Use1401

Programmer's Interface Library for Windows and Macintosh

Nov 2009

First edition	February	1993
Second edition	October	1993
Third edition	June	1997
Fourth edition	December	1999
Fifth edition	October	2006
Sixth edition	November	2009

Published by:

Cambridge Electronic Design Limited
Science Park
Milton Road
Cambridge
CB4 0FE
UK

Telephone:	Cambridge (01223) 420186
Fax:	Cambridge (01223) 420488
Email:	info@ced.co.uk
Web site:	www.ced.co.uk

Table of Contents

The Use1401 library

Overview

The Use1401 software is a library of routines used to access the 1401 device driver under Windows (3.x and 9x), the Macintosh operating system and Windows NT (4, 2000 and upwards). The Windows versions can be built as 16-bit and 32-bit code, though nowadays only the 32-bit version is used. The library is designed to be as similar as possible in all its configurations; there is only one source file that builds the Mac and all Windows versions.

32-bit Windows programs

The 32-bit Windows library is built as a DLL, `USE1432.DLL`, which can be used by 32-bit software written in a variety of programming languages. Currently, CED only supports the use of the DLL from Microsoft C/C++, support for other languages such as Delphi 2 and Visual BASIC version 5 has been developed and tested by CED customers and is distributed by CED.

To use the library, include `USE1401.H` in your source file and link with `USE1432.LIB`. Depending upon what other Windows include files are in use, you may need to include `WINIOCTL.H` before `USE1401.H`. To run the programs, `USE1432.DLL` must be in a suitable position for Windows to find and load it. Windows NT looks for DLLs in the current directory first, next in the Windows directory, then in the Windows system directories and lastly in all directories listed in the `PATH` environment variable.

32-bit Windows programs that use Use1401 can run under Windows 3.x using `WIN32s` by using co-operative behaviour between `USE1401.DLL` and `USE1432.DLL,` function calls are automatically passed from the 32-bit library down to the 16-bit library and then on to the Windows device driver. The `USE1401.DLL` and `USE1432.DLL` libraries are both required for this co-operation to work. When running on Windows 9x or NT, `USE1432.DLL` communicates directly with the device driver.

16-bit Windows programs

The 16-bit Windows library is built as a Dynamically Linked Library (DLL), `USE1401.DLL`, which allows it to be used by 16-bit software written using a variety of programming languages. Currently, CED supports the use of the 16-bit DLL from Microsoft C/C++, Borland Pascal and Visual BASIC versions 3 and 4. Additional support, produced by CED customers, provides support for use from Borland C++. CED customers have also verified that the Borland Pascal support also works with Delphi 1.

To use the library from C/C++, include `USE1401.H` in your source file and link with `USE1401.LIB`. Specifics of use with other languages is provided after the function documentation. The function documentation includes function prototypes as seen by programmers using other languages. To run the programs, `USE1401.DLL` must be in a suitable position for Windows to find and load it. Windows looks for DLLs in the current directory first, next in the Windows directory, then in the Windows system directory and lastly in all directories listed in the `PATH` environment variable.

16-bit Windows programs that use Use1401 can be run under Windows NT using the NT 1401 device driver. This is done by `USE1401.DLL` passing function calls on to the 32-bit DLL, `USE1432.DLL`, which needs to be stored with `USE1401.DLL`. This is able to convert the 16-bit parameters to 32-bit and then invoke the NT driver. If running under Windows 3.x or 9x, `USE1401.DLL` communicates directly with the Windows device driver.

Nowadays the use of 16-bit Windows software is entirely obsolete and information relating to this is provided only for historical reference.

Macintosh programs

The Macintosh library is supplied as a linkable object file usable with the `MPW C` software development system. To use the library, include `USE1401.H` in your source file and link with `USE1401.C.O`. Note that the `MPW MAKE` system takes care of this automatically.

The Use1401 Macintosh support only functions on the original Macintosh OS and is not usable with OS-X and later versions of the Unix-derived OS. Information on the Use1401 Macintosh support is provided only for historical reference.

Use1401 C/C++ support

Introduction

Use1401 is a set of functions written in C to control the 1401 data acquisition system. The code may be compiled on a PC under DOS/Windows using Microsoft C/C++ version 7.0 or greater, or on a Macintosh using MPW C/C++. The interface file `USE1401.H` is a set of C declarations (not C++) providing a full set of error codes, structures and functions for communication with the 1401 device. All exported functions and error codes have the prefix `U14` to indicate their origin. For use with other Windows languages, only a replacement for `USE1401.H` is required, the actual library functions are not changed.

In order to build the Use1401 library the `MACHINE.H` file is required. This is a CED header file which contains mappings of functions which vary in name between the Mac and PC environments but which are functionally equivalent. It also provides some `#defines` which are not available on one machine or the other.

The Windows version of Use1401 is intended to be built into a DLL and includes all the functions necessary to initialise the DLL at load time.

Other languages

This chapter specifically covers the use of Use1401 from programs written in C and C++, and all examples are provided for C/C++. Some examples are also given for other languages; you can get further examples of using the Use1401 library in the example programs provided along with the Use1401 libraries and include files. Because the Use1401 library is the same for all languages, this chapter contains much information which is relevant to all languages, generally only the names used to specify a type of variable or record structure change.

While most of the Use1401 functionality is available with all platforms and languages, there are a few details which only apply to one machine or the other, or vary with the programming language used. The individual function documentation gives some information on this, chapters following the function documentation give more machine and language-specific details.

Function calls and U14API

All the functions in the unit are defined with the `U14API(xxx)` macro. This macro defines the function as a `PASCAL` type function with the return type `xxx`. The macros are defined such that for:

```
U14API(short)
```

the result would be:

`short FAR PASCAL`	for Microsoft C on the PC
`pascal short`	on a Macintosh.

This provides a transparent mapping of the `PASCAL` key word to the correct place for each compiler. Similar adjustments are possible for other languages.

Several routines require pointer parameters. When used on a Macintosh a simple 'address of' is sufficient to generate a pointer, but for 16-bit programs under Windows it is important to provide a FAR pointer. This will usually be done automatically, but if your compiler is incapable of this it is easily done by casting e.g. `(short FAR *)&` ....

Handles

With very few exceptions, all of the functions in the library require a 1401 handle parameter. The documentation of the individual functions assumes this parameter and do not mention it. A handle is a short integer (16 bits) which contains a value identifying the 1401 to be used, the handle value is returned by `U14Open1401` and used thereafter. If you were writing a program that used two 1401s you would call `U14Open1401` twice, receiving back two handles that would be used separately to communicate with the two 1401s opened. Multiple 1401s are only currently supported under Windows NT. Once `U14Close1401` has been called the handle value is invalid and can no longer be used.

Error codes

Most of the functions in Use1401 return an error code. The complete list of error codes is shown in the listing of `USE1401.H` at the end of this document. It is important that programmers use the constants provided when looking at error codes and not their values, as CED reserves the right to change the actual values used. The ‘no error’ value returned is also defined (as `U14ERR_NOERROR`), but it is reasonable to use it’s known value of zero, as below (assuming the function result is in `sErr`):

`if (sErr == 0)` or even `if (!sErr)`

The Use1401 library contains a function, `U14GetErrorString`, which will convert an error code returned by Use1401 into a human-readable string. You are advised to make use of this function if at all possible as it will ensure that, if new error codes are added to Use1401, your software can still display meaningful information about the error.

MINDRIVERMAJREV

This constant is defined as the minimum major revision level of the device driver that this library is able to work with.

Use1401 Functions

Access and status functions

These are functions concerned with opening the 1401 for access and determining the condition of the 1401.

U14Open1401

This function must be called before any other function is used to communicate with the 1401, it initialises the library and checks for the presence of the 1401. The return value is either the handle to be used for all further communication with the 1401, or a negative error code.

C/C++
```
U14API(short) U14Open1401(short n1401);
```

Visual BASIC
```
Function U14Open1401(ByVal n1401 As Integer) As Integer
```

Borland Pascal/Delphi
```
Function U14Open1401(n1401:INTEGER):INTEGER;
```

The `n1401` parameter is used to select which 1401 is to be opened. If `n1401` is zero, the first available 1401 is opened. Other values of `n1401` specify the number of the 1401 to be opened, usually this is from 1 to 3. The current software for Windows and the Macintosh only supports a single 1401, a value of zero is recommended.

The `U14Open1401` function performs the following tests to determine if the 1401 is present and ready for use:

- It checks that the 1401 device driver is present.
- The 1401 device driver input and output buffers are emptied.
- Resets the 1401 using `U14Reset1401`, only if necessary.
- It requests the device driver revision level and fails if it is too old to be used.
- The 1401 status is requested from the driver.

A typical program to use the 1401 starts and finishes with code which resembles the following:

C/C++
```
#include <use1401.h>
int main( )
{
    short   sErr, hand;
    sErr=U14Open1401(0);
    if (sErr >= 0)                       /* all ok so carry on */
    {
        hand = sErr;          /* save the handle for future use */
        .....                       /* Use the 1401 as required */
        sErr=U14Close1401(hand);      /* check for close errors */
    }
    else
        return(1);                /* return with error if fails */
}
```

Visual BASIC
```
Dim sErr as Integer        'local
Dim handle1401 as Integer  'this could be in main declarations

sErr = U14Open1401(0)
If (sErr >= 0) Then        'all ok so carry on
  handle1401 = sErr        'save handle for future use
  .....                    'use the 1401 as required
  serr = U14Close1401(handle1401) 'close and check
Endif
```

Borland Pascal/Delphi

```
USES Use1401;

BEGIN
VAR sErr:INTEGER;
    hand:INTEGER;

sErr:=U14Open1401(0);
IF (sErr >= 0)                    { all ok so carry on }
   THEN BEGIN
        hand := sErr;      { save handle for future use }
         .....
        sErr := U14Close1401(hand);    { close and check }
        END;
END;
```

U14Close1401

This function should be called when an application has finished using the 1401. Once this function has been called, the 1401 can no longer be used and is available for use by other applications. See `U14Open1401` for an example of its use.

C/C++
```
U14API(short) U14Close1401(short hand);
```
Visual BASIC
```
Function U14Close1401(ByVal hand As Integer) As Integer
```
Borland Pascal/Delphi
```
Function U14Close1401(hand:INTEGER):INTEGER;
```

The return value is zero or an error code.

U14CloseAll

This procedure, which only does anything in the 32-bit Windows build, is intended as a last-ditch ‘crash exit’ attempt to tidy up the Use1401 library by closing all 1401s that are currently open – even if the software does not know what 1401s are open or have access to the 1401 handles.

C/C++
```
U14API(void) U14CloseAll(void);
```
Visual BASIC
```
Sub U14CloseAll()
```
Borland Pascal/Delphi
```
Procedure U14CloseAll;
```

U14Reset1401

This function performs a hardware reset of the 1401. This will stop any running commands and flush any input or output from the I/O buffers. Any DMA in progress is stopped. All loaded commands remain intact within the 1401. You can use this to get the 1401 out of a "hung" situation (for example when a dedicated 1401 command is waiting for a trigger which never comes), or for general-purpose initialisation.

C/C++
```
U14API(short) U14Reset1401(short hand);
```
Visual BASIC
```
Function U14Reset1401(ByVal hand As Integer) As Integer
```
Borland Pascal/Delphi
```
Function U14Reset1401(hand:INTEGER):INTEGER;
```

The return value is zero or a negative error code.

U14ForceReset

This function ensures that the next call to `U14Reset1401` will actually perform a reset of the 1401. We have found that, with some flavours of Windows, performing a 1401 reset over the USB bus can cause the 1401 device driver to lock up very occasionally. To avoid this the USB driver checks the 1401 state in `U14Reset1401` and, if everything is OK and the 1401 is responding OK, does not actually perform the reset. Use this function to override this behaviour and cause an actual reset on the next call to `U14Reset1401`.

