguity when there is more than one related file. For example, if the Orders file was listed as a secondary file as well as the Phones file, it would be important to know if the Orders file is related to the Phones file or the Customers file.

Similarly, the order in which the two file names appear is significant. The first file listed is always the “child” file, and the second file listed is always the “parent” file.

Finally, [OTHERS] means the following lines show other files accessed by the procedure, but not a control template. It also marks the end of the prior [PRIMARY] subsection.

[PROMPTS]

The prompts subsection is part of the common subsection. It lists the template prompts associated with the application, program, module, or procedure, plus the values supplied for the prompts by the developer. See the *Template Language Reference* in the on-line help for more information on template prompts.

Template prompts may be found in several different subsections, including [PERSIST] (see [APPLICATION]), [PROMPTS], and [FIELDPROMPT].

The template prompts and their associated values appear in two different formats: simple and dependent.

Simple Prompts

Both formats begin with the prompt name. The prompt name begins with the percent sign (%). The **simple format** follows the prompt name with a prompt type and a value enclosed in parentheses.

Available prompt types are @picture, LONG, REAL, STRING, FILE, FIELD, KEY, COMPONENT, PROCEDURE, and DEFAULT. The prompt type may optionally be further qualified as UNIQUE or as MULTI. MULTI means the prompt has multiple values. UNIQUE also indicates multiple values, however, the values are in ascending order and there are no duplicates.

The simple format syntax is diagrammed as follows, where the vertical columns show alternative parameters, the curly braces show optional parameters, and *value* is the value for the prompt:

%name	{UNIQUE}	@picture	(‘value’)
	{MULTI}	LONG	(‘value1’,’value2’,’valuen’)
		REAL	
		STRING	
		FILE	
		FIELD	
		KEY	
		COMPONENT	
		PROCEDURE	
		DEFAULT	

Following is a description of the available prompt types:

@picture	The value is a picture token.
LONG	The value is a number in LONG format (4 byte unsigned integer).
REAL	The value is a number in REAL format (8 byte floating point format).
STRING	The value is a character string.
FILE	The value is the label of a data file.
FIELD	The value is the label of a field in a data file.
KEY	The value is the label of a key.

COMPONENT The value is the label of a key component.

PROCEDURE The value is the label of a procedure.

DEFAULT Typically the same as STRING, but stored differently internally. DEFAULT variables do not have an explicit type. If they are assigned to, the variable takes the type of the value that is assigned to it.

For example:

```
[PROMPTS]
%LastProgramExtension DEFAULT  ('EXE')
%SizePreferences MULTI LONG (3,3)
%RangeField COMPONENT  (PHO:CustNumber)
%UpdateProcedure PROCEDURE  (UpdatePhones)
```

Let's examine each line of this example.

%LastProgramExtension is the template name of the stored value. DEFAULT indicates the value is stored as a string (since 'EXE' is a string). The ('EXE') indicates the value of %LastProgramExtension is 'EXE.'

%SizePreferences is the template name of the stored value. The value is stored in LONG format and MULTI means there are multiple values. The (3,3) indicates the values for %SizePreferences are 3 and 3.

%RangeField is the template name of the stored value. The value stored is the label of a KEY COMPONENT, and PHO:CustNumber is that label.

%UpdateProcedure is the template name of the information stored. The information stored is the label of a PROCEDURE and UpdatePhones is that label.

Dependent Prompts

Like simple prompts, dependent prompts begin with the prompt name. The prompt name begins with the percent sign (%). The **dependent format** follows the prompt name with the keyword DEPEND and is spread over multiple lines.

The dependent format is diagrammed as follows, where the vertical columns show alternative parameters, the curly braces show optional parameters, and the WHEN lines represent all the possible values for both the %Prompt and the %ParentSymbol:

```
%Prompt      DEPEND      %ParentSymbol{UNIQUE}     @picture      TIMES  n
                                      {MULTI}      LONG
                                                   REAL
                                                   STRING
                                                   FILE
                                                   FIELD
                                                   KEY
                                                   COMPONENT
                                                   DEFAULT
WHEN  ('ParentSymbolValue')  ('promptvalue')1
WHEN  ('ParentSymbolValue')  ('promptvalue')2
WHEN  ('ParentSymbolValue')  ('promptvalue')n
```

The %ParentSymbol following the DEPEND keyword represents a template symbol that the value of %Prompt depends upon. That is, the %ParentSymbol may have multiple different values, and the value of %Prompt depends on the current value of %ParentSymbol.

For example:

```
[PROMPTS]
%GenerationCompleted DEPEND %Module DEFAULT TIMES 4
WHEN  ('TUTORIAL.clw') ('1')
WHEN  ('TUTOR001.clw') ('1')
WHEN  ('TUTOR002.clw') ('1')
WHEN  ('TUTOR003.clw') ('1')
       .
```

Again, let's examine each line of the example. %GenerationCompleted is the name of the prompt for which the value is stored. The value of %GenerationCompleted DEPENDs on the value of %Module. The values are stored as strings. TIMES 4 means %Module has 4 different possible values.

