JBC CLIPWKS USER'S GUIDE

Introduction
CLIPWKS (pronounced "CLIP-WORKS") is a Clipper library that allows your program to transfer data to and from Lotus, Excel, and Quattro spreadsheets. The library is designed to permit you to create a spreadsheet with data and formulas, as well as specifying printer parameters, sort ranges and keys, calculation parameters, protection, borders, and global alignment. In addition, the library allows you to read existing spreadsheets and access any cell within the spreadsheet. Within limitations, it is also possible to overwrite cell values with data of the same type.

CLIPWKS provides a set of high-level functions that can be quickly linked into an application to provide spreadsheet import and export, and to produce spreadsheets from report form files. If you only need to add spreadsheet translation to an existing program, these functions should allow you to get up and running with a minimal amount of effort.

System Requirements
To develop applications using ***CLIPWKS***, you must have a computer running MS-DOS or PC-DOS (including DOS boxes within Windows), the Clipper 5.x compiler, and a linker capable of linking Microsoft object modules. You can also use ***CLIPWKS*** within a protected mode program.

To use the NG online help program, you must have a copy of the Norton Guides program (that comes with Clipper) or Expert Help and sufficient RAM to use them.

Installing CLIPWKS
CLIPWKS comes on a single floppy disk, which contains various versions of the library, a NG online help file, and complete source code to the library. You merely need to copy the appropriate LIB file to your Clipper library directory and the online help file to your NG directory. The source code is included and installed in a separate directory. If you modify the source code, you will need to have access to CLIPWKS.CH (included in the source directory). CLIPWKS.CH is not needed in your application programs.

Linking CLIPWKS
CLIPWKS is protected mode compatible and can be linked by any linker that supports Clipper code. There are three CLIPWKS lib files, depending upon which version of Clipper you are using:

- CLPWKS50.LIB - Clipper 5.01
- CLPWKS52.LIB - Clipper 5.2
- CLPWKS53.LIB - Clipper 5.3

Substitute the appropriate library file where shown in the examples below:

Exospace
```
C>exospace fi prog lib CLPWKS5x
```

Blinker (script file)
```
     FILE your_prog
    BEGINAREA
        ALLOCATE CLPWKS5x
        ALLOCATE extend
    ENDAREA
```

RTLINK

```
     C>RTLINK FILE your_prog LIBRARY CLPWKS5x
        -or-
    C>RTLINK your_prog, , ,CLPWKS5x
```

License

1. Basic use: Joseph D. Booth Consulting, Inc. grants you, the user, an unlimited duration, non-transferable license to install and use ***CLIPWKS*** (hereinafter called the software) on one computer at a time. The software may not be networked, i.e. you cannot install it on a network for group use. Each person using this software must have his or her own copy. A registered user of the software is granted permission to keep one copy at his primary work location and a second copy at home. This done not however permit a programmer to keep a copy of a company registered software at home, only the software registered to the programmer.

2. Copying: The software is not copy-protected. However, it is protected by United States copyright law, all rights reserved. It is ***illegal*** to create, distribute, or allow the distribution of unauthorized copies of the software. You may make a reasonable number of backup copies, provided they are strictly for the registered user's convenience and use.

3. Integration: You may compile and/or link the source included and/or generated by the software into your own executable program and distribute these programs without any royalty or payment due to Joseph D. Booth Consulting, Inc. You may not, however, distribute copies of the source code, even if modified by you, without the express, written permission of Joseph D. Booth Consulting, Inc.

4. User Manual: This user manual is protected by US copyright laws. It cannot be reproduced, transmitted, transcribed, stored in a retrieval system, or translated in any language in any form with the prior written consent of Joseph D. Booth Consulting, Inc.

5. **Governing law:** This license agreement is governed by the laws of the Commonwealth of Pennsylvania.

Limitation of Liability

Joseph D. Booth Consulting, Inc. will not be liable for consequential, special, indirect or other similar damages or claims, including loss of profits or any other commercial damages. In no event will Joseph D. Booth Consulting, Inc.'s liability exceed the price paid for the software, regardless of any form of claim.

Technical Support

In order to make using CLIPWKS as easy and enjoyable as possible, we provide several forms of support to registered users. In order to be eligible for support, you must register your serial number by either mailing in the registration card or sending a message to either Compuserve or through the internet.

Internet

You can reach tech support through the internet at Joe-Booth@WORLDNET.ATT.NET. Be sure to leave your serial number. Messages are usually checked at least twice a day.

Compuserve

You can send messages to account number 72040,2112 on Compuserve if you'd like for technical support. We usually check Compuserve messages at least twice a day.

FAX

You can reach tech support via the FAX at 610-409-8859. Faxes are usually responded to within 24 hours. Please include your serial number, and if possible, a sample of your CLIPWKS problem.

Telephone

You can call for voice tech support at 610-409-8858 between the hours of 9:00am and 3:00pm EST. Please make sure you have your serial number handy. Voice support is only provided for registered CLIPWKS users.

Trademarks

- CLIPWKS is a registered trademark of Joseph D. Booth Consulting, Inc.
- BLINKER is a registered trademark of Assembler Software Manufacturers, Inc.

•

Chapter 1 - Importing and Exporting

CLIPWKS consists of a variety of functions to allow you to create and to read spreadsheets without using the translation options built into many of the spreadsheet programs. There are two levels of functions in CLIPWKS. The first level, covered in this chapter, is a high-level set of functions to provide import and export ability between spreadsheets and .DBF's. The second level, covered in the following chapters, allows your application to create spreadsheets which can be directly used by a spreadsheet program, or to read data from spreadsheets into your Clipper application. Depending upon your application, you should find enough functionality in CLIPWKS to solve a wide range of spreadsheet access issues.

Importing data from a spreadsheet

CLIPWKS includes a program called ***import.prg***. This program serves as an example of using CLIPWKS as well as a useful function to quickly bring data into a .dbf file.

The Limport() function takes four parameters which control the spreadsheet data to be imported into the DBF file. These parameters are the filename, the DBF name, an optional range of cells, and a default field type to be used if CLIPWKS cannot make a field determination.

The first parameter is the spreadsheet name. It must exist and be a valid spreadsheet, otherwise Limport will return an error code.

The second parameter is the name of the DBF file to put the data into. If the DBF file exists, its structure is used and data is appended into the file. If you want a new file created, be sure to ZAP the file before calling Limport().

If the file does not exist, then Limport() uses the first row in the range of cells to determine the types and sizes of the fields in the .DBF file. Be sure to specify a cell range that does not include headers, since this would result in all fields being character type, which might not be what you want. For example:

	A	B	C
1	Name	Hire date	Salary
2	Bill Clinton	22-JAN-93	120000.00
3	Al Gore	22-JAN-93	80000.00

Specify range A2..C3, rather than A1..C3. If CLIPWKS can not find a cell within the range, it defaults to numeric. You can also specify the type of field to create if CLIPWKS cannot determine the cell type. This is the fourth parameter and is optional.

The third parameter is the range name to import. This can be a named range, such as TAX or SALARY (if these fields exist in the file) or cell coordinates separated by two periods (or a colon in Excel). If the range is not specified, the entire spreadsheet is imported. If the range cannot be found, then Limport() returns a error code.

Limport() contains a translation function which is used to try and convert spreadsheet data into the DBF field. It will handle must transformations, such as Str() or logical values.