C/C++
```
U14API(short) U14ForceReset(short hand);
```
Visual BASIC
```
Function U14ForceReset(ByVal hand As Integer) As Integer
```
Borland Pascal/Delphi
```
Function U14ForceReset(hand:INTEGER):INTEGER;
```

The return value is zero or a negative error code.

U14TypeOf1401

This function is used to determine the type of 1401 in use. It will only return a valid result after `U14Open1401` has been used successfully.

C/C++
```
U14API(short) U14TypeOf1401(short hand);
```
Visual BASIC
```
Function U14TypeOf1401(ByVal hand As Integer) As Integer
```
Borland Pascal/Delphi
```
Function U14TypeOf1401(hand:INTEGER):INTEGER;
```

The return value is one of the defined constants for 1401 types (see Use1401.H), `U14TYPEUNKNOWN`, or a negative error code.

U14NameOf1401

This function is used to provide the name of the 1401 (for example Power1401 mk II) in use so that applications can display the 1401 type without having to include (and update) a table of 1401 devices. It will only return a valid result after `U14Open1401` has been used successfully.

C/C++
```
U14API(short) U14NameOf1401(short hand,
                            LPSTR lpBuf, WORD wMax);
```
Visual BASIC
```
Function U14NameOf1401(ByVal hand As Integer,
                       ByVal lpStr As String,
                       ByVal wMax As Integer) As Integer
```
Borland Pascal/Delphi
```
Function U14NameOf1401(hand:INTEGER; lpStr:Pchar;
                       wMax:WORD):INTEGER;
```

`lpStr`	Pointer to a character array in which to store the 1401 name.
`wMax`	The maximum number of characters the buffer can hold including a terminating null character.

The return value is zero or a negative error code.

U14StateOf1401

This function returns a code indicating the state of the 1401. The state information is updated when the 1401 is opened, or whenever `U14Reset1401` is used. The return values are standard error codes.

C/C++
```
U14API(short) U14StateOf1401(short hand);
```

Visual BASIC
```
Function U14StateOf1401(ByVal hand As Integer) As Integer
```

Borland Pascal/Delphi
```
Function U14StateOf1401(hand:INTEGER):INTEGER;
```

The most likely return values are `U14ERR_NOERROR` (indicating all is OK) or:

```
U14ERR_OFF           The 1401 is turned off
U14ERR_NC            The 1401 is not connected
U14ERR_ILL           The 1401 is unwell
U14ERR_NO1401DRIV    The 1401 device driver is not loaded
U14ERR_DRIVTOOOLD    The 1401 device driver is too old
U14ERR_NOIF          There is no interface card
U14ERR_TIME          The 1401 timed out
U14ERR_BADSW         The interface card switches are set badly
U14ERR_LOCKFAIL      Memory locking for block transfers failed
U14ERR_NOINT         Could not get the interrupt channel wanted
U14ERR_INUSE         The 1401 is in use by another application
U14ERR_NODMA         DMA was wanted but could not get the channel
                     (results in programmed transfers being used)
```

Other return values are possible if there is a problem communicating with the device driver.

U14GetUserMemorySize

This function stores the 1401 user memory size, in bytes, in the location pointed to by the parameter. The 1401 user memory is the memory available for use by 1401 commands, it runs from address zero to `(size-1)`. Returns zero or an error code.

C/C++
```
U14API(short) U14GetUserMemorySize(short hand,
                    long FAR * lpSize);
```

Visual BASIC
```
Function U14GetUserMemorySize(ByVal hand As Integer,
                    size As Long) As Integer
```

Borland Pascal/Delphi
```
Function U14GetUserMemorySize(hand:INTEGER;
                    VAR size:LONGINT):INTEGER;
```

`lpSize`	points to a long that will be updated with the memory size.
`size`	a long integer passed by reference that will be updated with the memory size.

U14LastErrCode

This function returns the last error code from a call to the library for the specified 1401.

C/C++
```
U14API(short) U14LastErrCode(short hand);
```

Visual BASIC
```
Function U14LastErrCode(ByVal hand As Integer) As Integer
```

Borland Pascal/Delphi
```
Function U14LastErrCode(hand:INTEGER):INTEGER;
```

U14GetErrorString

This function converts a Use1401 error code to a human-readable string.

C/C++

```
U14API(void) U14GetErrorString(short sErr, LPSTR lpStr,
                        WORD wMax);
```

Visual BASIC

```
Function U14GetErrorString(ByVal sErr As Integer,
                        ByVal lpStr As String,
                        ByVal wMax As Integer) As Integer
```

Borland Pascal/Delphi

```
Function U14GetErrorString(hand:INTEGER; lpStr:Pchar;
                        wMax:WORD):INTEGER;
```

`lpStr`	Pointer to a character array in which to store the error description.
`wMax`	The maximum number of characters the buffer can hold including a terminating null character.

U14DriverVersion

This function returns the driver version. The high word of the function result is the major revision level and the low word is the minor revision level. A 1401 handle is not required, allowing the driver to be checked before `U14Open1401` is called. If the high word of the result (the major revision level) is less than the constant `MINDRIVERMAJORREV` then `U14Open1401` will always fail.

C/C++

```
U14API(long) U14DriverVersion( void );
```

Visual BASIC

```
Function U14DriverVersion() As Long
```

Borland Pascal/Delphi

```
Function U14DriverVersion:LONGINT;
```

U14DriverType

This function returns flags indicating the type of 1401 driver and 1401 interface card in use. A 1401 handle is not required, allowing the driver to be checked before `U14Open1401` is called.

C/C++

```
U14API(long) U14DriverType( void );
```

Visual BASIC

```
Function U14DriverType() As Long
```

Borland Pascal/Delphi

```
Function U14DriverType:LONGINT;
```

The return value is a negative error code, or a value indicating the type of device driver:

0	Windows using an ISA interface card
1	Windows using a PCI interface card
2	Windows using the USB interface

U14TransferFlags

This function returns flags providing information on the block transfer and other device driver capabilities available. Because the 1401 support software does a good job of adjusting to the capabilities available, most applications will operate identically regardless of the flag values, but there are occasions where this information is needed. A negative error code can be returned.

C/C++	`U14API(short) U14TransferFlags(short hand);`
Visual BASIC	`Function U14TransferFlags(ByVal hand As Integer) As Integer`
Borland Pascal/Delphi	`Function U14ransferFlags(hand:INTEGER):INTEGER;`

The value returned has bits set to indicate various capabilities. There are defined constants in `USE1401.H` set to the values of these bits:

`U14TF_USEDMA (bit 0)`	This bit is set if DMA will be used in block transfers. If clear, then programmed I/O will be used.
`U14TF_MULTIA (bit 1)`	This bit is set if multiple transfer areas can be used. If clear, then only transfer area zero is available.
`U14TF_FIFO (bit 2)`	This bit is set if an ISA interface card with FIFO buffer and demand-mode DMA capabilities is in use.
`U14TF_USB2 (bit 3)`	This bit is set if a USB interface running at USB2 speeds is in use.
`U14TF_NOTIFY (bit 4)`	This bit is set if the device driver is able to generate block transfer notification events (see the documentation on the `U14SetTransferEvent` function).
`U14TF_SHORT (bit 5)`	This bit is set if a PCI card interface that is capable of short PCI bus cycles is in use.
`U14TF_PCI2 (bit 6)`	This bit is set if a PCI card interface using the newer 1401-70 PCI interface card is in use.
`U14TF_CIRCTF (bit 7)`	This bit is set if the device driver supports circular-mode block transfers to the host PC (see the documentation on the `U14SetCircular` function and following functions)
`U14TF_DIAG (bit 8)`	This bit is set if the device driver supports the diagnostics and debug functions.
`U14TF_CIRC14 (bit 9)`	This bit is set if the device driver supports circular-mode block transfers to the 1401. At the time of writing circular-mode transfers to the 1401 have not been implemented on any device driver – and may never be.

U14MonitorRev

This function returns the 1401 monitor revision. Most software should not need to know this, but if your application requires that a given level of 1401 monitor ROM is present, then you can use this function to check for problems and alert the user. The value returned is the low part of the monitor revision level plus the high part of the revision times 1000. Generally, the high part of the revision does not vary. For example the current micro1401 monitor ROM is revision 3.13, for which this function would return the value 3013.

C/C++	`U14API(long) U14MonitorRev(short hand);`
Visual BASIC	`Function U14MonitorRev(ByVal hand As Integer) As Long`
Borland Pascal/Delphi	`Function U14DriverVersion(hand:INTEGER):LONGINT;`

Character transfer functions

These functions are all concerned with the transfer of text characters between the 1401 and the host computer. They include functions to send and receive strings, testing the state of character transfers and controlling timeouts.

All character communication with the 1401 is buffered (the characters are stored in arrays both before and after transmission) to allow for efficient communication and to avoid an application hanging while it waits for a string to be transmitted.

U14SendString

This function is sends a string to the 1401. If the buffers are full it waits (using the timeout) until space is available. The string must be terminated 'C-style', with a zero character. No terminator character is appended, so you will have to take care to use properly terminated strings. The return value is zero or an error code.

C/C++

```
U14API(short) U14SendString(short hand, LPSTR pStr);
```

Visual BASIC

```
Function U14SendString(ByVal hand As Integer,
                       ByVal pStr As String) As Integer
```

Borland Pascal/Delphi

```
Function U14SendString(hand:INTEGER; pStr:PChar):INTEGER;
```

`pStr`	Pointer to the string to be sent.

For example:

C/C++

```
short sErr;

sErr = U14SendString(hand, "ERR;");      /* ask for 1401 errors */
if (sErr == 0)                          /* If the string went OK */
{
    long   alVal[2];
    sErr = U14LongsFrom1401(hand, alVal, 2);
    if (sErr != 2)             /* Check we got two numbers back */
        ...
}
```

Visual BASIC

```
sErr As Integer
lVals(2) As Long

sErr = U14SendString(hand, "ERR;")          ' ask for 1401 errors
If (sErr = 0) Then
  sErr = U14LongsFrom1401(hand, lVals, 2)    ' Read back response
  If (sErr <> 2)                            ' Check we got 2 numbers
    ...
  End If
End If
```

Borland Pascal/Delphi

```
TYPE sErr:INTEGER;
     str:STRING;                { String to build up commands in }
     lVals:ARRAY[0..1] OF LONGINT;

str := 'ERR;'+CHR(0);             { build null terminated string }
sErr := U14SendString(hand, @str[1]);    { how to use string var }
IF (sErr = 0) THEN
   BEGIN
   sErr := U14LongsFrom1401(hand, @lVals, 2);
   IF (sErr <> 2) THEN                { Check we got two numbers }
     ...
   END;
```

U14GetString

This function reads the next line of input from the 1401 into a string variable, terminated with a `NULL` character. In normal circumstances, the last character in the string will be the terminating carriage-return. If a line of input is not immediately available, the function waits for data to arrive until it times out. It stops reading when a carriage return is encountered or when the maximum number of characters have been read. If there are several replies in the input buffer, several calls to `U14GetString` are needed to read them all. All commas in the string are converted to spaces.

C/C++

```
U14API(short) U14GetString(short hand, LPSTR pStr, WORD wMax);
```

Visual BASIC

```
Function U14GetString(ByVal hand As Integer,
                      ByVal pStr As String,
                      ByVal wMax As Integer) As Integer
```

Borland Pascal/Delphi

```
Function U14GetString(hand:INTEGER; pStr:Pchar;
                      wMax:WORD):INTEGER;
```

`pStr`	Pointer to a character array in which to store the data.
`wMax`	The maximum number of characters the buffer can hold including the terminating null character. The function stops reading when this number of characters have been read. The function fails if the maximum length is less than 2 as this would leave no room for any characters.

U14LongsFrom1401

This routine waits for a period defined by `U14SetTimeout` while trying to get a reply string from the 1401. If a reply is received it attempts to convert the string into longs using `U14StrToLongs`.

