WAVECREST Corporation

Application Programming Interface (API) User's Guide

200002-03 REV A

WAVECREST Corporation continually engages in research related to product improvement. New material, production methods, and design refinements are introduced into existing products without notice as a routine expression of that philosophy. For this reason, any current *WAVECREST* product may differ in some respect from its published description but will always equal or exceed the original design specifications unless otherwise stated.

WAVECREST Corporation
A Technologies Company
7626 Golden Triangle Drive
Eden Prairie, MN 55344
(952) 831-0030
(800) 733-7128
www.wavecrest.com

First Printing: September 1999

This page intentionally left blank.

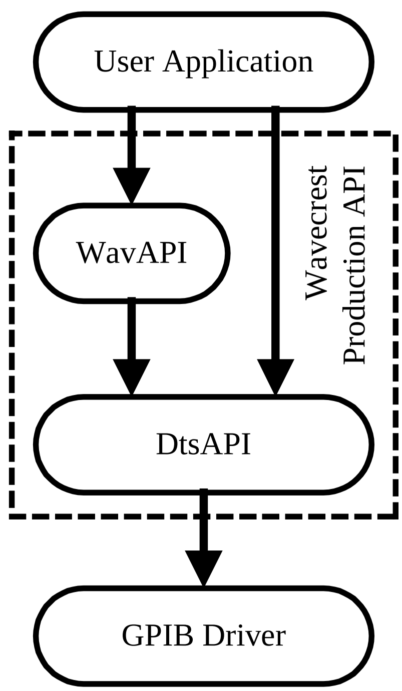
Table of Contents

CHAPTER 1 - INTRODUCTION

Wavecrest has implemented the Production API to provide direct access to the algorithms employed in the Visi6 software. It allows programmers to quickly integrate the functionality available in the Visi6 software into their own applications. Many tedious tasks such as GPIB interfacing and memory management are eliminated. A layered approach is utilized which provides access to all the statistics and plot data available in the Visi6 software, and versions are available for Microsoft Windows as well as many UNIX platforms. It also provides routines to leverage configurations established with the Visi6 software in order to streamline the transition from characterization laboratory to production floor.

1.1 ELEMENTS OF AN APPLICATION UTILIZING THE WAVECREST PRODUCTION API APPLICATION

An application utilizing the Wavecrest Production API is typically comprised of the following components:



Note that the Wavecrest Production API is divided into two blocks. The DtsAPI block provides a Hardware Abstraction Layer to isolate the higher level algorithms from the hardware itself. Although GPIB is the only physical medium supported at this time, this abstraction layer provides a means to easily migrate to other mediums such as Ethernet in the future.

The WavAPI block contains all the code required for the various Visi6 algorithms. It depends on the DtsAPI block for all lower level interactions with the hardware.

1.2 FILES COMPRISING THE WAVECREST PRODUCTION API

The Wavecrest Production API is comprised of a pair of header files and accompanying libraries. The header files are platform independent while the libraries are platform dependent. Libraries for Microsoft Windows applications are provided in the form of run-time Dynamic Link Libraries. Libraries for UNIX applications are provided in both static and shared forms on HP platforms and as static libraries only on SUN.

In addition to the header and library files, sample application source code and makefiles are also provided. There is also a directory containing various dataCOM patterns. Files are located on the CDROM in the following directory locations:

```
🗁─api
   │  api.pdf        // This manual in PDF form
   │  apitest.c      // Sample application source code
   │  dtsapi.h       // Low level header file
   │  wavapi.h       // High level header file
   🗁─hp10x
   │     libdts.a    // Low level static library
   │     libdts.sl   // Low level shared library
   │     libwav.a    // High level static library
   │     libwav.sl   // High level shared library
   │     makefile    // Makefile to build sample
   🗁─hp9x
   │     libdts.a    // Low level static library
   │     libdts.sl   // Low level shared library
   │     libwav.a    // High level static library
   │     libwav.sl   // High level shared library
   │     makefile    // Makefile to build sample
   🗁─solaris2
   │     libdts.a    // Low level static library
   │     libwav.a    // High level static library
   │     makefile    // Makefile to build sample
   🗁─sunos
   │     libdts.a    // Low level static library
   │     libwav.a    // High level static library
   │     makefile    // Makefile to build sample
   🗁─win32
   │     dtsapi.bas  // VBasic equivalent to include
   │     dtsapi.dll  // Low level shared library
   │     dtsapi.lib  // Stub header for linking
   │     makefile    // Makefile to build sample
   │     wavapi.bas  // VBasic equivalent to include
   │     wavapi.dll  // High level shared library
   │     wavapi.lib  // Stub header for linking
   🗁─patns          // Various dataCOM pattern files
```

1.3 WAVECREST PRODUCTION API INSTALLATION

To install the Wavecrest Production API, first create a target directory on the host system. Copy the files contained in the base directory (*apitest.c dtsapi.h wavapi.h*) as well as those from the particular platform directory to the newly created target directory.

1.4 BUILDING THE SAMPLE APPLICATION

Before attempting to build the sample application, the supported compiler should be installed and properly configured. This may include modifying the PATH environment variable so that the compiler executable can be launched from a command line. It may also involve setting INCLUDE and LIB environment variables so that the standard include files and libraries may be located by the compiler. Consult the compiler documentation for further information.

To build the sample application, on UNIX execute the following from a command prompt:

```
make
```

To build the sample application, on Microsoft Windows execute the following from a command prompt:

```
nmake
```

1.5 EXECUTING THE SAMPLE APPLICATION

Before attempting to execute the sample application, the supported GPIB interface card must be installed and properly configured. Consult the manufacturer's documentation for further information. The Wavecrest DTS207x should be powered, attached via GPIB cable, and the output from one of the Cal Signals should be connected to the Ch1 input. Test your configuration using Visi6 if possible.

To execute the sample application, issue the following from a command prompt:

```
./apitest
```

Note: proceeding the application name by "./" assures that the executable is launched even if the current directory is not included in the search path on UNIX.

If the sample application is successfully executed, the program should produce output similar to the following:

```
-Wavecrest Production API-
-   Sample Application   -

     Average: 5.002ns
     1-Sigma: 2.612ps
     Minimum: 4.992ns
     Maximum: 5.009ns
```

Congratulations! You have built your first application using the Wavecrest Production API.

1.6 REVIEWING THE SAMPLE APPLICATION

Let's examine the sample application in more detail.

❶
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "wavapi.h"

long main ( void )
  {
```
❷
```
  STAT tStat;
```
❸
```
  if ( DtsInitDev ( "dev5", 0, 5 ) )
    goto error;
```
❹
```
  memset ( &tStat, 0, sizeof ( STAT ) );
  WavDefStat ( &tStat );
```
❺
```
  if ( WavGetStat ( &tStat ) )
    goto error;
```
❻
```
  printf ( "-Wavecrest Production API-\n" );
  printf ( "-   Sample Application   -\n\n" );
  printf ( "     Average: %.3lfns\n", tStat.dMean * 1e9  );
  printf ( "     1-Sigma: %.3lfps\n", tStat.dSdev * 1e12 );
  printf ( "     Minimum: %.3lfns\n", tStat.dMini * 1e9  );
  printf ( "     Maximum: %.3lfns\n", tStat.dMaxi * 1e9  );
```
❼
```
  WavClrStat ( &tStat );
  DtsExitDev ( );
  return 0;

error:
  DtsExitDev ( );
  printf ( "DTS207x error\n" );
  return -1;
  }
```

Step 1: Declare Required Include Files

The Wavecrest Production API utilizes a number of custom structures which are declared in the two supplied include files. When ***wavapi.h*** is included, ***dtsapi.h*** is also automatically included.

Step 2: Allocate Required Structures

Each Visi6 window has a specific structure and several function calls to facilitate the data acquisition process. These structures contain input information concerning how to acquire the data, and output data as a result of the acquisition. The STAT structure is specific to the Statistics window.

Step 3: Initialize the DTS207x

DtsInitDev() must be called once at the beginning of your application in order to pass information concerning the GPIB configuration. The initialization values shown may need to be altered if a non-standard configuration is used. The first parameter is used to specify the GPIB device name on UNIX platforms and is ignored on Microsoft Windows. The second parameter is the board number, and the final parameter is the device number. See the documentation concerning this function call for complete details concerning configuration options.

All Production API functions return a non-zero value in the event of an error. These error codes are defined in the supplied include files. A successful call to DtsInitDev() must be accomplished before any other calls to the Wavecrest Production API.

Step 4: Initialize STAT Window Structure

Before utilizing an allocated Window Structure it must be initialized. This initialization may involve two or more steps.

The first step is to zero out the array using the standard memset() function. This step should only be performed once immediately after the structure is allocated and prior to it being used, as information concerning dynamic memory allocation is subsequently added to the structure.

The second step is to call the function call intended to initialize each of the particular structure parameters to their default values. In this case the WavDefStat() function is called. This step insures that all parameters contain reasonable values.

The final step is to manually modify any parameters from their default values. Great care should be used when manually adjusting parameters to insure that valid values are used.

Step 5: Perform Data Acquisition

A single call is made to perform the acquisition. Information concerning how to acquire the data is drawn from the STAT structure, and output data as a result of the acquisition is also returned in the STAT structure. If an error occurs during the acquisition a non-zero value is returned. See Appendix A for definition of error codes.

Note that the Wavecrest Production API performs it's own dynamic memory allocation as required. The calling application does not need to concern itself with memory management. However, since dynamic memory allocation information is contained within the structure, the supplied cleanup functions detailed below must be utilized in order to avoid memory leaks.

Acquisition functions may be called repeatedly with the same Window Structure. When doing so the output results contained within the structure are simply overwritten. Any dynamic memory previously allocated is re-utilized. Using the same Window Structure over and over again has the desirable attribute of reducing the memory fragmentation that would occur if memory was allocated, freed, and reallocated repeatedly.

Step 6: Print Results

Results to be printed are drawn directly from the STAT structure. Note that all results are returned in the units of Hertz, Volts, and seconds. Therefore a conversion factor may be required in order to display the results in more appropriate units.

Step 7: Cleanup and Terminate Application

Before terminating the application, the supplied cleanup functions should be called. WavClrStat() frees any dynamic memory which may have been allocated, and clears out the structure. DtsExitDev() closes the GPIB device driver. After this cleanup has been performed the application may terminate normally.

1.7 WHERE TO GO FROM HERE

This completes your introduction to the Wavecrest Production API. You should have installed the software, built a basic application, and reviewed it's composition. You should now have a basic understanding of the underlying framework, and be ready to leverage that understanding to further explore the interface. Subsequent chapters present additional detail concerning the structures and functions provided with the Wavecrest Production API.

CHAPTER 2 - HIGH LEVEL FUNCTIONS

The Wavecrest Production API provides four high level functions to implement each of the nine standard windows contained in the Visi6 software. Additional utility routines are provided to initialize parameters, perform a pulse-find operation, and interpret plot arrays. These routines relieve the programmer of many tedious tasks such as GPIB interfacing and memory management.

This chapter provides a general overview of these high level functions. To understand the particular input and output parameters involved in the context of a specific window, refer to the corresponding structures addressed in the following chapter.

2.1 STANDARD WINDOW ROUTINES

The four high level functions used to implement each of the nine standard windows contained in the Visi6 software are declared as follows:

```
void WavDefXxxx ( YYYY *tZzzz );
long WavGetXxxx ( YYYY *tZzzz );
void WavClrXxxx ( YYYY *tZzzz );
long WavCfgXxxx ( YYYY *tZzzz, char *sFile);
```

Where the following substitutions are made:

Window	Xxxx	YYYY	tZzzz
Oscilloscope	Osci	OSCI	tOsci
Histogram	Hist	HIST	tHist
Jitter Analysis	Jitt	JITT	tJitt
Function Analysis	Func	FUNC	tFunc
Time Digitizer	Tdig	TDIG	tTdig
dataCOM	Dcom	DCOM	tDcom
Eye Histogram	Eyeh	EYEH	tEyeh
Time Series	Tser	TSER	tTser
Statistics	Stat	STAT	tStat
Random Data	Rand	RAND	tRand

Note: _stdcall and DllCall are part of the function definitions in the header file, but can essentially be ignored. They are utilized to provide options when building and using DLL's on Microsoft Windows. They are implemented to allow the same header file to be used for both building the DLL and importing the DLL, insuring consistent declarations.

2.1.1 Fill a Window Structure with Default Parameters

```
void WavDefXxxx ( YYYY *tZzzz );
```

Input:

tZzzz Pointer to Window Structure

Return:

None

Example:

```
STAT tStat;
memset ( &tStat, 0, sizeof ( STAT ) );
WavDefStat ( &tStat );
```

This function is used to fill a Window Structure with default values. Using this function insures that all parameters contain reasonable values. It is recommended that this function is called first even if parameters within the structure will be subsequently adjusted manually.

During data acquisition dynamic memory will be acquired as necessary. This memory is tracked within the window structure. Before calling this function with a newly allocated Window Structure you should zero out the array using the standard memset() function. This step insures that information within the structure concerning dynamic memory allocation is cleaned out prior to using the structure. This step should be performed once and only once on a given structure.

In spite of owning memory, this function may be called repeatedly for a given window structure to reestablish default parameters, as it does not effect any of the parameters pertaining to memory allocation. Use the cleanup function detailed later in the chapter to clear out a structure after it has been used. Failure to use the cleanup function before discarding a window structure will result in a memory leak.

2.1.2 Perform a Data Acquisition

```
long WavGetXxxx ( YYYY *tZzzz );
```

Input:

tZzzz Pointer to Window Structure

Return:

0 on Success or Error Code on Failure

Example:

```
if ( WavGetStat ( &tStat ) )
        goto ErrorHandler;
```

This function call is used to perform a data acquisition. Information concerning how to acquire the data is drawn from the Window Structure, and output data as a result of the acquisition is also returned in the Window Structure.

Note that the Wavecrest Production API performs it's own dynamic memory allocation as required. The calling application does not need to concern itself with memory management. However, since dynamic memory allocation information is contained within the structure, the supplied cleanup functions detailed below must be utilized in order to avoid memory leaks.

Acquisition functions may be called repeatedly with the same Window Structure. When doing so the output results contained within the structure are simply overwritten. Any dynamic memory previously allocated is re-utilized. Using the same Window Structure over and over again has the desirable attribute of reducing the memory fragmentation that would occur if memory was allocated, freed, and reallocated repeatedly.

2.1.3 Clear a Window Structure Prior to Release

```
void WavClrXxxx ( YYYY *tZzzz );
```

Input:

tZzzz Pointer to Window Structure

Return:

None

Example:

```
if ( WavClrStat ( &tStat ) )
    goto ErrorHandler;
```

Before a Window Structure is released this function should be called. This function frees any dynamic memory that may have been allocated during a previous data acquisition, and then clears out the structure.

2.1.4 Load Settings from Visi6 Configuration File

```
long WavCfgXxxx ( YYYY *tZzzz, char *sFile );
```

Input:

tZzzz Pointer to Window Structure
sFile Pointer to File Name

Return:

0 on Success or Error Code on Failure

Example:

```
STAT tStat;
memset ( &tStat, 0, sizeof ( STAT ) );
WavCfgStat ( &tStat, “myconfig.stc” );
```

This function is used to load a Window Structure with values from a Visi6 configuration file. The ability to do so streamlines the transition from characterization laboratory to production floor. The requirements to zero the Window Structure prior to calling the function are the same as the function to load Default Parameters outlined above.

2.2 High Level Utility Routines

These high level utility routines are provided to initialize parameters, perform a pulse-find operation, and interpret plot arrays.

2.2.1 Get API Version

```
long WavGetVers ( void );
```

Input:

    None

Return:

    Major version in high byte, minor version in low byte

Example:

    VerNum = WavGetVers ( );

This function may be called to determine the current API version.

2.2.2 Fill a Parameter Structure with Default Values

```
void WavDefParm ( PARM *tParm );
```

Input:

tParm	Pointer to Parameter Structure

Return:

    None

Example:

    PARM tParm;
    WavDefParm (&tParm);

This function is used to fill a Parameter Structure with default values. These parameters could then be downloaded to the DTS207x by calling the DtsSetParm() function. Using this function insures that all parameters contain reasonable values.

This function is used internally by the API itself, but may be called by a user application as well. It would typically be used if an application were calling some of the lower level functions such as DtsRqstAcq(), DtsGetData(), or DtsGetMacr() to implement a user defined algorithm.

It is not necessary to clear a Parameter Structure using the standard memset() function prior to calling this function, as no dynamic memory allocation information is contained within the Parameter Structure.

2.2.3 Perform a Pulse-find Operation

```
long WavPulsFnd ( PARM *tParm, long lWind );
```

Input:

- **tParm** Pointer to Parameter Structure
- **lWind** Window Type, out of the following defined types:

```
WIND_OSCI   WIND_HIST   WIND_JITT
WIND_FUNC   WIND_TDIG   WIND_DCOM
WIND_EYEH   WIND_TSER   WIND_STAT
WIND_RAND
```

Return:

0 on Success or Error Code on Failure

Example:

```
STAT tStat;
memset ( &tStat, 0, sizeof ( STAT ) );
WavDefStat ( &tStat );
if ( WavPulsFnd ( &tStat.tParm, WIND_STAT ) )
    goto ErrorHandler;
```

This function is used to perform a pulse-find operation in conjunction with the high level window functions. The pulse-find feature determines minimum and maximum voltage levels for the selected channels and/or arms and sets the voltage thresholds based on the percentage set in the lFndPcnt field in the tParm structure. Although a lower level function DtsPulsFnd() exists, it should not be used in conjunction with the high level window functions.

2.2.4 Determine X-value in Plot Structure Based on Index

```
double WavGetXval ( PLOT *tPlot, long lIndx );
```

Input:

tPlot	Pointer to Plot Structure
lIndex	Index from which to determine X-value, range (0 to tPlot.lNumb-1)

Return:

X-value

Example:

```
JITT tJitt;
double XvalOfYmax;
memset ( &tJitt, 0, sizeof ( JITT ) );
WavDefJitt ( &tJitt );
if ( WavGetJitt ( &tJitt ) )
        goto ErrorHandler;
XvalOfYmax = WavGetXval ( &tJitt.tFftN,
                tJitt.tFftN.dYmaxIndx );
```

This function is used to assist a user application in extracting information from a Plot Structure. In order to reduce memory requirements, only Yaxis values are contained within Plot Structures. The Xaxis values can be calculated using this function.

The example above details how the maximum jitter frequency can be determined from an N-clock Jitter Analysis.

2.2.5 Determine Y-value in Plot Structure Based on Index

```
double WavGetYval ( PLOT *tPlot, long lIndx );
```

Input:

tPlot	Pointer to Plot Structure
lIndx	Index from which to determine Y-value, range (0 to tPlot.lNumb-1)

Return:

Y-value

Example:

```
JITT tJitt;
double Yval;
if ( WavGetJitt ( &tJitt ) )
        goto ErrorHandler;
Yval = WavGetYval ( &tJitt.tFftN, 0 );
```

This function is used to assist a user application in extracting information from a Plot Structure. It is primarily included to assist when programming in Microsoft Visual Basic. When programming in C the data array can be accessed directly, so this function adds unnecessary overhead.

2.2.6 Determine Tail-fit Y-value for a given X-value

```
double WavGetTfit ( SIDE *tSide, double dXval );
```

Input:

tSide	Pointer to Tail-fit Side Structure
dXval	X-value from which to determine Y-value

Return:

Y-value

Example:

```
HIST tHist;
double minYval, maxYval;
if ( tHist.tTfit.lGood )
  {
  minYval = WavGetTfit ( &tHist.tTfit.tL, tHist.tTfit.tL.dLoValu );
  maxYval = WavGetTfit ( &tHist.tTfit.tL, tHist.tTfit.tL.dHiValu );
  }
```

In order to reduce memory requirements, only coefficients for the idealized curve representing the fitted tails are stored when tail-fits are performed. This function can be used to generate curves representing the idealized curves. This function should only be applied after a tail-fit has been successfully completed, as indicated by the "lGood" flag in the TFIT structure.

The example above details how the two endpoints of the idealized tail-fit curve can be determined for the left tail of a Histogram window.

2.2.7 Free all internal memory

```
void WavFreeMem ( void );
```

Input:

None

Return:

None

Example:

```
WavFreeMem();
```

This function may be called in order to free any memory that was allocated for internal use. It is not normally necessary to call this function as all memory is freed when the application is terminated. However, if multiple threads of execution are used it may be desirable to call this function whenever an individual thread is terminated.

CHAPTER 3 – STRUCTURES

The Wavecrest Production API provides structures to be used in conjunction with the high level function calls detailed in the previous chapter. Each of these structures is specific to one of the nine standard windows contained in the Visi6 software. Additional utility structures are defined which are used within these standard window functions.

3.1 STANDARD WINDOW STRUCTURES

The following high level structures are used in conjunction with the function calls detailed in the previous chapter. Each is specific to one of the nine standard windows contained in the Visi6 software.

3.1.1 Structure used for Oscilloscope window

```
typedef struct
  {
  /* Input parameters  */
  PARM    tParm;
  FFTS    tFfts;
  long    lStrt, lStop, lIncr;
  /* Output parameters */
  long    lGood;
  PLOT    tTime[ POSS_CHNS ];
  PLOT    tFreq[ POSS_CHNS ];
  } OSCI;
```

tParm
    Contains acquisition parameters, see end of chapter for details

tFfts
    FFT window and analysis parameters, see end of chapter for details

lStrt
    Start time in picosec's (20,000 to 100,000,000), the default is 20000

lStop
    Stop time in picosec's (20,000 to 100,000,000), the default is 100000

lIncr
    Time increment in picoseconds, the default is 500 and minimum is 10

lGood
    Flag indicates valid output data in structure

tTime
    Time domain plot of voltage data

tFreq
    Frequency domain plot of voltage data

3.1.2 Structure used for Histogram window

```
typedef struct
  {
  /* Input parameters  */
  PARM    tParm;
  double  dUnitInt;
  long    lPassCnt, lErrProb;
  long    lTailFit, lForcFit;
  long    lMinHits;
  long    lFndEftv;
  long    lMinEftv, lMaxEftv;
  long    lAutoFix;
  /* Output parameters */
  long    lGood;
  long    lNormCnt;
  double  dNormMin, dNormMax;
  double  dNormAvg, dNormSig;
  long    lAcumCnt;
  double  dAcumMin, dAcumMax;
  double  dAcumAvg, dAcumSig;
  PLOT    tNorm;
  PLOT    tAcum;
  PLOT    tMaxi;
  PLOT    tBath;
  PLOT    tEftv;
  TFIT    tTfit;
  } HIST;
```

tParm
Contains acquisition parameters, see end of chapter for details

dUnitInt
Unit Interval to assess Total Jitter, only used if tail-fit is enabled. This value is entered in seconds, the default is 1e-9 seconds (1ns).

lPassCnt
WavGetHist() can be called repeatedly with the same HIST structure. Data is then accumulated in the **tAcum** and **tMaxi** plot structures. This parameter tracks acquisitions so far, and may be set to 0 to reset. When set to 0 the **tAcum** and **tMaxi** plot structures are flushed. It will be automatically incremented by the WavGetHist() function.

lErrProb
Error probability for Total Jitter, the valid range is -1 to -16 and the default value is -12. This value is used in conjunction with the bathtub curve after the successful completion of a tail-fit in order to project the value of Total Jitter.

lTailFit
If non-zero a tail-fit will be attempted on the **tAcum** data array, the default is to not attempt a tail-fit

lForcFit
If non-zero use the force-fit method, the default is disabled

lMinHits
Minimum hits before attempting a tail-fit in 1000's, the default is 50

lFndEftv
Flag to indicate that an effective jitter calculation is to be attempted
lMinEftv, lMaxEftv
Defines the range of the bathtub curve which is to be used to calculate an effective jitter value. The defaults for lMaxEftv and lMinEftv are -4 and -12 respectively. The valid range is -1 to -16 and lMinEftv must be less then lMaxEftv.
lAutoFix
If true perform a pulsefind as required
lGood
Flag indicates valid output data in structure
lNormCnt
Number of hits in **tNorm** plot array below
dNormMin
Minimum value in **tNorm** plot array below
dNormMax
Maximum value in **tNorm** plot array below
dNormAvg
Average value in **tNorm** plot array below
dNormSig
1-Sigma value in **tNorm** plot array below
lAcumCnt
Number of hits in **tAcum** plot array below
dAcumMin
Minimum value in **tAcum** plot array below
dAcumMax
Maximum value in **tAcum** plot array below
dAcumAvg
Average value in **tAcum** plot array below
dAcumSig
1-Sigma value in **tAcum** plot array below
tNorm
Histogram of data from latest acquisition only
tAcum
Histogram of data from all acquisitions combined
tMaxi
Histogram with the maximum value obtained for every particular bin across all of the acquisitions performed so far
tBath
Bathtub curves determined from PDF, only valid when a successful tail-fit has been performed
tEftv
Effective Bathtub curves if lFndEftv is set and a valid fit is obtained
tTfit
Structure containing tail-fit info, only valid when a successful tail-fit has been performed. See end of chapter for additional details

3.1.3 Structure used for Jitter Analysis window

```
typedef struct
  {
  /* Input parameters  */
  PARM    tParm;
  FFTS    tFfts;
  long    lIncStop;
  long    lMaxStop;
  long    lAutoFix;
  double  dCornFrq;
  double  dRjpjFmn;
  double  dRjpjFmx;
  long    lFftAvgs;
  /* Output parameters */
  long    lGood;
  double  dWndFact1Clk;
  double  dWndFactNClk;
  PLOT    tSigm;
  PLOT    tPeak;
  PLOT    tFft1;
  double  dPjit1Clk;
  double  dRjit1Clk;
  long   *lPeakData1Clk;
  long    lPeakNumb1Clk;
  long    lPeakRsvd1Clk;
  PLOT    tFftN;
  double  dPjit1Clk;
  double  dRjit1Clk;
  long   *lPeakData1Clk;
  long    lPeakNumb1Clk;
  long    lPeakRsvd1Clk;
  double  dFreq;
  } JITT;
```

tParm

Contains acquisition parameters, see end of chapter for details

tFfts

FFT window and analysis parameters, see end of chapter for details

lIncStop

Increase stop count between acquisitions in increments of this value, the default is 1. Stop counts range from **tParm.lStopCnt** to **lMaxStop**

lMaxStop

Maximum stop count to collect data for, the default is 256. The stop count will be incremented from the value in **tParm.lStopCnt** to this.

lAutoFix

If true calculate the above parameters based on the following corner frequency plus information measured on the live data signal

dCornFrq

Corner Frequency for RJ+PJ in Hertz. This value is used in conjunction with the Bit Rate and pattern to determine the maximum stop count to be used to acquire RJ+PJ data. A lower value increase acquisition time. The default value is 637e3.

dRjpjFmn
Minimum integration limit for RJ+PJ in Hertz, a negative value disables filter. This filter is disabled by default.

dRjpjFmx
Maximum integration limit for RJ+PJ in Hertz, a negative value disables filter. This filter is disabled by default.

lFftAvgs
This variable is raised to the power of 2 to determine the number of acquisitions to use in order to average the FFT output. The default is a value of 0 which uses a single acquisition, and hence no averaging.

lGood
Flag indicates valid output data in structure

dWndFact1Clk, dWndFactNClk
These values are all used internally, DO NOT ALTER!

tSigm
Contains the 1-Sigma plot array

tPeak
Contains the (max - min) plot array

tFft1
Frequency plot data on 1-clock basis

dPjit1Clk
Periodic jitter calculated on 1-clk basis

dRjit1Clk
Random jitter calculated on 1-clk basis

lPeakData1Clk
Tracks detected spikes in RJ+PJ data. This structure is not normally directly access by an application program.

lPeakNumb1Clk
Count of detected spikes, indicates the number of values in the **lPeakData1Clk** array.

lPeakRsvd1Clk
Used to track memory allocation for **lPeakData1Clk** values

tFftN
Frequency plot data on N-clock basis

dPjitNClk
Periodic jitter calculated on N-clk basis

dRjitNClk
Random jitter calculated on N-clk basis

lPeakDataNClk
Tracks detected spikes in RJ+PJ data. This structure is not normally directly access by an application program.

lPeakNumbNClk
Count of detected spikes, indicates the number of values in the **lPeakDataNClk** array.

lPeakRsvdNClk
Used to track memory allocation for **lPeakDataNClk** values

dFreq
Carrier frequency

3.1.4 Structure used for Function Analysis window

