Note: All functions return **Error_Memory** if the user supplies pointers to non-allocated memory.
Note: ITC1600_Errors are not included – see ITC1600.doc.
Note: All structures align to 8 bytes.

A. Global Management Functions

1. Global Configuration

```
LONG  ITC_GlobalConfig(void* GlobalConfig)
```

This function changes basic default settings of the driver.
Care should be taken when using this function.

GlobalConfig:

```
typedef struct
      {
      long SoftwareFIFOSize;
      long HardwareFIFOSize_A;
      long HardwareFIFOSize_B;
      long Reserved;

      long Reserved0;
      long Reserved1;
      long Reserved2;
      long Reserved3;
      }
ITCGlobalConfig;
```

SoftwareFIFOSize: used to specify how much memory the driver will allocate for data. The size will be automatically adjusted to fit to a 4K page boundary. If SoftwareFIFOSize value less then MinDefaultSizeInBytes driver will adjust SoftwareFIFOSize to MinDefaultSizeInBytes, which is currently 1MB.

HardwareFIFOSize_A: currently not user selectable.
HardwareFIFOSize_B: currently not user selectable.
Reserved - Reserved3: reserved for future expandability.

B. Device Management Functions

1. Get Number of Devices

```
LONG ITC_Devices( unsigned long    DeviceType,
                  unsigned long*   DeviceNumber);
```

This function searches the system for all specified **DeviceType,** and returns **DeviceNumber** – the total number of Devices.

Maximum number of available device types:

```
#define     MAX_DEVICE_TYPE_NUMBER        4
```

This version of the driver supports the following device types:

```
#define     ITC16_ID                      0     //ITC-16/PCI-16
#define     ITC18_ID                      1     //ITC-18/PCI-18
#define     ITC1600_ID                    2     //ITC-1600/PCI-1600
#define     ITC00_ID                      3     //Virtual Device
```

This version of the driver supports the following maximum number for each device:

```
#define     ITC16_MAX_DEVICE_NUMBER       16
#define     ITC18_MAX_DEVICE_NUMBER       16
#define     ITC1600_MAX_DEVICE_NUMBER     16
#define     ITC00_MAX_DEVICE_NUMBER       16
#define     ITC_MAX_DEVICE_NUMBER         16    //Max(ITC16, ITC18, ITC1600)
```

Example:

```
      unsigned long ITC16NumberOfDevices;
      unsigned long ITC18NumberOfDevices;
      unsigned long ITC1600NumberOfDevices;
      unsigned long ITC00NumberOfDevices;
      long Error;
      char ErrorText[256];

      Error = ITC_Devices (ITC16_ID, &ITC16NumberOfDevices);
      if(Error != ACQ_SUCCESS)
            {
            Error = ITC_GetStatusText(    DeviceHandle,
                                          Error,
                                          ErrorText,
                                          256);
            ...Process Error...
            }

      Error = ITC_Devices (ITC18_ID, &ITC18NumberOfDevices);
      if(Error != ACQ_SUCCESS)
            ...Process Error...

      Error = ITC_Devices (ITC1600_ID, &ITC1600NumberOfDevices);
```

```
if(Error != ACQ_SUCCESS)
      ...Process Error...

Error = ITC_Devices (ITC00_ID, &ITC00NumberOfDevices);
if(Error != ACQ_SUCCESS)
      ...Process Error...
```

2. Get Device Handle

```
LONG ITC_GetDeviceHandle(     unsigned long   DeviceType,
                              unsigned long   DeviceNumber,
                              HANDLE*         DeviceHandle);
```

Returns the **DeviceHandle** of specified **DeviceNumber** and **DeviceType**
If specified **DeviceType** is out of range, returns **Error_DeviceIsNotSupported.**
If specified **DeviceNumber** is out of range, returns **Error_DeviceIsNotSupported | 1.**
If Device is not Opened, **DeviceHandle** is set to **INVALID_HANDLE_VALUE == -1** and the return value is **Error_NotOpen.**

This function is used to get back the **DeviceHandle** of the specified device.
Please note that the function **ITC_OpenDevice** returns the **DeviceHandle** as well.