C/C++

```
U14API(short) U14LongsFrom1401(short hand, long FAR * palNums,
                      short sMax);
```

Visual BASIC

```
Function U14LongsFrom1401(ByVal hand As Integer, palNums As Any,
                      ByVal sMax As Integer) As Integer
```

Borland Pascal/Delphi

```
Function U14LongsFrom1401(hand:INTEGER; palNums:TpNums;
                      sMax:INTEGER):INTEGER;
```

`palNums`	Pointer to an array of longs to be filled in.
`sMax`	The size of the array.

The return value is the number of longs retrieved or an error code. If no values could be retrieved from the reply string, the function returns zero. For an example of use, see `U14SendString`.

U14StrToLongs

This is a utility routine which takes a pointer to a string and a pointer to an array of longs. The string is converted into as many longs as fit into the array, stopping when data from the string is used up. This function is used internally by `U14LongsFrom1401`.

C/C++

```
U14API(short) U14StrToLongs(LPSTR lpStr, long FAR *palNums,
                        short sMax);
```

Visual BASIC

```
Function U14StrToLongs(ByVal lpStr As String, palNums As Any,
                        ByVal sMax As Integer) As Integer
```

Borland Pascal/Delphi

```
Function U14StrToLongs(lpStr:PChar; palNums:TpNums;
                        sMax:INTEGER):INTEGER;
```

`lpStr`	Pointer to null-terminated string containing text.
`palNums`	Pointer to an array of longs to be filled in.
`sMax`	The size of the array.

The return value is the number of longs placed in the array or a negative error code. If no values could be retrieved from the string, the function returns zero.

U14SendChar

This function sends the single character `cChar` to the 1401, waiting (with timeouts) if the buffer is full. Returns zero or an error code.

C/C++

```
U14API(short) U14SendChar(short hand, char cChar);
```

Borland Pascal/Delphi

```
Function U14SendChar(hand:INTEGER; cChar:CHAR):INTEGER;
```

U14GetChar

This function fetches a single character from the 1401, waiting (with timeouts) for a character if one is not available, and places it in the variable pointed to by `lpcChar`. For Pascal use, the character is placed in the `cChar` variable is passed by reference. Returns zero or an error code.

C/C++

```
U14API(short) U14GetChar(short hand, LPSTR lpcChar);
```

Borland Pascal/Delphi

```
Function U14GetChar(hand:INTEGER; VAR cChar:CHAR):INTEGER;
```

U14Stat1401

This function is used to check for characters waiting to be read from the 1401.

C/C++

```
U14API(short) U14Stat1401(short hand);
```

Visual BASIC

```
Function U14Stat1401(ByVal hand As Integer) As Integer
```

Borland Pascal/Delphi

```
Function U14Stat1401(hand:INTEGER):INTEGER;
```

The return value is greater than zero if there is at least one line of pending input from the 1401 and zero if there is not. A negative error code may also be returned. The `U14Stat1401` function is normally used to allow other host activities to take place while we are waiting for the 1401 to complete some task. The trick is to get the 1401 to send some input to the host after it has finished the task so that `U14Stat1401` can be used to detect it. For example:

C/C++

```
long    alErr[2];                       /* to store our results */

U14SendString(hand, "ADCMEM,F,2,0,1000,0,0,CT,10,10;ERR;");
while (!U14Stat1401(hand))              /* while no response .. */
    DoIdle();                        /* ..do some idle operation */
U14LongsFrom1401(hand, (long FAR *)&alErr,2);     /* get errors
*/
```

Visual BASIC

```
Static alErr(2) As Long                     ' to store our results
Dim sErr As Integer

U14SendString(hand,"ADCMEM,F,2,0,1000,0,0,CT,10,10;ERR;")
While (U14Stat1401(hand) = 0)                   ' response ready?
   DoIdle();                         ' No response, do something
Wend
sErr=U14LongsFrom1401(handle1401, alErr, 2)         ' get errors
```

Borland Pascal/Delphi

```
alErr:ARRAY [0..1] OF LONGINT;          { to store our results }

sErr:=U14SendString(hand, 'ADCMEM,F,2,0,1000,0,0,CT,10,10;ERR;');
IF sErr=0
  THEN BEGIN
       WHILE (U14Stat1401(hand) = 0) DO { while no response .. }
           DoIdle;                  { ..do some idle operation }
       U14LongsFrom1401(hand, @alErr, 2);        { get errors }
       END;
```

In this example we have asked the 1401 to wait for a trigger, then to sample some data and finally to return the error status. As the ADCMEM command is run in dedicated mode using the F option there is no response from the ERR command until the sampling is over. In this case we make use of this waiting time to perform some idle time routines.

U14LineCount

This function returns a count of the current number of lines waiting in the input buffer to be read or a negative error code.

C/C++	`U14API(short) U14LineCount(short hand);`
Visual BASIC	`Function U14LineCount(ByVal hand As Integer) As Integer`
Borland Pascal/Delphi	`Function U14LineCount(hand:INTEGER):INTEGER;`

U14CharCount

This function returns a count of the current number of characters waiting in the input buffer to be read or a negative error code.

C/C++	`U14API(short) U14CharCount(short hand);`
Visual BASIC	`Function U14CharCount(ByVal hand As Integer) As Integer`
Borland Pascal/Delphi	`Function U14CharCount(hand:INTEGER):INTEGER;`

U14SetTimeout

Set the timeout period when waiting to send or receive characters from the 1401. The default timeout period is 3 seconds. There is no return value.

C/C++
```
U14API(void) U14SetTimeout(short hand, long lTimeOut);
```
Visual BASIC
```
Sub U14SetTimeout(ByVal hand As Integer,
                    ByVal lTimeOut As Long)
```
Borland Pascal/Delphi
```
Procedure U14SetTimeout(hand:INTEGER; lTimeOut:LONGINT);
```

`lTimeOut` The required timeout, in 1/60ths of a second.

U14GetTimeout

This function returns the current timeout period in 1/60ths of a second. This function cannot return an error.

C/C++
```
U14API(long) U14GetTimeout(short hand);
```
Visual BASIC
```
Function U14GetTimeout(ByVal hand As Integer) As Long
```
Borland Pascal/Delphi
```
Function U14GetTimeout(hand:INTEGER):LONGINT;
```

U14WhenToTimeOut

This function returns the time, in system ticks, at which the 1401 will time out. It does this by getting the current time, in system ticks, and adding the timeout period currently set. This function is used internally by the Use1401 functions and is made available for use elsewhere. Use this function with `U14PassedTime`.

C/C++
```
U14API(long) U14WhenToTimeOut(short hand);
```
Visual BASIC
```
Function U14WhenToTimeOut(ByVal hand As Integer) As Long
```
Borland Pascal/Delphi
```
Function U14WhenToTimeOut(hand:INTEGER):LONGINT;
```

U14PassedTime

This function returns non-zero if the time provided (presumed returned by `U14WhenToTimeOut`), has been passed, otherwise zero. This function cannot return an error.

C/C++
```
U14API(short) U14PassedTime(long lTime);
```
Visual BASIC
```
Function U14GetTimeout(ByVal lTime As Long) As Integer
```
Borland Pascal/Delphi
```
Function U14GetTimeout(lTime:LONGINT):INTEGER;
```

U14OutBufSpace

This function returns the number of characters of space available in the output buffer or a negative error code. A return value of 0 means no room is available in the output buffer.

C/C++
```
U14API(short) U14OutBufSpace(short hand);
```
Visual BASIC
```
Function U14OutBufSpace(ByVal hand As Integer) As Integer
```
Borland Pascal/Delphi
```
Function U14OutBufSpace(hand:INTEGER):INTEGER;
```

U14KillIO1401

This function flushes out the character input and output buffers in the device driver. This is not generally required, except when attempting to recover from an error condition.

C/C++	`U14API(short) U14KillIO1401(short hand);`
Visual BASIC	`Function U14KillIO1401(ByVal hand As Integer) As Integer`
Borland Pascal/Delphi	`Function U14KillIO1401(hand:INTEGER):INTEGER;`

Command loading functions

These functions are concerned with the loading of commands into the 1401. Please note that the `U14LdCmd` function is provided for special purposes only; normal applications should use the `U14Ld` function.

U14Ld

This function loads a list of commands into the 1401 if they are not already loaded. It automatically adds the correct file extension (`.CMD`, `.GXC`, `.ARM` or `.ARN`) to the commands in the list depending on the type of 1401 connected. This function uses block transfers (below) to transfer the command data into the 1401. Block transfer area zero is used, so this function will fail if area zero is already in use. Note that loading a command into the 1401 causes all currently loaded commands to be reset to a default state.

C/C++

```
U14API(long) U14Ld(short hand, LPSTR vl, LPSTR str);
```

Visual BASIC

```
Function U14Ld(ByVal hand As Integer, ByVal vl As String
                      ByVal command As String) As Long
```

Borland Pascal/Delphi

```
Function U14Ld(hand:INTEGER; vl:PChar; command:PChar):LONGINT;
```

`vl` This parameter is only needed on a PC and should be set to a null string or `NULL` on a Macintosh. It is a pointer to a string holding a directory path to be searched for 1401 commands. The path should end with a backslash '\' or a colon ':' as the command name is directly appended to form a file name.

`command` This is a string, or pointer to a string, holding the list of 1401 commands to be loaded. The command names are separated by commas and may not include any spaces. Upper case is preferred for command names for compatibility with old 1401s.

For Macintosh software, the commands are first searched for in the application resource, followed by resources in the `1401Commands` file in the system folder. For Windows software, the function searches for the commands in a number of directories in turn:

- The directory specified in the `vl` parameter, if non-blank.
- The `1401` directory in the application directory (32-bit library only)
- The directory specified in the `1401DIR` environment variable, if it exists.
- The `\1401` directory on the current drive.

The function returns an error code indicating the success or failure of the operation. The error code is formed from the command number (starting at 1) in the list in the high word and a normal Use1401 error code in the low word. For example:

```
short   sCmd, sErr;
long    lRetVal;
lRetVal=U14Ld(hand, "","ADCMEM,SS2");         /* load commands */
sErr=(short)(lRetVal & 0xFFFF);              /* get error code */
if (sErr != U14ERR_NOERROR)           /* see if error occurred */
{
    sCmd=(short)(lRetVal>>16);      /* index of failed command */
    switch (sCmd)
    {
        case 1:               /* ... ADCMEM failed to load ... */
        break;
        case 2:                  /* ... SS2 failed to load ... */
        break;
    }
}
else
                 /* ... All commands loaded successfully ... */
```

U14LdCmd

This function loads a single command into the 1401, even if it is already loaded. It returns the normal error codes. There are few occasions when this is required, so we suggest that the U14Ld function is used in all application software. Note that loading a command into the 1401 causes all currently loaded commands to be reset to a default state.

C/C++

```
U14API(short) U14LdCmd(short hand, LPSTR command);
```

Visual BASIC

```
Function U14LdCmd(ByVal hand As Integer,
                                ByVal command As String) As Integer
```

Delphi

```
Function U14LdCmd(hand:INTEGER; command:PChar):INTEGER;
```

`command`	This is the name of the file to be loaded. On a PC, if there is no `.CMD`, `.GXC`, `.ARM` or `.ARN` on the end of the name, the appropriate file extension is added. See the `U14Ld` documentation for details of where the function looks for the command.

`U14LdCmd` uses block transfers to transfer the command data into the 1401. Block transfer area zero is used, so this function will fail if area zero is already in use.

Block transfer functions

These functions are all concerned with the transfer of blocks of data between the 1401 and the host computer. These transfers normally use DMA mechanisms, but can operate using programmed I/O. In general, application programmers outside CED will only require the `U14ToHost` and `U14To1401` functions, the rest of the block transfer functions are only needed for more demanding programs.

U14ToHost & U14To1401

These two functions are used to pass blocks of data between memory buffers in the host computer and the 1401. The functions call `U14SetTransArea` and `U14UnSetTransfer` as required. Block transfer area number zero is used, so these functions will fail if area zero is already in use.