```
typedef struct
  {
  /* Input parameters  */
  PARM    tParm;
  FFTS    tFfts;
  long    lIncStrt;
  long    lMaxStrt;
  long    lAnlMode;
  long    lAutoFix;
  long    lSpanCnt;
  long    lDataPts;
  /* Output parameters */
  long    lGood;
  PLOT    tTime;
  PLOT    tDerv;
  PLOT    tFftT;
  PLOT    tFftD;
  PLOT    tSigm;
  PLOT    tPeak;
  PLOT    tMini;
  PLOT    tMaxi;
  double  dSigmAvg;
  double  dSigmMin;
  double  dSigmMax;
  double  dTimePos;
  double  dTimeNeg;
  long    lTimePosLoc;
  long    lTimeNegLoc;
  double  dDervPos;
  double  dDervNeg;
  long    lDervPosLoc;
  long    lDervNegLoc;
  double  dFreq;
  } FUNC;
```

tParm

Contains acquisition parameters, see end of chapter for details. Note that external Arm1 is enabled by default.

tFfts

FFT window and analysis parameters, see end of chapter for details

lIncStrt

Increase start count by this value, the default is 1. Data is collected for start counts ranging from **tParm.lStrtCnt** to **lMaxStrt**.

lMaxStrt

Maximum start count to collect data for, the default is 250. The start count will be incremented from the value in **tParm.lStrtCnt** to this.

lAnlMode

Relationship of start and stop counts, use one of:

ANL_FNC_FIRST	Arm start first
ANL_FNC_PLUS1	Stop = Start + 1
ANL_FNC_START	Stop = Start

lAutoFix
If true calculate the above parameters based on **lSpanCnt** and **lDataPts** plus information measured on the live data signal

lSpanCnt
The number of edges across which to measure

lDataPts
The total data points within span to measure

lGood
Flag indicates valid output data in structure

tTime
Time domain plot data

tDerv
1st derivative of time domain plot data

tFftT
Frequency domain plot data

tFftD
Frequency domain of 1st derivative plot data

tSigm
Contains the 1-Sigma plot array

tPeak
Contains the (max - min) plot array

tMini
Contains the Minimum plot array

tMaxi
Contains the Maximum plot array

dSigmAvg
Average 1-Sigma value

dSigmMin
Minimum 1-Sigma value

dSigmMax
Maximum 1-Sigma value

dTimePos
Maximum increase between time values

dTimeNeg
Maximum decrease between time values

lTimePosLoc
Index to maximum increase between values

lTimeNegLoc
Index to maximum decrease between values

dDervPos
Maximum increase between 1st derivative values

dDervNeg
Maximum decrease between 1st derivative values

lDervPosLoc
Index to maximum increase between 1st derivative values

lDervNegLoc
Index to maximum decrease between 1st derivative values

dFreq
Carrier frequency

3.1.5 Structure used for Time Digitizer window

```
typedef struct
  {
  /* Input parameters  */
  PARM    tParm;
  FFTS    tFfts;
  long    lAutoFix;
  double  dMaxFreq;
  long    lFftAvgs;
  /* Output parameters */
  long    lGood;
  PLOT    tTime;
  PLOT    tStmp;
  PLOT    tFft1;
  PLOT    tFftN;
  double  dCarFreq;
  double  dSmpRate;
  double  dFftNdBc;
  } TDIG;
```

tParm
Contains acquisition parameters, see end of chapter for details
lStampTm is enabled for this window by default

tFfts
FFT window and analysis parameters, see end of chapter for details

lAutoFix
If true calculate the above parameters based on **dMaxFreq** plus information measured on the live data signal

dMaxFreq
Maximum Frequency information that is desired

lFftAvgs
This variable is raised to the power of 2 to determine the number of acquisitions to use in order to average the FFT output. The default is a value of 0 which uses a single acquisition, and hence no averaging.

lGood
Flag indicates valid output data in structure

tTime
Time domain plot data

tStmp
Time stamp data array, not normally plotted

tFft1
Frequency plot data on 1-clock basis

tFftN
Frequency plot data on N-clock basis

dCarFreq
Carrier frequency

dSmpRate
Sampling rate

dFftNdBc
dBc assessed on 1-clock FFT data

3.1.6 Structures used for dataCOM window

```
typedef struct
  {
  /* Input parameters  */
  PARM    tParm;
  char    sPtnName[ 128 ];
  long    lAcqMode, lRndMode;
  long    lQckMode, lIntMode;
  long    lGetRate, lTailFit;
  long    lErrProb, lPassCnt;
  long    lFftAvgs;
  SPEC    tRateInf, tDdjtInf, tRjpjInf;
  double  dDdjtLpf, dDdjtHpf;
  double  dRjpjFmn, dRjpjFmx;
  double  dBitRate, dCornFrq;
  long    lHeadOff;
  long    lFndEftv;
  long    lMinEftv, lMaxEftv;
  long    lFiltEnb;
  /* Output parameters */
  long    lGood;
  PATN    tPatn;
  double  dWndFact;
  long    lMaxStop, lPosRoll, lNegRoll;
  DDJT   *tDdjtData;
  long    lDdjtRsvd;
  double *dMeasData[ 2 ];
  long    lMeasRsvd[ 2 ];
  double *dRjpjData[ 4 ];
  long    lRjpjRsvd[ 4 ];
  double *dTfitData[ 4 ];
  long    lTfitRsvd[ 4 ];
  long   *lPeakData[ 4 ];
  long    lPeakNumb[ 4 ], lPeakRsvd[ 4 ];
  double *dFreqData[ 4 ];
  long    lFreqRsvd[ 4 ];
  double *dTailData[ 4 ];
  long    lTailRsvd[ 4 ];
  long    lHits;
  double  dDdjt, dRang;
  double  dRjit[ 4 ], dPjit[ 4 ], dTjit[ 4 ];
  double  dEftvLtDj[ 4 ], dEftvLtRj[ 4 ];
  double  dEftvRtDj[ 4 ], dEftvRtRj[ 4 ];
  PLOT    tRiseHist, tFallHist;
  PLOT    tRiseMeas, tFallMeas;
  PLOT    tNormDdjt;
  PLOT    tHipfDdjt, tLopfDdjt;
  PLOT    tBathPlot[ 4 ];
  PLOT    tEftvPlot[ 4 ];
  PLOT    tSigmNorm[ 4 ], tSigmTail[ 4 ];
  PLOT    tFreqNorm[ 4 ], tFreqTail[ 4 ];
  } DCOM;
```

tParm

Contains acquisition parameters, see end of chapter for details

sPtnName

Name of pattern file to be used, the file must exist or an error will be returned. The first time WavGetDcom() is called the pattern is loaded into **tPatn** which is the internal representation of the pattern. If the pattern file is to be changed, WavClrDcom() should be called first to clear the internal representation so that the new pattern will be loaded. The default file is k285.ptn

lAcqMode

Mask defining modes for RJ+PJ acquire, set bits as follows:
Bit3: PW- Bit2: PW+ Bit1: Per- Bit0: Per+
The default mode is to acquire Per+ only.

lRndMode

Non-zero value enables random mode, valid when auto-arming only. This is not enabled by default.

lQckMode

Non-zero value enables quick mode, valid with external arm only. When enabled a sparse set is obtained for RJ+PJ analysis, which significantly reduces acquisition time. High frequency performance is reduced when this option is enabled. This is not enabled by default.

lIntMode

Interpolation mode for RJ+PJ analysis, non-zero value selects linear interpolation, otherwise cubic interpolation is used. Cubic interpolation is the default mode.

lGetRate

If non-zero Bit Rate will be measured, otherwise appropriate value must be supplied in **dBitRate** variable. The default is to measure the Bit Rate. This mode is NOT valid when using random mode, the value must be supplied.

lTailFit

If non-zero a tail-fit will be tried, valid with external arm only. Not enabled by default.

lErrProb

Error probability for Total Jitter, the valid range is -1 to -16 and the default value is -12. This value is used in conjunction with the bathtub curve after the successful completion of a tail-fit in order to project the value of Total Jitter.

lPassCnt

Acquisitions so far, set to 0 to reset

lFftAvgs

This variable is raised to the power of 2 to determine the number of acquisitions to use in order to average the FFT output. The default is a value of 0 which uses a single acquisition, and hence no averaging.

tRateInf

Parameters to acquire Bit Rate, see SPEC structure later in chapter

tDdjtInf

Parameters to acquire DCD+DDJ, see SPEC structure later in chapter

tRjpjInf
Parameters to acquire RJ+PJ, see SPEC structure later in chapter

dDdjtLpf
Low pass DCD+DDJ filter frequency in Hertz, negative value disables filter. This is only valid when external arming is enabled. This filter is disabled by default.

dDdjtHpf
High pass DCD+DDJ filter frequency in Hertz, a negative value disables filter. This is only valid when external arming is enabled. This filter is disabled by default.

dRjpjFmn
Minimum integration limit for RJ+PJ in Hertz, a negative value disables filter. This filter is disabled by default.

dRjpjFmx
Maximum integration limit for RJ+PJ in Hertz, a negative value disables filter. This filter is disabled by default.

dBitRate
Bit Rate, may be specified or measured. If **lGetRate** is non-zero this value is measured and placed in this field. If **lGetRate** is zero an appropriate value must be placed in the variable. This value nust be supplied when Random mode is being used.

dCornFrq
Corner Frequency for RJ+PJ in Hertz. This value is used in conjunction with the Bit Rate and pattern to determine the maximum stop count to be used to acquire RJ+PJ data. A lower value increase acquisition time. The default value is 637e3.

lHeadOff
Header offset, valid when external arming only. This offset value can be used to skip past header information and into the repeating data pattern stream. This can be useful when analyzing data from disk drives when the pattern marker may be synchronized with the start of frame data. The default value is 0.

lFndEftv
Flag to indicate that an effective jitter calculation is to be attempted

lMinEftv, lMaxEftv
Defines the range of the bathtub curve which is to be used to calculate an effective jitter value. The defaults for lMaxEftv and lMinEftv are -4 and -12 respectively. The valid range is -1 to -16 and lMinEftv must be less then lMaxEftv.

lFiltEnb
Flag to enable IDLE character insertion filter. When enabled any edge measurements that are not within +/-0.5 UI will be discarded.

lGood
Flag indicates valid output data in structure

tPatn
Internal representation of pattern, the internal details of this structure are not important from an application standpoint. The first time WavGetDcom() is called the pattern is loaded into **tPatn** which is used internally for all subsequent acquisition and analysis.

dWndFact, lMaxStop, lPosRoll, lNegRoll
These values are all used internally, DO NOT ALTER!

tDdjtData
Raw DCD+DDJ measurements, see DDJT structure later in chapter for additional details, this structure is not normally directly access by an application program.

lDdjtRsvd
Used to track memory allocation for **tDdjtData** structures

dMeasData
Raw all-measurements histogram data, only valid when auto-arming is used. This structure is not normally directly access by an application program.

lMeasRsvd
Used to track memory allocation for **dMeasData** values

dRjpjData
Raw variance data, this structure is not normally directly access by an application program.

lRjpjRsvd
Used to track memory allocation for **dRjpjData** values

dTfitData
Raw tail-fit data if tail-fit data is enabled and successful, as indicated by the **lGood** variable in the **tTfit** structure being non-zero. This structure is not normally directly access by an application program.

lTfitRsvd
Used to track memory allocation for **dTfitData** values

lPeakData
Tracks detected spikes in RJ+PJ data. This structure is not normally directly access by an application program.

lPeakNumb
Count of detected spikes, indicates the number of values in the **lPeakData** array.

lPeakRsvd
Used to track memory allocation for **lPeakData** values

dFreqData
Raw FFT output when averaging is enabled. This structure is not normally directly access by an application program.

lFreqRsvd
Used to track memory allocation for **dFreqData** values

dTailData
Raw tail-fit FFT output when tail-fit and averaging are both enabled. This structure is not normally directly access by an application program.

lTailRsvd
Used to track memory allocation for dTailData values

dHits
Total samples taken to calculate DDJT, RJ, and PJ values combined. Gives an indication of the actual data to support the calculated total jitter number.

dDdjt

DCD+DDJ jitter number in seconds.

dRang

Pk-Pk of all-measurements histogram, valid when auto-arming only.

dRjit

Random jitter number in seconds, for each of the enabled modes.

dPjit

Periodic jitter number in seconds, for each of the enabled modes.

dTjit

Total jitter number in seconds, for each of the enabled modes.

dEftvLtDj, dEftvLtRj, dEftvRtDj, dEftvRtRj

Effective jitter in seconds for each of the enabled modes is stored in this variables if calculated. In order to calculate the effective jitter **lFndEftv** must contain a non-zero value. Since the effective jitter is calculated by optimizing a curve-fit a result is not guaranteed. If the curve-fit fails a negative value will be returned in these variables.

tRiseHist

DCD+DDJ histogram of rising edges

tFallHist

DCD+DDJ histogram of falling edges

tRiseMeas

Rising all-measurements histogram, valid when auto-arming only.

tFallMeas

Falling all-measurements histogram, valid when auto-arming only.

tNormDdjt

DCD+DDJvsUI plot, valid when external arming is enabled only.

tHipfDdjt

High Pass Filtered DCD+DDJvsUI plot, valid when external arming is enabled only. This is only calculated when **dDdjtHpf** is a non-negative number. When calculated, the **dDdjt** value is adjusted based on this filter being applied.

tLopfDdjt

Low Pass filtered DCD+DDJvsUI plot, valid when external arming is enabled only. This is only calculated when **dDdjtHpf** is a non-negative number.

tBathPlot

Bathtub plots, one for each of the modes enabled in **lAcqMode**

tEftvPlot

Effective Bathtub curves if **lFndEftv** is set and a valid fit is obtained

tSigmNorm

1-Sigma plots, one for each of the modes enabled in **lAcqMode**

tSigmTail

1-Sigma tail-fits, only valid if tail-fit is enabled. One for each of the modes enabled in **lAcqMode**

tFreqNorm

Frequency plots, one for each of the modes enabled in **lAcqMode**

tFreqTail

Tail-fit FFT plots, only valid if tail-fit is enabled. One for each of the modes enabled in **lAcqMode**

```
typedef struct
  {
  long    lSampCnt;
  double  dMaxSerr;
  long    lPtnReps;
  } SPEC;
```

lSampCnt
Sample size to use when acquiring data, the default value is 100

dMaxSerr
Value of standard error which is tolerated, used to identify wrong pattern or other setup error. The default value is 0.5

lPtnReps
Patterns to sample across, the default values are 10 for **tRateInf** and 1 for **tDdjtInf** and **tRjpjInf**

```
typedef struct
  {
  double  dMean;
  double  dVars;
  double  dMini;
  double  dMaxi;
  double  dDdjt;
  double  dFilt;
  long    lNumb;
  } DDJT;
```

dMean
Average value for this span

dVars
Variance value for this span

dMini
Minimum value for this span

dMaxi
Maximum value for this span

dDdjt
Static displacement for this span (UI)

dFilt
DDJT after HPF is applied (UI)

lNumb
Number of measures in this span

3.1.7 Structure used for Eye Histogram window

```
typedef struct
  {
  /* Input parameters  */
  PARM    tParm;
  long    lPassCnt, lRefEdge;
  long    lErrProb;
  long    lClokSmp, lFiltSmp;
  long    lTailFit, lForcFit;
  long    lMinHits;
  long    lFndEftv;
  long    lMinEftv, lMaxEftv;
  double  dMinSpan;
  /* Output parameters */
  long    lGood;
  long    lRiseCnt, lFallCnt;
  double  dDataMin, dDataMax;
  double  dDataSig, dAvgSkew;
  double  dUnitInt;
  long    lUnitOff;
  double  dRiseMin, dRiseMax;
  double  dFallMin, dFallMax;
  long    lSpanCnt;
  PLOT    tRise, tFall;
  PLOT    tRiseProb, tFallProb;
  PLOT    tBath;
  TFIT    tTfit;
  } EYEH;
```

tParm

Contains acquisition parameters, see end of chapter for details

lPassCnt

WavGetEyeh() can be called repeatedly with the same EYEH structure. Data is then accumulated in the plot structures. This parameter tracks acquisitions so far, and may be set to 0 to reset. When set to 0 the plot structures are flushed. It will be automatically incremented by the WavGetEyeh() function.

lRefEdge

Clock edge which all data is in reference to, valid values are:

EDGE_FALL or EDGE_RISE

The default value is EDGE_RISE

lErrProb

Error probability for Total Jitter, the valid range is -1 to -16 and the default value is -12. This value is used in conjunction with the bathtub curve after the successful completion of a tail-fit in order to project the value of Total Jitter.

lClokSmp

Sample size while acquiring clock rate, the default value is 10000

lFltSmp

Sample size when finding filter limits, the default value is 1000

lTailFit

If non-zero a tail-fit will be tried, the default is disabled

lForcFit

If non-zero use the force-fit method, the default is disabled

lMinHits

Minimum hits before attempting a tail-fit in 1000's, the default is 50

lFndEftv

Flag to indicate that an effective jitter calculation is to be attempted

lMinEftv, lMaxEftv

Defines the range of the bathtub curve which is to be used to calculate an effective jitter value. The defaults for lMaxEftv and lMinEftv are -4 and -12 respectively. The valid range is -1 to -16 and lMinEftv must be less then lMaxEftv.

dMinSpan

Minimum span between clock and data edges in seconds, can be used to match trigger delay to correlate with oscilloscopes.

lGood

Flag indicates valid output data in structure

lRiseCnt

Number of hits in rising edge data

lFallCnt

Number of hits in falling edge data

dDataMin

Minimum value relative to clock edge

dDataMax

Maximum value relative to clock edge

dDataSig

1-Sigma of all values relative to clock

dAvgSkew

Average of all values relative to clock

dUnitInt

Measured Unit Interval, this is based on the clock

lUnitOff, dRiseMin, dRiseMax, dFallMin, dFallMax, lSpanCnt

These values are all used internally, DO NOT ALTER!

tRise

Histogram of rising edge data

tFall

Histogram of falling edge data

tRiseProb

Probability Histogram of rising edge data

tFallProb

Probability Histogram of falling edge data

tBath

Bathtub curves determined from PDF

tEftv

Effective Bathtub curves if **lFndEftv** is set and a valid fit is obtained

tTfit

Structure containing tail-fit info, see end of chapter for details

3.1.8 Structure used for Time Series window

```
typedef struct
  {
  /* Input parameters  */
  PARM    tParm;
  long    lNumb;
  double  dSpan;
  long    lAutoFix;
  /* Output parameters */
  long    lGood;
  double  dYstd;
  double  dAvar;
  PLOT    tMean;
  PLOT    tMini;
  PLOT    tMaxi;
  PLOT    tTime;
  PLOT    tSdev;
  PLOT    tPeak;
  } TSER;
```

tParm
Contains acquisition parameters, see end of chapter for details

lNumb
WavGetTser() can be called repeatedly with the same TSER structure. Data is then accumulated in the plot structures. This parameter tracks acquisitions so far, and may be set to 0 to reset. When set to 0 the plot structures are flushed. This parameter is automatically incremented by the WavGetTser() function.

dSpan
Time delay between measurements

lAutoFix
If true perform a pulsefind as required

lGood
Flag indicates valid output data in structure

dYstd
1-Sigma value calculated on all data

dAvar
Allan variance calculation

tMean
Contains the average plot array

tMini
Contains the minimum plot array

tMaxi
Contains the maximum plot array

tTime
Contains the time at which measurements were taken

tSdev
Contains the 1-Sigma plot array

tPeak
Contains the (max - min) plot array

3.1.9 Structure used for Statistics window

```
typedef struct
  {
  /* Input parameters  */
  PARM    tParm;
  long    lPfnd;
  long    lAutoFix;
  /* Output parameters */
  long    lGood;
  double  dMean;
  double  dMaxi;
  double  dMini;
  double  dSdev;
  double  dDuty;
  double  dFreq;
  double  dVmin[ POSS_CHNS ];
  double  dVmax[ POSS_CHNS ];
  } STAT;
```

tParm
Contains acquisition parameters, see end of chapter for details

tPfnd
If non-zero a pulse-find is performed before each measure, the default is to not perform a pulse-find

lAutoFix
If true perform a pulsefind as required

lGood
Flag indicates valid output data in structure

dMean
Contains the returned average value

dMaxi
Contains the returned maximum value

dMini
Contains the returned minimum value

dSdev
Contains the returned 1-Sigma value

dDuty
Contains the returned duty cycle, this is not measured if a TPD measurement is being performed

dFreq
Contains the frequency of the signal being measured

dVmin
Min voltage returned from last pulse-find

dVmax
Max voltage returned from last pulse-find

3.1.10 Structure used for Random Data window

```
typedef struct
  {
  /* Input parameters  */
  long    lCoun;
  long    lPcnt;
  DCOM    tDcom;
  /* Output parameters */
  long    lGood;
  double  dDjit;
  double  dRjit;
  double  dTjit;
  PLOT    tSigmTail;
  } RAND;
```

lCoun

Count of tail-fits to be performed, use one of the following:

RAND_AUTO	Continue to perform tailfits until RJ is within some percentage of the previous pass, see **lPcnt** below
RAND_FIT3	Perform 3 tailfits
RAND_FIT5	Perform 5 tailfits
RAND_FIT9	Perform 9 tailfits
RAND_FIT17	Perform 17 tailfits

lPcnt

Auto-mode succeed percentage, if selected

RAND_PCNT5	RJ within 5% of previous pass
RAND_PCNT10	RJ within 10% of previous pass
RAND_PCNT25	RJ within 25% of previous pass
RAND_PCNT50	RJ within 50% of previous pass

tDcom

Random data window uses a DCOM structure to hold most of the input and output parameters, see the dataCOM section for detailed information

lGood

Flag indicates valid output data in structure

dDjit, dRjit, dTjit

Deterministic, random, and total jitter values

tSigmTail

1-Sigma plot based on tail-fit results

3.2 UTILITY STRUCTURES

The following utility structures are used in the standard window functions:

3.2.1 Basic structure used to return plot data

```
typedef struct
  {
  double  *dData;
  long    lNumb, lRsvd;
  double  dXmin, dXmax;
  double  dYmin, dYmax;
  double  dYavg, dYstd;
  long    lXminIndx;
  long    lXmaxIndx;
  long    lYminIndx;
  long    lYmaxIndx;
  double  dAltXmin, dAltXmax;
  } PLOT;
```

dData
Pointer to y-axis data array

lNumb
Number of valid data points

lRsvd
Used to track memory allocation

dXmin, dXmax
X-axis values for ends of data array

dYmin, dYmax
Min & Max values in Y-axis data array

dYavg, dYstd
Average & 1-Sigma values for data array

lXminIndx, lXmaxIndx
Used by histograms to indicate location of first and last valid bins

lYminIndx, lYmaxIndx
Indicates the location where the Min & Max values occur in data array

dAltXmin, dAltXmax
Alternate X-axis values, if applicable. For graphs where it makes sense an alternate X-axis unit may be calculated. Examples include time or index on a Jitter Analysis 1-sigma plot, or unit interval or time on a dataCOM bathtub plot. If no applicable alternate unit is defined these variables will both be set to zero.

3.2.2 Structure used for parameters of one side of a tail-fit

```
typedef struct
{
  double  dCoef[ 3 ];
  double  dDjit;
  double  dRjit;
  double  dChsq;
  double  dLoValu, dHiValu;
  double  dMuValu;
  double  dEftvDj, dEftvRj;
  } SIDE;
```

dCoef
  Used by WavGetTfit() to generate idealized tail-fit curves
dDjit
  Deterministic jitter, this side only
dRjit
  Random jitter, this side only
dChsq
  ChiSquare indicator, goodness of fit
dLoValu, dHiValu
  dXval range over which tail was fitted
dMuValu
  Projected dXval where mu was determined
dEftvDj, dEftvRj
  Holds the effective jitter values if calculated. To calculate the effective jitter **lFndEftv** must contain a non-zero value. Since the effective jitter is calculated by optimizing a curve-fit a result is not guaranteed. If the curve-fit fails a negative value will be returned in these variables.

3.2.3 Structure used to hold tail-fit results for histograms

```
typedef struct
  {
  long    lGood;
  SIDE    tL, tR;
  double  dDjit;
  double  dRjit;
  double  dTjit;
  } TFIT;
```

lGood
  Flag to indicate successful tail-fit
tL, tR
  Structures containing individual left & right tail-fit data
dDjit
  Deterministic jitter, from both sides
dRjit
  Random jitter, average from both sides
dTjit
  Total jitter, calculated from bathtub

3.2.4 Structure used for Acquisition Parameters

```
typedef struct
  {                             // Defaults as follows:
  long    lFuncNum;             // FUNC_PER
  long    lChanNum;             // 1
  long    lStrtCnt;             // 1
  long    lStopCnt;             // 2
  long    lSampCnt;             // 300
  double  dStrtVlt;             // 0.0
  double  dStopVlt;             // 0.0
  long    lStrtArm;             // 1
  long    lStopArm;             // 1
  long    lOscTrig;             // CHAN1
  long    lOscEdge;             // EDGE_RISE
  long    lFiltEnb;             // 0
  double  dFiltMin;             // -2.49
  double  dFiltMax;             // 2.49
  long    lAutoArm;             // ARM_STOP
  long    lArm1Edg;             // 1
  long    lArm2Edg;             // 1
  double  dArm1Vlt;             // 0.0
  double  dArm2Vlt;             // 0.0
  long    lArm2Gat;             // 0
  long    lStampTm;             // 0
  long    lFndMode;             // PFND_PEAK
  long    lFndPcnt;             // PCNT_5050
  long    lFndTrg1;             // TRIG_ARM1
  long    lFndTrg2;             // TRIG_ARM1
  long    lFndTime[ 2 ][ 6 ]; // { { 20000,  30000, 100,
                                //       20000,  30000, 100, },
                                //    { 20000,  30000, 100,
                                //       20000,  30000, 100 } }
  long    lTimeOut;             // 2
  long    lDsmChan[ 2 ];        // MIN_BANK1_CHN
                                // MIN_BANK2_CHN
  } PARM;
```

lFuncNum

Function to measure, use any of the follow:

2-Channel:	FUNC_TPD_PP	TPD +/+
	FUNC_TPD_MM	TPD -/-
	FUNC_TPD_PM	TPD +/-
	FUNC_TPD_MP	TPD -/+
1-Channel:	FUNC_TT_P	Rising edge time
	FUNC_TT_M	Falling edge time
	FUNC_PW_P	Positive pulse width
	FUNC_PW_M	Negative pulse width
	FUNC_PER	Period
	FUNC_FREQ	Frequency

lChanNum

Channel to measure, the minimum value is 1, the maximum can be determined by calling DtsMaxChan()

lStrtCnt

Channel start count, the minimum value is 1, the maximum can be determined by calling DtsMaxCnts()

lStopCnt

Channel stop count, the minimum value is 1, the maximum can be determined by calling DtsMaxCnts()

lSampCnt

Sample size, the minimum value is 1, the maximum can be determined by calling DtsMaxVals()

lStrtVlt

Start voltage sets the reference voltage used to initiate the time measurement. The valid range is +/-1.1 volts

lStopVlt

Stop voltage sets the reference voltage used to terminate the time measurement. The valid range is +/-1.1 volts

lStrtArm

Arm to use for start event, only used if **lAutoArm** is set to ARM_EXTRN, the minimum value is 1

lStopArm

Arm to use for stop event, only used if **lAutoArm** is set to ARM_EXTRN, the minimum value is 1

lOscTrig

Channel to use for oscilloscope trigger, use any of the follow: TRIG_ARM1, TRIG_ARM2 TRIG_CHN1, TRIG_CHN2

lOscEdge

Edge to use to trigger oscilloscope, use any of the following: EDGE_FALL, EDGE_RISE

lFiltEnb

Filter enable, any non-zero value enables filters

dFiltMin

Filter minimum in seconds, only used if **lFiltEnb** is non-zero valid range is +/-2.49 seconds

dFiltMax

Filter maximum in seconds, only used if **lFiltEnb** is non-zero valid range is +/-2.49 seconds

lAutoArm

Auto arm enable and mode, use any of the following:

ARM_EXTRN	Arm using one of the external arms
ARM_START	Auto-arm on next start event
ARM_STOP	Auto-arm on next stop event
ARM_FIRST	Auto-arm insuring start before stop Note: this mode is frequency limited

lArm1Edg

Arm1 edge to use, only used if **lAutoArm** is set to ARM_EXTRN may be either EDGE_FALL or EDGE_RISE

lArm2Edg

Arm2 edge to use, only used if **lAutoArm** is set to ARM_EXTRN may be either EDGE_FALL or EDGE_RISE

dArm1Vlt

Arm1 voltage, the valid range is +/-1.1 volts
only used if **lAutoArm** is set to ARM_EXTRN

dArm2Vlt

Arm2 voltage, the valid range is +/-1.1 volts
only used if **lAutoArm** is set to ARM_EXTRN

lArm2Gat

Enable Arm2 gating, any non-zero value enables gating
When gating is enabled Arm2 edge and reference voltages are associated with gating.

lStampTm

Any non-zero value enables elapsed time stamping. To perform time stamping a signal must be present on Arm2, the nature of the signal is not important - the calibration signal is fine. A successful pulse-find must have also been performed on Arm2. When time stamping is enabled an array of time data can be downloaded after a sample is acquired using the DtsGetTime() function. Each value in this array represents the time at which it's sample was taken. When time stamping is enabled the maximum value of **lSampCnt** is one half its normal value.

lFndMode

Pulse find mode, may be one of the following:

PFND_FLAT	Use flat algorithm for pulse-find calculation
PFND_PEAK	Use peak value for pulse-find calculation
PFND_STRB	Use strobing method for pulse-find calc.

lFndPcnt

Pulse find percentage, may be one of the following:

PCNT_5050	Use 50/50 level for pulse-find calculation
PCNT_1090	Use 10/90 level for pulse-find calculation
PCNT_9010	Use 90/10 level for pulse-find calculation
PCNT_USER	Do NOT perform pulse-find, manual mode When this mode is selected valid voltages must be loaded in the **lStrtVlt**, **lStopVlt**, **lArmVlt1**, and **lArmVlt2** parameters
PCNT_2080	Use 20/80 level for pulse-find calculation
PCNT_8020	Use 80/20 level for pulse-find calculation

lFndTrg1

Ch1 StrobePF trigger, only valid if **lFndMode** is PFND_STRB
May be TRIG_ARM1, TRIG_ARM2 TRIG_CHN1, or TRIG_CHN2

lFndTrg2

Ch2 StrobePF trigger, only valid if **lFndMode** is PFND_STRB
May be TRIG_ARM1, TRIG_ARM2 TRIG_CHN1, or TRIG_CHN2

lFndTime

StrobePF times, only valid if **lFndMode** is PFND_STRB
Contains data pertaining to time range over which to perform a strobing pulse-find, all values are in picoseconds. Values are contained in a two dimensional array, the first index specifies which channel the data pertains to, the second index pertains to the following data:

max_start_delay, max_stop_delay, max_step_increment
min_start_delay, min_stop_delay, min_step_increment

lTimeOut

Seconds for timeout before returning an error

lDsmChan

DSM channel select, determines which channel of the optional switch matrix is selected if available. The first digit specifies the bank, the second digit specifies the channel. Valid values are 11-18 for the first bank and 21-28 for the second bank.

3.2.5 Structure with FFT window and analysis parameters

