Clipper to Max Migration Guide 5

Max Additional Features 47

Migrating to the Web 57

Clipper to Max Migration Guide

Clipper Migration Guide Version 2.00 2001-03-22a

About This Document

This guide will introduce you to Max, the evolution of Xbase. Using a well-understood implementation as a point of reference, we identify differences and improvements. If you are an Xbase programmer, this document will assist you as you assess your options.

This document aids Xbase programmers to:

- move program code from Summer ’87 and CA-Clipper 5.x applications to Max as quickly and smoothly as possible
- to aid programmers in understanding how Max has evolved the Xbase language and behaviors

This document is organized into two major sections:

- **Implementation Differences**- This section provides essential information to successfully run legacy Clipper code. The focus is making you a successful Max developer. Following our tips, you can quickly compile and run code you already have.
- **Extended Features**- Use this section to enhance your applications, to start new development, and to expand your understanding of what Max can do.

Why Max is the Evolution of Xbase

With years of Xbase experience, we wanted Max to be a little smarter.

As a simple example, dBASE II was written to minimize typing for people who had to work interactively at a dot prompt. So its designers avoided making the user type quotes when opening a file with the **use** command. Max allows you to use this "shortcut" notation, since we know it is your habit. But if you want greater precision and language consistency, Max stands with you.

Because you are a professional, we designed Max to be more sensible. Max offers you more formality if you choose to use it. And more functionality if you want to take advantage of the power. For more on how we’ve expanded upon the Xbase promise, see **Using Extended Mode** on page 17. This section gives you a broad overview at features we think you will want to exploit. But for a full briefing, be sure to consult the Max Developer’s Overview and the Max Language Reference.

Before expanding your Xbase talents, you probably want to know where Max differs from the Xbase environment you know best. The following section shows how to get your code running with a minimum of recoding.

•

Implementation Differences

Functions: Dummy Arguments

In some cases, Clipper supports the use of the logical data type placeholder arguments when you want to pass default values on optional parameters. In many cases, this logical value is holding the place for numeric or character types. Max maintains stricter type checking, so placeholders will generate errors if the data type expected does not match the placeholder's type. Max supports empty parameters.

Solution: Remove Clipper placeholder values and use empty parameters instead.

Benefits:

- Empty parameters clearly document where you depend upon the default value for a parameter
- Max's strict type checking enables the compiler to identify data type errors. .

Clipper	`dbedit( .t., .t., .t., .t., mFields )`
Max	`dbedit( ,,,,, mFields )`

File Formats*

	Clipper	Max
Memory Files (.MEM)	Files written by Max may not always be successfully read by Clipper or other Xbase products.	Files generated by other Xbase products are automatically recognized and read by Max, but are written in Max format.
Index files	.NTX files	.MTX files are a Max format. (Write code to regenerate your indexes or use the KEY clause IN **`USE`** or **`SET INDEX TO`**.)

* Future Max versions may support ODBC access to other popular Xbase file formats.

Preprocessor Directives

Max implements key preprocessor directives:

Directive	Meaning
#define	Define a manifest constant or pseudofunction
#else	Code to be used if #ifdef or #ifndef is false
#endif	Terminator for #ifdef and #ifndef directives
#ifdef	Conditionally include code if identifier exists
#ifndef	Conditionally include code if identifier does not exist
#include	Include a file into the current source
#undef	Removes a #defined identifier

Preprocessor: #command, #translate

- Unlike CA-Clipper 5.x, the Max preprocessor does not impose case sensitivity.
- Pseudofunctions- Max will accept function name substitutions but does not perform argument queueing.

Some Clipper 5.x preprocessor features are not implemented. For more information, see **Preprocessor: #command, #translate** on page 9.

Indexes

Filename Extension

Because the default Max index file extension in Max is .MTX, your legacy code will probably refer to the wrong file extension.

Example:

```
if .not. file("budget_partno.ntx") && not the Max extension
                                   && for an index
 && generates .mtx file
 index on budget->partno to budget_partno
endif
USE budget INDEX budget_partno
```

Solution: Inspect your source code for locations where your code tests for or specifies .NTX extensions. Replace all explicit .NTX references with .MTX.

Example:

```
USE budget INDEX budget_partno KEY budget->partno
```

Data Type for Index Key Expressions

Issue: Clipper allows you to specify an expression of any data type when an index key consists of a single element (not compound) expression. Max only supports character data types for index key expressions.

Example:

```
Field         Type   Size    Dec
ITEMCOUNT      N       5      0
index on itemcount to ndxcode  && this would be a numeric key
```

Solution: Use a data type conversion function if you specify a non-character field as a key expression.

Clipper	`INDEX on itemcount to ndxcode`
Max	`INDEX on STR(itemcount,5) to ndxcode`

Note: Max, Clipper, dBASE and FoxPro all require you to convert all elements to character strings when an index expression consists of a compound key.

Using The SEEK Command Against A Non-Character Field

Issue: Once you have generated an index by converting a non-character expression to a character data type, you must remember that SEEK (and FIND) will only succeed if you first convert a numeric expression to character. If the SEEK command does not have a character expression to work with, the result will always be a seek failure (EOF() = .T.)

Solution: Convert the seek expression to character:

```
var = 123
SEEK str(Var,12)
```

Unique Indexes Provide Integrity Protection

Clipper allows indexes to be UNIQUE, but it does not guarantee the uniqueness of keys. When you generate an index with the UNIQUE attribute in Max, Max will not insert duplicate index keys. In such cases, **REPLACE** fails. Then use **UNIQUEVIOL()**:

```
* REPLACE indexedField WITH duplicatedValue
if uniqueviol()
    ? "The uniqueness of this key would be violated."
    inkey(3)
    loop
endif
```

See the Max Language Reference for details.

Conflicting User Defined Names

Functions: User Defined Conflicting With Internal

Issue: Clipper permits you to write user defined functions with names that conflict with its own library of functions.

Solution: Use unique function names or provide a prefix:.

Clipper	`function seek&& acceptable`
	`parameters CustomerName`
	`return( CustomerName )`

Max	`function seek&& Max generates error`
	`parameters CustomerName`
	`return( CustomerName )`
	
	`function MySeek&& acceptable`
	`parameters CustomerName`
	`return( CustomerName )`

Solution: Prefix your UDF name to make it distinctive or use some other unique name.

Benefits:

- Max will not allow you to write function names that conflict with internal function names
- You cannot accidentally disable an internal function by naming one of yours identically.