C/C++

```
U14API(short) U14ToHost(short hand, LPSTR lpHost, DWORD dwSz,
                    DWORD dwA1401, short eSz);

U14API(short) U14To1401(short hand, LPSTR lpHost, DWORD dwSz,
                    DWORD dwA1401, short eSz);
```

Visual BASIC

```
Function U14ToHost(ByVal hand As Integer, lpHost As Any,
                    ByVal dwSz As Long, ByVal dwA1401 As Long,
                    ByVal eSz As Integer) As Integer

Function U14To1401(ByVal hand As Integer, lpHost As Any,
                    ByVal dwSz As Long, ByVal dwA1401 As Long,
                    ByVal eSz As Integer) As Integer
```

Borland Pascal/Delphi

```
FUNCTION U14ToHost(hand:INTEGER; lpHost:POINTER; dwSz:DWORD;
                    dwA1401:DWORD; eSz:INTEGER):INTEGER;

FUNCTION U14To1401(hand:INTEGER;  lpHost:POINTER; dwSz:DWORD;
                    dwA1401:DWORD; eSz:INTEGER):INTEGER;
```

`lpHost`	A pointer to the host buffer for the transfer. This region is where the data is put by `U14ToHost` or taken from by `U14To1401`.
`dwSz`	The number of data bytes to transfer. It is crucial that this number does not exceed the size of the buffer pointed at by `lpHost`. If it does a memory protection error will probably occur, if a protection error does not occur the transfer may write all over memory which you do not intend to change. The maximum allowed transfer size is 64 Kbytes.
`dwA1401`	The address of the start of the 1401 buffer. This is given as an unsigned long to allow access to the full address range in the 1401*plus*, micro1401 or Power1401.
`eSz`	This parameter is only relevant on the Macintosh and should be set to 0 on the PC. This defines what sort of data is to be transferred between the 1401 and the host machine, see the Macintosh specifics for more information.

The functions return a negative error code if there is a problem and U14ERR_NOERROR if all is successful. For example, to collect a buffer of 1024 two-byte data points from the start of the 1401 memory into an array:

```
short    asBuffer[1024];                /* buffer for our data */
short    sErr;                                /* error return */

sErr = U14ToHost(hand, (LPSTR)asBuffer, 2048, 0, 0);
```

U14SetTransArea

This function prepares an area of memory in the host computer for block transfers between the 1401 and the host computer. If necessary, the memory is locked so that it cannot be moved. The function returns an error code.

C/C++

```
U14API(short) U14SetTransArea(short hand, WORD wArea,
                        void FAR * lpvBuff,
                        DWORD dwLength, short eSz);
```

Visual BASIC

```
Function U14SetTransArea(ByVal hand As Integer,
                        ByVal wArea As Integer, lpvBuff As Any,
                        ByVal dwLength As Long,
                        ByVal eSz As Integer) As Integer
```

Borland Pascal/Delphi

```
Function U14SetTransArea(hand:INTEGER; wArea:WORD;
                        lpvBuff:POINTER; dwLength:DWORD;
                        eSz:INTEGER):INTEGER;
```

`wArea`	The number of the transfer area to set up. This must be a value in the range 0 to 7. Only area 0 is available for use with a standard 1401 but with later 1401 types areas from 0 to 7 are available. You also need an up-to-date device driver to use multiple areas, so your software should check the value returned by `U14TransferFlags` to see if multiple transfer areas can be used.
`lpvBuff`	The address of the buffer that data is transferred from or into, or (for Visual BASIC) the start of the buffer.
`dwLength`	The length of the transfer buffer, in bytes. The maximum allowed length is 1 Mbytes, but older device drivers and versions of Use1401 have a limit of 256 Kbytes.
`eSz`	The size of the data items being transferred. This is only needed on the Macintosh, as it is used to set the byte swapping on the interface card. On the PC, set this value to zero.

This function is only needed when you are using a command that uses `DMA` to transfer data to or from host memory such as `TOHOST, TO1401` or `CLOAD`. In fact these three commands are taken care of by the `U14ToHost, U14To1401` and `U14Ld` functions respectively, all of which call `U14SetTransArea` internally. Other commands that use block transfers include `PERI32`, which transfers and ‘unwraps’ data from it’s circular buffer and `MRHOST,` which transfers directly to and from mass RAM in the 1401. There are also a number of specialised commands that use block transfers.

An example of using the `U14SetTransArea` function with the `TOHOST` command is shown below:

```
char       acBuff[10000];              /* buffer for our data   */
long       er[2];                  /* for decoding 1401 replies */
short      sErr;                      /* for collecting errors  */

sErr=U14SetTransArea(hand, 0, &acBuf, 10000, 0);/* Mac use eSz */
if (sErr == U14ERR_NOERROR)                      /* no error   */
{
    U14SendString(hand, "TOHOST,0,10000,0;ERR;");
    sErr=U14LongsFrom1401(hand, (long FAR *)er,2);

    U14UnSetTransfer(hand, 0);                 /* free area now */
}
else
          ... here for SetTransArea error ...
```

U14SetTransferEvent

This function, which is only functional in the 32-bit Windows build of the library, specifies an event which will be set into the signalled state by the device driver whenever a specified block transfer is completed. Applications can use such an event to detect that action is required or to wake up a worker thread that handles incoming data. The function returns an error code.

C/C++

```
U14API(short) U14SetTransferEvent(short hand, WORD wArea,
                    long hEvent, BOOL bToHost
                    DWORD dwStart, DWORD dwLength);
```

Visual BASIC

```
Function U14SetTransArea(ByVal hand As Integer,
                    ByVal wArea As Integer,
                    ByVal hEvent As Long,
                    ByVal bToHost As Long
                    ByVal dwStart As Long,
                    ByVal dwLength As Long) As Integer
```

Borland Pascal/Delphi

```
Function U14SetTransArea(hand:INTEGER; wArea:WORD;
                    hEvent:LONGINT;
                    bToHost:LONGINT;
                    dwStart:DWORD;
                    dwLength:DWORD):INTEGER;
```

`wArea`	The number of the transfer area to which this event applies, from 0 to 7. Only one event can be associated with any given transfer area, the event must be set after U14SetTransArea or U14SetCircular is called and before the transfer area is used (any transfers occurring before the event is set will be ignored).
`hEvent`	The Win32 event handle, cast to a long (32-bit) integer.
`bToHost`	This specifies the direction of transfer that can set the event. Set this to 1 for transfers to the host PC, 0 for transfers to the 1401.
`dwStart`	This, along with `dwLength`, specifies the relevant area of the transfer buffer.
`dwLength`	This, along with `dwLength`, specifies the relevant area of the transfer buffer.

This function registers an event (a `Win32` object) that will can be set to the signalled state to indicate that application action is required. The event will be set whenever a block transfer in the set direction finishes. For non-circular transfers the transfer must overlap the section of the transfer buffer specified by `dwStart` and `dwLength` by at least 1 byte, for circular transfers the amount of data currently available in the circular buffering system (as reported by `GetCircBlk`) must be at least `dwLength` bytes.

A trivial example of using the `U14SetTransferEvent in Visual C++` is shown below:

```
HANDLE hEvent = CreateEvent(NULL, FALSE, FALSE, NULL);
if (!hEvent)                     // NULL returned if we failed
{
     Error handling here
}
i1401Err = U14SetTransferEvent(hand, 0, (long)hEvent, TRUE, 0, 1);
if (i1401Err != U14ERR_NOERROR)  // error check
{
    Error handling here
}
```

U14UnSetTransfer

This function releases a transfer area so that it is available for use elsewhere. If it is not used then further `U14SetTransArea` or `U14SetCircular` calls with the same area number will fail. It also unlocks the memory locked by `U14SetTransArea`. Locking memory reduces system performance, therefore it is important to only set up a transfer area when it is needed and close it down once it has been used. See above for an example of use.

C/C++

```
U14API(short) U14UnSetTransfer(short hand, WORD wArea);
```

Visual BASIC

```
Function U14UnSetTransfer(ByVal hand As Integer,
                    ByVal wArea As Integer) As Integer
```

Borland Pascal/Delphi

```
Function U14UnSetTransfer(hand:INTEGER; wArea:WORD):INTEGER;
```

`wArea`	The number of the transfer area to release. This must be a value in the range 0 to 7, and the same value as used in the corresponding `U14SetTransArea` call.

U14SetCircular

This function , which is only functional in the 32-bit Windows build of Use1401, prepares an area of memory in the host computer for circular-mode block transfers between the 1401 and the host computer. If necessary, the memory is locked so that it cannot be moved. The function returns an error code.

C/C++

```
U14API(short) U14SetCircular(short hand, WORD wArea,
                    BOOL bToHost,
                    void FAR * lpvBuff,
                    DWORD dwLength);
```

Visual BASIC

```
Function U14SetCircular(ByVal hand As Integer,
                    ByVal wArea As Integer,
                    ByVal bToHost As Long
                    lpvBuff As Any,
                    ByVal dwLength As Long) As Integer
```

Borland Pascal/Delphi

```
Function U14SetCircular(hand:INTEGER; wArea:WORD;
                    bToHost:LONGINT
                    lpvBuff:POINTER;
                    dwLength:DWORD):INTEGER;
```

`wArea`	The number of the transfer area to set up. This must be a value in the range 0 to 7. Only area 0 is available for use with a standard 1401 but with later 1401 types areas from 0 to 7 are available. You need an up-to-date device driver to use circular transfer mode, so your software should check the value returned by `U14TransferFlags` to see if circular transfers can be used.
`bToHost`	Set this value to 1 to use circular transfers in transfers from the 1401 to the host PC, or use 0 for transfers to the 1401. At the present time only circular transfers from the 1401 to the host PC are supported. The flags returned by `U14TransferFlags` includes a separate flag for circular transfers to the 1401 so that this can be checked-for.
`lpvBuff`	The address of the buffer that data is transferred from or into, or (for Visual BASIC) the start of the buffer.
`dwLength`	The length of the transfer buffer, in bytes. The maximum allowed length is 1 Mbytes, but older device drivers and versions of Use1401 have a limit of 256 Kbytes.

This function is used to set up a block transfer area in the same manner as `U14SetTransArea`, but for circular transfers. In circular transfers from the 1401 data sent to the host is automatically stored in the circular buffer, the application code uses `U14GetCircBlk` to find out what 1401 data is available and `U14FreeCircBlk` to tell the 1401 device driver that an area filled with 1401 data is available for use again. Circular transfer mode is a specialised form of block data transfer designed for efficient high-bandwidth data throughput – it is not intended for use by programmers outside CED. See below for more details of circular transfers

U14GetCircBlk

This function , which is only functional in the 32-bit Windows build of Use1401, returns information on the currently available data from a circular-mode transfer area.

C/C++

```
U14API(long) U14GetCircBlk(short hand, WORD wArea,
                    DWORD FAR * pdwOffs);
```

Visual BASIC

```
Function U14GetCircBlk(ByVal hand As Integer,
                    ByVal wArea As Integer,
                    dwOffs As Long) As Long
```

Borland Pascal/Delphi

```
Function U14GetCircBlk(hand:INTEGER; wArea:WORD;
                    VAR dwOffs:LONGINT):LONGINT;
```

`wArea`	The number of the transfer area to test, this must have been set up using `U14SetCircular`.
`pdwOffs`	Points to a DWORD that will be updated with the byte offset within the transfer area of the start of available data.

The return value of the function is the number of bytes of 1401 data available (or 0 for no data), or a negative error code.

In circular-mode data transfers, all management of the circular buffer is handled by the 1401 device driver. When there is data available for the host application, `U14GetCircBlk` will return the size and start offset of the area holding available data. Once the application has used this data, it should call `U14FreeCircBlk` to let the device driver know that this memory is available for re-use.

U14FreeCircBlk

This function , which is only functional in the 32-bit Windows build of Use1401, frees an area of a circular transfer buffer and returns information on any further available data.

