

```

/*****
/*          R E G S T A T . C          */
/*-----*/
/* Task      : Display statistical information about the Registry      */
/*            and display paths when searching for keys.              */
/*-----*/
/* Authors    : Michael Tischer and Bruno Jennrich                  */
/* developed on : 08/30/1995                                          */
/* last update  : 09/12/1995                                          */
/*****
#include <windows.h>
#include <stdio.h>

//---- Typedefs -----

typedef struct tagREGSTAT          // Receives Registry statistics
{
    int    iMaxLevel;              // Greatest nesting level

    DWORD  lcntKeys;               // Total number of keys
    DWORD  lcntValues;            // Total number of values

    DWORD  lcchMaxKeyNameLen;      // Number of characters in longest key

    DWORD  lcchMaxValueNameLen;    // Number of characters in longest value name
    DWORD  lcbMaxValueDataLen;     // Number of bytes in largest value

    DWORD  cntREG_BINARY;          // Number of value types
    DWORD  cntREG_DWORD;
    DWORD  cntREG_DWORD_BIG_ENDIAN;
    DWORD  cntREG_EXPAND_SZ;
    DWORD  cntREG_LINK;
    DWORD  cntREG_MULTI_SZ;
    DWORD  cntREG_NONE;
    DWORD  cntREG_RESOURCE_LIST;
    DWORD  cntREG_SZ;
} REGSTAT;
typedef REGSTAT *PREGSTAT;

// Prototype for Callback of RegStat() -----
typedef VOID (WINAPI* FOUND_CALLBACK)( PSTR    pKeyName,
                                       PSTR    pValueName,
                                       DWORD    dwValueType );

typedef struct tagKEYANDNAMES      // for receiving a predefined
{
    LPSTR  lpName;                // HKEY... key and its
    HKEY    hKey;                 // Key handle
} KEYHANDLENAMES;

char g_Buffer[ 1024 ];            // Buffer for receiving the entire key
int  g_iKeyPos;                  // Position of new SubKey to be added

/*****
/* RegStat : Gets status information about Registry key and          */
/*            its SubKeys.                                          */
/*-----*/
/* Parameters:    hKey        - Source key                          */
/*                iLevel      - Current nesting level              */
/*                pRS         - Address of REGSTAT structure        */
/*                pFindKey    - Address of key to be found          */
/*                pFindValue  - Address of a value to be found      */
/*                Callback    - Address of callback function that   */
/*                             is called when key/valuenam is      */
/*                             found.                                */
/* Return value : keiner                                           */
/*****
void RegStat( HKEY        hKey,
              int         iLevel,
              PREGSTAT    pRS,
              LPSTR       pFindKey,
              LPSTR       pFindValue,
              FOUND_CALLBACK pCallback )
{
    char chKeyClass[ MAX_PATH ];    // Buffer for Key class names
    PSTR pSubKeyName;              // Pointer to key name