On the following 4 lines, 1 for each possible value of %Module, WHEN indicates a possible value for %Module—'TUTORIAL.CLW', followed by the corresponding value for %GenerationCompleted—'1'.

Nested Dependent Prompts

Dependent prompts can also show more than one level of dependency. The WHEN instances are nested for each additional level of dependency. For example:

```
%ForegroundNormal DEPEND %Control DEPEND %ControlField LONG TIMES 2
WHEN  (‘?Browse:1’) TIMES 2
WHEN  (‘CUS:CustNumber’) (4294967295)
WHEN  (‘CUS:CompanyName’) (4294967295)
WHEN  (‘?Browse:2’) TIMES 4
WHEN  (‘CUS:Address’) (4294967295)
WHEN  (‘CUS:City’) (4294967295)
WHEN  (‘CUS:State’) (4294967295)
WHEN  (‘CUS:ZipCode’) (4294967295)
```

In this example, the value of %ForegroundNormal depends on the value of %Control, and then on the value of %ControlField. %Control can have 2 possible values: ?Browse:1 and ?Browse:2. For each of these values, there is a WHEN...TIMES line that shows the number of possible values of %ControlField associated with this value of %Control.

Then, following each WHEN...TIMES line, are more WHEN lines showing each possible value for %ControlField, followed by the corresponding value for %ForegroundNormal. Note the precedence, %controlfield is dependent on the current value of %control.

[EMBED]—[END]

The embed subsection is an optional part of the common subsection. It may contain several subsections and keywords that describe each embed point defined for this procedure, module, or program with the Embedded Source dialog. See Defining Embedded Source in the User’s Guide.

The [EMBED] subsection may contain the following subsections and keywords:

```
[EMBED]          optional
 EMBED           repeatable
  [INSTANCES]    optional, repeatable
   WHEN
   [DEFINITION]
    [SOURCE]     optional, repeatable
    [TEMPLATE]   optional, repeatable
    [PROCEDURE]  optional, repeatable
    [GROUP]      optional, repeatable
     INSTANCE 4
   [END]
  [END]
[END]
```

[EMBED]-[END]	Marks the beginning of this subsection that describes each embed point defined for this procedure (optional).
EMBED	The string following this keyword identifies the embed point (required). EMBED appears once for each “filled” embed point. For example: `EMBED %ControlPreEventHandling`
[INSTANCES]-[END]	Indicates there is more than one instance of this embed point (optional). See the example below.

WHEN	Indicates in which instance of the embed point the source statements are embedded (required). For example:
	`[INSTANCES]`
	`WHEN ‘?Change:2’`

[DEFINITION]-[END]	Marks the beginning of the subsection that defines the embedded source statements (required).

[SOURCE]	Identifies the source statements as free-form text from the Text Editor (optional). This may contain omittable information delimited by PROPERTY:BEGIN and PROPRETY:END statements. For example:
	`[SOURCE]`
	`PROPERTY:BEGIN`
	`PRIORITY 4000`
	`PROPERTY:END`
	`!This is a source embed point`

[TEMPLATE]	Identifies the source statements as free-form text from the Text Editor which includes Template Language statements (optional). For example:
	`[TEMPLATE]`
	`!This is a template coded embed point`
	`#FOR(%LocalData)`
	`! %LocalData`
	`#ENDFOR`

[PROCEDURE]	Identifies the embedded source statements as a procedure call from the Procedure to Call dialog. For example:
	`[PROCEDURE]`
	`EmbeddedProcedureCall`

[GROUP]	Identifies the embedded source as a code template.

INSTANCE	Identifies the instance number of the embedded code template. For example:
	`[GROUP]`
	`INSTANCE 4`

Example—[EMBED]-[END]

Let's examine a comprehensive example to illustrate each component of the [EMBED] subsection. First, notice how the .TXA text is very similar to the text in the Embedded Source dialog:

```
[EMBED]
EMBED %ControlPreEventHandling
[INSTANCES]
WHEN '?Change:2'
[INSTANCES]
WHEN 'Accepted'
[DEFINITION]
[SOURCE]
!This is a source embed point
[PROCEDURE]
EmbeddedProcedureCall
[GROUP]
INSTANCE 4
[END]
[END]
[END]
EMBED %ControlEventHandling
       . . .
[END]
```

[EMBED] marks the beginning of the subsection. The [EMBED] subsection always ends with [END].

EMBED %ControlPreEventHandling identifies the embed point where the source is embedded. Notice that the name for the embed point in the [EMBED] subsection is slightly different than in the **Embedded Source** dialog. The names in the **Embedded Source** dialog are expanded for maximum clarity.

[INSTANCES] indicates there is more than one instance of the %ControlPreEventHandling embed point, that is, there is one instance of this embed point for each control in the procedure. Each [INSTANCES] ends with [END].

WHEN '?Change:2' indicates in which instance of the embed point the source statements are embedded. At code generation time, there is a variable (or macro) named %Control that has the value '?Change:2' indicating which control, and thus which instance of the embed point, is processed.