C/C++

```
U14API(long) U14FreeCircBlk(short hand, WORD wArea,
                    DWORD dwOffs, DWORD dwSize,
                    DWORD FAR * pdwOffs);
```

Visual BASIC

```
Function U14FreeCircBlk(ByVal hand As Integer,
                    ByVal wArea As Integer,
                    ByVal dwOffs As Long,
                    ByVal dwSize As Long,
                    dwOffs As Long) As Long
```

Borland Pascal/Delphi

```
Function U14FreeCircBlk(hand:INTEGER; wArea:WORD;
                    dwOffs:DWORD, dwSize:DWORD,
                    VAR dwOffs:LONGINT):LONGINT;
```

`wArea`	The number of the transfer area to test, this must have been set up using `U14SetCircular`.
`dwOffs`	The byte offset within the transfer area of the start of the memory area to be freed.
`dwSize`	The size, in bytes, of the area to be freed.
`pdwOffs`	Points to a DWORD that will be updated with the offset within the transfer area of any available data.

The return value of the function is the number of bytes of 1401 data available after the area specified by the parameters is freed (or 0 for no data), or a negative error code.

In circular-mode data transfers, all management of the circular buffer is handled by the 1401 device driver. `U14FreeCircBlk` is used to inform the device driver that you have copied or otherwise handled an area of memory, it returns information about any data that remains available after the freeing. In order for circular-mode transfers to work efficiently the application software should ensure that buffer areas containing data are freed as quickly as possible.

The device driver code assumes that the application will only free buffer areas reported as holding available data and that the start offset of the freed area is the same as the start offset of the currently reported available area (this ensures that the data you process is always the oldest available data, and keeps the free buffer area contiguous). Therefore, if the device driver reports that 25000 bytes of data is available starting at byte offset 10000, the next call to `U14FreeCircBlock` **must** have a `dwOffs` parameter of 10000 and the `dwSize` parameter can be from 1 to 25000.

U14WorkingSet

This function adjusts the working set of a process; the amount of actual, physical, memory that is used by that process. This function is only available in the WIN32 build of Use1401, it is used to increase the amount of memory available to the process so that failures to lock memory in `U14SetTransArea` (resulting in error code -544) should not occur. Increasing the working set of a process requires a certain level of operating system privilege so calls to `U14WorkinSet` may fail if you do not have sufficient privilege. The function returns 1 if it was unable to access the process (probably because of insufficient privilege), 2 if reading the current settings failed and 3 if setting the new values failed. It returns 0 if all went OK.

C/C++
```
U14API(short) U14WorkingSet(DWORD dwMin, DWORD dwMax);
```

Visual BASIC
```
Function U14WorkingSet(ByVal dwMin As Long,
                       ByVal dwMax As Long) As Integer
```

Borland Pascal/Delphi
```
Function U14WorkingSet(DWORD dwMin, DWORD dwMax):INTEGER;
```

`dwMin`	The minimum working set required, in Kbyte, this must be large enough to allow the program to run but we expect the operating system to provide more than this. A default value of 400 or 800 is used in CED software.
`dwMax`	The maximum working set size required, in Kbyte. This should be a lot larger than `dwMin`, the operating system will allocate something between `dwMin` and `dwMax`. A default value of 4000 is used in CED software.

U14BlkTransState

Returns 1 if a block transfer is in progress, 0 if not or a negative error code.

C/C++	`U14API(short) U14BlkTransState(short hand);`
Visual BASIC	`Function U14BlkTransState(ByVal hand As Integer) As Integer`
Borland Pascal/Delphi	`Function U14BlkTransState(hand:INTEGER):INTEGER;`

Test and diagnostic functions

The following structures and functions are used for testing the 1401 in various ways. They are of little use to normal applications software and should be avoided by non-CED programmers. Many of these functions are only available on some platforms and languages.

TGET_TX_BLOCK

This structure is used by the `U14GetTransfer` function to return information about an area set up for block transfers, it allows debug and test software to see the behaviour of the Windows device drivers. Note that this structure is not used in the Macintosh version; the Macintosh `U14GetTransfer` function is different.

C/C++

```
#define  GET_TX_MAXENTRIES    257                   /* # blocks */

typedef struct             /* used for U14_GetTransfer results */
{                             /* Info on a single mapped block */
    long     physical;
    long     size;
} TXENTRY;

typedef struct TGetTxBlock  /* used for U14_GetTransfer results */
{
    long     size;
    long     linear;
    short    seg;
    short    reserved;
    short    avail;
    short    used;
    TXENTRY  entries[GET_TX_MAXENTRIES];  /* mapped block info */
} TGET_TX_BLOCK;

typedef TGET_TX_BLOCK FAR * LPGET_TX_BLOCK;
```

Borland Pascal/Delphi

```
CONST  GET_TX_MAXENTRIES = 257;                      { # blocks }

TYPE TXEntry=RECORD               { for U14_GetTransfer results }
             physical:LONGINT;       { Info on one mapped block }
             size:LONGINT;
             END;

     TGet_TX_Block=RECORD    { used for U14_GetTransfer results }
             size:LONGINT;          { matches structure in VXD }
             linear:LONGINT;
             seg:INTEGER;
             reserved:INTEGER;
             avail:INTEGER;
             used:INTEGER;
             entries: ARRAY[0..GET_TX_MAXENTRIES-1] OF TXEntry;
             END;
```

BYTE_SIZE, WORD_SIZE & LONG_SIZE

These constants are defined for use in the diagnostics functions where it is necessary to specify the width if a data read or write. Note that, for the standard 1401, the size is ignored as all reads and writes are 8 bit. Note also that the micro1401 cannot do word accesses and uses 4-byte access instead

```
#define BYTE_SIZE   1      /* 8-bit data access        */
#define WORD_SIZE   2      /* 16-bit data access       */
#define LONG_SIZE   3      /* 32-bit data access       */
```

U14StartSelfTest

This function starts off a self-test pass in the 1401. It returns zero or a negative error code.

C/C++
```
U14API(short) U14StartSelfTest(short hand);
```
Visual BASIC
```
Function U14StartSelfTest(ByVal hand As Integer) As Integer
```
Borland Pascal/Delphi
```
Function U14StartSelfTest(hand:INTEGER):INTEGER;
```

U14CheckSelfTest

This function checks to see if the self-test pass started by `U14StartSelfTest` has returned any errors, timed out, or finished. The function result is zero or an error code.

C/C++
```
U14API(short) U14CheckSelfTest(short hand, long far * pData);
```
Visual BASIC
```
Function U14StartSelfTest(ByVal hand As Integer,
                        pData As Any) As Integer
```
Borland Pascal/Delphi
```
Function U14StartSelfTest(hand:INTEGER; pData:TpNums):INTEGER;
```

`pData`	This points to an array of 3 longs that will hold the returned information on the progress of the self-test. The first long holds information on the status; 0 for still waiting, -1 for self-test finished, -2 for self-test timeout, all other values are self-test error codes. The other two longs hold information on the self-test error, if any.

U14GetTransfer

This function returns information about a (previously set up) transfer area. The function result is zero or a negative error code. The Macintosh version is different; see the Macintosh specifics.

C/C++
```
U14API(short) U14GetTransfer(short hand, LPGET_TX_BLOCK lpBlock)
```
Borland Pascal/Delphi
```
Function U14GetTransfer(hand:INTEGER;
                        VAR trBlock:TGet_TX_Block):INTEGER;
```

`lpBlock`	A pointer to a `GET_TX_BLOCK` structure that will be updated with transfer area information.
`trBlock`	A `TGet_TX_Block` record passed by reference that will be updated with transfer area information.

The first item in the structure, the size, should be filled in by the caller with the transfer area number for which information is required. The routine then fills in the structure with the current settings for that table entry. These values are undefined if the entry has never been used.

U14BaseAddr1401

This function returns the base address of the 1401 interface card registers. This is normally `0x300` for the ISA interface card, can be anything for the PCI card, is -1 for the USB interface and depends upon the interface card slot used on the Macintosh.

C/C++
```
U14API(long) U14BaseAddr1401(short hand);
```
Visual BASIC
```
Function U14BaseAddr1401(ByVal hand As Integer) As Long
```
Borland Pascal/Delphi
```
Function U14BaseAddr1401(hand:INTEGER):LONGINT;
```

U14Grab1401

This function causes the 1401 device driver to grab control of the 1401. This function must be called before any of the diagnostics functions described below are used. It must only used when the 1401 is in a quiescent state and most especially not when a block transfer is in progress.

C/C++
```
U14API(short) U14Grab1401(short hand);
```
Visual BASIC
```
Function U14Grab1401(ByVal hand As Integer) As Integer
```
Borland Pascal/Delphi
```
Function U14Grab1401(hand:INTEGER):INTEGER;
```

U14Free1401

This function allows the 1401 to free run again, cancelling the effect of `U14Grab1401`. You must call this function to undo the effects of `U14Grab1401`.

C/C++
```
U14API(short) U14Free1401(short hand);
```
Visual BASIC
```
Function U14Free1401(ByVal hand As Integer) As Integer
```
Borland Pascal/Delphi
```
Function U14Free1401(hand:INTEGER):INTEGER;
```

U14Poke1401

This diagnostic function writes specified data to an address inside the 1401. This function can only be used when the 1401 has been grabbed by the device driver.

C/C++
```
U14API(short) U14Poke1401(short hand, long lAddr, long lValue,
                                      long lSize, long lRepeats);
```
Visual BASIC
```
Function U14Poke1401(ByVal hand As Integer, ByVal addr As Long,
                     ByVal value As Long, ByVal size As Long,
                     ByVal repeats As Long) As Integer
```
Borland Pascal/Delphi
```
Function U14Poke1401(hand:INTEGER; addr:LONGINT; value:LONGINT;
                     size:LONGINT; repeats:LONGINT):INTEGER;
```

`lAddr`	The address, within the 1401 address space, to be written to. Please note that you need to know about the internals of the 1401 in order to choose addresses that are safe to write to.
`lValue`	The value to write to the address.
`lSize`	The size of the write access. This should be one of `BYTE_SIZE`, `WORD_SIZE` or `LONG_SIZE` as appropriate. Note that not all types of 1401 support all sizes of data access.
`lRepeats`	The number of times to write to the location. To write continuously until `U14StopDebugLoop` is called, use a repeat count of zero.

U14Peek1401

This diagnostic function reads from a specified address inside the 1401, the data read is saved internally and can be retrieved using `U14GetDebugData`. This function can only be used when the 1401 has been grabbed by the device driver.

C/C++

```
U14API(short) U14Peek1401(short hand, long lAddr, long lSize,
                    long lRepeats);
```

Visual BASIC

```
Function U14Peek1401(ByVal hand As Integer, ByVal addr As Long,
                    ByVal size As Long, ByVal repeats As Long)
                    As Integer
```

Borland Pascal/Delphi

```
Function U14Peek1401(hand:INTEGER; addr:LONGINT; size:LONGINT;
                    repeats:LONGINT):INTEGER;
```

`lAddr`	The address, within the 1401 address space, to be read from. You will need to be familiar with the 1401 internals in order to choose a suitable address.
`lSize`	The size of the read access. This should be one of `BYTE_SIZE, WORD_SIZE` or `LONG_SIZE` as appropriate. Note that not all types of 1401 support all sizes of data access.
`lRepeats`	The number of times to read from the location. To read continuously until `U14StopDebugLoop` is called, use a repeat count of zero.

U14Ramp1401

This diagnostic function writes a data ramp to an address inside the 1401. This function can only be used when the 1401 has been grabbed by the device driver.