Exporting data to a spreadsheet

CLIPWKS provides a function which can be used to create a spreadsheet from an existing .DBF file. The function's name is Lexport(), which is the opposite of the Limport() function above. It allows you to create a spreadsheet file from the contents of a DBF table. The parameters to the function are described below:

Specifying Output FileName

The first parameter is the name of the spreadsheet file to be created. It can be a file name or a fully qualified path and file designation.

Selecting Fields For Export

By default, the Lexport() function will transfer every field into a cell of a new spreadsheet. This can be changed by specifying the second parameter, a field array. If passed, the field array will specify which fields and in which order they should be written to the spreadsheet.

```
function demo
LOCAL  aList_ := {"Soc_Sec","Last","First","Salary"}
   select PAYROLL
   Lexport("Payroll.wks", aList_)
return NIL
```

The field array may contain a size designation, which is separated from the field name by chr(124). For example, if the third element in the example above were changed to:

```
        aList_[3] := "First | 1"
```

Then column C of the spreadsheet would only contain the first character of field->first. The field reference can also contain embedded functions, for example:

```
       aList_[3] := "trim(last) + ',' + substr(first,1,1)"
```

In the example above, the last name and the first initial would appear within the first column.

If your array element contains a function rather than a field name, be sure to pass the size parameter. If not Lexport() will default to 10 characters, which is the default width of a spreadsheet cell.

NOTE:

If headers are not desired in the spreadsheet, pass logical false as the third parameter to prevent headers from being written to the file.

Specifying Headers

Lexport() will create column headers based on the field names. The third parameter, an array of headers, can be used to override the default headers.

Selecting Records

The fourth parameter controls which records the Lexport() command should process. The default is all records.

ALL	Character string – transfer all records in the file.
FOR	Expression -- only records meeting the filter expression.

WHILE	Key, conditions – seek the key, and do while condition is meet.
TAG	A string the same number of characters as there are records in the database. If a check mark chr(251) is found in the corresponding position based on - substr(tag_list, recno(), 1) the record will be written. If any other character is found, the record will not be transfered into the spreadsheet.
EVERY	Every record is written into the spreadsheet. This allows a sample of the file to be extracted.

If an array is passed as the parameter it is assumed to contain record numbers of the records to be transferred into the spreadsheet.

Voters.dbf

Name	**Character**	**25**
Address	**Character**	**25**
City	**Character**	**15**
State	**Character**	**2**
Did_Vote	**Logical**	**1**
Party	**Character**	**1**

To produce a spreadsheet of all voters with no headers, we would use the following syntax:

```
    Lexport( "voters.wks", aList_, .F., "ALL" )
```

To produce a spreadsheet of only Republicans Party="R", we would use the following sybtax:

```
    Lexport(  "voters.wks", aList_, .F., "FOR:party="R" )
```

Chapter 2 - Creating Spreadsheets

CLIPWKS allows you to create spreadsheet files which can be read by the appropriate spreadsheet program without special translation. CLIPWKS is not limited to only writing data, but also allows you to write formulae and global settings. In this chapter, we will cover the functions needed to create a spreadsheet and place data into it. We will also cover the functions available to set some of the global settings.

Creating a spreadsheet

The Lcreate() function call is used to tell CLIPWKS you are going to create a spreadsheet. It expects two parameters, the name of the file you wish to create, and the type of spreadsheet to create. For example, to create an Excel 3.x file to contain payroll information, you could use the following code.

```
        aWks := Lcreate("PAYROLL","E3")
```

where PAYROLL is the name of the file, and E3 stands for Excel 3.x. If you do not specify a file extension, CLIPWKS will select the appropiate one for the spreadsheet version. For Excel spreadsheets, this would be .XLS. (Lotus 2.x uses .WK1, Lotus 3.x uses .WK3 Quattro uses .WKQ, and Quattro Pro uses .WQ1).

If the Lcreate() succeeds, an array will be returned. It is essential that this array is save since the other functions expect this array as a parameter. If the array is empty, then a problem occurred when attempting to create the file. The most likely problem is that the file already exists and cannot be overwritten or that you've specified a bad directory name. Immediately after creating a spreadsheet, your code should check to make sure that this array is not empty.

```
function MakePR()
LOCAL aWks := Lcreate( "PAYROLL","E3" )
if empty(aWks)
   Alert("Problem occurred creating PAYROLL.XLS")
endif
```

Ok, now what...

Once you've called Lcreate() and saved the array that was returned, you are ready to start writing to your spreadsheet. The array returned from Lcreate() must be passed as the first parameter to any function you call while writing the spreadsheet.

When writing to the spreadsheet, you should specify the settings first, and then write the data. For example, if you wish to turn global protection on, you would use the Lprotect() function after your call to Lcreate(). You could also set other parameters and then put your data into the file.

```
function MakePR()
LOCAL aWks := Lcreate( "PAYROLL","E3" )
if empty(aWks)
   Alert("Problem occurred creating PAYROLL.XLS")
endif
//
// Set global parameters
//
Lprotect(aWks,.T.)                // Turn on Global protection
Lfooter(aWks,"For Review Only")   // Print Footing
```

Global settings

The global settings determine how the spreadsheet is calculated, printed, and sorted. You can also specify cell alignment and can write named ranges into the file.

Print settings

CLIPWKS allows you to control how a spreadsheet is printed. You can specify margins, heading and footing strings, range of cells to print, etc. These settings are described in the
following section.

Lborders()	This function sets the row and column borders which will be printed along the top and left margin whenever the spreadsheet is printed.
Lfooter()	This function allows you to specify the footer string to be written on the bottom of each page when the spreadsheet is printed.
Lheader()	This function allows you to specify the header string to be written.
Lmargins()	This function allows you to specify the printer margins for the spreadsheet.
Lprint()	This function allows you to specify whether the spreadsheet is printed in a formatted (with borders and headers) or unformatted (just data) format.
LprtRange()	This function allows you to specify the range of cells to print when the sheet is printed.
Lsetup()	This function allows you to specify the printer setup string to use when printing the spreadsheet

Sort settings

With CLIPWKS, you can specify a sort range and up to two sort keys. Each key can be (D)escending or (A)scending depending upon your application's needs. This section describes the sort parameters. The sort range determines how much of the spreadsheet should be sorted and the sort keys determine which column(s) are used to sort the data.

Lsortkey()	This function specifies the primary column which the spreadsheet should be sorted on. You can specify both the column range and the sort direction.
Lsortkey2()	This function specifies the secondary column which the spreadsheet should be sorted on. You can specify both the column range and the sort direction.
LsortRange()	This function specifies which cells should be sorted

Spreadsheet appearance

CLIPWKS allows you to specify the global alignment for text cells and the column widths for various columns in the file. This section describes these functions.

Lalign()	This function allows you to set the global alignment to L)eft, R)ight or C)entered.

Lformat()	This function allows you to specify the format for a range of cells. It is only used with Lotus 3.x which does not store the cell formatting information with the cell.
Lwidth()	This function allows you to specify the column width for a particular column or to set the default column width for the spreadsheet

Now, for the actual data

Once the spreadsheet has been created and the global settings specified, you can start writing data into the various cells. As with the global settings, the commands to place data into cells also expect the array from Lcreate() as the first parameter.

Writing blank cells

Although blank cells are normally not stored when the spreadsheet is written, they can be stored if they are referenced as part of a formula of if they contain special formatting. You can use the Lblank() function in ***CLIPWKS*** to write a blank cell into the spreadsheet..