```
typedef struct
  {                            // Defaults as follows:
  long    lWinType;            // FFT_KAI
  long    lPadMult;            // 4
  double  dCtrFreq;            // 2500
  double  dRngWdth;            // 100
  double  dAlphFct;            // 8.0
  } FFTS;
```

lWinType

Window type, use one of the following:

FFT_RCT	Rectangular window
FFT_KAI	Kaiser-Bessel window
FFT_TRI	Triangular window
FFT_HAM	Hamming window
FFT_HAN	Hanning window
FFT_BLK	Blackman window
FFT_GAU	Gaussian window

lPadMult

Power of 2 to use for padding (0 - 5)

dCtrFreq

Frequency over which to assess dYavg in plot array (Hz)

dRngWdth

Width over which to assess dYavg (Hz)

dAlphFct

Alpha factor when using Kaiser-Bessel window

3.2.6 Structure used for Jitter Generator Parameters

```
typedef struct
  {                             // Defaults as follows:
  long    lSnthEnb;             // 0
  long    lOutpEnb;             // 0
  double  dOutpFrq;             // 500 MHz
  double  dDutyCyc;             // 50
  long    lSyncTyp;             // SYNC_BIT
  long    lSyncDiv;             // 1
  double  dSyncFrq;             // 500 MHz
  double  dEftvFrq;             // 500 MHz
  long    lOutpLvl;             // LEVL_CUSTOM1
  long    lSyncLvl;             // LEVL_CUSTOM1
  double  dOutpAmp;             // 1.0
  double  dSyncAmp;             // 1.0
  double  dOutpOff;             // 0.0
  double  dSyncOff;             // 0.0
  long    lOutpTrm;             // TERM_GRND
  long    lSyncTrm;             // TERM_GRND
  long    lJitEnab;             // 0
  long    lJitMode;             // JITT_PER
  long    lJitUnit;             // UNIT_SEC
  double  dJitAmpl;             // 0
  double  dJitFreq;             // 1 MHz
  long    lJitDist;             // DIST_SIN
  } JGEN;
```

lSnthEnb
Synthesizer enabled if non-zero

lOutpEnb
Output enabled if non-zero

dOutpFrq
Main clock frequency in Hertz

dDutyCyc
Duty cycle [0.0 < dDutyCyc < 100.0]

lSyncTyp
Sync signal source, use any of the following:

SYNC_JIT	Synchronized with jitter source
SYNC_BIT	Generated by bit clock
SYNC_IND	Independent of jitter or output

lSyncDiv
Sync divider, only used if **lSyncTyp** is SYNC_BIT or SYNC_IND

dSyncFrq
Sync frequency in Hertz, only used if **lSyncTyp** is SYNC_IND

dEftvFrq
Effective Sync Frequency – this is Read Only! This is calculated by the device based on the current settings of **lSyncType**, **lSyncDiv**, and **lSyncFrq**.

lOutpLvl

Output level, the following are valid values:

LEVL_ECLGND	-0.9 to –1.7 terminated 50Ω to GND
LEVL_ECLNEG2	-0.9 to –1.7 terminated 50Ω to –2V
LEVL_ECLOPEN	-0.9 to –1.7 terminated Open Circuit
LEVL_PECLPOS3	4.2 to 3.2 terminated 50Ω to +3V
LEVL_PECLOPEN	4.2 to 3.2 terminated Open Circuit
LEVL_TTLGND	2.65 to 0.15 terminated 50Ω to GND
LEVL_TTLOPEN	2.65 to 0.15 terminated Open Circuit
LEVL_CMOS3GND	2.65 to 0.15 terminated 50Ω to GND
LEVL_CMOS3OPN	2.65 to 0.15 terminated Open Circuit
LEVL_CMOS5OPN	2.65 to 0.15 terminated Open Circuit
LEVL_CUSTOM1	User selectable set
LEVL_CUSTOM2	User selectable set
LEVL_CUSTOM3	User selectable set

lSyncLvl

Sync level, valid values are the same as those defined for **lOutpLvl** above except that LEVL_ECLGND is not valid.

dOutpAmp

Output amplitude if one of the three custom levels is selected

dSyncAmp

Sync amplitude if one of the three custom levels is selected

dOutpOff

Output offset if one of the three custom levels is selected

dSyncOff

Sync offset if one of the three custom levels is selected

lOutpTrm

Output termination if one of the three custom levels is selected, use any of the following:

TERM_GRND	Terminated is 50Ω to 0 Volts
TERM_NEG2	Terminated is 50Ω to -2 Volts
TERM_POS3	Terminated is 50Ω to +3 Volts
TERM_OPEN	Terminated to Open Circuit

lSyncTrm

Sync termination if one of the three custom levels is selected, use any of the values listed for **lOutpTrm**

lJitEnab

Jitter enabled if non-zero

lJitMode

The means by which jitter amplitude is specified, use one of the following:

JITT_PER	Specified on a single period basis
JITT_CUM	Specified as a maximum across multiple repetitions of the waveform

lJitUnit

The units by which jitter amplitude is specified, use one of the following:

UNIT_SEC	Specified in seconds
UNIT_UI	Specified in unit intervals [0.0 – 1.0]
UNIT_DEG	Specified in degrees [0.0 – 360.0]

dJitAmpl

Jitter amplitude in selected units

dJitFreq

Jitter frequency in Hertz

lJitDist

Jitter distribution, may be one of the following:

DIST_SIN	Sine waveform
DIST_SAW	Sawtooth waveform
DIST_TRI	Triangular waveform
DIST_SSC	Spread Spectrum Curve
DIST_RND	Random Distribution

3.2.7 Structure used for Arm Generator Parameters

```
typedef struct
  {                             // Defaults as follows:
  PARM    tParm;                // Same as PARM in 3.2.4
                                // except: lSampCnt = 50
                                // and lAutoArm = ARM_EXTRN
  char    bPtnBits[ 10 ];       // All zeros
  char    bMskBits[ 10 ];       // All zeros
  char    sPtnName[ 128 ];      // "sof.ptn"
  long    lInvtPtn;             // 0
  long    lCyclDly;             // 0
  long    lFineDly;             // 0
  long    lFunctSw;             // 0
  long    lSpeedSw;             // 0
  long    lProtoSw;             // 0
  long    lCommDet;             // 0
  long    lCDlyByp;             // 0
  long    lEdgeCnt;             // 0x0F
} AGEN;
```

tParm

This structure contains the DT207x settings to be used when optimizing the marker position with the **ArmFindDly()** function. This is mainly used to specify the Arm and Channel, but may also be used to override default voltage thresholds or other parameters.

bPtnBits, bMskBits

These fields are used to hold the internal representation of the pattern, the details of these fields is not important from an application standpoint. The first time **ArmSetParm()** is called the pattern is loaded into these fields from the file named in the **sPtnName** field. This internal representation is used for all subsequent operations.

sPtnName

Name of pattern file to be used, the file must exist or an error will be returned. The first time **ArmSetParm()** is called, appropriate values are loaded into the **bPtnBits** and **bMskBits** fields. If the pattern file is to be changed, both theses fields should be cleared to all zeros so that the new pattern will be loaded on the next call to **ArmSetParm()**. The default file is sof.ptn

lInvtPtn

Invert the pattern bits if non-zero, this is used to compensate for sending a polarity sensitive signal through an amplifier stage which inverts the signal. This parameter is not used if **lFunctSw** is set to Edge Count Mode.

lCyclDly

Cycle Delay Increment [0 - 39]. The value of each is increment is dependent on the protocol. For 1X or 2X Fibre Channel each increment is equal to 941ps. For 1X or 2X GigaBit Ethernet each increment is equal to 800ps.

lFineDly

Fine Delay Increment [0 - 255]. Each increment is equal to approximately 15.686ps, giving a total possible delay of 4ns.

lFunctSw

Marker Generation Function, the following are valid values:

- 0 Pattern Match Mode
- 1 Edge Count Mode

lSpeedSw

Speed Switch, the following are valid values:

- 0 1X Fibre Channel or GigaBit Ethernet
- 1 2X Fibre Channel or GigaBit Ethernet

This parameter is not used if **lFunctSw** is set to Edge Count Mode.

lProtoSw

Protocol Switch, the following are valid values:

- 0 Fibre Channel
- 1 GigaBit Ethernet

This parameter is not used if **lFunctSw** is set to Edge Count Mode.

lCommDet

Enable comma detect in the AG-100's front end SERDES if non-zero.

lCDlyByp

Bypass cycle based delay circuitry if non-zero.

lEdgeCnt

Edge count to be used if **lFunctSw** is set to Edge Count Mode. It should be entered as either the count of positive edges or the count of negative edges (they must be the same), but not the sum of both.

CHAPTER 4 – LOW LEVEL FUNCTIONS

The Wavecrest Production API provides a number of low level functions to allow programmers to quickly integrate DTS207x functionality into their applications. Aside from the initialization and termination functions, these functions are not necessary if the high level window function calls detailed in Chapter 2 are used. However, these functions are provided in order to simplify many of the details involved in a programmer developing their own algorithms.

4.1 INITIALIZATION AND TERMINATION FUNCTIONS

These functions are provided to perform initialization tasks and cleanup prior to termination.

4.1.1 Initialize Device

```
long  DtsInitDev ( char *sDevName, long lBrdNumb,
                                   long lBrdAddr );
```

Input:

sDevName	Pointer to device name if UNIX platform
lBrdNumb	GPIB board number
lBrdAddr	GPIB board address

Return:

0 on Success or Error Code on Failure

Example:

DtsInitDev (“dev5”, 0, 5);

This function must be called once at the beginning of your application in order to pass information concerning the GPIB configuration. The first parameter is used to specify the GPIB device name on UNIX platforms and is ignored on Microsoft Windows. The second parameter is the board number, and the final parameter is the device number.

A successful call to DtsInitDev() must be accomplished before any other calls to the Wavecrest Production API.

Typical examples of sDevName parameter on Sun Platforms:

sDevName	Description
dev5	Device at bus address 5

Typical examples of sDevName parameter on HP-UX Platforms:

sDevName	**Description**
hpib,5	Device at bus address 5, and symbolic name hpib.
7,5	Device at bus address 5, and connected to an interface card at logical unit 7.
lan[128.10.0.3]:hpib,5	Connect to a LAN server at IP address 128.10.0.3 which contains an hpib interface with device at bus address 5.
lan[hpibsrv.wave.com]:7,5	Connect to a LAN server named hpibsrv.wave.com which contains an interface card at logical unit 7 with primary device at bus address 5.

4.1.2 Cleanup Prior to Application Termination

```
long DtsExitDev ( void );
```

Input:

None

Return:

0 on Success or Error Code on Failure

Example:

DtsExitDev ( );

Before terminating the application, the supplied cleanup function should be called. DtsExitDev() closes the GPIB device driver. After this cleanup has been performed the application may terminate normally.

4.2 INFORMATION FUNCTIONS

These functions provide various information services.

4.2.1 Get API Version

```
long DtsGetVers ( void );
```

Input:
    None
Return:
    Major version in high byte, minor version in low byte
Example:
    VerNum = DtsGetVers ( );

This function may be called to determine the current API version.

4.2.2 Get Maximum Channel Number

```
long DtsMaxChan ( void );
```

Input:
    None
Return:
    Maximum channel number supported on this device
Example:
    MaxChan = DtsMaxChan ( );

This function may be called to determine the maximum channel number on this device. The first channel is always number 1, and current devices only have 2 channels. This function is intended to support future expansion when devices with more than two channels become available.

4.2.3 Get Maximum Start/Stop Count Values

```
long DtsMaxCnts ( void );
```

Input:
    None
Return:
    Maximum number of start/stop count values obtained in a single measurement
Example:
    DtsMaxCnts ( );

This function may be called to determine the maximum number of start/stop counts that can be configured. This function is intended to support future expansion when additional counter values may be allowed.

4.2.4 Get Maximum Sample Values

```
long DtsMaxVals ( void );
```

Input:
    None
Return:
    Max. number of sample values obtained in a single measurement
Example:
    DtsMaxVals ( );

This function may be called to determine the maximum number of samples that can be taken with a single acquisition. This function is intended to support future expansion when additional samples may be taken in a single measurement.

4.3 UTILITY FUNCTIONS

These functions provide various utility services.

4.3.1 Enable or Disable Front Panel Display