C/C++

```
U14API(short) U14Ramp1401(short hand, long lAddr, long lDef,
                          long lEna, long lSize, long lRepeats);
```

Visual BASIC

```
Function U14Ramp1401(ByVal hand As Integer, ByVal addr As Long,
            ByVal def As Long, ByVal ena As Long,
            ByVal size As Long, ByVal repeats As Long) As Integer
```

Borland Pascal/Delphi

```
Function U14Ramp1401(hand:INTEGER; addr:LONGINT; def:LONGINT;
            ena:LONGINT; size:LONGINT; repeats:LONGINT):INTEGER;
```

`lAddr`	The address, within the 1401 address space, to be written to. Please note that you need to know about the internals of the 1401 in order to choose addresses that are safe to write to.
`lDef`	The default value to write to the address. This is the initial start of the data ramp and sets the state of any disabled bits.
`lEna`	A bit mask enabling bits to be ramped. Bits in the mask that are set will have the ramping data written to them, bits that are clear will have only the appropriate default value written.
`lSize`	The size of the write access. This should be one of `BYTE_SIZE`, `WORD_SIZE` or `LONG_SIZE` as appropriate. Note that not all types of 1401 support all sizes of data access. For `WORD_SIZE` and `LONG_SIZE` ramps, all of the bytes are ramped simultaneously.
`lRepeats`	The number of times to write the ramp to the location, a value of one selects a single 256 value ramp. To write continuously until `U14StopDebugLoop` is called, use a repeat count of zero.

U14RampAddr

This diagnostic function reads from a ramping address inside the 1401. The individual bytes within the address are ramped simultaneously. This function can only be used when the 1401 has been grabbed by the device driver.

C/C++

```
U14API(short) U14RampAddr(short hand, long lDef, long lEna,
                                             long lSize, long lRepeats);
```

Visual BASIC

```
Function U14Ramp1401(ByVal hand As Integer, ByVal def As Long,
                     ByVal ena As Long, ByVal size As Long,
                     ByVal repeats As Long) As Integer
```

Borland Pascal/Delphi

```
Function U14Ramp1401(hand:INTEGER; def:LONGINT; ena:LONGINT;
                     size:LONGINT; repeats:LONGINT):INTEGER;
```

`lDef`	The default address value to read from. This is the initial start of the address ramp and sets the state of any disabled bits in the address.
`lEna`	A bit mask enabling address bits to be ramped. Bits in the mask that are set will be used as part of the ramping address, bits that are clear will stick at the default value.
`lSize`	The size of the read access. This should be one of `BYTE_SIZE, WORD_SIZE` or `LONG_SIZE` as appropriate. Note that not all types of 1401 support all sizes of data access.
`lRepeats`	The number of times to read the ramp, a value of one selects a single 256 value ramp. To read continuously until `U14StopDebugLoop` is called, use a repeat count of zero.

U14StopDebugLoop

This function causes any continuous debug loops (reads or writes set up with a repeat count of zero) to terminate. This function can only be used when the 1401 has been grabbed by the device driver.

C/C++

```
U14API(short) U14StopDebugLoop(short hand);
```

Visual BASIC

```
Function U14StopDebugLoop(ByVal hand As Integer) As Integer
```

Borland Pascal/Delphi

```
Function U14StopDebugLoop(hand:INTEGER):INTEGER;
```

U14GetDebugData

This function returns the result of the last use of `U14Peek1401`, if the 1401 has been set to peek continuously then `U14StopDebugLoop` must be called first. This function can only be used when the 1401 has been grabbed by the device driver.

C/C++

```
U14API(short) U14StopDebugLoop(short hand, long* plValue);
```

Visual BASIC

```
Function U14StopDebugLoop(ByVal hand As Integer,
                          value As Long) As Integer
```

Borland Pascal/Delphi

```
Function U14StopDebugLoop(hand:INTEGER; VAR val:LONGINT):INTEGER;
```

`plValue`	This points to a long that will be updated with the value read from the 1401.

32-bit Windows specifics

Overview

The Windows version of the 32-bit Use1401 library has some extra features which are described below. The documentation gives C/C++ function prototypes only, though use from other languages would be equivalent.

DLL behaviour

The Use1401 code is designed to be used as a DLL. The unit includes the `DllMain` function required by Windows for it to use the unit as a DLL:

```
int APIENTRY DllMain(HANDLE hInst, DWORD dwReason,
                                        LPVOID lpReserved);
```

The `DllMain` code finds the windows version and sorts out any 16 to 32bit links that may be required to run 16-bit software on Windows NT or to run 32-bit software on Windows 3.x.

Direct calls to the driver

The device driver may be called directly by using the following functions, which are not currently exported by the standard DLL. These functions should not be used except for development.

```
U14API(short) U14Status1401(short h, long lCode, TCSBLOCK* pBlk);
U14API(short) U14Control1401(short h, long lCode, TCSBLOCK* pBlk);
```

The first parameter is the handle (or 1401 number for Open), the next parameter is a function code which is passed to the driver. All the standard function codes are defined in `USE1401.H`. The final parameter is a pointer to a control/status block used to hold information relevant to this function

16-bit Windows specifics

Overview

The Windows version of the 16-bit Use1401 library has some important extra features which are described below. The documentation gives C/C++ function prototypes only, though use from other languages would be equivalent. This information is provided for historical purposes only – 16-bit Windows software is obsolete.

DLL behaviour

The Use1401 code is designed to be used as a DLL. The unit includes the two functions required by Windows for it to use the unit as a DLL, `LibMain` and `WEP`:

```
BOOL FAR PASCAL LibMain(HANDLE hMod, WORD wDSeg,
                            WORD wHeapSz, LPSTR lpszCmdLine);
BOOL FAR PASCAL WEP(int nArgument);
```

The `LibMain` code finds the device driver for the 1401. If the unit is not compiled and linked as a DLL the driver will not be found and none of the functions will work. It is possible to write a routine to find the driver address if you do not wish to use the unit as a DLL, the following code would need to be added to the start of `U14Open1401`:

```
/* get API entry to CED_1401.386--if it is installed */
_asm
{
    mov     ax,1600h                        ;enhanced mode?
    int     2Fh                             ;api call
    test    al,7Fh                          ;enhanced mode?
    jz      Not_Running_Enhanced            ;no
    mov     ax,1684h                        ;get device API call
    mov     bx, CED_1401_Device_ID          ;for the CED_1401 VxD
    int     2Fh                             ;get api entry point
    mov     word ptr V1401D_API,di          ;save address
    mov     word ptr V1401D_API+2,es
    mov     ax,es                           ;is V1401D installed?
    or      ax,di
    jz      CED_1401_Not_Installed          ;if not, split
    mov     sDriverState,U14ERR_NOERROR     ;show success
    jmp     get_out
Not_Running_Enhanced:
    mov     sDriverState,U14ERR_NO386ENH    ;no enhanced windows!
    jmp     Get_Out                         ;return our error
code
CED_1401_Not_Installed:
    mov     sDriverState,U14ERR_NO1401DRIV   ;no v1401d installed
Get_Out:
}
```

Information callback functions

The 16-bit Windows 1401 driver has support for callbacks to application programs. This mechanism allows monitoring of character I/O between the driver and the 1401. The monitoring facilities may be extended in the future, or changed. CED does not recommend that these functions are used The information is passed to a user application by means of Windows messages that are issued whenever there is data available. To receive this information your application must first find the ID of the messages that the driver sends. The string constant called `WM_CEDCALLBACK_STR` is registered as a global Windows message name by the Use1401 library and the driver. The application must call `U14RegCallBackWnd` in order to find the message value. This also identifies to the Use1401 library and the driver a particular window that will receive the messages. Once the program has finished monitoring the system, the function `U14DeRegCallBackWnd` must be called. This frees up monitoring for use elsewhere. This mechanism is used by the CED utility program `CEDMON` which displays data passing between the driver and the 1401. It should be noted that in order to use the monitor functions there is no need to call `U14Open1401` or any of its associated routines. The function definitions are:

U14RegCallBackWnd

This function registers a window as the window to receive messages from the driver when there is information to collect.

```
U14API(short) U14RegCallBackWnd(short n1401, HWND hWnd, LPWORD
lpwMessCode);
```

The `n1401` parameter specifies the 1401 to be monitored, it's meaning is the same as the `U14Open1401` parameter. The `hWnd` parameter identifies the window which is to receive the messages. The word value pointed to by `lpwMessCode` is filled in with the value of the message to look for. Once a window has registered for messages no other window may register. The return value of the function is an error code indicating whether the window was successfully registered.

U14DeRegCallBackWnd

This functions releases the window from receiving messages, allowing another window to register.

```
U14API(short) U14DeRegCallBackWnd(HWND hWnd);
```

Monitoring

Once the window has been registered it starts receiving messages when there is data to collect. The data is held in a fixed size buffer in the driver. As soon as a line of data enters the buffer a message is sent to the registered window to inform it that there is data to collect. No more messages are sent until data has been collected. Once the monitor buffer has been collected the driver sends a message when the next line of data is stored in the buffer. It should be noted that a message is only sent once a complete line has been stored. The data in the buffer is stored with lines terminated with CR LF sequence rather than the single CR that the 1401 sends. A line is also deemed terminated by a ';' character. If the buffer becomes too full to store any more data a message is sent to the window to indicate this. The messages so far defined all have the message value defined by `U14RegCallBackWnd` and their `wParam` and `lParam` values are:

```
wParam lParam low   hi     Meaning
   0   MONDATAMSG*  n/a    Data is available from the buffer.
   0   MONFULLMSG*  count  The buffer has overflowed. The
                           number of characters to read to
*Defined in V14MON.H       clear the buffer is  in count.
```

The file `V14MON.H` defines the two message codes and also a constant `MONITOR_BUF_SZ` which is the size of the buffer in the driver. To retrieve the data `U14GetMonBuff` should be called:

U14GetMonBuff

This function reads as much data as is in the monitor buffer or `wMaxLen` characters, whichever is the smaller, into the buffer pointed at by `lpBuffer`. End of line in buffer data is stored as `CR LF` not just `CR`.

```
U14API(short) U14GetMonBuff(LPSTR lpBuffer, WORD wMaxLen);
```

This allows easier displaying of the data in the Windows environment. Once an application has received a `MONDATAMSG` it will receive no more until this function has been called. Even if the whole of the buffer is not read, the application will not receive a message to say there is more data until new data is added to the buffer. However, it is perfectly legal to make repeated calls to this routine until an empty buffer is returned. The end of the data is marked by a `NULL` character.

Here is an example of using these routines:

```
#include "windows.h"
#include "use1401.h"
#include "v14mon.h"

WORD    wCEDMessage;

long FAR PASCAL MainWndProc(HWND hWnd,WORD wMessage,
                            WORD wParam,LONG lParam)
{
    switch (wMessage)
    {
       /* WM_CREATE code, reg window and remember message code */
        case WM_CREATE:
          U14RegCallBackWnd(hWnd,(LPWORD)&wCEDMessage);
        break;

       /* WM_DESTROY code, de-registers the window */
        case WM_DESTROY:
           U14DeRegCallBackWnd(hWnd);
        break;

       /* if not a standard message it might be ours so check */
        default:
            if (wMessage==wCEDMessage)      /* our message ? */
                GetMonitorData();     /* go and get the data */
        break;
    }  /* end of switch */
};
```

If you are using the `MFC` classes the `ON_REGISTERED_MSG` macro should be used in the message map to achieve the same result.

The CED application `CEDMON.EXE` is an example of an `MFC` program which uses these routines.

Direct calls to the driver

The device driver may be called directly by the following function. This should not be used except for development.

```
U14API(short) U14CallDriver(short hand, BYTE bMainFn,
                    BYTE bSubFn, U14PARAM lpParams);
```

The first parameter is the handle (or 1401 number for Open), the next two parameters are function codes which are passed to the driver. All the standard function codes are defined in `USE1401.H`. The header also defines `U14_NOSUBFN` which is passed as the second parameter for most existing functions. There is also a type defined for passing data to the driver; `U14PARAM`, this is a pointer type and can be used to cast a pointer to another type when passing the pointer to the driver.

