L3 TECHNOLOGIES

# Meta Properties 2.0

# CONTENTS

## Introduction

Meta Properties are a set of properties that can be used to determine the features of a File Driver, or to determine or define the structure of a file at run-time. Meta Properties facilitate the development of applications that can load and manipulate any file without the need to hard-code the file definition. Developers can use Meta Properties to produce applications where the user can decide the structure of the data they want to store. The Clarion language can now be used to develop end-user databases.

There are several categories of Meta Properties:

- Driver Properties are all read-only and provide a means to determine the capabilities of the driver being used to access a file.
- File Properties are used to determine the overall characteristics of a file. Some properties are used to access arrays of key, memo or field descriptors.
- Key Properties are used to determine the characteristics of a files keys and/or indexes. Some properties are used to access arrays of key components.
- Memo Properties are used to determine the characteristics of a files' memo or BLOB fields.
- Record Properties are used to access the files record buffer.
- Field properties are used to determine the characteristics of a files' fields, and to access their values.

These properties work in conjunction with a small number of SEND() commands to provide a means to create or load and access file structures at run-time. The Meta Property values and associated declarations are in METAPROP.INC.

Meta Properties are implemented within an extension to the TopSpeed File Driver Kit and are used to provide dynamic file access facilities within all File Drivers developed by L3 Technologies. The Meta Properties Extension is available in source code form as a 'drop-in' extension to any file drivers developed with the TopSpeed File Driver Kit.

## Meta Property Usage

There are two major categories of usage for Meta Properties:

- Loading a file, determining its' structure and accessing it's data.
- Creating a new file structure.

In all cases it is necessary to declare a file on which to operate. This file is known as the Meta File. The processes by which the Meta File may be used to load or create files are described below.

### Declaring the Meta File

The Meta File may be used for loading or creating any file structure. It has no structure itself, it must be declared with an empty RECORD. It is used simply as a structure with which to communicate with a driver. The Meta File should be declared as follows:

```
Meta    FILE,DRIVER('Driver')
Key       INDEX()              ! Omit if keys are not used
Memo      BLOB()               ! Omit if memos and blobs are not
used
          RECORD
          END
        END
```

The 'Driver' string should be replaced with the name of the File Driver in use. The 'Key' element is known as the Meta Key and is used where a key label must be passed to a file command. Similarly the 'Memo' element is known as the Meta Memo and is used to reference Memos and Blobs.

### Loading a File

Before a file can be opened using the Meta File, the file must be loaded. This process invokes the File Driver to import the file structure. When the file has been loaded, the internal data structures for the Meta File will have been set up to represent the structure of the file.

Before a file can be loaded, its' name and path must be specified. This can be done in a number of ways. If a name is not specified, the file name defaults to the label of the Meta File. The name may also be specified in the usual way using the NAME() attribute. Finally, the name may be specified using the File:Path property.

Loading the file takes just two lines of code:

```
  Meta{File:Path} = FullPath   ! Just one way of specifying the path
  SEND(Meta, 'LOADFILE')       ! Load the file structure
```

At this point the Meta File can be used to determine the structure of the file identified by FullPath, and to access it's data.

### Determining the File Structure

The file structure can be determined by querying Meta Properties of the Meta File. The following code fragment to print the field definitions, illustrates this process:

```
  !*** Print the field definitions
  LOOP i# = 1 TO Meta{File:Fields}
```

```
    Meta{File:Field} = i#
    FieldName   = Meta{Field:Name}
    FieldLabel  = Meta{Field:Label}

    CASE Meta{Field:Type}
    OF Type:LONG
      FieldType = 'LONG'
    OF Type:REAL
      FieldType = 'REAL'
    OF Type:DECIMAL
      FieldType =
'DECIMAL('&Meta{Field:Digits}&','&Meta{Field:Places}&')'
    OF Type:STRING
      FieldType = 'STRING('&Meta{Field:Size}&')'
    ELSE
      FieldType = 'UNKNOWN'
    END

    Output(FieldLabel, FieldType, FieldName)
  END
```

The loop header determines the total number of fields in the record using the file property File:Fields.