Writing data into cells

The Lput() function is used to write data and formulas into your spreadsheet. You need to pass three parameters to Lput(), the first of which is the spreadsheet array returned from the Lcreate() function call. The second parameter is the data itself. The data type of this parameter determines what kind of cell ***CLIPWKS*** writes to the spreadsheet. The third parameter is the cell address that you'd like to write the data to, such as A3 or B15, etc. You can use the Lc() function to convert row/column coordinates into a spreadsheet cell reference. There is also a fourth parameter, which is optional. This parameter specifies the formatting to apply to the cell, such as numeric, date, percent etc.

Data type	Value	Description
Numeric	any	Writes a numeric cell
Logical	true	@true formula
	false	@false formula
Date	any	@date() function
Char	+ or @	Clipwks assumes this is a formula and attempts to parse your expression into a formula. If Clipwks can create a formula- it will write a formula cell into the spreadsheet.
	/x	The character following the slash is repeated across the width of the column. This is handy for underling headings- etc.
	any	A label cell is written with the text you pass

The option cell format parameter can be one of the following format attributes which a number of decimals.specifed.

Format	Meaning	
Dx	Date	1=DD-MMM-YY
		2=MMM-YY
		3=DD-MMM
Fd	Fixed	[d]ecimals
Pd	Percent	[d]ecimals
,d	Comma	[d]ecimals
Cd	Currency	[d]ecimals
Sd	Scientic Notation	[d]ecimals

Chapter 3 - Reading Spreadsheets

CLIPWKS provides two methods for reading cells from a spreadsheet. The first is LfindFirst() and LfindNext() which sequentially reads every cell from the spreadsheet. This is the fastest method CLIPWKS offers for reading a spreadsheet.

The code below shows a function that will read all the cells in the file and display the cell address and its contents.

```
procedure main(cFile)
LOCAL aWks   := Lread( cFile )
LOCAL aCell
if !empty(aWks)
   aCell := LfindFirst(aWks)
   while !empty(aCell)
      ? padr(aCell[4],8)," ",aCell[1]
      aCell := LfindNext(aWks)
   enddo
   Lclose(aWks)
else
   Alert("Error occurred opening the spreadsheet!")
endif
return
```

CLIPWKS only requires three function calls to read any spreadsheet. First you must open the spreadsheet using Lread(). This function returns an array of information used by the other CLIWPKS functions. The Lget() function expects the array from Lread() and can be used to retrieve the contents of any cell within the file. The final function, Lclose(), takes the array as the parameter and closes the file.

The code below shows a function that will open the spreadsheet and return the contents of a specified array of cells.

```
procedure main(cFile,aCells)
  LOCAL aWks   := Lread( cFile )
  LOCAL nSize  := len(aCells)
  LOCAL arr_   := array(nSize)
  LOCAL x

  if !empty(aWks)
     for x := 1 to nSize
        arr_[x] := Lget( aWks,aCells[x] )
     next
     Lclose(aWks)
  else
     Alert("Error occurred opening the spreadsheet...")
     arr_ := {}
  endif
return arr_
```

Chapter 4 - Function Listing

The following section lists all the functions with the CLIPWKS library. They are grouped together by usage class.

Writing functions

Lalign()	Set label alignment
Lblank()	Write a blank cell into the worksheet
Lborders()	Write print borders into the worksheet
Lcalcmode()	Set calculation mode
Lcalcorder()	Set calculation order
Lcreate()	Creates a new spreadsheet
Lfirstcell()	Set first cell address
Lfooter()	Write footer string into the spreadsheet
Lformat()	Set the format for a range of cells
Lheader()	Write header string into the spreadsheet
Literate()	Set the formula recalculation iteration flag
Lmargins()	Write printer margin information into the spreadsheet
Lname()	Write a named range into the spreadsheet
Lprint()	Write select formatted or unformatted spreadsheet printing
Lprotect()	Set the global protection flag in the spreadsheet
Lprtrange()	Write print range of cells to spreadsheet
Lput()	Writes data to the spreadsheet
Lrange()	Write number of rows/columns into spreadsheet
Lsetup()	Write printer setup string into the spreadsheet
Lsortkey()	Writes the primary sort key out to the spreadsheet
Lsortkey2()	Writes the secondary sort key out to the spreadsheet
Lsortrange()	Writes the sort range out to the spreadsheet
Lwidth()	Set the column to the specified width

High-level functions

Lexport()	To export a .DBF file to a spreadsheet
Limport()	To import a spreadsheet into a .DBF file
Frmtowks()	To convert a report form file to a spreadsheet

Reading functions

Lcols()	To determine number of columns in spreadsheet
Lfindfirst()	To return the first cell in the spreadsheet
Lfindnext()	To return subsequent cells from either Lfindfirst() or Lget()
Lget()	To return the contents of a cell
Lgetcell()	To return an array of cell attributes
Lgetrange()	To return an array of spreadsheet cells
Lread()	To open a spreadsheet for reading
Lreplace()	To replace an existing cell's contents
Lrows()	To determine number of rows in spreadsheet
Ltype()	To return the type of a cell

Miscellaneous functions

Lc()	To convert row and column coordinates to spreadsheet notation

Lclose()	To write the EOF op code and close the file handle
Liswks()	To test if a file is a valid spreadsheet
Lr()	To convert row and column coordinates to spreadsheet range
Lversion()	To determine the version of the spreadsheet

Lexport() EXPORT.PRG

Purpose To export a .DBF file to a spreadsheet
Syntax Lexport(cSpreadsheet, aFld_exp, aHdg_exp, cRecords, cVersion)
Parameters

cSpreadsheet	name of the spreadsheet to create
aFld_exp	array of fields to export
aHdg_exp	array of heading expressions
cRecords	which records to write to spreadsheet
cVersion	type of spreadsheet to produce

Returns nStatus
Notes Lexport() is a high-level function use to transfer records from a .DBF file into a spreadsheet. A great degree of flexibility is provided, including control of the fields and headings, and the selection of records to write to the spreadsheet.

cWksfile is the name of the spreadsheet to create. It defaults to the work area name

aFields is either an array or a semi-colon delimited list of fields/functions to place in each column of the spreadsheet.

aHeaders is a corresponding array or semi-colon delimited list of header strings. Each element contains the header for the parallel entry in the aFields array. If aHeaders is a logical .F., no headers will be written to the spreadsheet.

cRecs is a character string indicating which records to transfer into the spreadsheet.

CVersion is a two character abbreviation indicating the type of spreadsheet you with to create

L1 - Lotus 1.x	***QU - Quattro***
L2 - Lotus 2.x	***QP - Quattro Pro***
L3 - Lotus 3.x	***SY - Symphony***
E2 - Excel 2.x	***E3 - Excel 3.x for Windows***
E4 - Excel 4.x	

Limport() IMPORT.PRG

Purpose To import a spreadsheet into a .DBF file

Syntax Limport(cSpreadsheet,cDBF_file [,cRange] [,cDefType])

Parameters

cSpreadsheet	name of the spreadsheet to read
cDBF_file	name of .DBF file to import into
cRange	optional range of cells to import
cDefType	optional default field type

Returns nStatus

0 = Ok
-1 = Missing spreadsheet
-2 = Problem opening the file
-3 = Invalid range specified
-4 = Field type unknown in header
-5 = Problem creating the .DBF file

Notes If the <cDBF_file> exists, it's structure is used and data is appended into the file. If you want a new file created, be sure to ZAP the file before Limport().