Variables: Conflicting With Reserved Words

Issue: Max accepts reserved words as variable names.But if you use that variable on a command line that also uses that same name as a reserved word, Max will generate an error.

Solution:

- Enclose the variable reference with parentheses. This creates an expression context; within expressions there are no reserved words.

Example:

```
CURSOR = .T.
Set CURSOR CURSOR       && Max generates error
Set CURSOR (CURSOR)     && This is acceptable
```

Multidimensional Arrays

Max suports multidimensional arrays with an approach that is more consistent with other conventional programming languages. Max does not support the CA-Clipper 5.x convention of "ragged arrays" with cells pointing to other cells. (See **Multidimensional Arrays** on page 27.)

Expanded Color Handling

In almost every respect, Max allows you to specify color strings identically to Clipper, and dBASE.

In some language elements, Max uses the fourth segment of the color string, normally reserved for the “unselected color.” (For example, a field that is not currently visited by the screen cursor.)

Example:

```
set color to w/n, ;       && 1st color: Normal
              b/w, ;      && 2nd color: highlighted
              n,   ;      && 3rd color: border
              i           && 4th color: unselected
```

Max extends functionality in these commands and functions:

Command/Function	4th color segment is used for the following purpose:
@ ... GET	Color of unselected fields (pending GET)
@ ... PROMPT	Unselected menu choice. Also colors the message line displayed by SET MESSAGE
ACCEPT	Color of data after the user submits the value
INPUT	Color of data after the user submits the value
BROWSE	Color of currently selected field
dbedit( )	Color of currently selected field
EDIT MEMO	Color of the marked block
memoedit( )	Color of the marked block
achoice( )	Color of unselected items

Tip: If you prefer a single color setting, use the same color specification in the first and fourth segments of the color string.

Note: The **COLOR** clause is available in all user interface commands, allowing you to specify colors within each user interface object. The **COLOR** clause may be used with:

?	**??**	**PRINT**	**@**
ACCEPT	**LIST**	**CLEAR**	**SET SCOREBOARD**
TYPE	**WAIT**		

FILE()

Issue: Max does not support filemasks as parameters to the FILE( ) function.

Example:

```
FILE( "*.dbf" )          && Does not work with Max
```

Solution: Use the ADIR() function as an alternative:

```
ADIR( "*.dbf" ) != 0     && Use this instruction instead
```

Compiling A Tree of Files

Max supports "tree" compilation just like Clipper does. In Max, when you use the DO command, do not specify file extensions if you depend upon tree compilation to include all code for your application. Max automatically identifies the relevant file with a .PRG extension:.

Clipper	`DO first`
	
	`*first.prg`
	`DO second.prg`

Max	`DO first`
	
	`*first.prg`
	`DO second`

Miscellaneous

Line continuation when calling user defined functions

- As in Clipper, dBASE and FoxPro, Max uses the semicolon as a line continuation character allowing you to spread code to the following line. If you break a call to a function, do not place the continuation character before the opening parenthesis:

Max Generates Error	`myValue = ValidateFields ;` `( .T., "Test" )`
Acceptable	`myValue = ValidateFields( ;` `.T., "Test" )`

SET KEY <keyname> TO <function(withParameter)>

Issue: Clipper and Max permit you to specify parameterless user defined functions when binding a routine to a key on the keyboard. If you specify parameters anyway, Clipper nor Max can process them. Clipper and Max behave differently under this misuse of the language

Example:

```
Set Key -1 to UserFunction( date() )
```

Clipper	Max
Ignores parameter and runs the function when user presses the specified key.	Generates an error.

Solution: Remove any parameters. If necessary, recode the function to perform correctly when no user-defined parameters are available.

Using achoice() With lastkey()

Issue: In Clipper, the value of **`lastkey()`** is always reset to 0 after the function is called. Max preserves the **`lastkey()`** value so it can be retrieved in subsequent calls. If you are migrating Clipper code, your calls to **`achoice()`** may expect lastkey() to return 0.

Solution: Reset **`lastkey()`** value by calling **`lastkey(0`**) immediately before calling **`achoice()`**.

Status Codes: achoice(), dbedit(), memoedit()

achoice( ) **dbedit( )** **memoedit()**	• **`achoice()`** passes 5 status codes • **`dbedit()`** passes 5 status codes • **`memoedit()`** passes 5 status codes	• **`achoice()`** passes 7 status codes. • **`dbedit()`** passes 8 status codes • **`memoedit()`** passes 6 status codes Max provides for more state handling than Clipper does. ***NOTE:*** If your application seems to be "frozen" while one of these functions was called, check to see if your application is properly handling status codes. (See **Functions** on page 25.)

Extended Features

Max was designed to quickly compile and run legacy Clipper code. But you can add considerable power and reduce your coding effort if you take advantage of Max's superset features. This section summarizes Max benefits.

Using Extended Mode

As you go through the Max Language Reference, you will find details about extended mode features and behaviors. When you use certain commands and functions in extended mode, Max offers changed functionality. You can also find some introductory information about extended features in this document:

- **Default SET Values** on page 20.
- **Idle State Handling: idleb(), idlep(), idlem()** on page 26.
- **Array Handling** on page 35.
- **Miscellaneous** on page 15.

How to Invoke Extended Mode

- At the compiler command line- this will have a global effect on all code compiled for this application.
- In the source code- you can switch into $extended mode and back to $standard mode at any time.

Invoking Extended Mode When Compiling

```
max -extended myprogram.prg
```

Invoking Extended Mode Within Your Source Code

```
* myapp.prg
dfile = "customer.dbf"
$extended
USE dfile
$standard
USE mydbf INDEX myindex ALIAS myindex
```

Compiler Directives

Place these directives within your source code to trigger specific compile time actions:.

Directive	Meaning
`$echo <"echostring">`	Prints the message string <"echostring"> at compile time (if and when that line is compiled). • Used in conjunction with the pre-processor, you can issue messages indicating whether certain blocks of code are being compiled. • Place $echo directives in key places within your source code to assist in finding problem code areas or to test control flow.
`$extended`	Use extended Max features
`$standard`	Use Clipper Summer ‘87 syntax and behaviors
`$warn`	Invoke/suppress compiler warnings (equivalent to `-W` compiler flag)

Function Prototype Directive

Xbase's informality makes development fast, but can also lead to intensive debugging due to ambiguities, data type mismatches, and parameter mismatches. Max introduces function prototypes to Xbase. If you prototype your functions and invoke the appropriate warning level, Max provides more rigorous compile time checking of your function calls than has ever been offered in an Xbase product.

A prototype takes the following form:

```
$prototype  <returnType> funcName( varType, varType, @, ?, ...)
```

Where:

Symbols for Parameter Types:

Symbol:	Parameter Data Type Represented:
CHR	Character
NUM	Numeric
LOG	Logical
DAT	Date
?	Indeterminate
VAR	Variable name or Array name
@	Argument passed by reference. (Can be used in combination with other type symbols.)

Symbols for Return Type:

Symbol:	Return Data Type Represented:
CHR	Character
NUM	Numeric
LOG	Logical
DAT	Date
?	Indeterminate

For more information on compiler directives, see **Compiler Directives** on page 18.

Also see **Functions** on page 25.

Introductory information on functions can also be found in the Max Developer's Overview.

Default SET Values

In extended mode, Max implements different defaults for these SET commands:

	Clipper/Max Standard	Max Extended Mode
SET DATE	`AMERICAN`	`BRITISH`
SET WRAP	`OFF` `.F.`	`ON` `.T.`

Literals vs. Quoted Strings vs. Variables

Because Max is an Xbase language environment, you can use macros just as extensively as with any earlier Xbase product. But we've taken the Max a few steps further so that you can avoid the performance penalties that come with macros.

The most obvious place to start is with the command lines that open DBF data tables, index files and define the alias. Max allows you to use quoted strings or variables without requiring macros:.

References to filenames	Clipper	Max

Literals / Strings	`USE myDBF INDEX dx`	`USE myDBF INDEX dx`
		or:
		`USE "myDBF" INDEX "dx"`

Variables	`dfile=MyPath+"\"+MyDBF`
	`* need macro`
	`USE &dfile`
	`* or expression`
	`USE (dfile)`	

There is no need to do anything special to signal to Max that you are using quoted strings; you can compile in standard mode. So you may use the classic Xbase/Clipper syntax, or evaluate quoted strings.

Max automatically parses your literals, then it will correctly identify and evaluate an expression if Max finds:

- a quote `"` as the first character of a literal
- an opening parenthesis `(`

Millenium Solution (Year 2000)

Max provides a fix for interpreting date variables and fields lacking 4-digits:

- Use the **`SET EPOCH`** command to specify a year serving as the beginning of a 100-year period.
- Also **`SET CENTURY ON`** to ensure that all future data entry will require users to enter 4-digit years.

Consult the Max Language Reference for details on these two commands.

Expanded Alias References

Clipper allows programmers to point an alias to functions that don't accept the alias as a parameter. :

Clipper	Max
`recnow = cust->( RECNO() )`	`recnow = RECNO( "cust" )`

You may use the alias as a paraameter in these functions::

`afields()`	`deleted()`	`found()`	`lock()`
`bof()`	`eof()`	`header()`	`lupdate()`
`dbfilter()`	`fcount()`	`indexkey()`	`recno()`
`dbrelation()`	`field() / fieldname()`	`indexord()`	`recsize()`
`dbrselect()`	`flock()`	`lastrec() / reccount()`	`used()`

Operators

Assignment Operators:

Operator	Action
:=	Same as =. `memVar := 5 * 3` ***NOTE:*** This cannot be used as a logical equality operator. Use of this operator for assignments makes your coding intentions more specific than the = operator.
+=	This compound operator adds operand2 to operand1: `operand1 += operand2`
-=	This compound operator subtracts operand2 from operand1: `operand1 -= operand2`
*=	This compound operator multiplies operand1 by operand2: `operand1 *= operand2`
/=	This compound operator divides operand1 by operand2: `operand1 /= operand2`
^=	This compound operate raises operand1 to the power of operand2. (Operand2 is the exponent of operand1.) `operand1 ^= operand2`
%=	This compound operator divides operand1 by operand2 and returns the remainder: `operand1 %= operand2`

Universal Concatenation Operator

The universal operator is a special Max language feature permitting you to quickly force a non-character expression into a character type while concatenating::

Operator	Action
|	Concatenates multiple data types into one string; place between each element.

Increment and Decrement Operators

Increment and decrement operators are a concise way to add or subtract numeric values from a variable:

Operator Type	Operator	Action
Increment	**++**	Increases value of variable while performing operation.
Decrement	**--**	Decreases value of variable while performing operation.

The increment and decrement operators can appear as prefix or postfix operators:

- Postfix- If the operator appears *after* the variable name, increment operation happens *after* any other operations that use that variable:

```
nVar1 :=  5
nVar2 :=  nVar1++       && first assign to nVar2, then
increment
? nVar1              && result: 6
? nVar2&& result: 5
```

- Prefix- If the operator appears *before* the variable name, increment operation happens *before* any other operations that use that variable:

```
nVar1 :=  5
nVar2 :=  ++nVar1
? nVar1              && result: 6
? nVar2&& result: 6
```

Functions

Function Improvements

The following functions may behave differently in certain circumstances:

Function	Clipper	Max
afields( ) **acopy()**	• If an array has not already been declared, Clipper will not create the array. • If the retrieved data requires more rows than already declared in the array, Clipper will not resize the array.	• When an array has not already been explicitly declared, this function will automatically create it for you. • Max will automatically resize an array as necessary. **Note:** These behaviors only occur within regions of code compiled with the **$extended** directive. See the Functions reference for details.
achoice( ) **dbedit( )** **memoedit()**	• `achoice()` passes 5 status codes • `dbedit()` passes 5 status codes • `memoedit()` passes 5 status codes	• `achoice()` passes 7 status codes. • `dbedit()` passes 8 status codes • `memoedit()` passes 6 status codes Max provides for more state handling than Clipper does. **Issue 1:** You may have written UDF's that test for previously non-existent status codes. **Issue 2:** Your user defined functions may not adequately use all codes. **Solution for 1 & 2:** In your called functions, use `DO CASE` or `ELSEIF` to test for all possible modes returned by Max. ***NOTE:*** If your application seems to be "frozen" while one of these functions was called, check to see if your application is properly handling status codes.

Also see **Function Prototype Directive** on page 19.

Code Blocks

Code blocks are short pieces of executable code, much like a user defined function but without a function name. Like a macro, code blocks evaluate expressions.

- Code blocks perform better because they are compiled (unlike macros, which must be expanded and evaluated at runtime).
- You can use code blocks to execute short code fragments directly at the location where the code will be called.
- Code blocks may be assigned to variables.

Added Index Features

- Max provides a **KEY** clause for index commands. This enables your application to automatically create index files when do not exist. Specify the **KEY** clause in the **USE** or **SET INDEX** commands.

NOTE: In a multiuser environment, the regeneration of index files can impose unacceptable delays due to file locking. Index generation is best left to small tables used for lookups and light-duty purposes. Well designed applications should include administration modules for system administrators to regenerate large indexes at appropriate times when operations will not be impacted by the delay.

- Max provides a **FOR** clause, allowing you to create filtered indexes which include only the subset of keys meeting a conditional expression you specify.
- Max provides uniqueness integrity so you can implement true primary keys. (See **Unique Indexes Provide Integrity Protection** on page 11.)

Idle State Handling: idleb(), idlep(), idlem()

When you implement achoice(), dbedit() and memoedit(), Max calls your UDF during idle states. If you would like background tasks to execute during idle states, call **idleb()**, **idlep()** and **idlem()**.

See the Max Language Reference for details.f

Multidimensional Arrays

Max suports multidimensional arrays. A series of array-handling functions are added to work better with multidimensional arrays: :

`mcopy()`	`mdel()`	`mdir()`
`mfields()`	`mins()`	`msort()`

In addition to providing expanded data manipulation, Max multidimensional arrays are ideal data structures for storing lists. Before Max you called adir() and afields() and dumped the results into multiple parallel arrays. With Max, you only need one multidimensional array.

```
DECLARE MyArray[2,4]
```

Details on these functions may be found in the Max Language Reference. Multidimensional arrays are described in the Max Developer's Overview.

Variable Scoping

There are four classes of variables: PUBLIC, PRIVATE, LOCAL and STATIC. Each has its own scope characteristics. The scope of a variable determnes its lifetime and visibility:

	Lifetime	Visibility
PUBLIC	From creation until the termination of the program or the program or RELEASE.	From creation until the termination of RELEASE.
PRIVATE	From creation until control returns to any function or procedure above the module in module in which the variable was created.	From creation until control returns to any function or procedure above the which the variable was created.

	Lifetime	Visibility
LOCAL	Only as long as the module where the variable was created variable was created is still running.	Only as long as the module where the is still running.
STATIC	From creation until the termination of the program or variable was created is still running.	Only as long as the module where the RELEASE.

Field/Variable Precedence (SET PRECEDENCE)

Xbase is ambiguous in naming and referencing variables and fields. We know that **`cust->lastname`** is a field and **`m->lastname`** is a variable. The alias pointers tell us that explicitly. But is **`lastname`** a field or is it a variable?

By default, Xbase searches first for a field by that name. If none is available, it will then look for a memory variable with that name. Routines produce unexpected results because their variables refer to a database field that in certain contexts may or may not be available. This behavior can result in long debugging sessions for you.

Use **`SET PRECEDENCE TO VARIABLES`** among your application's initialization instructions to *reverse* the evaluation order Max employs when evaluating the name. (The result: Max evaluates ambiguous references by first consulting its list of in-scope variables. And then if no such variable exists, Max accessess the in-context fields.) This ensures you are referring to variables rather than fields when the name is ambiguous. But it still permits Max to consult the available fields if no such variable exists.

NOTE: Don't make implicit use of default behaviors. We strongly recommend that Max programmers use explicit references to fields by using work area pointers. This makes your applications more bug-free and it also documents your intent within the source code.

NOTE: When you use the CA-Clipper 5.x /V compiler switch, you are globally declaring all ambiguous references to be variables. (This is as if you applied m-> pointers to all ambiguous references within the code you have compiled.)

For more details, consult the Max Language Reference.

Integrated Debugger

The debugger is part of MaxRun. You activate it:

- by pressing ALT+D while running the app
- by specifying the -d switch on maxrun's command line

To view source, the source modules must in the same physical directory as the .max file.

Added Commands & Functions

Max's language set adds significant functionality. You can run your Clipper programs without extending functionality, but we recommend that you familiarize yourself with the many valuable benefits Max adds:

UNIX Support

termlog()	returns the name of the device associated with the current terminal login, like : "/dev/tty1a"
termname()	returns the terminal emulation type for the current login, like: ansi, vt100, etc
termwrite()	bypasses screen buffers and all screen output translation routines, and writes directly to the screen device

IBM Mainframe Support

asc2ebc()	converts a character expression in ASCII a a character string in EBCDIC
ebc2asc()	is used to convert a character string in EBCDIC to a character string in ASCII

Database Handling

BROWSE	provides dbedit() functionality using dBASE / FoxPro syntax. (See the list of clauses to control the browes behavior.)
uniqueviol( )	determines if there was an index key uniqueness violation in any index associated with the specified database file work area

Color Handling

asciicor()	converts a color (in the format "foreground/background") to a number that encodes the colors requested
decodcor()	decodes a color code into a string in the format "foreground/ background"

Numeric Handling

bin2f( )	converts a character string with 8 bytes in IEEE format to a number
f2bin()	converts a floating-point number to a character string with 8 bytes in IEEE floating-point format
lennum( )	determines the number of significant digits of a number
log10( )	calculates the base 10 logarithm of the number specified
mod( )	returns the remainder of a division
mod10( )	calculates the check control digit of a string based on mod10 algorithm
mod11( )	calculates the check control digit of a string based on mod11 algorithm
rand( )	generates a random number between 0 and 0.999999
sign( )	returns the sign of a number
w2bin( )	converts a number to a 2 character string

String Handling

difference( )	compares two strings, on the basis of SOUNDEX codes
like( )	compares a string with a wild card pattern
initcap( )	converts the initial characters of all words on the specified string to capital letters
pad( )	returns a new string of the length specified by pad_length (Clipper 5.x compatible)
raw( )	returns a string with all uppercase letters and converts all accented characters to equivalent characters without accents

Multiuser/Network Support

SET AUTOLOCK	When enabled, automatically lockes index files during each user update to the DBF.
network( )	returns whether the application is running on a multiuser environment.

Branching Logic

switch()	evaluates two or more expressions sequentially, returning the result of one of them according a the conditions evaluated. It is similar a the DO...CASE command, except that it can be used within expressions

Error Handling

criticalerror( )	is called when a critical error is issued by the OS. This function can be redefined by the user for custom behavior
uniqueviol( )	determines if there was an index key uniqueness violation in any index associated with the specified database file work area
usererror( )	is executed whenever an error occurs (before userexit( ))
userexit( )	is executed when the application is terminating

File Handling

attrib( )	returns and optionally changes attributes of a file
cd( )	changes the current directory
fcommit( )	forces the OS to write to disk any buffered data for the specified file
fpath()	returns the full path name of the specified file
indexalias()	returns the alias for the DBF associated with this index file
indexfor()	returns the filter expression when indexing on a **FOR** condition
md( )	creates a directory
rd( )	removes a directory

Keyboard Handling

autoclose()	closes the file opened by autorec( ) or autorun( ).
autorec( )	records all keyboard data entry into a file, storing the key pressed and the time intervals.
autorun( )	plays a previously recorded file created by autorec( ), reproducing keyboard data entry.
capslock( )	returns and optionally changes the status of the CAPSLOCK key
isalt( )	determines the current status of the ALT key.
isctrl( )	determines the current status of the CTRL key.
isins( )	determines the current status of the INS key.
islshift( )	determines the current status of the left SHIFT key
isrshift( )	determines the status of the right SHIFT key.
numlock( )	returns the current status of the NUMLOCK key and optionally sets NUMLOCK with a new status.
restkey( )	restores the function that was associated with a key before it was saved with savekey( ).
savekey()	saves the function associated to a key with the SET KEY.
scrlock( )	returns and optionally changes the status of the SCROLL LOCK key.
xlastkey( )	returns the code for the last key read from the keyboard buffer and optionally allows this code to be changed

Array Handling

adeclare( )	is used to declare an array when its name and number of elements are defined only at run time.
ainit( )	declare/initializes the elements of an array
alen()	returns the number of elements on multi-dimensional arrays
aredim( )	redimensions an array, keeping its old elements and if the array does not exist, aredim( ) declares it
arraycont()	allows indirect access a elements of an array specifing a character expression as the array name and a numeric expression as the array index. (Instead of using a macro to refer to the array.)
arraylen( )	returns the number of elements for the specified array. The array name is a character expression. (Instead of using a macro to refer to the array.)
arraystore( )	allows indirect assignment a array elements using a character expression a specify the array name. (Instead of using a macro to refer to the array.)
axchg( )	exchanges array elements
mcopy( )	duplicates an array with one or multiple dimensions
mdel( )	deletes elements in multi-dimensional arrays
mdir( )	is the multi-dimensional array version of adir( )
mfields( )	is the multi-dimensional version of afields( )
mins( )	inserts elements in multi-dimensional arrays
msort( )	is the multi-dimensional version of asort( )

User Interface Handling

AT row, col	same as @ row, col
BROWSE	allows you to inspect and edit the data of one or more database files in a familiar tabular format
EDIT MEMO	allows you to edit or just display the contents of any string
POPUP	allows you to create popup menus as with achoice()
SET PICTURE	specifies a format mask that determines the display and entry of data using SAY AND GET
SET POINT	allows you to define a character which will be used to separate the digits about the decimal point of a number for the purposes of displaying and entering data
SET SEPARATOR	SET SEPARATOR sets which character will be used to separate groups of three (3) digits to the left of the decimal point
SET TOPCHARS	defines a translation array for ASCII codes above 127
asciicor()	converts a color (in the format "foreground/background") to a number that encodes the colors requested
decodcor( )	decodes a color code into a string in the format "foreground/ background"
editing( )	determines if the current GET is being processed
explode()	draws an exploding box in the window defined by specified coordinates
gotoget( )	returns the number of the current GET being processed and also permits the user to jump to a specified GET
poscur( )	positions the screen cursor at a specified position. The limits of these coordinates are defined by maxrow( ) and maxcol( ), respectively
redraw( )	forces the entire screen to be redrawn

idleb( )	gives the current status of the BROWSE/dbedit() idle mode and optionally sets the BROWSE/dbedit() idle mode to active or not active
idlem( )	gives the current status of the EDIT MEMO/memoedit() idle mode and optionally sets the EDIT MEMO/memoedit() idle mode
idlep()	checks or sets the POPUP idle mode status. This permits the repeated execution of the function associated with POPUP during the wait for any key pressed
maxcol( )	returns the number of the last column of the screen
maxrow( )	returns the number of the last row of the screen

Date/Time Handling

SET HOURS	permits you to set whether 12 or 24 hour format is used for formatting hours, in all related commands
SET MARK	defines which character will be used to separate the day, month and year fields of all displayed dates and date data-entry masks
ampm()	converts a character expression in 24 hour time format ("HH:MM:SS") a a character string in 12 hour format ("HH:MM:SS am/pm")
days( )	calculates the number of days from a given number of seconds. (1 day= 86,400 secs)
dmy( )	converts a date to a date in a more readable format. This format is "dd month_name YY/YYYY", depending on the status of SET CENTURY
elaptime()	calculates the difference between end_hour and begin_hour.
mdy( )	converts a date to a string in long format. Its format is "month name dd, YY/YYYY" depending on the status of SET CENTURY
secs()	converts a time string in the format "HH:MM:SS" into seconds
stod( )	converts a character string in the format "YYYYMMDD" to a date value
tstring()	converts the number of seconds specified to a time character string in the format "HH:MM:SS"

seek( )	searches the master index of the database file
set( )	returns the current state of the specified SET (Clipper 5.x compatible)
store( )	stores an expression into a specified memory variable
videomode( )	returns the current mode of your video card. This mode can be changed with videomode( ) in DOS
vtype( )	returns the type of the variable or field that is specified

Unimplemented Features

Max embraces the good things that Clipper has to offer. But there is a short list unimplemented features to note:

Preprocessor: #command, #translate

Unlike CA-Clipper 5.x, the Max preprocessor does not impose case sensitivity.

Max does not implement the language creation/simulation features.:

- #command
- #translate
- the ability to simulate parameter passing for pseudofunctions.

For information on the implemented preprocessor features, see the Max Developer's Overview and **Preprocessor Directives** on page 9.

Pseudoclasses

- Error
- Get
- TBrowse
- TBColumn

Don't Abbreviate Commands

Max does not always recognize abbreviations. So if you develop errors in your application for no obvious reason, check if an abbreviation is the cause. (Example: `LOCA` is not an acceptable replacement for `LOCATE`.)