The first line of the loop sets the current (or target) field which subsequent field properties will refer to. There is an important observation to be made at this point. The fields in the record structure are treated internally as an array of field descriptors. It would seem natural to access fields using array syntax, however this would preclude using array syntax to access dimensioned fields. Consequently array syntax is reserved for accessing dimensioned fields, and in order to access a field, memo or key descriptor it is first necessary to specify the index of the target descriptor. Note that indexes are 1 based.

The Field:Name property is used to determine the name of the target field. This is the external name of the field. Certain File Drivers may return field names that are augmented to include additional type information.

The Field:Label property is used to determine the label of the target field. The label is a legal Clarion label, omitting any augmentation applied to the field's external name.

The File:Type property is used to determine the Clarion data type for the field. This is returned as a coded value which is matched using the Type:??? equates in METAPROP.INC.

Note that decimals and strings have additional properties, Field:Digits, Field:Places and Field:Size respectively. The Field:Size property is available for all types and returns the number of bytes of storage required by the field.

All other file structures are accessed in much the same way.

## Accessing the Data

At this point accessing the data will seem relatively straightforward, as illustrated in the following code fragment:

```
  !*** Print the field values
  LOOP i# = 1 TO Meta{File:Fields}
    Meta{File:Field} = i#

    IF Meta{Field:Dim} = 1 THEN
```

```
      FieldValue = Meta{Field:Value}
    ELSE
      FieldValue    = ''
      j#            = 1

      LOOP
        FieldValue = CLIP(FieldValue) & Meta{Field:Value, j#}

        j# += 1
        IF j# <= Meta{Field:Dim} THEN
          FieldValue = CLIP(FieldValue) & ','
        ELSE
          BREAK
        END
      END
    END
    Output(FieldValue)
  END
```

The loop header and target field selection appear as above.

The Field:Dim property returns the dimension of the field. Note that Clarion handles dimensioned fields as one-dimensional arrays internally.

The Field:Value property returns the value of the field. This is returned as a string but is formatted so that it may be converted to the appropriate type. The Field:Value property is an array property, it's index ranges from 1 to the value of the Field:Dim property. The index is used to reference an element in a dimensioned field. If no index is specified then the first element is returned.

## Unloading the File

When the Meta File is no longer in use, it is necessary to ensure that the resources allocated to the file are released. This is achieved using the UNFIXFILE command:

```
  SEND(Meta, 'UNFIXFILE')
```

This will release any resources that are not normally released when the file is closed.

## Creating a New File Structure

The Meta File may be used to dynamically create files of any legal Clarion structure. The process of creating a file involves defining the file, field, memo, and key characteristics, then 'fixing' the file so that it can be created. This process is illustrated in the following code fragment:

```
  !*** Create the following file structure
  ! F1                FILE,DRIVER('Driver'),CREATE
  ! PKey                KEY(F1:I), PRIMARY
  !                     RECORD
  ! I                     SHORT
  ! S                     STRING(10)
  !                     END
  !                   END

  !*** Set file attributes
  Meta{File:Path} = 'F1'
  Meta{File:Create} = TRUE

  !*** Set up fields
  Meta{File:Field} = 1
  Meta{Field:Name} = 'I'
  Meta{Field:Type} = Type:SHORT

  Meta{File:Field} = 2
  Meta{Field:Name} = 'S'
  Meta{Field:Type} = Type:STRING
  Meta{Field:Size} = 10

  !*** Set up keys
  Meta{File:Key} = 1
  Meta{Key:Name} = 'PKey'
  Meta{Key:Primary} = TRUE
  Meta{Key:Component} = 1
  Meta{Key:Field} = 1

  !*** Fix and create the file
  SEND(Meta, 'FIXFILE')
  IF ERRORCODE() THEN
    MESSAGE('Error fixing file: ' & ERROR())
  ELSE
    ChkCreate(Meta)
    SEND(Meta, 'UNFIXFILE')
  END
```

The Key:Component property sets the target component for the key. The Key:Field property then determines the field number for the component. Note that fields that are components in a key, must be defined before the key can be defined.

## Meta Keys

A Meta Key is declared using the same syntax as a dynamic index.  The Meta Key is bound to a dynamically created key and passed to file commands that take a key parameter.  Given the above Meta File definition, the following illustrates how to process the file in keyed order, using the Meta Key to specify the key on which to order:

```
  !*** Process the file ordered by the first key
  Meta{File:MetaKey} = 1

  SET(Meta:Key)

  LOOP
    NEXT(Meta)
    IF ERRORCODE() THEN
      BREAK
    END
    !*** Fetch field values as illustrated above
  END
```

Note that if the MetaKey has not been bound, then a 'Bad Key File' error is posted when the key is used.

Using Meta Properties v2.0 under Clarion 4, it is possible to reference Key Meta Properties via a key:

```
  !*** Obtain the label of the key
  KeyLabel = Key{Key:Label}
```

## Meta Command Definitions

Meta Commands are used to invoke specific processes within the File Driver. The following details the Meta commands and their usage.

### FIXFILE

The FIXFILE command causes the driver to process the file structure internally to determine that it represents a legal Clarion structure. The FIXFILE command must be issued once the file structure has been set up using file properties, and before the file is created or opened.

### LOADFILE

The LOADFILE command causes the driver to import the file structure from a file on disk. The file structure does not need to be fixed if it loaded in this way.

### UNFIXFILE

The UNFIXFILE command releases resources used internally to maintain the Meta File. It should be called when the file is no longer needed.

Currently, the Meta File is automatically 'unfixed' after a call to CLOSE(), COPY(), CREATE(), REMOVE() or RENAME(). This means that if the file is to be accessed after a call to one of these functions, it is necessary to re-specify the file structure.

## Meta Property Definitions

The following details the Meta Properties and indicates whether they are readable (R) and/or writeable (W). Many Meta Properties may be used to determine properties of non-Meta files, these are marked as R+ and/or W+ as appropriate. When updating properties for non-Meta files it is necessary to ensure that the file representation is not made inconsistent with the disk file.

### Driver Properties

All driver properties are read-only, they return information about the File Driver that is being used to access the file.

#### Driver:Name (R+)

The Driver:Name property returns a string, no more than 20 characters in length, containing the name of the File Driver. This is the string that appears in a file's DRIVER() attribute.

#### Driver:DllName (R+)

The Driver:DllName property returns a string, no more than 12 characters in length, containing the name of the File Driver .DLL.

#### Driver:CopyRight (R+)

The Driver:CopyRight property returns a string, no more than 40 characters in length, containing the copyright notice for the File Driver.

#### Driver:Description (R+)

The Driver:Description property returns a string, no more than 30 characters in length, containing the full description of the File Driver.

#### Driver:Create (R+)

The Driver:Create property returns a BOOL which is TRUE if the File Driver supports the CREATE() command, FALSE otherwise.

#### Driver:Owner (R+)

The Driver:Owner property returns a BOOL which is TRUE if the File Driver supports the OWNER attribute for files, FALSE otherwise.

#### Driver:Encrypt (R+)

The Driver:Encrypt property returns a BOOL which is TRUE if the File Driver supports the ENCRYPT attribute for files, FALSE otherwise.

#### Driver:Reclaim (R+)

The Driver:Reclaim property returns a BOOL which is TRUE if the File Driver supports the RECLAIM attribute for files, FALSE otherwise.

#### Driver:Keys (R+)

The Driver:Keys property returns a BOOL which is TRUE if the File Driver supports keys, FALSE otherwise.

#### Driver:KeyOrder (R+)

The Driver:KeyOrder property returns an integer which has the following values:

| | |
|---|---|
| 0 | Driver supports ascending keys only |
| 1 | Driver supports keys with ascending and descending components |

| | |
|---|---|
| 2 | Driver supports descending keys |

**Driver:KeyDup (R+)**

The Driver:KeyDup property returns a BOOL which is TRUE if the File Driver supports the DUP attribute for keys, FALSE otherwise.

**Driver:KeyCase (R+)**

The Driver:KeyCase property returns a BOOL which is TRUE if the File Driver supports the NOCASE attribute for keys, FALSE otherwise.

**Driver:KeyOpt (R+)**

The Driver:KeyOpt property returns a BOOL which is TRUE if the File Driver supports the OPT attribute for keys, FALSE otherwise.

**Driver:Indexes (R+)**

The Driver:Indexes property returns a BOOL which is TRUE if the File Driver supports indexes, FALSE otherwise.

**Driver:IndexOrder (R+)**

The Driver:IndexOrder property returns an integer which has the following values:

| | |
|---|---|
| 0 | Driver supports ascending indexes only |
| 1 | Driver supports indexes with ascending and descending components |
| 2 | Driver supports descending indexes |

**Driver:IndexCase (R+)**

The Driver:IndexCase property returns a BOOL which is TRUE if the File Driver supports the NOCASE attribute  for indexes, FALSE otherwise

**Driver:IndexOpt (R+)**

The Driver:IndexOpt property returns a BOOL which is TRUE if the File Driver supports the OPT attribute for indexes, FALSE otherwise.

**Driver:Dynamic (R+)**

The Driver:Dynamic property returns a BOOL which is TRUE if the File Driver supports dynamic indexes, FALSE otherwise.

**Driver:DynOrder (R+)**

The Driver:DynOrder property returns an integer which has the following values:

| | |
|---|---|
| 1 | Driver supports ascending dynamic indexes only |
| 2 | Driver supports dynamic indexes with ascending and descending components |
| 3 | Driver supports descending dynamic indexes |

**Driver:DynCase (R+)**

The Driver:DynCase property returns a BOOL which is TRUE if the File Driver supports the NOCASE attribute for dynamic indexes, FALSE otherwise.

**Driver:DynOpt (R+)**

The Driver:DynOpt property returns a BOOL which is TRUE if the File Driver supports the OPT attribute for dynamic indexes, FALSE otherwise.

**Driver:Memos (R+)**

The Driver:Memos property returns a BOOL which is TRUE if the File Driver supports memos or BLOBs, FALSE otherwise.

**Driver:MemoBinary (R+)**

The Driver:MemoBinary property returns a BOOL which is TRUE if the File Driver supports the BINARY attribute on memos, FALSE otherwise.

**Driver:MemoSize (R+)**

The Driver:MemoSize property returns a USHORT containing the maximum size for a memo supported by the File Driver.

**Driver:Pipe (R+)**

The Driver:Pipe property returns a LONG containing the address of the File Driver's entry point (also called the pipe function).

**Driver:StringSize (R+)**

The Driver:StringSize property returns a USHORT containing the maximum size for a STRING field supported by the File Driver.

**Driver:CStringSize (R+)**

The Driver:CStringSize property returns a USHORT containing the maximum size for a CSTRING field supported by the File Driver.

**Driver:RecordSize (R+)**

The Driver:RecordSize property returns a USHORT which contains the maximum record size supported by the File Driver.

**Driver:Dim (R+)**

The Driver:Dim property returns a SHORT containing the maximum number of field dimensions supported by the File Driver.

**Driver:IsSQL (R+)**

The Driver:IsSQL property returns a BOOL which is TRUE if the File Driver is an SQL File Driver, FALSE otherwise.

**Driver:Type (R+)**

The Driver:Type property returns a BOOL which is TRUE if the File Driver supports the specified data type, FALSE otherwise.  The values that specify the File Driver commands are defined in METAPROP.INC and are prefixed 'Type:'.

**Driver:Proc (R+)**

The Driver:Proc property returns a BOOL which is TRUE if the File Driver implements the specified file command, FALSE otherwise.  The values that specify the File Driver commands are defined in METAPROP.INC and are prefixed 'Proc:'.

## File Properties

All file properties are readable, most may be set until the file structure is fixed. File properties return information about the file.

**File:Create (R+ W+)**

Set the File:Create property to TRUE to specify that a file may be created using the CREATE() function, this is similar to specifying the CREATE attribute on a non-Meta file declaration.  The property may be set for a non-Meta file only if it has been declared BINDABLE.

**File:DriverString (R+ W+)**

The File:DriverString property specifies the driver string for a file.  The driver string is the second parameter to the DRIVER() attribute on a file.

**File:Encrypt (R+ W+)**

Set the File:Encrypt property to TRUE to specify that the file must be encrypted.

**File:Field (R+ W+)**

The File:Field property specifies the target (current) field to which subsequent field properties will refer.

**File:Fields (R+)**

The File:Fields property returns the number of fields in the files record.

**File:Key (R+ W+)**

The File:Key property specifies the target (current) key to which subsequent key properties will refer.

**File:Keys (R+)**

The File:Keys property returns the number of keys for the file.

**File:Label (R+ W+)**

The File:Label property specifies the label for a file.  The label may be different from the name that will specify the external name for the file.

**File:Memo (R+ W+)**

The File:Memo property specifies the target (current) memo to which subsequent memo properties will refer.

**File:Memos (R+)**

The File:Memos property returns the number of memos and BLOBs for the file.

**File:MetaKey (W)**

The File:MetaKey property specifies the key which will be bound to the MetaKey.

**File:MetaMemo (W)**

The File:MetaMemo property specifies the memo or blob which will be bound to the MetaMemo.

**File:Path (R+ W+)**

The File:Path property specifies the external name for a file.  In the absence of a name, the external name is derived from the label.

**File:Oem (R+ W+)**

Set the File:Oem property to TRUE to specify that the File Driver must perform OEM character conversion for the file.

### File:Owner (R+ W+)

The File:Owner property specifies the value of the OWNER() attribute for a file. This is the password used for data encryption.

### File:Reclaim (R+ W+)

Set the File:Reclaim property to TRUE to specify that the File Driver should reclaim unused space in a file.

### File:RecordSize (R+)

The File:RecordSize property returns the size of the file's record buffer. This property will return 0 unless the file structure has been 'fixed'.

## Key Properties

All key properties are readable, most may be set until the file structure is fixed. Key properties operate on the target key selected using the File:Key property or on the Meta Key selected by the File:MetaKey property. Under Clarion 4, key properties may also be selected via the key itself.

### Key:Component (R+ W+)

The Key:Component property specifies the target (current) component. The field referenced by the component is accessed using the Key:Field property.

### Key:Components (R+)

The Key:Components property returns the number of components in the key.

### Key:Dup (R+ W+)

Set the Key:Dup property to TRUE to specify that the key allows duplicate entries.

### Key:Dynamic (R+ W+)

Set the Key:Dynamic property to TRUE to specify that the key is a dynamic index that has no components.

### Key:Field (R+ W+)

The Key:Field property specifies the field referenced by the target component.

### Key:Index (R+ W+)

Set the Key:Index property to TRUE to specify that the key is an index.

### Key:Label (R+ W+)

The Key:Label property specifies the label of the key. This is different from the value of Key:Name which is the external name for the key. However if no value is specified for Key:Name, the external name is derived from the label.

### Key:Name (R+ W+)

The Key:Name property specifies the external name for the key. However if no value is specified for Key:Name, the external name is derived from the label.

### Key:NoCase (R+ W+)

Set the Key:NoCase property to TRUE to specify that the key is case-insensitive.

### Key:Opt (R+ W+)

Set the Key:Opt property to TRUE to exclude null entries from the key or index file.

#### Key:Ord (R+ W+)

Set the Key:Ord property to TRUE to specify that the target component in the key is to be sorted in descending order.  The sort order options for the driver may be determined using the Driver:KeyOrder, Driver:IndexOrder and Driver:DynOrder properties.

#### Key:Primary (R+ W+)

Set the Key:Primary property to TRUE to specify that the key is the primary key.

### Memo Properties

All memo properties are readable, most may be set until the file structure is fixed. Memo properties operate on the target memo selected using the File:Memo property or on the Meta Memo selected by the File:MetaMemo property.  Under Clarion 4, memo properties may also be selected via the memo itself.

#### Memo:Label (R+ W+)

The Memo:Label property specifies the label of the memo.  This is different from the value of Memo:Name which is the external name for the memo.  However if no value is specified for Memo:Name, the external name is derived from the label.

#### Memo:Name (R+ W+)

The Memo:Name property specifies the external name for the memo.  However if no value is specified for Memo:Name, the external name is derived from the label.

#### Memo:Size (R+ W+)

The Memo:Size property specifies the maximum size of the memo.  Blobs have a size of 0.

#### Memo:Type (R+ W+)

The Memo:Type property is used to specify the type of the memo.  The following values may be used:

| | |
|---|---|
| 1 | The memo is a text MEMO |
| 2 | The memo is a binary MEMO |
| 4 | The memo is a BLOB |

#### Memo:Value (R+ W+)

The Memo:Value property is used to reference the contents of the memo buffer.

### Record Properties

All record properties are readable, Record:Value may only be written when the file is fixed

#### Record:Value (R+ W+)

The Record:Value property is used to reference the entire contents of the files record buffer.  The data is returned unformatted, whereas referencing the fields individually returns data formatted for appropriate conversion.

#### Record:Size (R+)

The Record:Size property returns the size of the files record buffer.

### Field Properties

All field properties are readable, most may be set until the file structure is fixed. Field properties operate on the target field selected using the File:Field property.

**Field:Digits (R+ W)**

The Field:Digits property specifies the total number of digits in a DECIMAL field. A field of type DECIMAL(5,2) has Field:Digits=5 and Field:Places=2.

**Field:Dim (R+ W)**

The Field:Dim property specifies the number of elements in the field. Note that although Clarion allows multi-dimensional fields, the driver will treat the field as an array whose number of elements is the product of the dimensions.

**Field:Label (R+ W+)**

The Field:Label property specifies the label of the field. This is different from the value of Field:Name which is the external name for the field. However if no value is specified for Field:Name, the external name is derived from the label.

**Field:Name (R+ W+)**

The Field:Name property specifies the external name for the field. However if no value is specified for Field:Name, the external name is derived from the label.

**Field:Offset (R+)**

The Field:Offset property specifies the offset of the field within the record buffer. This does not usually need to be set since the offset of the field can be determined from the types of prior fields.

**Field:Over (R+ W)**

Set the Field:Over property to the number of a prior field that the field will map over.

**Field:Picture (R+ W)**

Set the Field:Picture property to a Clarion picture token for the field.

**Field:Places (R+ W)**

The Field:Places property specifies the total number of digits after the point in a DECIMAL field. A field of type DECIMAL(5,2) has Field:Digits=5 and Field:Places=2.

**Field:Size (R+ W)**

The Field:Size property specifies the size in bytes of the field. It is only necessary to specify the size of STRING, CSTRING or PSTRING fields, though the the Field:Size property will return the size of any field type.

**Field:Type (R+ W)**

The Field:Type property specifies the data type of the field. Use the Type:??? equates in METAPROP.INC to specify field types.

**Field:Value (R+ W+)**

The Field:Value property is used to reference the value of the field. The data is returned as a string formatted for appropriate conversion.

## Changes in Version 2.0

There have been some minor changes in Meta Properies v2.0 to accommodate new functionality in Clarion 4. The changes include:

- The File:Name property is replaced by File:Path to avoid a conflict.
- The File:Size property is replaced by File:RecordSize to avoid a conflict.
- It is now possible, under Clarion 4, to select key properties via the Meta Key or any other key: Key{Key:Label}.
- It is now possible, under Clarion 4, to select memo properties via the Meta Memo or any other memo: Memo{Memo:Label}.

## Limitations

Currently, the following limitations exist which will be addressed in a future version of the Meta Properties:

- There is no mechanism for defining GROUP structures within files.
- There is no means of binding fields in the file.