There is one function which is not given an interface call. This is the hard close of the 1401. If the following call is made it always closes the 1401 even if it is being used by another application. This should only be used in debugging programs and tools:

```
U14CallDriver(U14_CLOSE1401, U14_CLOSE1401, (U14PARAM)NULL);
```

See the device driver documentation for details on what happens when the call is made.

ESZBYTES, ESZWORDS & ESZLONGS

These constants are defined for use in block transfer routines where it is necessary to define the size of an element. In the IBM-PC case this is only used for dummy parameters. In the Mac case these constants determine what the byte swapping hardware on the interface card does with the data. The constant `ESZUNKNOWN` is also defined for use when the size is unknown or irrelevant.

```
#define ESZBYTES   0       /* BYTE element size value        */
#define ESZWORDS   1       /* WORD element size value        */
#define ESZLONGS   2       /* long element size value        */
#define ESZUNKNOWN 0       /* unknown element size value     */
```

When using the 1401 from Macintosh software, use the constant corresponding to the size of the elements in the data being transferred. Not that, if you are using complex commands that transfer data structures, all the elements of the data structure must be the same size for the byte swapping to work correctly.

TRANSFERDESC

This structure is used to return information about a transfer area. Note that a different `TRANSFERDESC` structure is used by the Windows version of Use1401 to return information about transfer areas.

```
typedef struct TransferDesc
{
    WORD        wArea;                /* number of transfer area
*/
    void FAR *  lpvBuff;           /* address of transfer area
*/
    DWORD       dwLength;          /* length of area to set up
*/
    short       eSize;                   /* size (for swapping)
*/
}TRANSFERDESC;

typedef TRANSFERDESC FAR *  LPTRANSFERDESC;
```

At present only area number 0 is available for use with a standard 1401 but with a 1401*plus*, micro1401 or Power1401 areas from 0 to 7 are available. You need an up-to-date device driver to use multiple areas, so your software should check the value returned by `U14TransferFlags` to see if multiple transfer areas can be used. Most 1401 commands use the default area 0.

U14GetTransfer

This function returns information about a (previously set up) transfer area.

```
U14API(short) U14GetTransfer(short hand,
             LPTRANSFERDESC lpTransDesc);
```

The implementation of this function for the Macintosh takes a pointer to a transfer description structure as its parameter. The first item in the structure, the area number, should be filled in by the caller. The routine then fills in the structure with the current settings for that table entry. These values are undefined if the entry is not in use.

Direct calls to the driver

The device driver may be called directly by the following functions. This should not be used except for development.

```
U14API(short) U14Status1401(short h, short sC, LPTCSBLOCK pBlk);
U14API(short) U14Control1401(short h, short sC, LPTCSBLOCK pBlk);
```

The first parameter is the handle or 1401 number for the open operation. The second is a function code, which is passed to the driver. All the standard functions have a status or control code defined in `USE1401.H`. The third parameter is a pointer to a control/status block structure, which is used to hold parameters for the operation.

Global variables

The `USE1401.H` file declares some global variables for the Macintosh:

```
extern short appResFile;   /* ref num for the application  */
extern short cmdResFile;   /* and for command resource files */
```

These two variables are the handles to the resource fork in the application and the 1401 command resource file in the system folder. they are used by Use1401 when trying to load commands.

Delphi & Borland Pascal specifics

Introduction

Use1401 is a set of functions written in C providing access to the 1401 data acquisition system, which may be compiled to produce a Windows DLL. This DLL can be used by software written using Borland Pascal for Windows or Delphi 1. The file `USE1401.PAS` declares a unit with constants, types and external functions providing a complete interface to the Use1401 DLL. By including this unit in your program, you are able to make use of all the functions in the Use1401 DLL to gain access to the 1401. Similarly, the file `USE1432.PAS` declares the Use1432 DLL as a unit for Delphi 2 software. Runtime linkage to the DLL is automatic and does not require any coding. All functions and error codes defined have the prefix U14 to indicate their origin.

The documentation for the Use1401 functions, and many other aspects of the library, are given as part of the C/C++ documentation because use from Borland Pascal or Delphi is simply a matter of getting access to the functions in the DLL, the actual functions used, the function parameters and the behaviour of the functions are identical for all languages.

If you are using the Borland IDE or Delphi to develop software, correct use of Use1401 is automatic. When developing using the command line driven Borland Pascal compiler, you must compile `USE1401.PAS` to produce USE1401.TPW, the compiled unit. To do this, use the following command at the DOS prompt:

```
        BPC USE1401 /CW
```

There is an example program, `WINDOW14` which gives some examples of using Use1401 from Borland Pascal. The files making up this example are `WINDOW14.PAS, WINDOW14.RES and WINDOW14.INC`.

Types and records

A number of types and structures are defined in `USE1401.PAS` and `USE1432.PAS` for use by the various routines. Many of these are unimportant or are provided for specialised purposes, the C documentation gives more details of these, the more important ones are documented below:

DWORD

This type defines an (assumed) unsigned 4 byte integer, for general use in the interface and for similarity to the underlying C code.

```
TYPE DWORD = LONGINT;
```

TpNums

This type defines a pointer to an array of `LONGINTs`, this array is assumed to be of an unknown length. This type is used as an argument to the `U14LongsFrom1401` function and other similar functions.

Visual Basic specifics

Introduction

The Use1401 library can be used from Visual BASIC Windows programs. 16-bit Visual BASIC programs produced by Visual-BASIC versions 3 and 4 use the 16-bit DLL `USE1401.DLL`, while 32-bit programs produced using Visual BASIC version 5 can use the 32-bit DLL `USE1432.DLL`.

The file `USE1401.BAS` declares a module with constants, types and external functions that provides a complete interface for communication with the 1401 (the file `USE1401.TXT` is a text version of this file). By including this module in your program, you are able to make use of the declared functions in the Use1401 DLL. Runtime linkage to the DLL is automatic and does not require any coding. All functions and error codes defined have the prefix U14 to indicate their origin.

Similarly, the file `USE1432.BAS` declares a module with constants, types and external functions that provides a complete interface for communication with the 1401 (the file `USE1432.TXT` is a text version of this file). By including this module in your program, you are able to make use of the declared functions in the Use1432 DLL. This 32-bit support is currently untested.

The documentation for the Use1401 functions, and many other aspects of the library, are given as part of the C/C++ documentation because use from Visual BASIC is simply a matter of getting access to the functions in the DLL, the actual functions used, the function parameters and the behaviour of the functions are identical for all languages.

Get1401String

This function is provided in the Visual BASIC support to provide a more convenient Visual Basic string read mechanism. It ensures that the string passed to the `U14GetString` routine is long enough to receive any string that the 1401 may return.

```
Function Get1401String (ByVal handle1401 As Integer,
             strP As String, ByVal maxlen As Integer) As Integer
```

This function calls `U14GetString` and returns the same error codes. It reads up to `maxLen` characters from the 1401 and times out if there are none ready. `strP` may safely be empty before you call this function. For example:

```
Dim strP As String
strP = ""
If Get1401String (hand1401,strP,20) Then 'read up to 20 characters
                                       'we have characters in strP
End If
```

Getting started

We suggest that you work through the first few chapters of the Visual Basic Programmer's Guide until you can find your way around the Project. Chapters 1 to 7 introduce properties, code, controls and menu design. Once you are familiar with the Visual Basic environment you can add the Use1401 support library and add code to access the 1401.

There are example projects `CED_VB01`, `CED_VB02`, and `CED_VB03` demonstrating the use of Use1401 from Visual BASIC. The files for these examples have the extensions `.FRM`, `.TXT` and `.MAK` and there is also a text file `README.DOC` describing them. You could either create a new project or use code from one or more of these example projects as a starting point, copying code from one project to another.