The fifth and sixth lines indicate yet another level of instances for this embed point. Not only does the embed point have an instance for each control in the procedure, it has an instance for each event associated with each control. At code generation time, there is a variable (or macro) named %ControlEvent that has the value 'Accepted' indicating which control event, and thus which instance of the embed point, is processed.

[DEFINITION] marks the beginning of the subsection that defines the embedded source statements. The [DEFINITION] ends with [END].

[SOURCE] indicates the type of embedded source: free-form source from the Text Editor. The free-form source follows on the next line and continues until the next .TXA subsection begins.

[PROCEDURE] indicates another type of embedded source: a procedure call. The procedure call follows on the next line.

[GROUP] indicates the third type of embedded source: a code template. Code templates are described in the [ADDITION] subsection. INSTANCE 4 indicates the instance subsection within the addition subsection where the embedded code template is described. (The name [GROUP], rather than [CODE] is used for historical reasons.)

[ADDITION]

The addition subsection is an optional part of the common subsection and appears once for each code template used. It may contain several subsections and keywords that describe each control, code, and extension template defined for this procedure, module, or program. See Using Control, Code, and Extension Templates in the User's Guide.

The [ADDITION] subsection may contain the following subsections and keywords:

```
[ADDITION]       repeatable
 NAME
 [FIELDPROMPT]   optional
 [INSTANCE]      repeatable
  INSTANCE
 PARENT          optional
 PROCPROP        optional
 [PROMPTS]       optional
```

[ADDITION] Marks the beginning of the subsection (optional). Appears once for each template *type* used. That is, if BrowseBox is used twice in a procedure, there is only one BrowseBox [ADDITION] in the [PROCEDURE] section, but with multiple [INSTANCE]s (see below).

NAME Identifies the template class and the specific template invoked (required). Appears once for each [ADDITION] subsection. For example:

```
NAME Clarion BrowseUpdateButtons
```

[FIELDPROMPT]

Indicates a prompt and its associated type and value (optional). This is only generated if you use a #FIELD statement in your templates. The prompts begin on the following line. See [PROMPTS] for a full discussion of prompt syntax. For example:

```
[FIELDPROMPT]
%MadeItUp LONG  (1)
```

[INSTANCE] Introduces the INSTANCE number on the following line. Appears once for each control, code, or extension template in the application, program, module, or procedure. See NAME above.

INSTANCE Indicates the instance number (identification number) of this particular template addition. For example:

```
[INSTANCE]
INSTANCE 2
```

PARENT Indicates this control template depends on another control template (optional). PARENT is followed by the instance number of the control template upon which this control template depends. For example:

```
PARENT 1
```

PROCPROP Means the prompts for this control template are shown in the **Procedure Properties** dialog (optional). If PROCPROP is absent, the prompts will not be displayed in the **Procedure Properties** dialog. For example:

```
PROCPROP
```

[PROMPTS] Marks the beginning of the list of prompts for this control template, and the values supplied by the developer for each prompt (optional). The prompts begin on the following line and continue until the beginning of the next .TXA subsection. See the [PROMPTS] section above for a complete discussion of prompt syntax. For example:

```
[PROMPTS]
%UpdateProcedure PROCEDURE  (UpdatePhones)
%EditViaPopup LONG  (1)
```

Example—[ADDITION]

Again, let's examine a comprehensive example to illustrate each component of the [ADDITION] subsection.

```
[ADDITION]
NAME Clarion BrowseUpdateButtons
[FIELDPROMPT]
%MadeItUp LONG  (1)
[INSTANCE]
INSTANCE 2
PARENT 1
PROCPROP
[PROMPTS]
%UpdateProcedure PROCEDURE  (UpdatePhones)
%EditViaPopup LONG  (1)
       .
[INSTANCE]
INSTANCE 4
PARENT 3
       .
```

[ADDITION] marks the beginning of the subsection.

NAME identifies the template class (Clarion) and the specific template (BrowseUpdateButtons) invoked.

[FIELDPROMPT] indicates a prompt and its associated type and value. This is only generated if you use a #FIELD statement in your templates. The prompts begin on the following line. See [PROMPTS] above for a full discussion of prompt syntax.

[INSTANCE] introduces the INSTANCE number. On the following line, INSTANCE 2 indicates the instance number of this particular template addition.

PARENT 1 indicates this control template depends on another control template whose INSTANCE number is 1. That is, the BrowseUpdateButtons template only makes sense if there is also a BrowseBox template. Further, the BrowseUpdateButtons are associated with this particular BrowseBox whose INSTANCE number is 1. This is very important when there is more than one BrowseBox in the procedure.

PROCPROP means the prompts for this control template are shown in the **Procedure Properties** dialog. If PROCPROP is absent, the prompts will not be displayed in the **Procedure Properties** dialog.

[PROMPTS] marks the beginning of the list of prompts for this control template, and the values supplied by the developer for each prompt. The prompts begin on the following line and continue until the beginning of the next .TXA subsection. See the [PROMPTS] section above for a complete discussion of this subsection.

Index: