Getting Started with the FrequenC Library

W.J. Riley
Hamilton Technical Services

Introduction

The FrequenC Library © is a collection of over 100 C functions for the analysis of frequency stability, implemented in the form of a Microsoft Windows dynamic link library (DLL). This document describes how to begin using the FrequenC.dll in a C language program.

Prerequisites

The following resources are needed to use the FrequenC DLL:

1. A copy of the FrequenC.dll. This file is distributed with the Stable32 software package.
2. A licensed copy of Stable32. Not only is this needed to obtain the FrequenC.dll, but it also serves as a means to validate calculations made by FrequenC functions in another program. The Stable32 software package also includes a User Manual that describes much of the FrequenC library functionality.
3. The FrequenC.h header file. This file contains functions prototypes, macro definitions and other information needed to use the FrequenC library.
4. A copy of the FrequenC.lib import library file. This file is needed to link with the FrequenC functions that are called by the target program.
5. A C programming environment, including source editor, preprocessor, compiler and linker. It should be possible to use the FrequenC.dll with other programming languages (e.g., C++, LabWindows, Visual Basic, etc.) but those are not specifically supported or discussed in this document.
6. Documentation describing the FrequenC library function syntax and usage. This is the most important resource for correctly using the library functions. Each function is described in a standard form that includes its purpose, arguments, and usage.
7. For commercial use, separate licenses are required for the Science and Engineering Tools and Numerical Recipes in C functions used internally by some of the FrequenC library functions.

Programming Examples

The following programming examples show the use of the FrequenC library in a Microsoft Win32 console application. Other programming environments (e.g., a Windows GUI program) can also be used, and their code is very similar.

FrequenC Library Programming

A C program that uses the FrequenC library must include the FrequenC.h header file and link with the FrequenC.lib import library. To run, the program must have access to the FrequenC.dll.

It is recommended that the following code be used to load the FrequenC DLL near the beginning of the program.

```
// DLL Handle
HMODULE hFCLib; // FrequenC DLL module handle

// Load 32-bit FrequenC DLL
hFCLib=LoadLibrary("FREQUENC.DLL");
```

```
// Check if FrequenC DLL loaded OK
if((UINT)hFCLib<=32)
{
      // Show error message
      printf("Unable to Load FrequenC DLL\n");

      // End program
      return 1;
}
```

Example #1

As a first example of using the FrequenC library, consider the `CheckFrequenC()` function, which takes no argument and simply returns an integer that indicates the FrequenC version number. The return value is 300 for the current version 3.00 of the FrequenC library. Calling this function is recommended before using any of the other functions to confirm the correct library version and its basic operation. This function also serves here as a simple first example of the library usage.

The documentation sheet for this function is as follows:

The FrequenC Library	
NAME: `CheckFrequenC`	**FUNCTION:** `Check FrequenC version number`
SYNOPSIS: `int CheckFrequenC(void)`	
RETURN: `int`	`FrequenC version number`
REMARKS: `FrequenC version number changes with major Stable32 revisions that incorporate additional functions into FrequenC.dll`	

EXAMPLE:
`#include <frequenc.h>                    /* FrequenC header file */`
`int ver;                                   /* FrequenC version # */`
`.`
`.`
`ver=CheckFrequenC();                             /* call function */`
`printf("\nFrequenC Version # = %d", ver);         /* display ver # */`	

SEE ALSO:	
REFERENCE:	

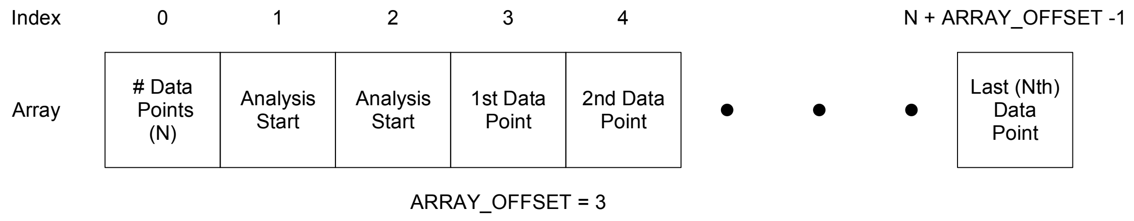
Example code for calling this function is shown below:

```
// Macro for FrequenC library version number
#define FREQUENC_DLL_VERSION_NUMBER 300

// Check FrequenC DLL version #
if(CheckFrequenC()!=FREQUENC_DLL_VERSION_NUMBER)
{
      // Show error message
      printf("Invalid FrequenC DLL Version\n");

      // End program
      return 0;
}
```

Data Format

Many of the FrequenC library functions use an one-dimensional array format for phase or frequency data where the first three elements hold the number of data points, and the start and end analysis limits. The data are assumed to represent equally-spaced points, with gaps denoted by a value of zero. All zeros are considered gaps in frequency data, while only embedded zeros are considered gaps in phase data. The sizes of the data arrays are determined by the calling program. The same data array format is also used to store timetags.

The FrequenC phase, frequency, and timetag double data arrays contain "headers" that hold the number of data points and the start and end analysis limits in their first three array elements (0-based indices 0, 1 and 2):



Example #2

After the FrequenC DLL has been loaded and its version and functionality confirmed, this second example shows the use of the `CalcFreqSigma()` function to calculate the original (non-overlapping) Allan deviation from a set of fractional frequency data using the data format described above. The documentation sheet for this function is as follows:

The FrequenC Library

NAME:	**FUNCTION:**
CalcFreqSigma	Calculate Allan deviation for frequency data

SYNOPSIS: `int CalcFreqSigma(F_TYPE y[], F_TYPE *p_sig)`

F_TYPE y[]	Frequency data array:
	y[0] = # data points
	y[1] = analysis start point
	y[2] = analysis end point

F_TYPE *p_sig	Pointer to sig, the Allan deviation for the frequency data
RETURN: int	The # of non-gap data points processed, or -1 if error

REMARKS:
Only data points between start and end analysis limits are analyzed.
There must be at least 2 non-gap analysis points.
All zeros are treated as gaps in frequency data.

EXAMPLE:
```
#include "frequenc.h"                        /* FrequenC header file */
F_TYPE y[512+ARRAY_OFFSET-1];               /* frequency data array */
F_TYPE sig;                                         /* Allan deviation */
int num;                                    /* # data points processed */
.
.
num=CalcFreqSigma(y, &sig);                              /* calc sigma */
if(num==-1)                                         /* check for error */
{
    printf("\nError");                                 /* error message */
}
else
{
    printf("\nSigma = %e", sig);                       /* display sigma */
    printf("\n# Data Points Analyzed = %d", num);      /* display num */
}
```

SEE ALSO: CalcPhaseSigma()

REFERENCE: NIST Technical Note 1337

Example code for calling this function is shown below:

```
// Create a set of classic freq test data as a static F_TYPE (double) array
// See NBS Monograph 140 Annex8.E
F_TYPE fDat[] = {9, 1, 9, 892, 809, 823, 798, 671, 644, 883, 903, 677};

// Sigma value
F_TYPE fSig;
```

```
// Call the Allan deviation function for these data
CalcFreqSigma(fDat, &fSig);

// Display the result
printf("Sigma = %e\n", fDat);
```

The expected Allan deviation result is 91.23.

Note that no tau or averaging factor is used for this function. The tau is that of the data set and the averaging factor is 1.

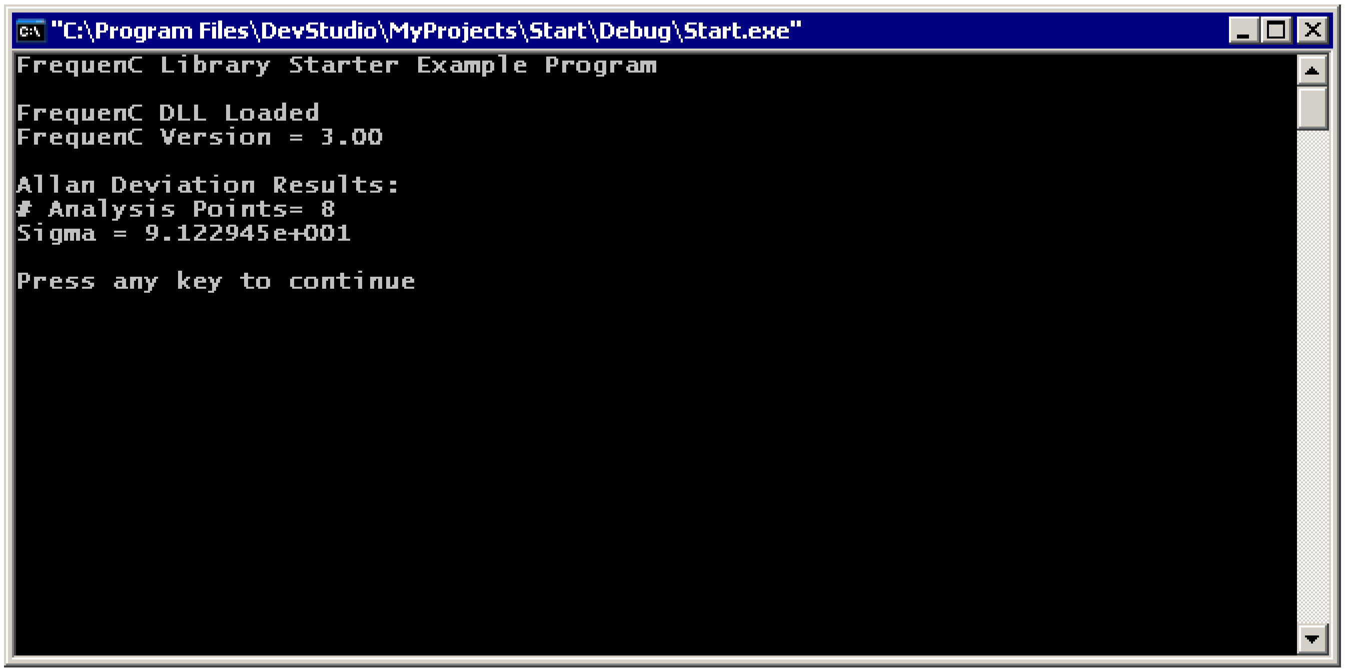
Win32 Console Application Test Program

Code for a complete Win32 console application test program is shown below:

```
int main(void)
{
      // Create a set of classic freq test data as a static F_TYPE array
      // See NBS Monograph 140 Annex8.E
      // This array includes header info: # pts, analy start, analy end
      // and 9 freq data points
      F_TYPE fDat[] = {9, 1, 9, 892, 809, 823, 798, 671, 644, 883, 903, 677};

      // Local variables
      int nNum;          // # analysis points
      int nVer;          // FrequenC version code
      F_TYPE fSig=1.0;   // Sigma

      // Show heading
      printf("FrequenC Library Starter Example Program\n\n");

      // Load 32-bit FrequenC DLL
      hFCLib=LoadLibrary("FREQUENC.DLL");

      // Check if FrequenC DLL loaded OK
      if((UINT)hFCLib<=32)
      {
            // Show error message
            printf("Unable to Load FrequenC DLL\n");

            // End program
            return 1;
      }
      else // OK
      {
            // Show success message
            printf("FrequenC DLL Loaded\n");
      }

      // Check FrequenC DLL version #
      if((nVer=CheckFrequenC())!=FREQUENC_DLL_VERSION_NUMBER)
      {
            // Show error message
            printf("Invalid FrequenC DLL Version\n");

            // End program
            return 1;
      }
      else // OK
      {
```

```
		// Show FrequenC version
		printf("FrequenC Version = %3.2f\n", nVer/100.);
	}

	// Call the Allan deviation function for test data array
	nNum=CalcFreqSigma(fDat, &fSig);

	// Show results message
	printf("\nAllan Deviation Results:\n");

	// Display the # analysis points
	printf("# Analysis Points= %d\n", nNum);

	// Show sigma result
	printf("Sigma = %e\n\n", fSig);

	// Done
	return 0;
}
```

This test program should produce the following results:



"C:\Program Files\DevStudio\MyProjects\Start\Debug\Start.exe"

```
FrequenC Library Starter Example Program

FrequenC DLL Loaded
FrequenC Version = 3.00

Allan Deviation Results:
# Analysis Points= 8
Sigma = 9.122945e+001

Press any key to continue
```