Maximum number of available device types:

```
#define     MAX_DEVICE_TYPE_NUMBER        4
```

This version of the driver supports the following device types:

```
#define     ITC16_ID                      0     //ITC-16/PCI-16
#define     ITC18_ID                      1     //ITC-18/PCI-18
#define     ITC1600_ID                    2     //ITC-1600/PCI-1600
#define     ITC00_ID                      3     //Virtual Device
```

This version of the driver supports the following maximum number for each device:

```
#define     ITC16_MAX_DEVICE_NUMBER       16
#define     ITC18_MAX_DEVICE_NUMBER       16
#define     ITC1600_MAX_DEVICE_NUMBER     16
#define     ITC00_MAX_DEVICE_NUMBER       16
#define     ITC_MAX_DEVICE_NUMBER         16    //Max(ITC16, ITC18, ITC1600)
```

Example:

```
      HANDLE DeviceHandle;
      long Error;
      char ErrorText[256];

      Error = ITC_GetDeviceHandle(ITC18_ID, 0, &DeviceHandle);
      if(Error != ACQ_SUCCESS)
            {
            Error = ITC_GetStatusText(    DeviceHandle,
                                          Error,
                                          ErrorText,
                                          256);
            ...Process Error...
            }
```

3. Get Device Type and Number by Handle (subset)

```
LONG ITC_GetDeviceType( HANDLE          DeviceHandle,
                        unsigned long*  DeviceType,
                        unsigned long*  DeviceNumber);
```

This is an additional function (provided in the subset of **GetDeviceInfo**). Returns **DeviceType** and **DeviceNumber** for Handle.

Note: **DeviceType** and/or **DeviceNumber** may point to NULL in order to discard this information.

If **DeviceHandle** is not valid, this function sets **DeviceType** and **DeviceNumber** to **INVALID_HANDLE_VALUE == -1**.

Example:

See **ITC_GetDeviceInfo()**.

4. Open Device

```
LONG ITC_OpenDevice (   unsigned long   DeviceType,
                        unsigned long   DeviceNumber,
                        unsigned long   Mode,
                        HANDLE*         DeviceHandle);
```

This function will search for specified **DeviceType** and **DeviceNumber.** if successful a driver will be opened, necessary memory will be allocated and all internal structures are initialized. **ITC_OpenDevice()** does not change the state of the hardware. Also note that same device cannot be opened multiple times.

If multiple application programs need communicate with the same device then each application must open and close the device to allow each program to communicate with the hardware.

For each device, **DeviceNumber** starts from 0.

Mode must be either:

NORMAL_MODE: Used to emulate functionality of Instrutech’s previous driver libraries. All data are multiplexed in the single FIFO.

SMART_MODE: All channels have their own FIFO for data

DeviceHandle: if this function is successful, DeviceHandle will return the handle to this device. **DeviceHandle** is used by all driver functions.

Errors returned:
If specified **DeviceType** is out of range returns **Error_DeviceIsNotSupported**
If specified **DeviceNumber** is out of range returns **Error_DeviceIsNotSupported | 1**
If device is already opened – **Error_AreadyOpen**
If computer is out of memory – **Error_MemoryAllocation**

If device is busy – **Error_DeviceIsBusy**
NOTE: If this error occurs, the user application should retry opening the device

User can change default setting for memory size used in "smart" mode for all devices and FIFO size for ITC by using the function **ITC_GlobalConfig.**

After **ITC_OpenDevice()**, the user can call **ITC_GetDeviceInfo()** function to find the current state of the opened device.

Maximum number of available device types:

```
#define     MAX_DEVICE_TYPE_NUMBER        4
```

This version of the driver supports the following device types:

```
#define     ITC16_ID                      0     //ITC-16/PCI-16
#define     ITC18_ID                      1     //ITC-18/PCI-18
#define     ITC1600_ID                    2     //ITC-1600/PCI-1600
#define     ITC00_ID                      3     //Virtual Device
```

This version of the driver supports the following maximum number for each device:

```
#define     ITC16_MAX_DEVICE_NUMBER       16
#define     ITC18_MAX_DEVICE_NUMBER       16
#define     ITC1600_MAX_DEVICE_NUMBER     16
#define     ITC00_MAX_DEVICE_NUMBER       16
#define     ITC_MAX_DEVICE_NUMBER         16    //Max(ITC16, ITC18, ITC1600)
```

Example:

```
      HANDLE DeviceHandle;
      long Error;
      char ErrorText[256];
      long DeviceType = ITC1600;
      long DeviceNumber = 0;

      Error = ITC_OpenDevice( DeviceType, DeviceNumber, SMART_MODE,
                              &DeviceHandle);
      if(Error != ACQ_SUCCESS)
            {
            Error = ITC_GetStatusText(    DeviceHandle,
                                          Error,
                                          ErrorText,
                                          256);
            ...Process Error...
            }
```

5. Close Device

```
LONG ITC_CloseDevice (HANDLE   DeviceHandle);
```

This function will stop any data transfer and release all resources allocated for hardware, but will not change the mode of operation.

In current form this function will free all memory reserved for an application via this driver.