```
long DtsSetDisp ( long lDisp );
```

Input:

lDisp Non-zero value to enable, zero to disable

Return:

0 on Success or Error Code on Failure

Example:

DtsSetDisp (1);

This function may be called to turn the front panel display on or off. Performance is improved if the front panel display is disabled.

4.3.2 Download Acquisition Parameters to Device

```
long DtsSetParm ( PARM *tParm );
```

Input:

tParm Pointer to Parameter Structure

Return:

0 on Success or Error Code on Failure

Example:

DtsSetParm (&tParm);

This function may be called to download the complete set of acquisition parameters to the device. Note that the Parameter Structure contains all the information necessary to completely define a basic measurement. After successfully issuing this command an acquisition may be performed using the DtsRqstAcq() or DtsGetData() command.

In order to optimize performance, this function keeps track of parameters that have been configured and only downloads parameters that have changed since the last time it was called. However, parameters which are manually sent using the DtsTalkDev() function will not be tracked, and could therefore cause unpredictable results. If this function is used to configure parameters, it should be used exclusively, and no parameters should be manually sent.

4.3.3 Perform a Pulse-find Operation

```
long DtsPulsFnd ( PARM *tParm );
```

Input:

tParm Pointer to Parameter Structure

Return:

0 on Success or Error Code on Failure

Example:

DtsPulsFnd (&tParm);

This function is used to perform a pulse-find operation based on the channel, arming, and pulse-find options in the Parameter Structure. On successful completion, the resulting voltages are returned in the appropriate fields of the Parameter Structure.

A higher level function WavPulsFnd() exists, which should be used in conjunction with the higher level window functions. In particular Oscilloscope and Time Digitizer windows require the extra steps taken by the higher level pulse-find function.

4.3.4 Update Voltage Information

```
long DtsGetVolt ( PARM *tParm );
```

Input:

tParm Pointer to Parameter Structure

Return:

0 on Success or Error Code on Failure

Example:

DtsGetVolt (&tParm);

This function is used to update the threshold voltage information in the Parameter Structure. On successful completion, the threshold voltages currently active in the DTS207X are returned in the appropriate fields of the Parameter Structure.

4.3.5 Device Reset

```
long DtsRsetDev ( void );
```

Input:

    None

Return:

    0 on Success or Error Code on Failure

Example:

    DtsRsetDev ( );

This function will reset the device to the power-up state. The existing machine state is lost, and all parameters are restored to their default values.

4.4 COMMUNICATION FUNCTIONS

These functions provide various communication services.

4.4.1 Send Command String to Device

```
long DtsTalkDev ( char *sCmnd );
```

Input:

sCmnd	Pointer to Command String

Return:

0 on Success or Error Code on Failure

Example:

DtsTalkDev (":ACQ:COUN 32000");

This function may be used to send individual command strings to the device. This function should be used whenever no response is expected from the device.

4.4.2 Send Command String and Receive ASCII Response

```
long DtsRqstAsc ( char *sCmnd, char *sSval, long lLeng );
```

Input:

sCmnd	Pointer to Command String
sSval	Pointer to Buffer to Hold Response String
lLeng	Length of Buffer to Hold Response String

Return:

0 on Success or Error Code on Failure
Response is placed in Response Buffer on Success

Example:

char buffer[128];
DtsRqstAsc (":ACQ:FUNC?", buffer, 128);

This function may be used to send individual command strings to the device when an ASCII response is expected.

4.4.3 Send Command String and Receive Double Precision Floating Point Number as Response

```
long DtsRqstDbl ( char *sCmnd, double *dDval );
```

Input:

sCmnd	Pointer to Command String
dDval	Pointer to double to Hold Response

Return:

0 on Success or Error Code on Failure
Response is placed in Double Precision Number on Success

Example:

```
double mean;
DtsRqstDbl ( ":MEAS:AVER?", &mean );
```

This function may be used to send individual command strings to the device when a Double Precision Floating Point number is expected as a response.

4.4.4 Send Command String and Receive Long Integer as Response

```
long DtsRqstInt ( char *sCmnd, long *lIval )
```

Input:

sCmnd	Pointer to Command String
lIval	Pointer to Long Integer to Hold Response

Return:

0 on Success or Error Code on Failure
Response is placed in Long Integer on Success

Example:

```
long switch;
DtsRqstInt ( ":CHAN:SWIT?", &switch );
```

This function may be used to send individual command strings to the device when a Long Integer is expected as a response.

4.5 ACQUISITION FUNCTIONS

These functions provide various acquisition services.

4.5.1 Request Data Acquisition

```
long DtsRqstAcq ( long lFunc, double *dMean, double *dSdev,
                  double *dMini, double *dMaxi );
```

Input:

lFunc Function Number – any of the following constants:

Constant	Description	Channels
FUNC_TPD_PP	TPD +/+	2-Chan
FUNC_TPD_MM	TPD -/-	2-Chan
FUNC_TPD_PM	TPD +/-	2-Chan
FUNC_TPD_MP	TPD -/+	2-Chan
FUNC_TT_P	Rising edge	1-Chan
FUNC_TT_M	Falling Edge	1-Chan
FUNC_PW_P	Positive pulse width	1-Chan
FUNC_PW_M	Negative pulse width	1-Chan
FUNC_PER	Period	1-Chan
FUNC_FREQ	Frequency	1-Chan

dMean Pointer to double to hold Mean or NULL
dSdev Pointer to double to hold 1-Sigma or NULL
dMini Pointer to double to hold Minimum or NULL
dMaxi Pointer to double to hold Maximum or NULL

Return:

0 on Success or Error Code on Failure

Example:

```
double mean;
DtsRqstAcq ( FUNC_PER, &mean, NULL, NULL, NULL );
```

This function may be used to request that a data acquisition be performed with statistics returned. If you do not require any of the individual statistics to be returned, you can pass NULL instead of a valid pointer.

4.5.2 Request Data Acquisition with Raw Data Returned

```
long DtsGetData ( long lFunc, double *dMean, double *dSdev,
                  double *dMini, double *dMaxi,
                  long *lNumb, void *pData, long lSize );
```

Input:

lFunc Function Number – any of the following constants:

Constant	Description	Channels
FUNC_TPD_PP	TPD +/+	2-Chan
FUNC_TPD_MM	TPD -/-	2-Chan
FUNC_TPD_PM	TPD +/-	2-Chan
FUNC_TPD_MP	TPD -/+	2-Chan
FUNC_TT_P	Rising edge	1-Chan
FUNC_TT_M	Falling Edge	1-Chan
FUNC_PW_P	Positive pulse width	1-Chan
FUNC_PW_M	Negative pulse width	1-Chan
FUNC_PER	Period	1-Chan
FUNC_FREQ	Frequency	1-Chan

dMean Pointer to double to hold Mean or NULL
dSdev Pointer to double to hold 1-Sigma or NULL
dMini Pointer to double to hold Minimum or NULL
dMaxi Pointer to double to hold Maximum or NULL
lNumb Pointer to Long Integer to hold Number of Raw Data Values
pData Pointer to Data Array to hold Raw Data Values
lSize Long Integer Indicating size of Data Type for Raw Data Values

Return:

0 on Success or Error Code on Failure

Example:

```
long numb;
double *data = malloc ( 32000 * sizeof ( double ) );
DtsGetData ( FUNC_PER, &mean, NULL, NULL, NULL,
                    &numb, data, sizeof ( double ) );
```

This function may be used to request that a data acquisition be performed with statistics and raw data values returned. If you do not require any of the individual statistics to be returned, you can pass NULL instead of a valid pointer. The application is responsible for allocating a sufficient data array to contain all of the raw data values. The size returned in “lNumb” may be different than would be expected by the sample size due to filters being enabled.

4.5.3 Perform Analysis Macro

```
long DtsGetMacr ( long lCmnd, long lFunc, long lChan,
                  long lStrt, long lStop, long lIncr,
                  long lXtra, float *fData, long lDesc );
```

Input:

lCmnd Type of Analysis Macro – one of the following:

Constant	Description
ANAL_FUNC	Function analysis macro
ANAL_JITT	Jitter analysis macro
ANAL_RANG	Range analysis macro

lFunc Function Number – any of the following constants:

Constant	Description	Channels
FUNC_TPD_PP	TPD +/+	2-Chan
FUNC_TPD_MM	TPD -/-	2-Chan
FUNC_TPD_PM	TPD +/-	2-Chan
FUNC_TPD_MP	TPD -/+	2-Chan
FUNC_TT_P	Rising edge	1-Chan
FUNC_TT_M	Falling Edge	1-Chan
FUNC_PW_P	Positive pulse width	1-Chan
FUNC_PW_M	Negative pulse width	1-Chan
FUNC_PER	Period	1-Chan
FUNC_FREQ	Frequency	1-Chan

lChan Channel to perform macro on: 1 or 2

lStrt, **lStop**, **lIncr**, **lXtra**

Parameters which are based on **sCmnd** as follows:

ANAL_FUNC	
lStrt	Beginning start count
lStop	Ending start count
lIncr	Start Count Increment
lXtra	Relationship of Stop Count to Start the following constants may be used:

Constant	Description
ANL_FNC_FIRST	Arm start first
ANL_FNC_PLUS1	Stop = Start+1
ANL_FNC_START	Stop = Start

ANAL_JITT	
lStrt	Start count for all measurements
lStop	Beginning Stop count
lIncr	Stop Count Increment
lXtra	Ending Stop count

ANAL_RANG	
lStrt	Start count for all measurements
lStop	Beginning Stop count
lIncr	Stop Count Increment
lXtra	Ending Stop count

fData	Pointer to Single Precision Data Array to hold Event Data
lDesc	Descriptor indicating values per Event defined as follows:

ANAL_FUNC	
2	Mean and Std. Deviation
4	Mean Std. Deviation, Min, & Max

ANAL_JITT	
2	Std. Deviation and Mean
3	Std. Deviation, Min, & Max

ANAL_RANG	
2	Std. Deviation and Mean
3	Std. Deviation, Min, & Max

Return:

0 on Success or Error Code on Failure

Example:

```
long ValuesPerEvent = 2;
long StartCount = 1, StopIncr = 1;
long MinStopCount = 2, Spans = 250;
long MaxStopCount = MinStopcount + Spans – 1;
float *data = malloc ( Spans * ValuesPerEvent * sizeof ( float ) );
DtsGetMacr ( ANAL_JITT, FUNC_PER, 1, StartCount,
    MinStopCount, StopCountIncr, MaxStopCount, data,
    ValuesPerEvent );
```

This function may be used to improve performance when statistics are required across a series of spans. These macros are primarily suited for the Jitter Analysis and Function Analysis windows. The results are returned in a single interleaved array of floats. The application is responsible for allocating a sufficient data array to contain the entire series of statistics.

4.5.4 Request Time Stamp Data

```
long DtsGetTime ( void *pData, long lNumb );
```

Input:

pData	Pointer to array of doubles to hold Time Values
lNumb	Number of Time Values to Read

Return:

0 on Success or Error Code on Failure

Example:

```
long numb;
double *data = malloc ( 16000 * sizeof ( double ) );
double *time = malloc ( 16000 * sizeof ( double ) );
DtsGetData ( FUNC_PER, &mean, NULL, NULL, NULL,
                        &numb, data, sizeof ( double ) );
DtsGetTime ( time, numb );
```

This function may be used to request the time stamp data after a data acquisition is performed. It is only valid when elapsed time stamping is enabled (stamp_tm field enabled in PARM structure). Note that when time stamping is enabled only half the maximum sample size is available (the DtsMaxVals() function can be used to obtain the maximum sample size). Also note that a signal must be present on Arm2 with arming enabled, and a valid pulse-find must have been previously completed. The calibration signal is suitable for this purpose.

This function returns an array of time values detailing when measurements were taken, these values are returned in seconds. By analyzing this array, the average sampling rate can be determined.

4.5.5 Request Strobing Oscilloscope Data

```
long DtsStrbWin ( long lChan, long lStar, long lStop,
                  long lIncr, double *dMean, long *lNumb,
                  double *dData );
```

Input:

lChan	Channel to be measured
lStar	Start of Strobe Window in picoseconds, valid range is 20,000 – 100,000,000
lStop	End of Strobe Window in picoseconds, valid range is 20,000 – 100,000,000
lIncr	Increment between strobed values, 10 is the minimum valid value
dMean	Pointer to double to hold average voltage
lNumb	Pointer to Long Integer to hold Number of Raw Data Values
dData	Pointer to array of doubles to hold Voltage Values

Return:

0 on Success or Error Code on Failure

Example:

```
long numb;
long values = ( 40000 – 20000 ) / 10 + 1;
double mean, *data = malloc ( values * sizeof ( double ) );
DtsStrbWin ( 1, 20000, 40000, 10, &mean, &numb, data );
```

This function may be used to request an array of voltage data from the strobing oscilloscope. The trigger source and voltage threshold must have been previously set. The application is responsible for allocating a sufficient data array to contain all of the raw data values.

4.6 CALIBRATION FUNCTIONS

These functions provide various calibration services.

4.6.1 Request External Calibration

```
long DtsExtnCal ( long lDoDC, long ( *pNext )( void ) );
```

Input:

- **lDoDC** A non-zero value causes a DC calibration to be performed first
- **pNext** Pointer to a function which is called whenever the user must be prompted to change input source, if a non-zero value is returned execution is continued, if 0 is returned execution is aborted

Return:

0 on Success or Error Code on Failure

Example:

```
char *prompt[] = {"\nConnect Ch1 to Cal1 AND Ch2 to Cal2...",
                  "\nCross cables at calibration signals..." };
long mesg;

long pNext ( void )
    {
    printf ( prompt[ mesg++ ] );
    getch ( );
    return 1;
    }

void main ( void )
    {
    mesg = 0;
    if ( DtsExtnCal ( 0, pNext ) )
        printf ( "\nAborted due to error..." );
    }
```

This function may be used to request that an external calibration be performed. Since user interaction is required during the calibration, a callback function must be passed to this function which is called allowing the application to provide prompts as required.

4.6.2 Request Internal Calibration

```
long _stdcall DtsIntnCal ( long lMult );
```

Input:

lMult Multiplier indicating the length of calibration

Return:

0 on Success or Error Code on Failure

Example:

DtsIntnCal (1);

This function may be used to request that an internal calibration be performed. A multiplier is provided which lengthens the calibration time, thereby increasing the quality of the calibration. The standard calibration time is approximately 5-1/2 minutes.

4.6.3 Request Strobe Calibration

