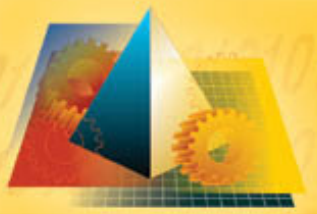


Clarion

Advanced Programming Resources

SoftVelocity Incorporated
2335 East Atlantic Blvd. Suite 410
Pompano Beach, Florida 33062
(954) 785-4555
www.softvelocity.com

This document supports Clarion 7

Printed in the United States of America (0409)

Contents:

.DCTX File Format 5
.DCTX Files: Clarion Dictionaries in XML Format 5
.DCTX XML Schema 5
.TXD File Format 9
.TXD Files: Clarion Dictionaries in ASCII Format 9
.TXD File Organization 10
.TXD Skeleton 10
.TXD File Sections 12
[DICTIONARY] 12
[FILES] 14
[ALIASES] 27
[RELATIONS] 28
Common Subsections 33
[LONGDESC] 33
[USEROPTION] 33
[TOOLOPTION] 33
[SCREENCONTROLS] 34
[REPORTCONTROLS] 34
.TXA File Format 35
.TXA Files: Clarion Applications in ASCII Format 35
.TXA File Organization 36
.TXA Skeleton 36
.TXA File Sections 39
[APPLICATION] 39
[PROJECT] 41
[PROGRAM]—[END] 42
[MODULE]—[END] 44
[PROCEDURE] 45
Common Subsections 48
[COMMON] 48
[DATA] 49
[FILES] 52
[PROMPTS] 55
[EMBED]—[END] 58
[ADDITION] 60
Index: 63

.DCTX File Format

.DCTX Files: Clarion Dictionaries in XML Format

XML is a standard format commonly used to represent data in a textual form.

Clarion 7 versions and greater now employ the use of the XML file format (.DCTX) to represent any Dictionary (.DCT) file.

The Dictionary Editor allows you to import a textual representation of a dictionary into the current active dictionary. You can import from a pre-C7 text source (TXD), or from the new C7 XML based format (DCTX).

You can also export the active dictionary contents to the C7 XML-based DCTX format.

.DCTX XML Schema

The following is the XML Schema used in the DCTX format:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="DataDictionary" xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
  <xs:element name="Audit">
    <xs:complexType>
      <xs:attribute name="CreateUser" type="xs:string" />
      <xs:attribute name="CreateDate" type="xs:string" />
      <xs:attribute name="CreateTime" type="xs:string" />
      <xs:attribute name="CreateVersionNumber" type="xs:string" />
      <xs:attribute name="ModifiedUser" type="xs:string" />
      <xs:attribute name="ModifiedDate" type="xs:string" />
      <xs:attribute name="ModifiedTime" type="xs:string" />
      <xs:attribute name="ModifiedVersionNumber" type="xs:string" />
    </xs:complexType>
  </xs:element>
  <xs:element name="Field">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="WindowControl" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Line" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:attribute name="Text" type="xs:string" />
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element ref="Audit" minOccurs="0" maxOccurs="unbounded" />
        <xs:element name="Validity" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:attribute name="Check" type="xs:string" />
            <xs:attribute name="Lookup" type="xs:string" />
          </xs:complexType>
        </xs:element>
        <xs:element ref="Field" minOccurs="0" maxOccurs="unbounded" />
      </xs:sequence>
      <xs:attribute name="Guid" type="xs:string" />
      <xs:attribute name="Name" type="xs:string" />
      <xs:attribute name="DataType" type="xs:string" />
      <xs:attribute name="Size" type="xs:string" />
      <xs:attribute name="ScreenPicture" type="xs:string" />
      <xs:attribute name="ScreenPrompt" type="xs:string" />
      <xs:attribute name="ReportHeading" type="xs:string" />
```