If the <cDBF_file> does not exist, then Limport() uses the first row in the range of cells to determine the types and sizes of the fields in the .DBF file. Be sure to specify a cell range that does not include headers, since this would result in all fields being character type, which might not be what you want.

FrmtoWks() FRM2WKS.PRG

Purpose To convert a report form file to a spreadsheet

Syntax Frmtowks(cReport_form, cSpreadsheet, cVersion)

Parameters

cReport_form	name of the report form file to use
cSpreadsheet	name of the spreadsheet to create- the extension is optional
cVersion	type of spreadsheet to produce- optional and will default to L2

Returns nStatus - -1 Form file is missing
-2 I/O error reading form file
-3 Invalid form file
-4 Error creating spreadsheet
-5 No current work area

Notes: FRMTOWKS() is used to read standard report form files and create spreadsheets from their contents. This allows you to give the user the option of printing to the screen/printer/ or a spreadsheet. The valid spreadsheet types are:

L1 - Lotus 1.x	***QU - Quattro***
L2 - Lotus 2.x	***QP - Quattro Pro***
L3 - Lotus 3.x	***SY - Symphony***
E2 - Excel 2.x	***E3 - Excel 3.x for Windows***
E4 - Excel 4.x	

Example

```
function DoReport( cFile,cOutput )
    LOCAL nResult
    LOCAL nChoice := 0
    LOCAL aErrors := { "No such form file",;
                       "Problem reading form file",;
                       "Invalid report form file",;
                       "Disk error creating spreadsheet",;
                       "No active work area selected"    }

    @ 10,10 prompt "Print report"
    @ 11,10 prompt "Display to the screen"
    @ 12,10 prompt "Make a spreadsheet"

    menu to nChoice

    do case
        case nChoice == 1
            report form (cFile) to printer
        case nChoice == 2
            report form (cFile) to console
        case nChoice == 3
            nResult := FrmToWks( cFile,cOutput )
            if nResult < 0
                Alert( aErrors[abs(nResult)] )
            endif
    endcase
return NIL
```

Lalign() MISCWKS.PRG

Purpose	Set label alignment, (L)eft,(C)enter,(R)ight	
Syntax	Lalign(aSpreadsheet,cAlignment)	
Parameters	aSpreadsheet	- Spreadsheet array handle
	cAlignment	- Alignment method
Returns	nStatus	0 All ok
		6 Invalid parameters

Notes: Text in a cell may be justified left, right, or centered. The Lalign() function is used to set the global alignment to be used for text being written to cells. Lalign() is not used with Excel spreadsheets.

Example
```
function DoReport( cFile,cOutput )
LOCAL aWks  := Lcreate( "TAXES","L2" )
   Lalign( aWks,"C")            // Set (C)entered alignment
   Lfooter( aWks,"Pennsylvania taxes" )
   Lclose( aWks )
return NIL
```

Lblank() WRITEWKS.PRG

Purpose Write a blank cell into the worksheet
Syntax Lblank(aSpreadsheet, cCell, cFormat)
Parameters:

aSpreadsheet	spreadsheet handle array
cCell	Cell address to write blank into

cFormat	Format to write in cell
	Dx = Date 1- DD-MMM-YY
	2- MMM-YY
	3- DD-MMM
	Fd = Fixed [d] decimals
	Pd = Percent [d] decimals
	,d = Comma [d] decimals
	Cd = Currency [d] decimals
	Sd = Sci Notation [d] decimals

Returns	nStatus	0 All ok
		4 Invalid cell
		6 Invalid parameters

Notes Lblank() allows you to write a blank cell into a spreadsheet. Unless the cell is referred to by a formula or has a different format, blank cells are normally not saved within a spreadsheet.

Lborders() MISCWKS.PRG

Purpose Write print borders into the worksheet
Syntax Lborders(aSpreadsheet,cTop_range,cLeft_range)

Parameters

aSpreadsheet	Spreadsheet handle array
cTop_range	Range of cells for top border
cLeft_range	Range of cells for left border

Returns	nStatus	0 All ok
		5 Invalid range
		6 Invalid parameters

Notes: Borders are ranges of cells that are printed along the left margin and the top row of a printed spreadsheet. Be sure to keep the borders in mind when you set the print range of the spreadsheet.

Example
```
function main
    LOCAL aWks  := Lcreate( "PARYOLL","L3" )
    if !empty(aWks)
        Lborders( aWks,"A1..G2","A2..A10")
        Lclose( aWks )
    endif
return ""
```

Lcalcmode() MISCWKS.PRG

Purpose Set calculation mode
Syntax Lcalcmode(aSpreadsheet,cMode)
Parameters

aSpreadsheet	Spreadsheet handle array
cMode	Calculation mode (A)utomatic (M)anual (B)ackground

Returns	nStatus	0 All ok
		6 Invalid parameters

Notes The calculation mode determines how often a spreadsheet's formula are recomputed. In you choice (A)utomatic, the spreadsheet will update all formulas as soon as a cell value affecting the formula is changed. If you choice (M)anual, you need to press a key, (usually F9), to recalculate the file. (B)ackground allows the spreadsheet to be recalculated while it is waiting for keystrokes from the end-user. If you create a spreadsheet using CLIPWKS and set the calcmode to (M)anual, be sure to have your end-user press F9 as soon they read in the spreadsheet. CLIPWKS does not calculate the values from the formulas it writes to the spreadsheet.

Lcalcorder() MISCWKS.PRG

Purpose Set calculation order
Syntax Lcalcorder(aSpreadsheet,cOrder)
Parameters

aSpreadsheet	Array of spreadsheet parameters
cOrder	Calculation order

	(R)owwise (C)olumn (N)atural

Returns	nStatus	0 All ok 6 Invalid parameters

Notes Calculation order determines when dependent cells are recalculated in order to resolve any formulas. Natural calculation order means that before a formula is computed, each cell it refers to is calculated first to ensure accurary. In column recalculation order, calculation starts in the first column and proceeds down. It ignores formulas in other columns. Rowwise works similarly, calculating rows at a time. The preferred method is (N)atural.

Lcreate() WRITEWKS.PRG

Purpose	Creates a new spreadsheet
Syntax	Lcreate(cFilename, cVersion)

Parameters

cFilename	Name of new spreadsheet to create
cVersion	Spreadsheet version

Valid spreadsheet versions are:

QP - Quattro Pro	L1 - Lotus 1.x	E2 - Excel 2.x
QU - Quattro	L2 - Lotus 2.x	E3 - Excel 3.x
	L3 - Lotus 3.x	E4 - Excel 4.x

Returns	aSpreadsheet

The spreadsheet is an array which needs to be passed to other ***CLIPWKS*** functions used when writing spreadsheet files.

Lfirstcell() MISCWKS.PRG

Purpose Set first cell address
Syntax Lfirstcell(aSpreadsheet,cAddress)
Parameters

aSpreadsheet	Spreadsheet array handle
cAddress	First cell address

Returns nStatus 0 All ok
6 Invalid parameters

Notes: Setting the first cell allows you to have an automatic macro startup when the user invokes your spreadsheet.

Lfooter() PRINTWKS.PRG

Purpose Write footer string into the spreadsheet
Syntax Lfooter(aSpreadsheet,cFooting)
Parameters

aSpreadsheet	Spreadsheet handle array
cFooting	Footer string

Returns nStatus 0 All ok
-6 Invalid parameters

Notes The footer string appears on the bottom of each page whenever the spreadsheet is printed. It may be up to 240 characters in length. Lfooter() is used to specify the footing string.