```
long DtsStrbCal ( long ( *pNext )( void ) );
```

Input:

pNext Pointer to a function which is called whenever the user must be prompted to change input source, if a non-zero value is returned execution is continued, if 0 is returned execution is aborted

Return:

0 on Success or Error Code on Failure

Example:

```
char *prompt[] = {"\nConnect Cal1 to Ch1 AND Cal2 to Arm1..",
                  "\nMove Cal2 from Arm1 to Arm2..........",
                  "\nMove Cal1 from Ch1 to Ch2............", };
long mesg;

long pNext ( void )
    {
    printf ( prompt[ mesg++ ] );
    getch ( );
    return 1;
    }

void main ( void )
    {
    mesg = 0;
    if ( DtsStrbCal ( pNext ) )
        printf ( "\nAborted due to error..." );
    }
```

This function may be used to request that a strobe calibration be performed. Since user interaction is required during the calibration, a callback function must be passed to this function which is called allowing the application to provide prompts as required.

4.7 GENERIC GPIB COMMUNICATION FUNCTIONS

These functions provide access to generic GPIB devices. They can be used to access pattern generators, voltmeters, etc. This interface handles the low-level communication tasks. However, knowledge of the programming language specific to the target device will be required.

4.7.1 Open a Generic GPIB Device

```
long  GpibDevOpn ( char *sDevName, long lBrdNumb,
                                   long lBrdAddr );
```

Input:

sDevName	Pointer to device name if UNIX platform
lBrdNumb	GPIB board number
lBrdAddr	GPIB board address

Return:

A valid device descriptor on Success or DTS_ERROR on Failure

Example:

GpibDevOpn (“dev5”, 0, 5);

This function must be called once at the beginning of your application in order to pass information concerning the GPIB configuration. The first parameter is used to specify the GPIB device name on UNIX platforms and is ignored on Microsoft Windows. The second parameter is the board number, and the final parameter is the device number.

A successful call to GpibDevOpn() must be accomplished before any other calls to the Wavecrest Production API concerning this device.

The device descriptor that is returned must be used on all subsequent calls to access this device.

Typical examples of sDevName parameter on Sun Platforms:

sDevName	Description
dev5	Device at bus address 5

Typical examples of sDevName parameter on HP-UX Platforms:

sDevName	Description
hpib,5	Device at bus address 5, and symbolic name hpib.
7,5	Device at bus address 5, and connected to an interface card at logical unit 7.
lan[128.10.0.3]:hpib,5	Connect to a LAN server at IP address 128.10.0.3 which contains an hpib interface with device at bus address 5.
lan[hpibsrv.wave.com]:7,5	Connect to a LAN server named hpibsrv.wave.com which contains an interface card at logical unit 7 with primary device at bus address 5.

4.7.2 Read Data from a Generic GPIB Device

```
long GpibDevGet( long lDevNumb, char *sBuff, long *lNumb );
```

Input:

lDevNumb	Device descriptor returned by GpibDevOpn
sBuff	Pointer to buffer to hold response
lNumb	Pointer to Long to hold length of response. On Entry this variable should contain the number of byte to attempt to read. On return it will be updated to reflect the actual number of bytes read.

Return:

0 on Success or Non-Zero number on Failure
Response is placed in Response Buffer on Success

Example:

```
long lDevNum;
char buffer[256];
lDevNum = GpibDevOpn ( "dev6", 0, 6 );
GpibDevSnd ( lDevNum, "*IDN?" );
GpibDevGet ( lDevNum, buffer, strlen ( buffer ) );
```

This function may be called to read data back from a generic GPIB device. You normally use this command in conjunction with a GpibDevSnd() command requesting information. The GpibDevGet() command is then used to retrieve the response.

4.7.3 Send Data to a Generic GPIB Device

```
long GpibDevSnd ( long lDevNumb, char *sCmnd );
```

Input:

lDevNumb	Device descriptor returned by GpibDevOpn
sCmnd	Pointer to command string

Return:

0 on Success or Non-Zero number on Failure

Example:

long lDevNum;
lDevNum = GpibDevOpn (“dev6”, 0, 6);
GpibDevSnd (lDevNum, “*RST?”);

This function may be called to send data to a generic GPIB device. A successful call to GpibDevOpn() must have been previously performed in order to obtain a device descriptor to the device.

4.7.4 Cleanup Prior to Application Termination

```
long GpibDevCls ( long lDevNumb );
```

Input:

lDevNumb	Device descriptor returned by GpibDevOpn

Return:

0 on Success or Error Code on Failure

Example:

long lDevNum;
lDevNum = GpibDevOpn (“dev6”, 0, 6);
GpibDevCls (lDevNum);

Before terminating the application, the supplied cleanup function should be called. GpibDevCls() closes the GPIB device driver. After this cleanup has been performed the application may terminate normally.

4.8 DTS550 JITTER GENERATOR FUNCTIONS

These functions provide access to a Wavecrest DTS550 Jitter Generator.

4.8.1 Initialize Jitter Generator Device

```
long  GenInitDev ( char *sDevName, long lBrdNumb,
                                long lBrdAddr );
```

Input:

sDevName	Pointer to device name if UNIX platform
lBrdNumb	GPIB board number
lBrdAddr	GPIB board address

Return:

0 on Success or Error Code on Failure

Example:

GenInitDev (“dev5”, 0, 5);

This function must be called once at the beginning of your application in order to pass information concerning the GPIB configuration. The first parameter is used to specify the GPIB device name on UNIX platforms and is ignored on Microsoft Windows. The second parameter is the board number, and the final parameter is the device number.

A successful call to GenInitDev() must be accomplished before any other calls to a Jitter Generator using the Wavecrest Production API.

Typical examples of sDevName parameter on Sun Platforms:

sDevName	Description
dev5	Device at bus address 5

Typical examples of sDevName parameter on HP-UX Platforms:

sDevName	Description
hpib,5	Device at bus address 5, and symbolic name hpib.
7,5	Device at bus address 5, and connected to an interface card at logical unit 7.
lan[128.10.0.3]:hpib,5	Connect to a LAN server at IP address 128.10.0.3 which contains an hpib interface with device at bus address 5.
lan[hpibsrv.wave.com]:7,5	Connect to a LAN server named hpibsrv.wave.com which contains an interface card at logical unit 7 with primary device at bus address 5.

4.8.2 Cleanup Prior to Application Termination

```
long GenExitDev ( void );
```

Input:
None
Return:
0 on Success or Error Code on Failure
Example:
GenExitDev ( );

Before terminating the application, the supplied cleanup function should be called. GenExitDev() closes the GPIB device driver. After this cleanup has been performed the application may terminate normally.

4.8.3 Enable or Disable Front Panel Display

```
long GenSetDisp ( long lDisp );
```

Input:
lDisp Non-zero value to enable, zero to disable
Return:
0 on Success or Error Code on Failure
Example:
GenSetDisp (1);

This function may be called to turn the front panel display on or off.

4.8.4 Retrieve Setup Parameters from Jitter Generator

```
long GenGetParm ( JGEN *tJgen );
```

Input:
tJgen Pointer to Jitter Generator Parameter Structure
Return:
0 on Success or Error Code on Failure
Example:
GenGetParm (&tJgen);

This function may be called to retrieve the complete set of jitter generator parameters. Note that the Jitter Generator Parameter Structure contains all the information necessary to completely define an output state.

4.8.5 Download Setup Parameters to Jitter Generator

```
long GenSetParm ( JGEN *tJgen );
```

Input:

tJgen Pointer to Jitter Generator Parameter Structure

Return:

0 on Success or Error Code on Failure

Example:

GenSetParm (&tJgen);

This function may be called to download the complete set of jitter generator parameters. Note that the Jitter Generator Parameter Structure contains all the information necessary to completely define an output state.

In order to optimize performance, this function keeps track of parameters that have been configured and only downloads parameters that have changed since the last time it was called. However, parameters which are manually sent using the GenTalkDev() function will not be tracked, and could therefore cause unpredictable results. If this function is used to configure parameters, it should be used exclusively, and no parameters should be manually sent.

4.8.6 Fill a Jitter Generator Structure with Default Values

```
void GenDefParm ( JGEN *tJgen );
```

Input:

tJgen Pointer to Jitter Generator Parameter Structure

Return:

None

Example:

JGEN tJgen;
GenDefJgen (&tJgen);

This function is used to fill a Jitter Generator Parameter Structure with default values. These parameters could then be downloaded to the DTS550 by calling the GenSetParm() function. Using this function insures that all parameters contain reasonable values.

It is not necessary to clear a Parameter Structure using the standard memset() function prior to calling this function, as no dynamic memory allocation information is contained within the Parameter Structure.

4.8.7 Jitter Generator Reset

```
long GenRsetDev ( void );
```

Input:
    None
Return:
    0 on Success or Error Code on Failure
Example:
    GenRsetDev ( );

This function will reset the device to the power-up state. The existing machine state is lost, and all parameters are restored to their default values.

4.8.8 Send Command String to Device

```
long GenTalkDev ( char *sCmnd );
```

Input:

sCmnd	Pointer to Command String

Return:
    0 on Success or Error Code on Failure
Example:
    GenTalkDev (":JITT:FREQ MAX");

This function may be used to send individual command strings to the device. This function should be used whenever no response is expected from the device.

4.8.9 Send Command String and Receive ASCII Response

```
long GenRqstAsc ( char *sCmnd, char *sSval, long lLeng );
```

Input:

sCmnd	Pointer to Command String
sSval	Pointer to Buffer to Hold Response String
lLeng	Length of Buffer to Hold Response String

Return:
    0 on Success or Error Code on Failure
    Response is placed in Response Buffer on Success
Example:
    char buffer[128];
    GenRqstAsc (":JITT:FREQ?", buffer, 128);

This function may be used to send individual command strings to the device when an ASCII response is expected.

4.8.10 Send Command String and Receive Double Precision Floating Point Number as Response

```
long GenRqstDbl ( char *sCmnd, double *dDval );
```

Input:

sCmnd	Pointer to Command String
dDval	Pointer to double to Hold Response

Return:

0 on Success or Error Code on Failure
Response is placed in Double Precision Number on Success

Example:

double freq;
GenRqstDbl (":JITT:FREQ?", &freq);

This function may be used to send individual command strings to the device when a Double Precision Floating Point number is expected as a response.

4.8.11 Send Command String and Receive Long Integer as Response

```
long GenRqstInt ( char *sCmnd, long *lIval )
```

Input:

sCmnd	Pointer to Command String
lIval	Pointer to Long Integer to Hold Response

Return:

0 on Success or Error Code on Failure
Response is placed in Long Integer on Success

Example:

long preset;
GenRqstInt (":JITT:PRES?", &preset);

This function may be used to send individual command strings to the device when a Long Integer is expected as a response.

4.9 AG-100 ARM GENERATOR FUNCTIONS

These functions provide access to a Wavecrest AG-100 Arm Generator.

4.9.1 Initialize Arm Generator Device

```
long  ArmInitDev ( char *sDevName, long lBrdNumb,
                                   long lBrdAddr );
```

Input:

sDevName	Pointer to device name if UNIX platform
lBrdNumb	GPIB board number
lBrdAddr	GPIB board address

Return:

0 on Success or Error Code on Failure

Example:

ArmInitDev (“dev7”, 0, 7);

This function must be called once at the beginning of your application in order to pass information concerning the GPIB configuration. The first parameter is used to specify the GPIB device name on UNIX platforms and is ignored on Microsoft Windows. The second parameter is the board number, and the final parameter is the device number.

A successful call to ArmInitDev() must be accomplished before any other calls to a Arm Generator using the Wavecrest Production API.

Typical examples of sDevName parameter on Sun Platforms:

sDevName	Description
dev5	Device at bus address 5

Typical examples of sDevName parameter on HP-UX Platforms:

sDevName	Description
hpib,5	Device at bus address 5, and symbolic name hpib.
7,5	Device at bus address 5, and connected to an interface card at logical unit 7.
lan[128.10.0.3]:hpib,5	Connect to a LAN server at IP address 128.10.0.3 which contains an hpib interface with device at bus address 5.
lan[hpibsrv.wave.com]:7,5	Connect to a LAN server named hpibsrv.wave.com which contains an interface card at logical unit 7 with primary device at bus address 5.

4.9.2 Cleanup Prior to Application Termination

```
long ArmExitDev ( void );
```

Input:
    None
Return:
    0 on Success or Error Code on Failure
Example:
    ArmExitDev ( );

Before terminating the application, the supplied cleanup function should be called. ArmExitDev() closes the GPIB device driver. After this cleanup has been performed the application may terminate normally.

4.9.3 Download Setup Parameters to Arm Generator

```
long ArmSetParm ( AGEN *tAgen );
```

Input:
    tAgen   Pointer to Arm Generator Parameter Structure
Return:
    0 on Success or Error Code on Failure
Example:
    ArmSetParm (&tAgen);

This function may be called to download the complete set of arm generator parameters. Note that the Arm Generator Parameter Structure contains all the information necessary to completely define an output state.

In order to optimize performance, this function keeps track of parameters that have been configured and only downloads parameters that have changed since the last time it was called. However, parameters which are manually sent using the ArmTalkDev() function will not be tracked, and could therefore cause unpredictable results. If this function is used to configure parameters, it should be used exclusively, and no parameters should be manually sent.

4.9.4 Fill a Arm Generator Structure with Default Values

```
void ArmDefParm ( AGEN *tAgen );
```

Input:

tAgen Pointer to Arm Generator Parameter Structure

Return:

None

Example:

AGEN tAgen;
ArmDefAgen (&tAgen);

This function is used to fill an Arm Generator Parameter Structure with default values. These parameters could then be downloaded to the AG-100 by calling the ArmSetParm() function. Using this function insures that all parameters contain reasonable values.

It is not necessary to clear a Parameter Structure using the standard memset() function prior to calling this function, as no dynamic memory allocation information is contained within the Parameter Structure.

4.9.5 Arm Generator Reset

```
long ArmRsetDev ( void );
```

Input:

None

Return:

0 on Success or Error Code on Failure

Example:

ArmRsetDev ( );

This function will reset the device to the power-up state. The existing machine state is lost, and all parameters are restored to their default values.

4.9.6 Send Command String to Device

```
long ArmTalkDev ( char *sCmnd );
```

Input:

    sCmnd Pointer to Command String

Return:

    0 on Success or Error Code on Failure

Example:

    ArmTalkDev (":PATT C14FAC14FA");

This function may be used to send individual command strings to the device. This function should be used whenever no response is expected from the device.

4.9.7 Send Command String and Receive ASCII Response

```
long ArmRqstAsc ( char *sCmnd, char *sSval, long lLeng );
```

Input:

sCmnd	Pointer to Command String
sSval	Pointer to Buffer to Hold Response String
lLeng	Length of Buffer to Hold Response String

Return:

    0 on Success or Error Code on Failure
    Response is placed in Response Buffer on Success

Example:

    char buffer[128];
    ArmRqstAsc (":PATT?", buffer, 128);

This function may be used to send individual command strings to the device when an ASCII response is expected.

4.9.8 Find Arm Delay for Optimal Marker Placement

```
long ArmFindDly ( AGEN *tAgen );
```

Input:

tAgen Pointer to Arm Generator Parameter Structure

Return:

0 on Success or Error Code on Failure

Example:

ArmFindDly (&tAgen);

This function may be called to find the delay that provides the optimal marker placement. The settings contained in the **tParm** member of the AGEN structure are DTS207x parameters used as feedback for assessing the marker placement. When this function successfully returns, the **lCyclDly** and **lFineDly** parameters will be altered to the values that were determined to provide the greatest jitter tolerance.

This page intentionally left blank.

CHAPTER 5 – CODE SAMPLES

The following code samples are provided in order to aid in getting started using the Wavecrest Production API. These code samples are provided for instructional purposes only.

5.1 MODIFYING WINDOW STRUCTURE PARAMETERS

The following code snippet shows how parameters pertaining to a high-level window structure may be modified.