Clipper 5.x and Max

Introduction

Max represents the language features at the core of Clipper and Xbase. PlugSys believes that CA-Clipper 5.x introduced many truly useful features. But unfortunately it also unnecessarily complicated application development.

The introduction of "pseudoclasses" mimicked superficial aspects of object oriented programming. But CA-Clipper 5.x did not allow programmers to subclass these objects nor did the product support inheritance.

Fox Products and Max

Introduction

Once upon a time FoxBase and FoxPro was manufactured as cross platform products. But it's no longer possible to develop Fox applications for UNIX.

Max shares an Xbase heritage with Fox products. There is a broad common set of commands and functions recognizable to Fox programmers. But you'll also need to make some adjustments. This chapter offers some practical information to help you think about the migration process.

- Language differences- As you attempt to compile code, you will discover some language constructs unsupported in Max. Some commands and functions will have subtle but differences. (Different or missing parameters.) Check the Max Language Reference for the details.
- Compiled vs. Interactive- Fox products, like dBASE before them, were intended to be used in an interactive mode. Max is intended to compile source code. These differences surface in many areas. For example Fox allows you to call **`BROWSE`** without a single parameter. The Max **`BROWSE`** command requires a field list because it does not have an interactive environment to examine the DBF structure at runtime. Similarly, Max does not suppy a native interactive environment to create new DBF table files. (More applications are connecting to SQL databases, so over time this becomes less significant.)
- Text-oriented displays- Like SCO FoxBase, Max runs text-oriented applications. FoxPro and Visual FoxPro developers will not find spinners and combo boxes in Max applications. (If graphical user interfaces are important to you, this could be one of many good reasons for you to consider web development.)

Commands

BROWSE

CREATE STRUCTURE

Max offers language constructs to permit DBF file creation. For more information, see **Creating DBF Files** on page 45.

SCATTER and GATHER

Functions

Tips

Don’t Abbreviate Commands

When dBASE II was around, it was small (the entire program fit on a single 160K diskette). And the number of commands was small. Because dBASE was interactive, its designers allowed users to save typing by abbreviating commands to 4 characters.

Max does not always recognize abbreviations. So if you develop errors in your application for no obvious reason, check if an abbreviation is the cause. (Example: `LOCA` is not an acceptable replacement for `LOCATE`.)

How To Structure Your Source Code

- Procedures and functions must have unique names if they appear anywhere in the application. Max does not support `DO <procName> IN <PRGfile>`.

Compiling And Running The Application

Creating DBF Files

Max Additional Features

Introduction

Once upon a time there was an effort to establish and ANSI/ISO language standard for Xbase. You can read more about this effort at http://www.database.org/x3j19/. Our assumption here is that all Xbase programmers are familiar with the language features of dBASE III Plus, which constituted the base from which the Xbase Language Standard (X3J19) started its work.

This chapter introduces you briefly to features that extend Xbase beyond the dBASE III Plus/ Clipper Summer ’87 language core.

- Some of these features appear in Xbase implementations that followed dBASE III Plus and Clipper Summer 87.
- Other elements listed here are unique Max ease-of-use features.

Long Names

- **Variable Names:** can exceed 10 characters. Max does not truncate variable names. Xbase products that enforce the 10-character maximum quietly truncate the variable names and will cause problems (`invoicetot` and `invoicetotal2` are seen as the same in products of that type.).
- **UDF Names:** There are no practical restrictions on the length of UDF names.

Note: Fieldnames are limited by the database structure to which Max is connected. By default, Max connects to Xbase .DBF files, which impose a maximum 10-character fieldname.

Intrinsic (In-Line) Parameters

Parameters can be declared in the function header:

```
Function Func1(Name, City, State)
```

Scoping: LOCAL and STATIC

Classic Xbase did not provide a good way to automatically "protect" variables in one function or procedure from interactions with other subroutines.

A private variable hides itself from routines above the function or procedure that declared it. But no protection was provided for that same variable in lower level routines.

Max provides the LOCAL variable scope declaration. So you can be assured that the lifetime and visibility of the variable are both limited to the function or procedure in which it is created.

Max provides the STATIC variable scope declaration. So you can hold values and make them available whenever a certain function or procedure regains control. (For example to retain and process a total invoice amount.)

Lifetime and Visibility

	Lifetime	Visibility
PUBLIC	From creation until the termination of the program or RELEASE.	From creation until the termination of the program or RELEASE.
PRIVATE	From creation until control returns to any function or procedure above the module in which the variable was created.	From creation until control returns to any function or procedure above the module in which the variable was created.
LOCAL	Only as long as the module where the variable was created is still running.	Only as long as the module where the variable was created is still running.
STATIC	From creation until the termination of the program or RELEASE.	Only as long as the module where the variable was created is still running.

Code Delimiters

Block Enclosure Characters

Long command lines do not require semi-colons on every line. Just place the block enclosure characters before and after the command line:

```
replace CUST->NAME with M->NAME, ;;
CUST->CITY with M->CITY,
CUST->STATE with M->STATE,
CUST->ZIP with M->ZIP,
for CUST->STATE = "CA"
...
;.
```

String Enclosure Operator

Allows specification of long strings that spread across several lines. This becomes especially useful for SQL queries and HTML:

```
Mask := {{
<table width="100%" cellspacing="0" cellpadding="0">
<tr bgcolor="#CCCC99">
<td><font face="Verdana">
<font color="#000000" face="Atial">
<b>@LocCategory</b></font></font></td>
</tr>
</table>
}}
```

Line-splitting Array Initializer

Arrays can be initialized across several lines without the need to use the semi-colon at the end of each line.

```
Array1 := { 1,
2,
3,
4,
5
}
```

C-style remark operator

If you would prefer to avoid confusing your code comments with the macro operator, use this comment marker: :

```
? InvoiceTotal // Showing the invoice total
```

Compiler Controls

Warnings

Max supports 4 levels of warning severity.

The compiler automatically checks and detects argument type mismatch in expressions using internal functions. The compiler knows the argument and return types of all internal functions. It is also possible to prototype UDFs so that calls to these UDFs will also be checked for potential errors at compile time.

Platform Appropriateness Checking

The compiler warns when you attempt to use language features which are clearly inapplicable to the current environment.