```
      <xs:attribute name="ReportControl" type="xs:string" />
      <xs:attribute name="Places" type="xs:string" />
      <xs:attribute name="Binary" type="xs:string" />
      <xs:attribute name="NoPopulate" type="xs:string" />
      <xs:attribute name="Over" type="xs:string" />
    </xs:complexType>
  </xs:element>
  <xs:element name="Dictionary">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="DictionaryVersion" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:attribute name="Version" type="xs:string" />
            <xs:attribute name="Description" type="xs:string" />
          </xs:complexType>
        </xs:element>
        <xs:element name="Table" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element ref="Audit" minOccurs="0" maxOccurs="unbounded" />
              <xs:element ref="Field" minOccurs="0" maxOccurs="unbounded" />
              <xs:element name="Key" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element ref="Audit" minOccurs="0" maxOccurs="unbounded" />
                    <xs:element name="Component" minOccurs="0" maxOccurs="unbounded">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element ref="Audit" minOccurs="0" maxOccurs="unbounded" />
                        </xs:sequence>
                        <xs:attribute name="Guid" type="xs:string" />
                        <xs:attribute name="FieldId" type="xs:string" />
                        <xs:attribute name="Order" type="xs:string" />
                        <xs:attribute name="Ascend" type="xs:string" />
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                  <xs:attribute name="Guid" type="xs:string" />
                  <xs:attribute name="Order" type="xs:string" />
                  <xs:attribute name="Name" type="xs:string" />
                  <xs:attribute name="KeyType" type="xs:string" />
                  <xs:attribute name="Unique" type="xs:string" />
                  <xs:attribute name="Primary" type="xs:string" />
                  <xs:attribute name="CaseSensitive" type="xs:string" />
                  <xs:attribute name="ExternalName" type="xs:string" />
                </xs:complexType>
              </xs:element>
              <xs:element name="Alias" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element ref="Audit" minOccurs="0" maxOccurs="unbounded" />
                  </xs:sequence>
                  <xs:attribute name="Guid" type="xs:string" />
                  <xs:attribute name="Name" type="xs:string" />
                  <xs:attribute name="Prefix" type="xs:string" />
                </xs:complexType>
              </xs:element>
            </xs:sequence>
            <xs:attribute name="Guid" type="xs:string" />
            <xs:attribute name="Name" type="xs:string" />
            <xs:attribute name="Prefix" type="xs:string" />
            <xs:attribute name="Driver" type="xs:string" />
            <xs:attribute name="Owner" type="xs:string" />
            <xs:attribute name="Path" type="xs:string" />
            <xs:attribute name="Thread" type="xs:string" />
          </xs:complexType>
        </xs:element>
        <xs:element name="Relation" minOccurs="0" maxOccurs="unbounded">
```

```
          <xs:complexType>
            <xs:sequence>
              <xs:element ref="Audit" minOccurs="0" maxOccurs="unbounded" />
              <xs:element name="ForeignMapping" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:attribute name="Guid" type="xs:string" />
                  <xs:attribute name="Field" type="xs:string" />
                </xs:complexType>
              </xs:element>
              <xs:element name="PrimaryMapping" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:attribute name="Guid" type="xs:string" />
                  <xs:attribute name="Field" type="xs:string" />
                </xs:complexType>
              </xs:element>
            </xs:sequence>
            <xs:attribute name="Guid" type="xs:string" />
            <xs:attribute name="PrimaryTable" type="xs:string" />
            <xs:attribute name="ForeignTable" type="xs:string" />
            <xs:attribute name="PrimaryKey" type="xs:string" />
            <xs:attribute name="ForeignKey" type="xs:string" />
            <xs:attribute name="PrimaryAlias" type="xs:string" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="Name" type="xs:string" />
      <xs:attribute name="Version" type="xs:string" />
      <xs:attribute name="DctxFormat" type="xs:string" />
    </xs:complexType>
  </xs:element>
  <xs:element name="DataDictionary" msdata:IsDataSet="true" msdata:UseCurrentLocale="true">
    <xs:complexType>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element ref="Audit" />
        <xs:element ref="Field" />
        <xs:element ref="Dictionary" />
      </xs:choice>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

.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.