Example
```
function main
LOCAL aWks   := Lcreate("PAYROLL","L3")
if !empty(aWks)
    Lheader(aWks,"Joseph D. Booth Consulting, Inc.")
    Lfooter(aWks,"For review only")
    Lclose(aWks)
endif
return NIL
```

Lformat() MISCWKS.PRG

Purpose Set the format for a range of cells
Syntax Lformat(aSpreadsheet, cRange,cFormat)
Parameters

aSpreadsheet	Spreadsheet array handle
cRange	Range of cells

cFormat	Format to write in cell
	Dx = Date 1- DD-MMM-YY
	2- MMM-YY
	3- DD-MMM
	Fd = Fixed [d]ecimals
	Pd = Percent [d]ecimals
	,d = Comma [d]ecimals
	Cd = Currency [d]ecimals
	Sd = Sci Notation [d]ecimals

Returns nStatus - 0 All ok
-5 Invalid range
-6 Invalid parameters

Notes: The Lformat() function is only needed with Lotus 3.x files. Lotus 3.x stores the format information in the header rather than the individual cells. If you need special formats in Lotus 3.x from CLIPWKS, you must use the Lformat() function first.

Lheader() PRINTWKS.PRG

Purpose Write header string into the spreadsheet
Syntax Lheader(aSpreadsheet,cHeading)
Parameters

aSpreadsheet	Spreadsheet handle array
cHeading	Heading string

Returns nStatus - 0 All ok
- 6 Invalid parameters

Notes The header string gets printed at the top of each page whenever the spreadsheet is printed. The Lheader() function is used to specify this string. It may be up to 240 characters long.
Example:

```
 function HeaderDemo
LOCAL aWks   := Lcreate("PAYROLL","L3")

if !empty(aWks)
   Lheader(aWks,"Joseph D. Booth Consulting, Inc.")
   Lfooter(aWks,"For review only")
   Lclose(aWks)
endif
return NIL
```

Literate() MISCWKS.PRG

Purpose Set the formula recalculation iteration flag

Syntax Literate(aSpreadsheet, nCount)
Parameters

aSpreadsheet	Array of spreadsheet parameters
nCount	Iteration count

Returns nStatus - 0 = All Ok
- 6 = Invalid parameters

Notes: On some complex formulas, multiple iterations are needed to get an acceptable degree of accuracy. You may specify the number of iterations to perform on formula calculations using the Literate() function. The default value is one iteration.

Lmargins() PRINTWKS.PRG

Purpose Write printer margin information into the spreadsheet
Syntax Lmargins(aSpreadsheet, nLeft, nRight, nPage, nTop, nBottom)
Parameters

aSpreadsheet	Spreadsheet handle array
nLeft	Left margin in columns
nRight	Right margin in columns
nPage	Lines per page
nTop	Top margin in rows
nBottom	Bottom margin in rows

Returns nStatus - 0 = All Ok
- 6 = Invalid parameters

Notes Lmargins() is used to specify the margins and page length. The left and right margin can be between zero and 254 characters. The top and bottom margin can be between zero and 32 lines. The page length can be between one and 100 lines.

Default values for various margins are listed below:

Left	3	Excel margins are measured in
Right	78	inches, so be sure to keep this
Top	3	mind when specifying margins.
Bottom	3	The defaults for Excel are an
Lines	66	inch on each side.

Lname() MISCWKS.PRG

Purpose Write a named range into the spreadsheet
Syntax Lname(aSpreadsheet, cName, cRange)
Parameters

aSpreadsheet	Array of spreadsheet parameters
cName	Name for specified range
cRange	Range of cells

Returns	nStatus	-0 = All Ok
		-5 = Invalid range
		-6 = Invalid parameters
		-7 = Missing range name

Notes: A spreadsheet allows a range of cells to have a name assigned to them. This assists in documenting function calls. The Lname() function writes an range name out to the spreadsheet. If a formula references the range, it will display the range name when the formula is view.

Example

```
 function DemoOne
LOCAL aWks := Lcreate("PAYROLL","L3")
if !empty(aWks)
   Lname(aWks,"SALARY","A1..A15")
   Lname(aWks,"TAXES","B1..B15")
   Lclose(aWks)
endif
return NIL
```

Lprint() PRINTWKS.PRG

Purpose Write select formatted or unformatted spreadsheet printing

Syntax Lprint(aSpreadsheet,cMethod)

Parameters

aSpreadsheet	Spreadsheet handle array
cMethod	(F)ormatted,(U)nformatted

Returns	nStatus	-0 = All Ok
		-6 = Invalid parameters

Notes: Spreadsheets are normally printed in formatted mode, showing both row and column headers. You can instruct the spreadsheet to print either formatted or unformatted using the Lprint() function.

Example:

```
function PrnDemo
    LOCAL aWks := Lcreate("SALES","L3")
    if !empty(aWks)
       LPrint(aWks,"F")  // Specify formatted printing
       Lclose(aWks)
    endif
return nil
```

Lprotect() MISCWKS.PRG

Purpose: Set the global protection flag in the spreadsheet

Syntax: Lprotect(aSpreadsheet,cSetting | lSetting)

Parameters

aSpreadsheet	Array of spreadsheet parameters
cSetting	ON | P - Enable global protection
	OFF | U - Disable global protection
lSetting	TRUE - Enable FALSE - disable

Returns	nStatus	- 0 = All Ok

- 6 = Invalid parameters

Notes: Lprotect is used to enable or disable global protection. When protection is enabled, protected cells may not be modified.

Example

```
function ProtectDemo
    LOCAL aWks := Lcreate("SALES","L3")
    if !empty(aWks)
       LProtect(aWks,.T.)  // Enable global protection
       Lclose(aWks)
    endif
return nil
```

Lprtrange() PRINTWKS.PRG

Purpose Write print range of cells to spreadsheet

Syntax Lprtrange(aSpreadsheet,cRange)

Parameters

aSpreadsheet	Spreadsheet handle array
cRange	Range of cells to print

Returns nStatus - 0 = All Ok
- 5 = Invalid range
- 6 = Invalid parameters

Notes: The print range determines what cells will be printed when the user prints the spreadsheet. Be sure to keep in mind the value of the Lborders() settings when specifying the print range.

Example

```
function Demo
   LOCAL aWks := Lcreate("SALES","L3")
   if !empty(aWks)
      LPrtRange(aWks,"A1..G50")
      Lclose(aWks)
   endif
return nil
```

Lput() WRITEWKS.PRG

Purpose Writes data to the spreadsheet

Syntax Lput(aSpreadsheet, xData, cCell_address, cFormat)

Parameters

aSpreadsheet	spreadsheet handle array
xData	Data to write
cCell_address	Cell address to write data at
cFormat	Format byte for the cell

Returns nStatus

Notes: The type of data determines what gets written in the cell. Numeric data is written out directly, logical data is converted to either the @TRUE or @FALSE function, and dates are converted to the @DATE function. Character data is interpreted by the first byte. If the first byte is a plus sign + or an ampersand @, then it is assumed to be a formula and the formula will be written out. Any other first character will result in the data being written directly to the spreadsheet.