Preprocessor Directives

C and C++ programmers have benefitted from these directives for years. They're easy to implement but allow you to improve the readability and maintainability of your code:

```
#include
#define
#ifdef...#else...#endif
#undef
```

Inline compiler directives

These allow you to specify compiler behaviors within areas of code:

$standard	**(Default mode)** This is the best choice for migrating Clipper Summer 87 code.
$extended	Provides improved functionality. (For details on extended mode, see Max Language Reference.) `You may also invoke extended mode at the compiler comamnd line with the -extended parameter.`
$echo	
$warn	

Inline Operators

Compound-assignment operators

+=	
-=	
***=**	
/=	
^=	
|=	

Unary increment/decrement operators

++	
--	

Change values while performing an operation:

```
customers=0
DO WHILE .NOT. EOF()
...
customers++
ENDDO
```

Inline Assignment

The optional use of assignment operator provides a clear way to differentiate between equality tests and value assignments. Allows assignments inside expressions, as in C.

```
* Set value of myvar1 equal to myvar2
* Compare if it is greater than 5
IF (myvar1 := myvar2) > 5
…
ENDIF
```

Codeblocks

Code blocks, highly regarded in Clipper 5.x, permit dynamic evaluation of code operations that would otherwise require a function. These are ideal for small code fragements that would otherwise require writing a UDF.

Arrays

Multidimensional Arrays

Other Xbase implementations have arrived at non-standard array implementations. Rather than using ragged arrays as in Clipper 5.x (where individual array elements can point to other arrays), Max supports true multidimentional arrays. Also there are the necessary utility functions to sort and manipulate tabular data.

Array Declaration

See **Line-splitting Array Initializer** on page 51.

Pointer Operations

Allows you to capture and retrieve the address of specific functions into POINTER variables. Indirect calls can be made using pointer variables through the CALL( ) function. Code invoked this way offers a significant performance advantage over macros or expression evaluation.

```
TaxCalcRoutine = Address( "Calc1" )

CheckOut( TaxCalcRoutine )
...
Function CheckOut( TaxRoutineAddress )

Call( TaxRoutineAddress, TotalSale, State )
```

Universal Concatenator

Cut coding normally required to converting and display compound strings based on multiple data types.

```
? "Total: " | TotFields | " - done on " | Date() | " at " | Time()
```


Quoted String Expressions: Alternative to Macros and Literals

When opening files in Xbase, experienced programmers prefer holding the filenames in a layer of abtraction removed from the code. You can do this in two ways:

- Treating the filenames as "constants" which may be evaluated at compile time. (This is achieved using the Max preprocessor. For more information, see **Preprocessor Directives** on page 52)
- Resolving the names at runtime (using memory variables).

Either approach allows you to "name the files" in one place in your code or even create a user-controlled configuration module where the filenames are defined. In order to do this with most Xbase products, you need macros to expand the filenames at runtime.

Max supports macros (which are indicated with the **&** flag). But you can improve performance and enhance readability by eliminating macros using this more orthogonal quoted string syntax::

```
$extended
USE "mydbf" INDEX "myindex" ALIAS "mine"
DBFfile := "mydbf"
Nxfile := "myindex"
AliasName := "mine"
USE DBFfile INDEX Nxfile
ALIAS AliasName
$standard
```

Note the use of the inline compiler directives (**$extended** and **$standard**). These permit Max to resolve this unique-to-Max feature (**$extended**) and return to legacy behaviors (**$standard**). See **Inline compiler directives** on page 52.

KEY clause regenerates indexes

The KEY clause is where you specify the key expression for a given index. This clause is available for commands that open index files. (USE, SET INDEX TO)

```
USE mydbf INDEX myName KEY "UPPER(lastname + firstname)"
```

The KEY clause:

- Saves you the bother of writing code for regenerating index files.
- Saves you the work of invoking the key regeneration routine.
- Provides a self-documenting way to declare index expressions.

Migrating to the Web

Introduction

If you've read this far, you must have some Xbase applications that would benefit from a beauty makeover and an architectural update. This chapter assumes your code already compiles and runs successfully under Max as per the recommendations elsewhere in this book.

This chapter helps you prepare applications written in Max's flavor of Xbase to run as web applications.

1. Isolate All User Interface Code and Forms

The web provides rich user interface objects (such as radio buttons, picklists, checkboxes, and push buttons). To get the benefit of these in your MaxWeb application, you'll need to remove all user interface code that may be mixed with other procedures and functions.

The easiest approach is to take each cluster of user interface code (such as GETs, dbedit(), memoedit(), etc.) and create a separate Xbase function as if this was an Xbase FORM (as in SET FORM TO).

2. Transform Xbase User Interface to HTML

Typical Xbase Form

```
SET COLOR TO bg+/b,r/w
CLEAR
STORE SPACE(20) to namefirst, namelast, address, city, state
STORE SPACE(2048) to comm
STORE .F. to pen, pencil, eraser, notebook, pad, breakfast, lunch,
dinner

@ 2, 2 SAY "First name" GET namefirst
@ 2,40 SAY "Last name" GET namelast
@ 4, 2 SAY "Address   " GET address
@ 6, 2 SAY "City      " GET city
@ 6,40 SAY "State" GET state

@10, 2 say "Items required:"
@11, 5 say "Pen" get pen PICTURE "@! Y"
@11,20 say "Pencil" GET pencil PICTURE "@! Y"
@11,35 say "Pad" GET pad PICTURE "@! Y"
@11,50 say "Notebook" GET notebook PICTURE "@! Y"

@13, 2 SAY "Meals:"
@14, 5 say "Breakfast" get breakfast PICTURE "@! Y"
@14,25 say "Lunch" get lunch PICTURE "@! Y"
@14,45 say "Dinner" get dinner PICTURE "@! Y"


comm = memoedit(comm, 18, 2, 26, 60)
READ
```

HTML Equivalent Form

```
<HTML>
<BODY BGCOLOR="#CCCC99">
<FONT FACE="Verdana,Arial,Helvetica,sans-serif">
<FORM>
<TABLE>
  <TR>
    <TD>First name</TD>
    <TD><INPUT TYPE="text" NAME="namefirst" SIZE="20"></TD>
  </TR>
  <TR>
    <TD>Last name</TD>
    <TD><INPUT TYPE="text" NAME="namelast" SIZE="20"></TD>
  </TR>
  <TR>
    <TD>Address</TD>
    <TD><INPUT TYPE="text" NAME="address" SIZE="20"></TD>
  </TR>
  <TR>
    <TD>City</TD>
    <TD><INPUT TYPE="text" NAME="city" SIZE="20"></TD>
  </TR>
  <TR>
    <TD>State</TD>
    <TD><INPUT TYPE="text" NAME="state" SIZE="20"></TD>
  </TR>
</TABLE>
<P>
Pen <INPUT TYPE="radio" NAME="supply" VALUE="pen">
Pencil <INPUT TYPE="radio" NAME="supply" VALUE="pencil">
Pad <INPUT TYPE="radio" NAME="supply" VALUE="pad">
Pen <INPUT TYPE="radio" NAME="supply" VALUE="notebook">
</P>
Breakfast <INPUT TYPE="checkbox" NAME="breakfast" VALUE="1">
Lunch <INPUT TYPE="checkbox" NAME="lunch" VALUE="1">
Dinner <INPUT TYPE="checkbox" NAME="dinner" VALUE="1">
<P>
<TEXTAREA ROWS="5" COLS="40"></TEXTAREA>
</P>
</FORM>
</FONT>
</BODY>
</HTML>
```