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

Import	Because a .TXD file can be imported easily into Clarion, it is currently intended for backward compatibility only. Clarion now uses a new DCTX format, which is documented elsewhere in this document.
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 of a prior C7 version may exhibit some strange behavior in the development environment. Importing 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.

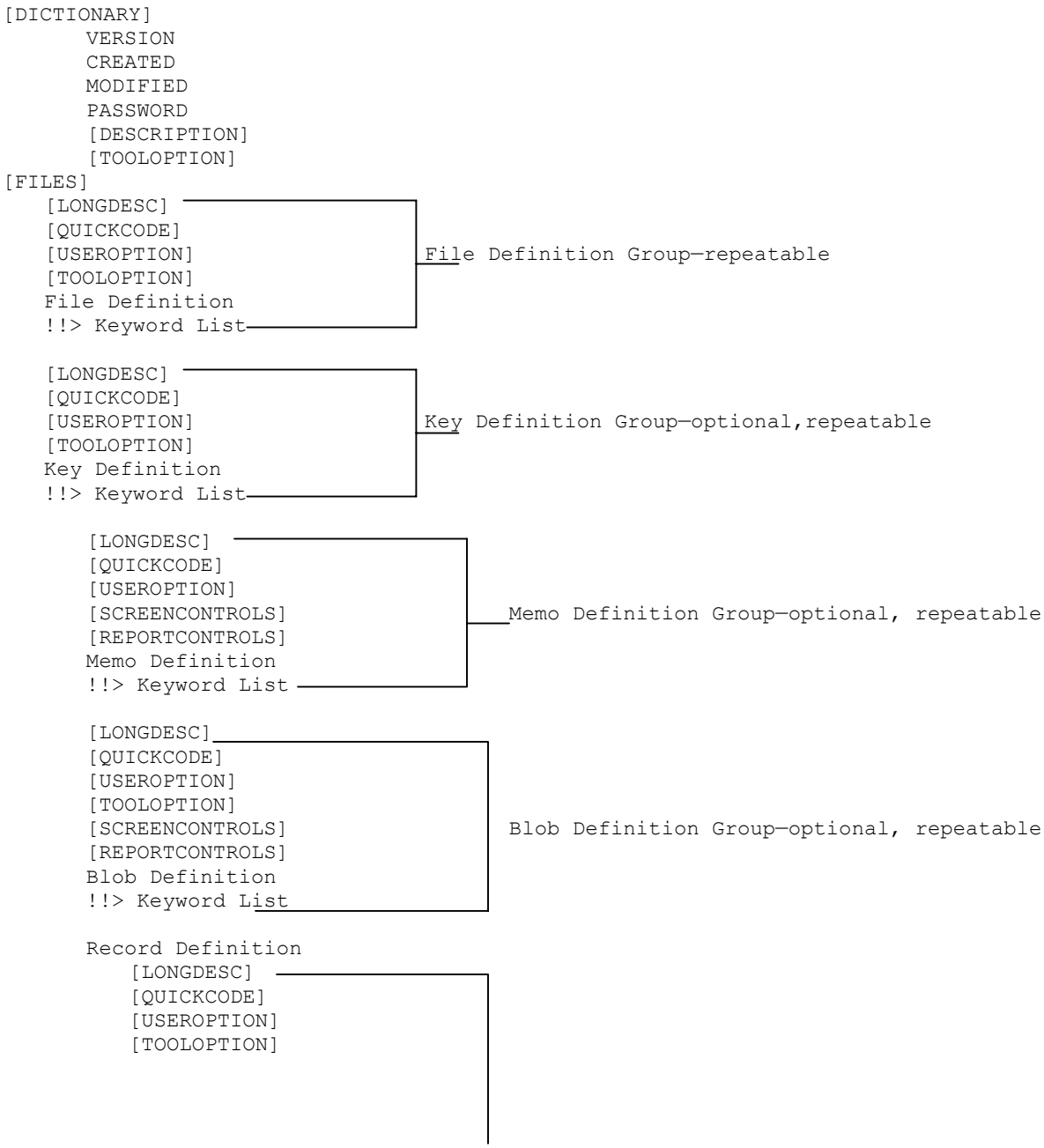
Each section begins with its title enclosed in square brackets and ends with the beginning of another section. Each section may contain subsections and keywords.