```
/* Allocate window structure */
STAT tStat;

/* Zero out the structure, and initialize to defaults */
memset ( &tStat, 0, sizeof ( STAT ) );
WavDefStat ( &tStat );

/* Change input parameters from default */
tStat.tParm.lFuncNum = FUNC_PW_P; /* Function PW+ */
tStat.tParm.lChanNum = 2;         /* Channel 2    */
tStat.tParm.lAutoArm = ARM_EXTRN; /* External Arm */
tStat.tParm.lStrtArm = 2;         /* Start Arm 2  */
tStat.tParm.lStopArm = 2;         /* Stop Arm 2   */
tStat.tParm.lSampCnt = 500;       /* Sample Size  */
tStat.tParm.lStopCnt = 11;        /* Stop Count   */
```

5.2 PERFORMING TAIL-FIT

The following code snippet shows how a tail-fit can be performed in a Histogram Window. Note that it may take many passes for the tail-fit to succeed. Therefore you may want to error if not successfully in a certain number of passes. Set the **lPass** parameter to 0 to start a new tail-fit analysis.

```
/* Allocate window structure, and initialize to defaults */
HIST tHist;
memset ( &tHist, 0, sizeof ( HIST ) );
WavDefHist ( &tHist );

/* Enable tail-fit */
tHist.lTailFit = 1;

/* Loop until tail-fit is successful */
while ( !tHist.tTfit.lGood )
  {
  if ( WavGetHist ( &tHist ) )
    goto ErrorHandler;
  }
```

5.3 DRAWING FROM A PLOT STRUCTURE

This code snippet shows how to draw from a plot structure. The example is for Microsoft Visual C++, but can be modified for other platforms.

```
void DrawPlot ( CDC *pCdc,   // Pointer to device context.
                CRect *wind, // Window to draw within
                             // in device coordinates.
                PLOT *plot,  // Source plot structure.
                double xmin, // Plot extents to use when
                double xmax, // drawing, this allows a
                double ymin, // margin to be added around
                double ymax )// plot or overlay of plots
  {                          // with differing extents.
  long i;
  double x, y;

  // First plot X point as a percent of window extents
  x =  ( plot->dXmin - xmin ) / ( xmax - xmin );

  // First plot X point in device coordinates
  x = ( double ) ( wind->right - wind->left )
    * x + ( double ) wind->left;

  // First plot Y point as a percent of window extents
  y = ( plot->dData[ 0 ] - ymin ) / ( ymax - ymin );

  // First plot Y point in device coordinates
  y = ( double ) ( wind->bottom - wind->top )
    * ( 1.0 - y ) + ( double ) wind->top;

  // Move current location to the first plot point
  pCdc->MoveTo ( ( int ) x, ( int ) y );

  for ( i = 1; i < plot->lNumb; i++ )
    {
    // Calculate what the next X point is
    x = ( ( plot->dXmax - plot->dXmin ) * ( double ) i
      / ( double ) ( plot->lNumb - 1 ) + plot->dXmin );

    // This plot X point as a percent of window extents
    x =  ( x - xmin ) / ( xmax - xmin );

    // This plot X point in device coordinates
    x = ( double ) ( wind->right - wind->left )
      * x + ( double ) wind->left;

    // This plot Y point as a percent of window extents
    y = ( plot->dData[ i ] - ymin ) / ( ymax - ymin );

    // This plot Y point in device coordinates
    y = ( double ) ( wind->bottom - wind->top )
      * ( 1.0 - y ) + ( double ) wind->top;

    // Draw line to this plot point
    pCdc->LineTo ( ( int ) x, ( int ) y );
    }
  }
```

5.4 PERFORMING A DATACOM MEASUREMENT

This code snippet shows how a dataCOM measurement can be taken. Error checking is performed at each step, and several acquisition parameters are overridden. A pulsefind is used to determine suitable voltage levels, and results are printed.

```
/* Declare required include files */
#include <stdio.h>
#include <string.h>
#include "wavapi.h"

int main(void)
  {
  /* Local variables */
  DCOM tDcom;
  int RetCode;

  /* Initialize DTS207x device */
  RetCode = DtsInitDev("hpib,5", 0, 5);
  if (RetCode)
    {
    fprintf(stderr,
      "\nDtsInitDev failed, return code = %i\n", RetCode);
    DtsExitDev();
    return -1;
    }

  /* Initialize structure to defaults */
  memset(&tDcom, 0, sizeof (DCOM));
  WavDefDcom(&tDcom);

  /* Override to use external arming */
  tDcom.tParm.lAutoArm = ARM_EXTRN;
  /* Select the pattern to use */
  strcpy(tDcom.sPtnName, "2^7-1.ptn");
  /* Do not measure the Bit Rate */
  tDcom.lGetRate = 0;
  /* Assign the Bit Rate to use */
  tDcom.dBitRate = 1.0625e9;

  /* Perform a pulsefind */
  RetCode = WavPulsFnd(&tDcom.tParm, WIND_DCOM);
  if (RetCode)
    {
    fprintf(stderr,
      "\nWavPulsFnd failed, return code = %i\n", RetCode);
    DtsExitDev();
    return -1;
    }
```

```
/* Acquire the measurement */
RetCode = WavGetDcom(&tDcom);
if (RetCode)
  {
  fprintf(stderr,
    "\nWavGetDcom failed, return code = %i\n", RetCode);
  DtsExitDev();
  return -1;
  }

/* Print the results in picoseconds */
fprintf(stderr,
  "Deterministic Jitter: %.3lfps\n", tDcom.dDdjt * 1e12);
fprintf(stderr,
  "Random Jitter: %.3lfps\n", tDcom.dRjit[0] * 1e12);
fprintf(stderr,
  "Total Jitter: %.3lfps\n", tDcom.dTjit[0] * 1e12);

/* Release the memory */
WavClrDcom(&tDcom);

/* Release the device */
RetCode = DtsExitDev();
if (RetCode)
  {
  fprintf(stderr,
    "\nDtsExitDev failed, return code = %i\n", RetCode);
  return -1;
  }

/* Indicate successful completion of the program */
return 0;
}
```

CHAPTER 6 – BUILD CONSIDERATIONS

6.1 SUPPORTED COMPILERS FOR THE WAVECREST PRODUCTION API

The Wavecrest Production API was built and is supported using the following compilers. Other compilers may be used and provide satisfactory results, although performance is not guaranteed.

Win32 (Win95, Win98, and WinNT 4.0)
Microsoft Visual C++ 5.0 and above
Microsoft C/C++ Optimizing Compiler 11.00
Microsoft Visual Basic 6.0
HP-UX 9.05
HP C/ANSI C Developer's Bundle A.B9.05.3A
HP-UX 10.2
HP C/ANSI C Developer's Bundle B.10.20.03
Sun 4.1.x (Solaris 1)
SPARCompiler C 3.0.1
Sun 2.5.1 or above (Solaris 2)
SPARCompiler C 3.0.1

6.2 BUILD REQUIREMENTS

When building an application using the Wavecrest Production API the following requirements need to be considered.

6.2.1 Win32 (Win95, Win98, and WinNT 4.0)

A static stub library and dynamic library link library (DLL) are supplied for developing under Microsoft Windows. You can link to the static stub library which relieves all the programming of the chores normally associated with linking to a DLL. The DLL libraries must be available in the current directory or somewhere in the PATH in order to execute the application.

The define **WIN32** must be supplied to enable options specific to Microsoft Windows platforms. If you are developing within the Visual C++ environment, this define is automatically supplied for you. If you are using a command line compiler, this define may be supplied as follows:

```
cl -c -DWIN32 apitest.c
```

The define **CPLUSPLUS** must be supplied if you are developing a Microsoft C++ application. This informs the compiler that the DLL was created as a C library, and does not contain the additional information that is normally contained in a C++ library. If you are developing a standard C application, supplying this define will result in an error. If you are using a command line compiler, this define may be supplied as follows:

```
cl -c -DWIN32 -DCPLUSPLUS apitest.c
```

In order to produce a DLL which is compatible for use with Visual Basic, fully packed structures are used. As long as the supplied include files are used in conjunction with the Microsoft compiler, this requirement is fulfilled by a supplied **#pragma** statement which removes all structure padding. If you are using another compiler, consult its documentation in order to enable full structure packing.

When developing under Visual Basic the two files **dtsapi.bas** and **wavapi.bas** are substituted for the normal C include files. These two files should be added as modules in your project, and contain all function call and structure declarations. The two DLL files need to be available in the current directory or somewhere in the PATH in order to execute the resulting application.

6.2.2 All UNIX Platforms

The define **WIN32** must NOT be defined when compiling under UNIX platforms. This define enables options which are not suitable under UNIX platforms.

6.2.3 HP-UX 9.05 and HP-UX 10.20

The ANSI C compiler must be used. ANSI compatibility is enabled from a command line by specifying the **-Aa** option as follows:

```
cc -c -Aa apitest.c
```

Required HPIB support is supplied by linking to the Standard Instrument Control Library. This library must already be installed per manufacturers documentation. This library can be included by adding **-lsicl** to the link command. The resulting link command including the Wavecrest API libraries takes the form:

```
cc -Aa apitest.o -ldts -lwav -lsicl -lm -o apitest
```

6.2.4 Sun 4.1.x (Solaris 1)

The ANSI C compiler must be used. ANSI compatibility is enabled from a command line by using the **acc** command as follows:

```
acc -c apitest.c
```

Required GPIB support is supplied by linking to the National Instruments GPIB Library. This library must already be installed per manufacturers documentation. This library can be included by adding **-lgpib** to the link command. The resulting link command including the Wavecrest API libraries takes the form:

```
acc apitest.o -ldts -lwav -lgpib -o apitest
```

6.2.5 Sun 2.5.1 or above (Solaris 2)

The standard ANSI C compiler must be used. The command line would appear as follows:

```
cc -c apitest.c
```

Required GPIB support is supplied by linking to the National Instruments GPIB Library. This library must already be installed per manufacturers documentation. This library can be included by adding **-lgpib** to the link command. The resulting link command including the Wavecrest API libraries takes the form:

```
cc apitest.o -ldts -lwav -lgpib -lm -o apitest
```

This page intentionally left blank.

APPENDIX A – ERROR CODES

Define	Value	Description
SUCCESS	0	Success
DTS_ERROR	-1	Error communicating with DTS
MEM_ERROR	-2	Required memory could not be allocated
CMD_ERROR	-3	Invalid parameters passed to function
VER_ERROR	-4	Invalid DTS version or DLL version
FIT_ERROR	-5	Failure applying tail-fit
LIM_ERROR	-6	Results exceed specified limits
FIO_ERROR	-7	File I/O error
ARM_ERROR	-8	No suitable arm signal detected
TRG_ERROR	-9	No suitable trigger signal detected
USR_ERROR	-10	Operation was terminated by user
UNT_ERROR	-11	Unit interval data exceeds limits
DDJ_ERROR	-12	DCD+DDJ data exceeds limits
VAR_ERROR	-13	Variance data for RJ+PJ exceeds limits
LRN_ERROR	-14	Learn Mode data exceeds limits
INT_ERROR	-15	Insufficient points for interpolation

APPENDIX B – VBASIC EXAMPLE

The following shows what the sample program in Chapter 1 might look like written as a Visual Basic subroutine:

```
Private Sub Sample_Click()
' Step #1  Allocate Required Structures
Dim tStat As STAT

' Step #2 Initialize the DTS207x
If (DtsInitDev("dev5", 0, 5) <> 0) Then
  mainDisplay.Text = "DtsInitDev failed..."
  GoTo ExitPoint:
End If

' Step #3 Initialize STAT Window Structure
'         memset() is not necessary, in VBasic
'         objects are automatically cleared
WavDefStat tStat

' Step #4 Perform Data Acquisition
If (WavGetStat(tStat) <> 0) Then
  mainDisplay.Text = "WavGetStat failed..."
  GoTo ExitPoint:
End If

' Step #5 Print Results
mainDisplay.Text = "-Wavecrest Production API-" & _
  vbCrLf & "-   Sample Application   -" & vbCrLf & _
  vbCrLf & "     Average:  " & _
  Format(tStat.dMean * 1000000000#, "0.000") & "ns" & _
  vbCrLf & "     1-Sigma:  " & _
  Format(tStat.dSdev * 1000000000000#, "0.000") & "ps" & _
  vbCrLf & "     Minimum:  " & _
  Format(tStat.dMini * 1000000000#, "0.000") & "ns" & _
  vbCrLf & "     Maximum:  " & _
  Format(tStat.dMaxi * 1000000000#, "0.000") & "ns"

' Step #6 Cleanup and Return
WavClrStat tStat

ExitPoint:
DtsExitDev
End Sub
```

WAVECREST Corporation

World Headquarters:
7626 Golden Triangle Drive
Eden Prairie, MN 55344
(952) 831-0030
FAX: (952) 831-4474
Toll Free: 1-800-733-7128
www.wavecrest.com

West Coast Office:
1735 Technology Drive, Suite 400
San Jose, CA 95110
(408) 436-9000
FAX: (408) 436-9001
1-800-821-2272

Europe Office:
Lilenthalalle 25
D-80939 Munchen
011-49-89-32225330
FAX: 011-49-89-32225333

200002-03 REV A