Example:

```
      HANDLE DeviceHandle;
      long Error;
      char ErrorText[256];
      Error = ITC_CloseDevice(DeviceHandle);
      if(Error != ACQ_SUCCESS)
            {
            Error = ITC_GetStatusText(    DeviceHandle,
                                          Error,
                                          ErrorText,
                                          256);
            ...Process Error...
            }
```

6. Initialize Hardware

```
LONG ITC_InitDevice (   HANDLE   DeviceHandle,
                        void*    sHWFunction);
```

Load LCA's and/or DSP codes to Hardware.
Initialize hardware to the Default State.

Notes: If **(sHWFunction == NULL)**, the driver loads the standard default configuration.
This function will get hardware Serial Numbers (if available) and store them in the structure **DeviceInfo**.
If the state of the device needs to be preserved, do not call this function. Use ITC_GetDeviceInfo() function to get current device state.

sHWFunction:

```
typedef struct
{
long Mode;
void* U2F_File;
unsigned long SizeOfSpecialFunction;      - size of ITC18_Special_HWFunction or
                                          ITC1600_Special_HWFunction structures
void* SpecialFunction;

unsigned long Reserved;
unsigned long id;
}
HWFunction;
```

Mode:

ITC18 Modes:

```
#define     ITC18_STANDARD          0
#define     ITC18_DYNAMICCLAMP      1
#define     ITC18_PHASESHIFT        2
```

ITC1600 Modes:

```
#define     ITC1600_INTERNAL_CLOCK              0x0
#define     ITC1600_INTRABOX_CLOCK              0x1
#define     ITC1600_EXTERNAL_CLOCK              0x2
#define     ITC1600_CLOCKMODE_MASK              0x3
#define     ITC1600_PCI1600_RACK                0x8

#define     ITC1600_RACK_RELOAD                 0x10
```

if “-1” only initialize driver defaults (does not load DSP and FPGAs)

U2F_File:

Specified full path name.
If NULL, the driver will search for default U2F file (ITC18.U2F or ITC1600.U2F) in the following order for Windows OS:

- path specified in the system registry
- windows system subdirectory
- application working directory

for classic MacOS, default U2F file (ITC18.U2Z or ITC1600.U2Z) is located in “System Folder/Extensions” folder
For MacOSX, default U2F file (ITC18.U2Z or ITC1600.U2Z) is

hidden inside the kernel driver (ITC_Driver.kext)

SizeOfSpecialFunction:
defined size of **SpecialFunction**

SpecialFunction:
Individual for each device type
For ITC16:
not supported

For ITC18:

```
typedef struct
{
unsigned long Function;             // HWFunction
void* InterfaceData;                // LCA for Interface side
void* IsolatedData;                 // LCA for Isolated side
unsigned long Reserved;
}
ITC18_Special_HWFunction;
```

Function:

```
#define     ITC18_STANDARD_FUNCTION             0
#define     ITC18_PHASESHIFT_FUNCTION           1
#define     ITC18_DYNAMICCLAMP_FUNCTION         2
#define     ITC18_SPECIAL_FUNCTION              3
```

If **Function** is set to ITC18_SPECIAL_FUNCTION, then the pointers **InterfaceData** and **IsolatedData** point to LCA data. If NULL then default hardware configuration is loaded.

For ITC1600:

```
typedef struct
{
unsigned long Function;                     // HWFunction
unsigned long DSPType;                      // DSP code ID
unsigned long HOSTType;                     // LCA ID for Host Card
unsigned long RACKType;                     // LCA ID for Rack Unit
}
ITC1600_Special_HWFunction;
```

Function:

```
#define     ITC1600_STANDARD_FUNCTION            0
#define     ITC1600_TEST_FUNCTION                0x10
```

DSPType:

```
#define     ITC1600_STANDARD_DSP                 0
#define     ITC1600_TEST_DSP                     4
```

HOSTType:

```
#define     ITC1600_STANDARD_HOST                0
```

RACKType:

```
#define     ITC1600_STANDARD_RACK                0
```

id:

Identify type of ITC1600 rack unit.
Currently set to 0.