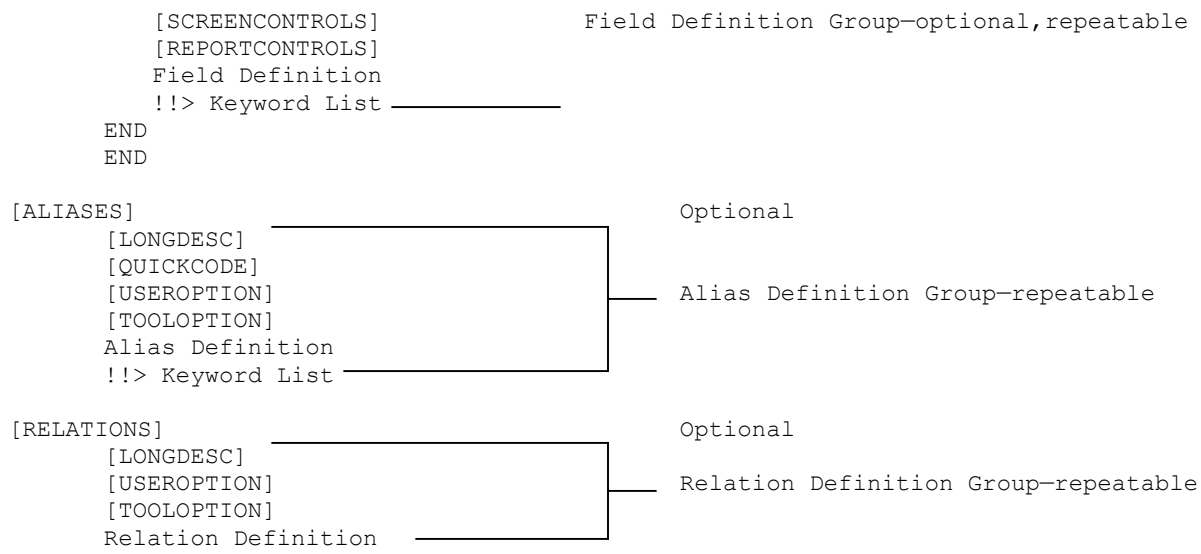
Subsections begin just like the major sections—with its title surrounded by square brackets.

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.




[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

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

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 (including the 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.