Appendix 1: Listing of USE1401.H

For reasons of space, the complete `USE1401.H` file is not included here, as it incorporates a large number of types, functions and #defines that are obsolete or only used internally to Use1401. Consult the actual `USE1401.H` file if you require complete information.

```
#ifndef __USE1401_H__                  /* Protect against multiple includes */
#define __USE1401_H__

#ifndef RC_INVOKED

#include "machine.h"

#ifdef macintosh
#define U14API(retType) pascal retType
#ifndef __TYPES__
#include <types.h>
#endif
#endif

#if (defined(_IS_MSDOS_) || defined(_IS_WINDOWS_)) && !defined(_MAC)
#define U14API(retType) retType FAR PASCAL
#endif

#ifdef _MAC
#undef U14API
#define U14API(retType) retType
#endif

#endif                                         /* End of ifndef RC_INVOKED */

/****************************************************************************/
/*                                                                          */
/* Return codes from functions                                              */
/*                                                                          */
/****************************************************************************/

#define U14ERR_NOERROR        0          /* no problems                      */

#define U14ERR_OFF            -500       /* 1401 there but switched off    */
#define U14ERR_NC             -501       /* 1401 not connected             */
#define U14ERR_ILL            -502       /* if present it is ill           */
#define U14ERR_NOIF           -503       /* I/F card missing               */
#define U14ERR_TIME           -504       /* 1401 failed to come ready      */
#define U14ERR_BADSW          -505       /* I/F card bad switches          */
#define U14ERR_PTIME          -506       /* 1401+ didn't come ready UNUSED */
#define U14ERR_NOINT          -507       /* couldn't grab the int vector   */
#define U14ERR_INUSE          -508       /* 1401 is already in use         */
#define U14ERR_NODMA          -509       /* couldn't get DMA channel       */
#define U14ERR_BADHAND        -510       /* handle provided was bad        */
#define U14ERR_BAD1401NUM     -511       /* 1401 number provided was bad   */

#define U14ERR_NO_SUCH_FN     -520       /* no such function               */
#define U14ERR_NO_SUCH_SUBFN  -521       /* no such sub function           */
#define U14ERR_ERR_NOOUT      -522       /* no room in output buffer       */
#define U14ERR_ERR_NOIN       -523       /* no input in buffer             */
#define U14ERR_ERR_STRLEN     -524       /* string longer than buffer      */
#define U14ERR_LOCKFAIL       -525       /* failed to lock memory          */
#define U14ERR_UNLOCKFAIL     -526       /* failed to unlock memory        */
#define U14ERR_ALREADYSET     -527       /* area already set up            */
#define U14ERR_NOTSET         -528       /* area not set up                */
```

```
#define U14ERR_BADAREA        -529        /* illegal area number            */

#define U14ERR_NOFILE         -540        /* command file not found         */
#define U14ERR_READERR        -541        /* error reading command file     */
#define U14ERR_UNKNOWN        -542        /* unknown command                */
#define U14ERR_HOSTSPACE      -543        /* not enough host space to load  */
#define U14ERR_LOCKERR        -544        /* could not lock resource/command*/
#define U14ERR_CLOADERR       -545        /* CLOAD command failed           */

#define U14ERR_TOXXXERR       -560        /* tohost/1401 failed             */

#define U14ERR_NO386ENH       -580        /* not 386 enhanced mode          */
#define U14ERR_NO1401DRIV     -581        /* no device driver               */
#define U14ERR_DRIVTOOOLD     -582        /* device driver too old          */

#define U14ERR_TIMEOUT        -590        /* timeout occurred               */

#define U14ERR_BUFF_SMALL     -600        /* buffer for getstring too small */
#define U14ERR_CBALREADY      -601        /* there is already a callback    */
#define U14ERR_BADDEREG       -602        /* bad parameter to deregcallback */

#define U14ERR_DRIVCOMMS      -610        /* failed talking to driver       */
#define U14ERR_OUTOFMEMORY    -611        /* neede memory and couldnt get it*/

#define U14TYPE1401           0           /* standard 1401                  */
#define U14TYPEPLUS           1           /* 1401 plus                      */
#define U14TYPEU1401          2           /* u1401                          */
#define U14TYPEPOWER          3           /* power1401                      */
#define U14TYPEU14012         4           /* u1401 mk II                    */
#define U14TYPEPOWER2         5           /* power1401 mk II                */
#define U14TYPEU14013         6           /* u1401-3                        */
#define U14TYPEUNKNOWN        -1          /* dont know                      */

// Transfer flags to allow driver capabilities to be interrogated
#define U14TF_USEDMA          1           /* Transfer flag for use DMA      */
#define U14TF_MULTIA          2           /* Transfer flag for multi areas  */
#define U14TF_FIFO            4           /* for FIFO interface card        */
#define U14TF_USB2            8           /* for USB2 interface and 1401    */
#define U14TF_NOTIFY          16          /* for event notifications        */
#define U14TF_SHORT           32          /* for PCI can short cycle        */
#define U14TF_PCI2            64          /* for new PCI card 1401-70       */
#define U14TF_CIRCTH          128         /* Circular-mode to host          */
#define U14TF_DIAG            256         /* Diagnostics\debug functions    */
#define U14TF_CIRC14          512         /* Circular-mode to 1401          */

#define ESZBYTES              0           /* BYTE element size value        */
#define ESZWORDS              1           /* WORD element size value        */
#define ESZLONGS              2           /* long element size value        */
#define ESZUNKNOWN            0           /* unknown element size value     */

/* These define required access types for the debug\diagnostiscs function */
#define BYTE_SIZE             1           /* 8-bit access                   */
#define WORD_SIZE             2           /* 16-bit access                  */
#define LONG_SIZE             3           /* 32-bit access                  */

/****************************************************************************/
/*                                                                          */
/* TypeDefs                                                                 */
/*                                                                          */
/****************************************************************************/
```

```
#if (defined(_IS_MSDOS_) || defined(_IS_WINDOWS_)) && !defined(_MAC)
#ifndef RC_INVOKED
#pragma pack(1)
#endif
#endif

/* This is the structure used to set up a transfer area */
typedef struct VXTransferDesc    /* use1401.c and use1432x.x use only       */
{
   WORD        wArea;            /* number of transfer area to set up       */
   WORD        wAddrSel;         /* 16 bit selector for area                */
   DWORD       dwAddrOfs;        /* 32 bit offset for area start            */
   DWORD       dwLength;         /* length of area to set up                */
} VXTRANSFERDESC;

typedef  void FAR *        U14PARAM;

/* This lot defines the structure used to retrieve information about a
   transfer area and the blocks into which it is split */
#define  MAXAREAS   8   /* The number of transfer areas supported by driver */

#define  GET_TX_MAXENTRIES   257       /* Array size for GetTransfer struct */

typedef struct                          /* used for U14_GetTransfer results */
{                                          /* Info on a single mapped block */
   long     physical;
   long     size;
} TXENTRY;

typedef struct TGetTxBlock              /* used for U14_GetTransfer results */
{                                               /* matches structure in VXD */
   long     size;
   long     linear;
   short    seg;
   short    reserved;
   short    avail;
   short    used;
   TXENTRY  entries[GET_TX_MAXENTRIES];       /* Array of mapped block info */
} TGET_TX_BLOCK;

typedef TGET_TX_BLOCK FAR *    LPGET_TX_BLOCK;

typedef struct                  /* used for get/set standard 1401 registers */
{
   short   sPC;
   char    A;
   char    X;
   char    Y;
   char    stat;
   char    rubbish;
} T1401REGISTERS;

typedef union     /* to communicate with 1401 driver status & control funcs */
{
   char           chrs[22];
   short          ints[11];
   long           longs[5];
   T1401REGISTERS registers;
} TCSBLOCK;

#if defined(WIN32) || defined(_MAC)
```

```
typedef TCSBLOCK*  LPTCSBLOCK;
#else
typedef TCSBLOCK FAR *  LPTCSBLOCK;
#endif

#if (defined(_IS_MSDOS_) || defined(_IS_WINDOWS_)) && !defined(_MAC)
#ifndef RC_INVOKED
#pragma pack()
#endif
#endif

#ifdef __cplusplus
extern "C" {
#endif

U14API(long) U14WhenToTimeOut(short hand);
U14API(short) U14PassedTime(long lCheckTime);
U14API(short) U14LastErrCode(short hand);

U14API(short) U14Open1401(short n1401);
U14API(short) U14Close1401(short hand);
U14API(short) U14Reset1401(short hand);
U14API(short) U14ForceReset(short hand);
U14API(short) U14TypeOf1401(short hand);
U14API(short) U14NameOf1401(short hand, LPSTR lpBuf, WORD wMax);

U14API(short) U14Stat1401(short hand);
U14API(short) U14CharCount(short hand);
U14API(short) U14LineCount(short hand);

U14API(short) U14SendString(short hand, LPCSTR lpString);
U14API(short) U14GetString(short hand, LPSTR lpBuffer, WORD wMaxLen);
U14API(short) U14SendChar(short hand, char cChar);
U14API(short) U14GetChar(short hand, LPSTR lpcChar);

U14API(short) U14LdCmd(short hand, LPCSTR command);
U14API(DWORD) U14Ld(short hand, LPCSTR vl, LPCSTR str);

U14API(short) U14SetTransArea(short hand, WORD wArea, void FAR * lpvBuff,
                                            DWORD dwLength, short eSz);
U14API(short) U14UnSetTransfer(short hand, WORD wArea);
U14API(short) U14SetTransferEvent(short hand, WORD wArea, long hEvent,
                                  BOOL bToHost, DWORD dwStart, DWORD dwLength);

U14API(short) U14ToHost(short hand, LPSTR lpAddrHost,DWORD dwSize,DWORD dw1401,
                                                            short eSz);
U14API(short) U14To1401(short hand, LPCSTR lpAddrHost,DWORD dwSize,DWORD dw1401,
                                                            short eSz);

U14API(short) U14SetCircular(short hand, WORD wArea, BOOL bToHost, void FAR * lpvBuff,
                                         DWORD dwLength);

U14API(long)  U14GetCircBlk(short hand, WORD wArea, DWORD FAR * pdwOffs);
U14API(long)  U14FreeCircBlk(short hand, WORD wArea, DWORD dwOffs, DWORD dwSize,
                                         DWORD FAR * pdwOffs);

U14API(short) U14StrToLongs(LPCSTR lpszBuff,long FAR *lpalNums,short sMaxLongs);
U14API(short) U14LongsFrom1401(short hand, long FAR * lpalBuff,short sMaxLongs);

U14API(void)  U14SetTimeout(short hand, long lTimeout);
U14API(long)  U14GetTimeout(short hand);
U14API(short) U14OutBufSpace(short hand);
U14API(long)  U14BaseAddr1401(short hand);
U14API(long)  U14DriverVersion(void);
U14API(long)  U14DriverType(void);
U14API(short) U14GetUserMemorySize (short hand, long FAR * lpMemorySize);
U14API(short) U14KillIO1401(short hand);

U14API(short) U14BlkTransState(short hand);
U14API(short) U14StateOf1401(short hand);

U14API(short) U14Grab1401(short hand);
U14API(short) U14Free1401(short hand);
U14API(short) U14Peek1401(short hand, long lAddr, long lSize, long lRepeats);
U14API(short) U14Poke1401(short hand, long lAddr, long lValue, long lSize, long lRepeats);
U14API(short) U14Ramp1401(short hand, long lAddr, long lDef, long lEnable,
                                                              long lSize, long lRepeats);
U14API(short) U14RampAddr(short hand, long lDef, long lEnable, long lSize, long lRepeats);
U14API(short) U14StopDebugLoop(short hand);
U14API(short) U14GetDebugData(short hand, long* plValue);

U14API(short) U14StartSelfTest(short hand);
U14API(short) U14CheckSelfTest(short hand, long FAR * lpData);
U14API(short) U14TransferFlags(short hand);
U14API(void)  U14GetErrorString(short nErr, LPSTR lpStr, WORD wMax);
U14API(long)  U14MonitorRev(short hand);
U14API(void)  U14CloseAll(void);

#ifdef __cplusplus
}
#endif

/****************************************************************************/
/*                                                                          */
/* Windows 3.1 /Dos Specifics                                               */
/*                                                                          */
/****************************************************************************/
#if ((defined(_IS_MSDOS_) || defined(_INC_WINDOWS)) && !defined(WIN32))

#define  MINDRIVERMAJREV   1    /* minimum driver revision level we need    */

#define  CED_1401_Device_ID   0x2952            /* VxD ID for CED_1401.386  */

#define  U14_NOSUBFN       0

#define  WM_CEDCALLBACK_STR   "CEDCALLBACK"

BOOL FAR PASCAL LibMain(HANDLE hMod,WORD wDSeg,
                        WORD wHeapSz,LPSTR lpszCmdLine);
BOOL FAR PASCAL WEP(int nArgument);

U14API(short) U14CallDriver(short hand, BYTE bMainFn, BYTE bSubFn, U14PARAM lpParams);

U14API(short) U14RegCallBackWnd(short n1401, HWND hWnd, LPWORD lpwMessCode);
U14API(short) U14DeRegCallBackWnd(HWND hWnd);
U14API(short) U14GetMonBuff(LPSTR lpBuffer, WORD wMaxLen);
U14API(short) U14GetCircSelector(void);
U14API(short) U14GetTransfer(short hand, LPGET_TX_BLOCK lpTransBlock);

#define  U14Status1401(H,X,Y)   U14CallDriver(H,X,U14_NOSUBFN,(U14PARAM)Y)
#define  U14Control1401(H,X,Y)  U14CallDriver(H,X,U14_NOSUBFN,(U14PARAM)Y)
#endif
```

```
/****************************************************************************/
/*                                                                          */
/* Windows NT Specifics                                                     */
/*                                                                          */
/****************************************************************************/
#if defined(WIN32) && !defined(_MAC)
                             /* if we are in NT/Win95/Win32s we have extra bits       */

#define  MINDRIVERMAJREV   1    /* minimum driver revision level we need    */

#ifndef RC_INVOKED
#pragma pack(1)
#endif

typedef struct paramBlk
{
    short       sState;
    TCSBLOCK    csBlock;
} PARAMBLK;

typedef PARAMBLK*   PPARAMBLK;


#ifndef RC_INVOKED
#pragma pack()
#endif

U14API(short) U14GetTransfer(short hand, LPGET_TX_BLOCK lpTransBlock);
U14API(short) U14WorkingSet(DWORD dwMinKb, DWORD dwMaxKb);
U14API(short) U14Status1401(short sHand, LONG lCode, TCSBLOCK* pBlk);
U14API(short) U14Control1401(short sHand, LONG lCode, TCSBLOCK* pBlk);

#endif

/****************************************************************************/
/*                                                                          */
/* Macintosh Specifics                                                      */
/*                                                                          */
/****************************************************************************/
#if defined(macintosh) || defined(_MAC)

#define  MINDRIVERMAJREV   2     /* minimum driver revision level we need   */

#define U14_RES1401COMMAND     '1401'
#define U14_RESPLUSCOMMAND     '1402'
#define U14_RESU1401COMMAND    '1403'
#define U14_RESPOWERCOMMAND    '1404'

#define k1401CommandFile    "\p1401Commands"
#define k1401DriverName     "\p.Driver1401"

#define  MAXAREAS   8   /* The number of transfer areas supported by driver */

/* Structure for GetTransfer results(NB also used internally by NT version) */
typedef struct TransferDesc
{
   WORD        wArea;            /* number of transfer area to set up       */
   void FAR *  lpvBuff;          /* address of transfer area                */
   DWORD       dwLength;         /* length of area to set up                */
   short       eSize;            /* size to move (for swapping on MAC)      */
} TRANSFERDESC;

typedef TRANSFERDESC FAR *    LPTRANSFERDESC;


U14API(short) U14GetTransfer(short hand, LPTRANSFERDESC lpTransDesc);

U14API(short) U14Status1401(short hand, short csCode, LPTCSBLOCK pBlock);
U14API(short) U14Control1401(short hand, short csCode, LPTCSBLOCK pBlock);

#endif

#endif                                       /* End of ifndef __USE1401_H__ */
```

Index

Index