Example:

```
        Lput(aWks,"+A5*@SUM(B2..B6)","A1")
        Lput(aWks,date(),"A2")
        Lput(aWks,PAYROLL->salary,"A3")
```

Lrange() MISCWKS.PRG

Write number of rows/columns into spreadsheet

Purpose **Syntax** Lrange(aSpreadsheet, nRows, nColumns)

Parameters

aSpreadsheet	Array of spreadsheet parameters
nRows	Number of rows in spreadsheet
nColumns	Number of columns in spreadsheet

Returns nStatus

Notes: The Lrange() command is used to specify how many active rows and columns the spreadsheet has.

Example:

```
function main
    LOCAL aWks := Lcreate("SALES","L3")
    if !empty(aWks)
       Lrange(aWks,20,5)  // 20 rows by 5 columns
       Lclose(aWks)
    endif
return nil
```

Lsetup() PRINTWKS.PRG

Purpose Write printer setup string into the spreadsheet

Syntax Lsetup(aSpreadsheet,cSetup)

Parameters

aSpreadsheet	Spreadsheet handle array
cSetup	Printer setup string

Returns nStatus

Example

```
function DemoWks
    LOCAL aWks := Lcreate("SALES.WK1")
    if !empty(aWks)
       Lsetup(aWks,"/015")
       Lclose(aWks)
    endif
return nil
```

Notes: The printer setup string can include imbedded ASCII codes by using the backward slash followed by the three digit ASCII code. For example, \015 would be used to set condensed mode on most Epson printers. If multiple characters are needed, no space should be used to separate them.

Lsortkey() SORTWKS.PRG

Purpose Writes the primary sort key out to the spreadsheet

Syntax Lsortkey(aWks, cRange, cDirection)

Parameters

aSpreadsheet	Spreadsheet handle array
cRange	Range of cells
cDirection	(A)scending or (D)escending

Returns nStatus

- 0	= All ok
- 5	= Invalid range
- 6	= Invalid parameters

Example

```
procedure main
    LOCAL aWks := Lcreate("TAXES.WK1")
    if !empty(aWks)
       LsortKey(aWks,"A1..A50","A")
       Lclose(aWks)
    endif
return
```

Notes: Lsortkey() defines the primary key that your spreadsheet should be sorted on. The range may be sorted either (A)scending or (D)escending depending upon the needs of your application.

Lsortkey2() SORTWKS.PRG

Purpose Writes the secondary sort key out to the spreadsheet

Syntax Lsortkey2(aSpreadsheet, cRange, cDirection)

Parameters

aSpreadsheet	Spreadsheet handle array
cRange	Range of cells
cDirection	(A)scending or (D)escending

Returns nStatus

- 0	= All ok
- 5	= Invalid range
- 6	= Invalid parameters

Example

```
procedure main
    LOCAL aWks := Lcreate("TAXES.WK1")
    if !empty(aWks)
       LsortKey(aWks,"A1..A50","A")
       LsortKey2(aWks,"B1..B50","D")
       Lclose(aWks)
    endif
return
```

Notes: Lsortkey2() defines the secondary key that your spreadsheet should be sorted on. The range may be sorted either (A)scending or (D)escending depending upon the needs of your application.

Lsortrange() SORTWKS.PRG

Purpose Writes the sort range out to the spreadsheet
Syntax Lsortrange(aSpreadsheet,cRange)
Parameters

aSpreadsheet	Spreadsheet handle array
cRange	Range of cells

Returns nStatus - 0 = All ok
- 5 = Invalid range
- 6 = Invalid parameters

Example

```
procedure main
LOCAL aWks := Lcreate("TAXES.WK1")
if !empty(aWks)
    LsortRange(aWks,"A1..H50")
    LsortKey(aWks,"A1..A50","A")
    LsortKey2(aWks,"B1..B50","D")
    Lclose(aWks)
 endif
 return
```

Notes: The sort range determines which cells are sorted when the /DS command is executed. It is normally all active cells within the spreadsheet.

Lwidth() WRITEWKS.PRG

Purpose Set a column's width or to set the global width in the spreadsheet.
Syntax Lwidth(aSpreadsheet [,cColumn|nColumn], nWidth)
Parameters

aSpreadsheet	Spreadsheet handle array
cColumn	Column letter
nColumn	Numeric letter
nWidth	Width to set

Returns nStatus -0 = All ok
-6 = Invalid parameters

Notes: Lwidth() is used to set a column width. You may specify the column using letter notation or by referring to the column number. Widths should be set prior to any data being written to the spreadsheet.

Example

```
#include "DBSTRUCT.CH"
function main
LOCAL aWks   := Lcreate("SAMPLE","L3")
LOCAL aStruc := PAYROLL->( dbstruct() )
LOCAL i
LOCAL nSize  := len(aStruc)
```

```
   if !empty(aWks)
      for i := 1 to nSize
         Lwidth(aWKs,i,aStruct[i,DBS_LEN])
      next
      Lclose(aWks)
   endif
   return .T.
```

Lcols() READWKS.PRG

Purpose To determine number of columns in spreadsheet

Syntax Lcols(aWks)

Parameters

aWks	Spreadsheet handle array

Returns nCols - Number of columns or -6 if invalid parameters

Example

```
function main
    LOCAL aWks := Lread("SALES.WK1")
    if !empty(aWks)
        ? "There are "+alltrim(str(Lrows(aWks)))+" and "
        ?? alltrim(str(Lcols(aWks)))+" in this spreadsheet"
        Lclose(aWks)
    endif
return nil
```

Notes: Lcols() is primarily used to build loops to read all cells in the spreadsheet, although Lfindfirst() and Lfindnext() are the fastest methods of reading cells in sequential order.

Lfindfirst() READWKS.PRG

Purpose: To return the first cell in the spreadsheet

Syntax: Lfindfirst(aWks [,xColumn])

Parameters

aWks	Spreadsheet handle array
xColumn	Column letter to find first cell of
nColumn	Column number to find first cell of

Returns aCell - Array of cell information

1) Cell data
2) Cell type
3) Physical size in spreadsheet
4) Cell address
5) Row number
6) Column number
7) Physical offset in file

Notes: Lfindfirst() positions the spreadsheet to the first cell in the spreadsheet or in the specified column. It also returns an array of cell information. Subsequent calls to Lfindnext() will obtain the next cell.

Example

```
function main
```

```
    LOCAL aWks   := Lread("PAYROLL")
    LOCAL aCell
    if !empty(aWks)
       aCell := LfindFirst(aWks)
       while !empty(aCell)
          ? aCell[1]
          aCell := LfindNext(aWks)
       enddo
       Lclose(aWks)
    endif
return ""
```

Lfindnext() READWKS.PRG

Purpose To return subsequent cells from either Lfindfirst() or Lget()

Syntax Lfindnext(aWks)

Parameters

aWks	Spreadsheet handle array

Returns aCell - Array of cell information

1) Cell data
2) Cell type
3) Physical size in spreadsheet
4) Cell address
5) Row number
6) Column number
7) Physical offset in file

Notes: Lfindnext() reads the next cell in the spreadsheet. It may start as a result of either a call to Lfindfirst() or to Lgetcell(). If a cell is found, an array of cell information is returned. If no cell is found, an empty array is returned.