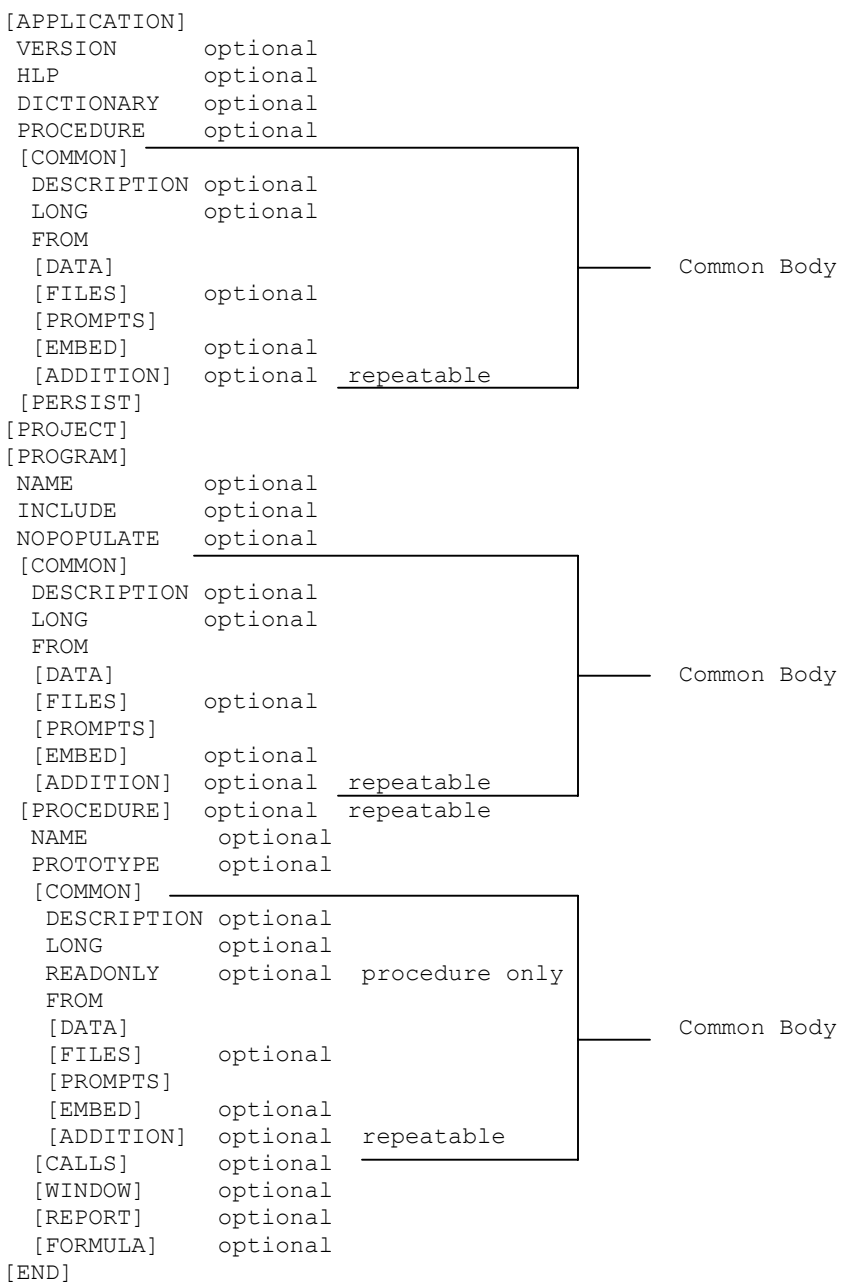
Each section begins with its title enclosed in square brackets and ends with [END], or with the beginning of another section. Each section may contain subsections and keywords.

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]                                  Common Body
  [FILES]       optional
  [PROMPTS]
  [EMBED]       optional
  [ADDITION]    optional  repeatable
 [PROCEDURE]    optional  repeatable

[END]
```

.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 [PROJECT] 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~Company
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:2)
  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]
```

ble—[PROCEDURE]

```
URE]
owseCustomers
PE 'LONG Count, REAL Sum'[CALLS]

.
.
.

ustomers
rders

ndow WINDOW('Browse Customers'),AT(,,358,188),SYSTEM,GRAY,MDI
T(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),|
S(CUS:CustNumber,CUS:CompanyName,CUS:Address,CUS:City,CUS:State,CUS:Zip)
N('&Insert'),AT(207,148,45,14),USE(?Insert:2),#SEQ(2),#ORIG(?Insert),#LINK(?Change:2)
N('&Change'),AT(256,148,45,14),USE(?Change:2),DEFAULT,#SEQ(2),#ORIG(?Change),#LINK(?Delete:2)
N('&Delete'),AT(305,148,45,14),USE(?Delete:2),#SEQ(2),#ORIG(?Delete),#LINK(?Insert:2)

A]
OrderTax
OrderTax
fter Lookups
TION Calculate State Tax
Total * 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} {MULTI}	@picture LONG REAL STRING FILE FIELD KEY COMPONENT DEFAULT	TIMES n
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:

Common TXA Subsections 48
TXA File Format 35
TXA File Organization 36
TXA File Sections 39
TXA Skeleton 36
TXD Common Subsections 33
TXD File Format 9
TXD File Organization 10
TXD File Sections 12
TXD Skeleton 10