```

```

PSTR pValueName;                                // Pointer to value name

DWORD lcchKeyName;                               // Number of characters of current key name
DWORD lcchClass;                                 // Number of characters of current class name
DWORD lSubKeys;                                  // Number of SubKeys in current key
DWORD lcchMaxSubkey;                             // Number of characters in largest SubKey
DWORD lcchMaxClass;                              // Number of characters in largest class
DWORD lcValues;                                  // Number of values of a key
DWORD lcchMaxValueName; // Number of characters in largest value name
DWORD lcbMaxValueData;                           // Size of maximum value data
DWORD lcbSecurityDescriptor;                     // Size of Security Descriptor

DWORD i;                                         // Counter
FILETIME ft;                                    // FileTime
HKEY hNewKey;                                   // Handle for new key

pRS->lcntKeys++;                                // Increment key counter

// Get information about subkeys -----
lcchClass = sizeof( chKeyClass );
if( SUCCEEDED( RegQueryInfoKey ( hKey,
                                chKeyClass,
                                &lcchClass ,
                                NULL,
                                &lSubKeys,
                                &lcchMaxSubkey,
                                &lcchMaxClass,
                                &lcValues,
                                &lcchMaxValueName,
                                &lcbMaxValueData,
                                &lcbSecurityDescriptor,
                                &ft ) ) )
{
    // Update maximum key name and value name found so far -----
    if( lcchMaxValueName > pRS->lcchMaxValueNameLen )
        pRS->lcchMaxValueNameLen = lcchMaxValueName;

    if( lcbMaxValueData > pRS->lcbMaxValueDataLen )
        pRS->lcbMaxValueDataLen = lcbMaxValueData;

    // Allocate memory for value name -----
    pValueName = malloc( lcchMaxValueName + 1 );
    if( pValueName )
    {
        // Browse every single value of the current key -----
        for( i = 0; i < lcValues; i++ )
        {
            DWORD lcchValue;
            DWORD dwType;

            pRS->lcntValues++;                    // Increment value counter

            lcchValue = lcchMaxValueName;

            // Get information about i-th value -----
            if( SUCCEEDED( RegEnumValue( hKey,
                                        i,
                                        pValueName,
                                        &lcchValue,
                                        NULL,
                                        &dwType,
                                        NULL,
                                        NULL ) ) )
            {
                if( pFindValue )                // Search for value name?
                    if( !strcmp( pValueName, pFindValue ) )
                        if( pCallback )
                            pCallback( g_Buffer, pFindValue, dwType );

                // Increment value type counter -----
                switch( dwType )
                {
                    case REG_BINARY:
                        pRS->cntREG_BINARY++;
                        break;
                    case REG_DWORD:

```

```

        pRS->cntREG_DWORD++;
        break;
        case REG_DWORD_BIG_ENDIAN:
            pRS->cntREG_DWORD_BIG_ENDIAN++;
            break;
        case REG_EXPAND_SZ:
            pRS->cntREG_EXPAND_SZ++;
            break;
        case REG_LINK:
            pRS->cntREG_LINK++;
            break;
        case REG_MULTI_SZ:
            pRS->cntREG_MULTI_SZ++;
            break;
        case REG_NONE:
            pRS->cntREG_NONE++;
            break;
        case REG_RESOURCE_LIST:
            pRS->cntREG_RESOURCE_LIST++;
            break;
        case REG_SZ:
            pRS->cntREG_SZ++;
            break;
    }
}
}
// Release value name buffer -----
free( pValueName );
}

// Update maximum nesting level -----
if( pRS->iMaxLevel < iLevel )
    pRS->iMaxLevel = iLevel;

// Update maximum key name length -----
if( lcchMaxSubkey > pRS->lcchMaxKeyNameLen )
    pRS->lcchMaxKeyNameLen = lcchMaxSubkey;

// Allocate buffer for key name -----
pSubKeyName = malloc( lcchMaxSubkey + 1 );
if( pSubKeyName )
{
    int iPos = g_iKeyPos;          // Save current key text position
    // Browse individual subkeys -----
    for( i = 0; i < lSubKeys; i++ )
    {
        lcchClass = sizeof( chKeyClass );
        lcchKeyName = lcchMaxSubkey + 1;

        // Get information about key -----
        if( SUCCEEDED( RegEnumKeyEx( hKey,
                                     i,
                                     pSubKeyName,
                                     &lcchKeyName,
                                     NULL,
                                     chKeyClass,
                                     &lcchClass,
                                     &ft ) ) )
        {
            g_Buffer[ iPos ] = 0;
            strcat( &g_Buffer[ iPos ], "\\\" );
            strcat( &g_Buffer[ iPos ], pSubKeyName );
            // Open subkey and take next recursion step -----
            if( SUCCEEDED( RegOpenKeyEx( hKey,
                                         pSubKeyName,
                                         0,
                                         0,
                                         &hNewKey ) ) )
            {
                // Search for key name ? -----
                if( pFindKey )
                    if( !strcmp( pSubKeyName, pFindKey ) )
                        if( pCallback )
                            pCallback( g_Buffer, NULL, 0 );

                // Browse subkeys of key that was just opened -----

```

```

        g_iKeyPos = strlen( g_Buffer );

        RegStat( hNewKey,
                  iLevel + 1,
                  pRS,
                  pFindKey,
                  pFindValue,
                  pCallback );

        g_iKeyPos = iPos;

        // Release key
        RegCloseKey( hNewKey );
    }
}
}
free( pSubKeyName );
}
}

/*****
/* RegAllStat : Gets information about all default keys
/*-----*/
/* Parameters:    pRS        - Address of REGSTAT structure
/*                pFindKey   - Address of key to be found
/*                pFindValue - Address of a value to be found
/*                Callback   - Address of callback function that is
/*                             called when key/value name is
/*                             found.
/*
/* Return value : none
*****/
void RegAllStat( PREGSTAT    pRS,
                 LPSTR       pFindKey,
                 LPSTR       pFindValue,
                 FOUND CALLBACK pCallback )
{
    int i;
    KEYHANDLENAMES kh[] = { {"HKEY_CLASSES_ROOT", HKEY_CLASSES_ROOT},
                             {"HKEY_CURRENT_USER", HKEY_CURRENT_USER},
                             {"HKEY_LOCAL_MACHINE", HKEY_LOCAL_MACHINE},
                             {"HKEY_USERS", HKEY_USERS},
                             {"HKEY_CURRENT_CONFIG", HKEY_CURRENT_CONFIG},
                             {"HKEY_DYN_DATA", HKEY_DYN_DATA},
                             {NULL, 0}};

    memset( pRS, 0, sizeof( REGSTAT ) );

    i = 0;
    while( kh[i].lpName )
    {
        strcpy( g_Buffer, kh[i].lpName );
        g_iKeyPos = strlen( kh[i].lpName );
        RegStat( kh[i].hKey, 0, pRS, pFindKey, pFindValue, pCallback );
        i++;
    }
}

/*****
/* FoundCallback : Callback function called by RegAllStat
/*                when either a key name or a value name
/*                was found.
/*-----*/
/* Parameters:    pKeyName   - Entire path of found key
/*                pValueName - <> 0 if value name was found
/*                dwValueType - If pValueName < > 0 dwValueType
/*                             contains the type of found value.
/*
/* Return value : none
*****/
VOID WINAPI FoundCallback( PSTR    pKeyName,
                          PSTR    pValueName,
                          DWORD    dwValueType )
{
    // Output entire key path -----
    if( pKeyName ) printf( "Key: %s\n", pKeyName );
}

```

```

}

/*****
/* main : Start function */
/*-----*/
/* Parameters:      argv[1] - Name of key to find */
/*                  If argv[1] is not specified, */
/*                  Registry statistics are given. */
/* Return value : none */
*****/
void main( int argc, char *argv[] )
{
    REGSTAT rs;
    PSTR pFindKey = NULL;

    printf("REGSTAT - (c) 1995 by Michael Tischer & Bruno Jennrich\n");
    printf("-----\n");

    // Search for all occurrences of a key? -----
    if( argc == 2 ) pFindKey = argv[ 1 ];

    // Start statistics and search -----
    RegAllStat( &rs, pFindKey, NULL, FoundCallback );

    // If no search was initiated, only statistics will be output -----
    if( !pFindKey )
    {
        printf("Maximum key level           : %ld\n", rs.iMaxLevel );
        printf("Total number of keys                : %ld\n", rs.lcntKeys );
        printf("Total number of key values           : %ld\n", rs.lcntValues );
        printf("Maximum length of a key name         : %ld\n", rs.lcchMaxKeyNameLen );
        printf("Maximum length of a value name       : %ld\n", rs.lcchMaxValueNameLen );
        printf("Maximum key value                    : %ld\n", rs.lcbMaxValueDataLen );
        printf("\nTotal number of different types of key values\n");
        printf("-----\n");
        printf("Binary data                          : %ld\n", rs.cntREG_BINARY );
        printf("DWORDs                               : %ld\n", rs.cntREG_DWORD );
        printf("Big-Endian DWORDs                    : %ld\n", rs.cntREG_DWORD_BIG_ENDIAN );
        printf("Strings                              : %ld\n", rs.cntREG_SZ );
        printf("String arrays                        : %ld\n", rs.cntREG_MULTI_SZ );
        printf("Strings with environment variables: %ld\n", rs.cntREG_EXPAND_SZ );
    }
}

```