Example:

```
function main
    LOCAL aWks   := Lread("PAYROLL")
    LOCAL aCell
    if !empty(aWks)
       aCell := LfindFirst(aWks)
       while !empty(aCell)
          ? aCell[1]
          aCell := LfindNext(aWks)
       enddo
       Lclose(aWks)
    endif
return ""
```

Lget() READWKS.PRG

Purpose To return the contents of a cell

Syntax Lget(aSpreadsheet,cCell)

Parameters

aSpreadsheet	Spreadsheet handle array
cCell	Cell address to look for

Returns xCell_contents - NIL if cell not found

Notes: Lget() returns the contents of the specified cell address. The data type will vary depending upon the cell's contents. If the cell contains the formulas @FALSE or @TRUE, a logical value will be returned. If the cell is formatted as a date, then CLIPWKS will return a date value. Otherwise, CLIWPKS will return either a numeric or character data. If the cell is not found, NIL will be returned.

Example:

```
function ReadPayWk
    LOCAL aWks   := Lread("PAYROLL.WK1")
    if !empty(aWks)
       ? Lget(aWks,"A1")
       ? Lget(aWks,"B2")
       Lclose(aWks)
    endif
return nil
```

Lgetcell() READWKS.PRG

Purpose To return an array of cell attributes
Syntax Lgetcell(aSpreadsheet,cCell)
Parameters:

aSpreadsheet	Spreadsheet handle array
cCell	Cell address to look for

Returns aCell_info - 1= Contents
2 = Cell type
3 = Cell length
4 = Cell address
5 = Row number
6 = Column number
7 = File offset

Notes Lgetcell() returns an array with information about the specified cell address. If the cell is not found, then NIL will be returned.

Example

```
function Sample
    LOCAL aWks   := Lread("PAYROLL.WK1")
    LOCAL aCell
    if !empty(aWks)
       ? aCell := Lgetcell(aWks,"A1")
       ? aCell[4],aCell[1]
       ? aCell := Lgetcell(aWks,"B2")
       ? aCell[4],aCell[1]
       Lclose(aWks)
    endif
return nil
```

Lgetrange() READWKS.PRG

Purpose To return an array of spreadsheet cells
Syntax Lgetrange(aWks,cRange)
Parameters

aWks	Spreadsheet handle array
cRange	Range specifier or range name

Returns aContents - Contents of all cells within range

Notes: Lgetrange() returns an array with the contents of the range. The range may be either a named range, such as SALARY or a specified range such as A1..A15.

Example:
```
function OpenPayWks
   LOCAL aWks   := Lread("PAYROLL.WK1")
   LOCAL aRange := {}
   if !empty(aWks)
       ? aRange := Lgetrange(aWks,"A1..B5")
       Lclose(aWks)
   endif
return nil
```

Lread() READWKS.PRG

Purpose To open a spreadsheet for reading
Syntax Lread(cFilename,lUpdate,lDates)
Parameters

cFilename	Name of spreadsheet to read
lUpdate	Whether CLIPWKS can update the spreadsheet
lDates	Whether to recognize dates in Lotus 3.x

Returns aWks - Spreadsheet handle array

Notes: Lread() is used to open an existing spreadsheet file for read access. You can also specify TRUE as the second parameter, which will allow you to update the spreadsheet using Lreplace(). Lread() returns an array of information that is used by the other functions to retrieve data from the spreadsheet. The third parameter is used when you are opening Lotus 3.x spreadsheets. Lotus 3.x does not store format information with the individual cells, but rather in the headers. CLIPWKS needs to evaluate these headers and create a bitmap to indicate which cells are formatted as dates. This processing time can be eliminated if you know there are no dates in the spreadsheet. Passing a FALSE as the third parameter controls this option. If the spreadsheet is large, this option can improve peformance when opening the file.

Example:
```
function main
    LOCAL aWks := Lread("SALES.WK3",.T.,.T.)
    if !empty(aWks)
       Lget(aWks,"A2")
       Lget(aWks,"A3")
       Lclose(aWks)
    endif
return nil
```

Lreplace() READWKS.PRG

Purpose To replace an existing cell's contents
Syntax Lreplace(aWks, xData, cCell)
Parameters

aWks	- Spreadsheet handle array
xData	- New data to replace into the cell
cCell	- Cell address to change

Returns nStatus - 0 = All ok
- 6 = Invalid parameters
- 12 = Spreadsheet open readonly

Example:
```
function main
    LOCAL aWks := Lread("PRESDENT.WK3",.T.,.T.)
    if !empty(aWks)
       Lreplace(aWks,"Bill Clinton","A2")
       Lreplace(aWks,"Al Gore","A3")
       Lclose(aWks)
    endif
return NIL
```
Notes:
Lreplace() allows you to change the values in an existing spreadsheet file. It also will allow you to add new values into the file. By default, when you open a spreadsheet it is opened readonly so you do not accidently replace data. See Lread() for the syntax to open spreadsheets for update.

Lrows() READWKS.PRG

Purpose To determine number of rows in spreadsheet
Syntax Lrows(aWks)
Parameters

aWks	Spreadsheet handle array

Returns nRows - Number of rows or -6 if invalid parameters

Example:
```
function main
    LOCAL aWks := Lread("SALES.WK1")
    if !empty(aWks)
       ? "There are "+alltrim(str(Lrows(aWks)))+" and "
       ?? alltrim(str(Lcols(aWks)))+" in this spreadsheet"
       Lclose(aWks)
    endif
return nil
```
Notes
Lrows() is primarily used to build loops to read all cells in the spreadsheet, although Lfindfirst() and Lfindnext() are the fastest methods of reading cells in sequential order.

Ltype() READWKS.PRG

Purpose To return the type of a cell
Syntax Ltype(aSpreadsheet,cCell)
Parameters

aSpreadsheet	Spreadsheet handle array
cCell	Cell address to look for

Returns	xCell_type	NIL - if cell not found
		C - Character
		D - Date
		L - Logical
		N - Numeric
		F - Formula
		B - Blank
		E - Error Cell
		NA - Not Applicable Cell

Lc() MISCWKS.PRG

Purpose To convert row and column coordinates to spreadsheet notation
Syntax Lc(nRow,nColumn [,nSheet])
Parameters

nRow	Row number
nColumn	Column number
nSheet	Worksheet number- Lotus 3.x only

Returns cCell_address

Notes Lc() is used to convert from row and column notation into spreadsheet cell addresses. It is most often used in loops, as shown in the example below:

Example
```
function main
LOCAL k
LOCAL j
for k=1 to 10
   goto k              // Goto record 'k'
   for j=1 to 5
       Lput( aWks,fieldget(j),lc(k,j) )
   next
next
return nil
```

The row may be an integer between 1 and 8192 and the column an integer between 1 and 256.

LClose() WRITEWKS.PRG

Purpose To write the EOF op code and close the file handle
Syntax Lclose(aSpreadsheet)
Parameters

aSpreadsheet	Spreadsheet handle array

Returns	nStatus	0 All ok
		2 I/O error occurred while closing
		6 Invalid parameters

Notes: After any spreadsheet is opened, either through Lcreate() or Lread(), you must close it to release its buffer and its handle. If you do not close a newly created spreadsheet, there is a good chance that some data will not be written out to the disk. Be sure to close all spreadsheets after you are done with them.

LisWks() MISCWKS.PRG

Purpose To test if a file is a valid spreadsheet
Syntax Liswks(cFile|nHandle)
Parameters

cFile	File name
nHandle	Handle file is opened on

Returns lSuccess
Notes Liswks() tests the beginning and ending bytes of the file to see if the file appears to be a valid spreadsheet file. It is useful to confirm the validity of a file if your application allows a user to type in a spreadsheet name.