3. Turn HTML Into Xbase Print Statements

```
? [<HTML>]
? [<BODY BGCOLOR="#CCCC99">]
? [<FONT FACE="Verdana,Arial,Helvetica,sans-serif">]
? [<FORM>]
? [<TABLE>]
? [  <TR>]
? [    <TD>First name</TD>]
? [    <TD><INPUT TYPE="text" NAME="namefirst" SIZE="20"></TD>]
? [  </TR>]
? [  <TR>]
? [    <TD>Last name</TD>]
? [    <TD><INPUT TYPE="text" NAME="namelast" SIZE="20"></TD>]
? [  </TR>]
? [  <TR>]
? [    <TD>Address</TD>]
? [    <TD><INPUT TYPE="text" NAME="address" SIZE="20"></TD>]
? [  </TR>]
? [  <TR>]
? [    <TD>City</TD>]
? [    <TD><INPUT TYPE="text" NAME="city" SIZE="20"></TD>]
? [  </TR>]
? [  <TR>]
? [    <TD>State</TD>]
? [    <TD><INPUT TYPE="text" NAME="state" SIZE="20"></TD>]
? [  </TR>]
? [</TABLE>]
? [<P>]
? [Pen <INPUT TYPE="radio" NAME="supply" VALUE="pen">]
? [Pencil <INPUT TYPE="radio" NAME="supply" VALUE="pencil">]
? [Pad <INPUT TYPE="radio" NAME="supply" VALUE="pad">]
? [Pen <INPUT TYPE="radio" NAME="supply" VALUE="notebook">]
? [</P>]
? [Breakfast <INPUT TYPE="checkbox" NAME="breakfast" VALUE="1">]
? [Lunch <INPUT TYPE="checkbox" NAME="lunch" VALUE="1">]
? [Dinner <INPUT TYPE="checkbox" NAME="dinner" VALUE="1">]
? [<P>]
? [<TEXTAREA ROWS="5" COLS="40"></TEXTAREA>]
? [</P>]
? [</FORM>]
? [</FONT>]
? [</BODY>]
? [</HTML>]
```

4. Integrate All MaxWeb Code

5. Register Function and Procedure Prototypes

6. Compile MaxWeb Code

7. Test By Running Your Code With the MaxWeb Simulator

8. Install Code On Web Server

9. Embed Call to Invoke MaxWeb Within HTML Form

10. Run Your MaxWeb Application

Simple MaxWeb Application

```
* ------------------------------------------------------------
* Inserts email address in database MAILING.DBF
* (Called from within an HTML form - for use with MaxWEB)
*
* Input:  EMAIL
*
* Output: Send a page informing the operation status
*
*
* Copyright (C) PlugSys International, 1999. All rights Reserved
* ------------------------------------------------------------
   * Max offers // as an additional comment signal
Function SUBSCRIBE( Email )
Private  ResultPage ,;
        MWBody     ,;
        MWLink

   // Check if Email is valid
   // ! is equivalent to .not.

   if ! IsValidEmail( Email )
      // := is equivalent to = as assignment operator
      MWBody := "Please provide a valid email address."
      SendPage("mwpages\Standard.html")
      return .T.
   end

   // Check if email address is already in the database
   use "us\MAILING" index "us\INDMAIL" shared
   seek lower(M->Email)
```

```
if Found()
   MWBody := {{ Thanks for your submission.
                <br><br>
                (<B>MaxWeb</B> found your email address already
                 in the <B>PlugSys International</B> mailing list.)
             }}
else
   append blank
   replace Mailing->EMAIL with M.Email,  ;
           Mailing->DATE  with Date(),   ;
           Mailing->TIME  with Time()

   * Note the {{ and }} as the start and termination characters
   * as a convenience for long strings

   MWBody := {{ Thank you for joining the <B>PlugSys
                International</B> mailing list.
                 <br><br><br>
                <B>MaxWeb</B> successfully added you to our database.
             }}
endif

MWBody += {{ <br><br>
             By pressing the <I>Submit</I> button you have
             executed an Xbase routine that stores your email
             address into a DBF file.
             <br>
          }}

MWLink := {{ <A href="/cgi-bin/MaxWEB.dll?webprocs:SHOWCODE">
             <B>Inspect</B></A> the <B>MaxWeb</B> source code behind
             this mailing list applet.
          }}
SendPage("mwpages\Standard-MW.html")
close MAILING
return .T.  // Terminate: do not leave app running
```

```
* ------------------------------------------------------------
* This procedure shows this PRG module in HTML format
*
* Input:  none
*
* Output: Send a page containing this PRG module
*
*
* Copyright (C) PlugSys International, 1999. All rights Reserved
* ------------------------------------------------------------


Function SHOWCODE()
Private     nHandle, ;
           nFileSize, ;
           MWCode

   nHandle = fopen(_MWPath+"\webprocs\subscrib.prg")

   if nHandle == -1
      MWCode = "<B>Error loading source file</B>"
   else
     nFileSize = fseek(nHandle, 0, 2)    // To the end of file: file size
      fseek(nHandle, 0)            // Point file handle to start of file
      if fread(nHandle, @MWCode, nFileSize) != nFileSize
         MWCode = "<B>Error loading source file</B>"
      else
         MWCode = TranslateHTML(MWCode)
      end
      fclose(nHandle)
   endif

   SendPage("mwpages\Standard-ShowCode.html")
   return .T.  // Terminate: do not leave app running
```

Clipper to Max Migration Guide

Max Additional Features

Migrating to the Web

Clipper to Max Migration Guide

Max Additional Features

Migrating to the Web