Lr() MISCWKS.PRG

Purpose To convert row and column coordinates to spreadsheet range
Syntax Lr(nTopRow, nTopColumn, nBottomRow, nBottomColumn [,nSheet])
Parameters

nTopRow	Row number
nTopColumn	Column number
nBottomRow	Bottom row number
nBottomColumn	Bottom column number
nSheet	Spreadsheet number - Lotus 3.x only

Returns cRange_address
Notes Lr() takes four coordinates and converts them into a cell address range. It is used to allow the program to work in terms of numeric row and columns and let CLIPWKS handle the translation into spreadsheet syntax.

Lversion Miscwks.prg

Purpose To determine the version of the spreadsheet
Syntax Lversion(cFile|nHandle)
Parameters

cFile	File name
nHandle	Handle file is opened on

Returns cVersion

L1 - Lotus 1.x
L2 - Lotus 2.x
L3 - Lotus 3.x
L4 - Lotus 4.x
LJ - Lotus J
QU - Quattro

QP	- Quattro Pro
E2	- Excel 2.x
E3	- Excel 3.x
E4	- Excel 4.x

Notes
If the file|handle appears to be a valid spreadsheet, this function will return a two character code indicating the version of spreadsheet it appears to be. If the file or handle does not appear to be a spreadsheet, then an empty string will be returned.

Chapter 5 - Trouble Shooting

There are various problems that might occur using CLIPWKS. Some of the more common problems are explained here. Be sure to review this section before calling for technical support.

Empty array from Lcreate()

Lcreate() uses the Fcreate() function to open a spreadsheet file. If Fcreate() fails, then an empty array will be returned. Examine the value of FERROR()immediately after the call to Lcreate(). The error returned from Ferror() will indicate what caused the problem. The most likely DOS errors you will run into are:

Ferror	*Meaning*
4	Too many files open - so increase your file handles
5	Access denied - probably because a file already exists and is set to read-only or the network denies access to the file
15	Invalid drive specified - check you file's drive letter
19	Attempted to write to a write-protected - move the write protect notch or write to a hard disk
21	Drive not ready - so close the drive door

Empty array from Lread()

Lread() uses the Fopen() function to open a spreadsheet file. If Fopen() fails, then an empty array will be returned. Examine the value of FERROR() immediately after the call to Lread(). The error returned from Ferror() will indicate what caused the problem. The most likely DOS errors you will run into are:

Ferror()	*Meaning*
2	File not found, check your spelling
3	Path not found, check your path name and network rights
4	Too many files open, so increase your file handles
15	Invalid drive specified, check you file's drive letter
21	Drive not ready, so close the drive door

Why are global parameters not getting updated?

Global settings and parameters must be written before any Lput() function calls are made. CLIPWKS saves all your settings in a buffer and writes them to disk after you call Lput(). Any settings made after the disk is written, will not be saved.

Blank or data cells are not properly formatted?

If you are using Lotus 3.x, keep in mind that you must use the Lformat() command to specify the range of cells to format. Individual cells are not formatted in Lotus 3.x files

Why won't CLIPWKS open a spreadsheet?

CLIPWKS always attemps to verify that the file is a valid spreadsheet file. It does this by reading the signature bytes in the file. Some new versions or spreadsheets created by other applications might not be recognized by CLIPWKS. If you have such a beast, please send us a sample so that future CLIPWKS releases can support it. CLIPWKS does not support Excel 5.x or above due to these files being stored in an undocumented structure storage format. Microsoft does not recommend trying to read these files from DOS, but rather through Windows DLL's.

Chapter 6 - Examples

The following programs show some examples of how you might use CLIPWKS.

Depreciation Table
Assume we need to build a chart comparing depreciation methods for a group of assets. The assets are stored in a database file called ASSETS.DBF, which has the following structure.

Field	Type	Size
Desc	Character	25.0
Cost	Numeric	12.2
Salvage	Numeric	12.2
Life	Numeric	3.0

To produce a spreadsheet called DEPREC.WK1, which contains each asset, its cost, savlage value, life, as well as formulas for computing various depreciation amounts, we could use the following code:

```
procedure DumpDepr
LOCAL wks_
LOCAL k := 2
LOCAL params

wks_ := Lcreate( "DEPREC.WK1","L2")
if !empty(wks_)
   Lrange( wks_, reccount()+2, 7 )
   Lwidth( wks_,"A",26 )
   Lwidth( wks_,"B",13 )
   Lwidth( wks_,"C",13 )
   Lwidth( wks_,"D", 4 )
   Lwidth( wks_,"E",13 )
   Lwidth( wks_,"F",13 )
   Lwidth( wks_,"G",13 )
   **************************
   ** Write column headers **
   **************************
   Lput(wks_,"Asset Description","A1")
   Lput(wks_,"Cost","B1")
   Lput(wks_,"Scrap","C1")
   Lput(wks_,"Life","D1")
   Lput(wks_,"S/L","E1")
   Lput(wks_,"SYD","F1")
   Lput(wks_,"DDB","G1")
   **********************
   ** Write underlines **
   **********************
   Lput(wks_,"\=","A1")
   Lput(wks_,"\=","B1")
   Lput(wks_,"\=","C1")
   Lput(wks_,"\=","D1")
   Lput(wks_,"\=","E1")
   Lput(wks_,"\=","F1")
   Lput(wks_,"\=","G1")
   go top
   while !eof()
       params := lc(++k,2)+","+;
                 lc(k,3)+","+lc(k,4)
       Lput(wks_,ASSETS->desc,lc(k,1))
       Lput(wks_,ASSETS->cost,lc(k,2))
       Lput(wks_,ASSETS->salvage,lc(k,3))
       Lput(wks_,ASSETS->life,lc(k,4))
       Lput(wks_,"@SLN("+params+")",lc(k,5))
       Lput(wks_,"@SYD("+params+",5)",lc(k,6))
       Lput(wks_,"@SLN("+params+",5)",lc(k,7))
       skip +1
    enddo
    lclose(wks_)
else
     Alert("Couldn't create DEPREC.WK1 file")
endif
close databases
return
```

Budget update from a spreadsheet

This example reads an existing spreadsheet and an existing database, and creates a new spreadsheet. The database contains the chart of accounts and the spreadsheet contains budget information for the upcoming year. We assume, for simplicity sake, that the spreadsheet is in the same order as the database file, i.e. record one corresponds to row one in the worksheet. The file structure of the chart of accounts is:

Field	Type	Size
Account	Character	12.0
Desc	Character	25.0
Budget	Numeric	12.2

To produce a spreadsheet called BUDGET.WK1, which contains the updated budget information, we could use the following code. This code will also update the ***budget*** field from the database.

```
LOCAL   oldwks
LOCAL   newwks
LOCAL   k := 0
LOCAL   newamt

use CHART new
oldwks  := Lread("budget.wk1")
newwks  := Lcreate("Budget.new","L2")

if !empty(oldwks) .and. !empty(newwks)
   while !eof()
      newamt := lget(oldwks,lc(++k,3))
      replace CHART->budget with newamt
      Lput(newwks,CHART->account,lc(k,1))
      Lput(newwks,CHART->desc,lc(k,2))
      Lput(newwks,newamt,lc(k,3))
      skip +1
   enddo
   Lclose(newwks)
   Lclose(oldwks)
   rename budget.wk1 to budget.old
   rename budget.new to budget.wk1
else
   Alert("Problem with files")
endif
close all
        return
```