Example:

```
      HANDLE DeviceHandle;
      long Error;
      char ErrorText[256];

      Error = ITC_InitDevice(DeviceHandle, NULL);
      if(Error != ACQ_SUCCESS)
            {
            Error = ITC_GetStatusText(    DeviceHandle,
                                          Error,
                                          ErrorText,
                                          256);
            ...Process Error...
            }
```

B. Static Information Functions

1. Get Device Information

```
LONG ITC_GetDeviceInfo(HANDLE DeviceHandle, void*   sDeviceInfo);
```

This function returns current hardware settings as defined below.

sDeviceInfo:

```
typedef struct
{
unsigned long DeviceType;
unsigned long DeviceNumber;
unsigned long PrimaryFIFOSize;                  // In Points
unsigned long SecondaryFIFOSize;                // In Points

unsigned long LoadedFunction;
unsigned long SoftKey;
unsigned long Mode;
unsigned long MasterSerialNumber;

unsigned long SecondarySerialNumber;
unsigned long HostSerialNumber;
unsigned long NumberOfDACs;
unsigned long NumberOfADCs;

unsigned long NumberOfDOs;
unsigned long NumberOfDIs;
unsigned long NumberOfAUXOs;
unsigned long NumberOfAUXIs;

unsigned long Reserved;
unsigned long Reserved;
unsigned long Reserved;
unsigned long Reserved;
}
GlobalDeviceInfo;
```

DeviceType: one of the following devices:

```
#define     ITC16_ID                     0     //ITC-16/PCI-16
#define     ITC18_ID                     1     //ITC-18/PCI-18
#define     ITC1600_ID                   2     //ITC-1600/PCI-1600
#define     ITC00_ID                     3     //Virtual Device
```

DeviceNumber: each initialized device type is assigned an ascending number starting from 0.

PrimaryFIFOSize:
Size of "Hardware FIFO" (for ITC16/18) or "Software FIFO" (for ITC1600) in points; 2 bytes for each point.

SecondaryFIFOSize:
Size of "Software FIFO" in points; 2 bytes for each point

This FIFO exists only in "SMART" mode and is used for individual channel FIFO.

LoadedFunction:
Current hardware configuration

Following functions are currently available:

```
#define     ITC18_STANDARD_FUNCTION             0x8000
#define     ITC18_PHASESHIFT_FUNCTION           0x8001
#define     ITC18_DYNAMICCLAMP_FUNCTION         0x8002
#define     ITC18_SPECIAL_FUNCTION              0x8003

#define     ITC16_STANDARD_FUNCTION             0x8000
#define     ITC1600_STANDARD_FUNCTION           0x8000
```

SoftKey:
Soft key is a special signature for each application using the driver.
This signature is valid if the Device is initialized.
ITC_InitDevice() will reset the SoftKey to ZERO.

Mode:
Driver Mode (Currently, SMART_MODE or NORMAL_MODE)

MasterSerialNumber:
ITC18/1600 Rack Serial Number. In case of ITC1600 (multiple rack configuration), this serial number is for Rack 0.

SecondarySerialNumber:
In case of ITC1600 (multiple rack configuration), this serial number is for Rack1. This number is not available for ITC16/18.

HostSerialNumber:
Serial number of PCI host card. Please note, that older production cards may not have programmed serial numbers. If serial number is required, please contact Instrutech Corp.

NumberOfDACs:
Number of available DA Outputs
NumberOfADCs:
Number of available AD Inputs
NumberOfDOs:
Number of available Digital Outputs (in number of ports)
NumberOfDIs:
Number of available Digital Inputs (in number of ports)
NumberOfAUXOs:
Number of available auxiliary outputs (see hardware manuals for description)
NumberOfAUXIs:
Number of available auxiliary inputs (see hardware manuals for description)

Example:

```
      HANDLE DeviceInfo;
      long Error;
      char ErrorText[256];
      GlobalDeviceInfo MyInfo;

      Error = ITC_GetDeviceInfo(DeviceHandle, &MyInfo);
```

```
if(Error != ACQ_SUCCESS)
      {
      Error = ITC_GetStatusText(    DeviceHandle,
                                    Error,
                                    ErrorText,
                                    256);
      ...Process Error...
      }
```

2. Get Versions

```
LONG ITC_GetVersions(   HANDLE   DeviceHandle,
                        void*    ThisDriverVersion,
                        void*    KernelLevelDriverVersion,
                        void*    HardwareVersion);
```

This function return embedded driver versions.

Version Format:

```
typedef struct
      {
      unsigned long Major;
      unsigned long Minor;
      char description[80];
      char date[80];
      }
VersionInfo;
```

Pointers can be to NULL for unneeded arguments.

ThisDriverVersion:
Current Driver version
KernelLevelDriverVersion:
Current low level system driver
HardwareVersion:
for ITC1600 is equivalent to the DSP version.

Example:

```
      HANDLE DeviceHandle;
      VersionInfo dV, kV, hV;
      char ErrorText[256];

      Error = ITC_GetVersions(DeviceHandle, &dV, &kV, &hV);
      if(Error != ACQ_SUCCESS)
            {
            Error = ITC_GetStatusText(    DeviceHandle,
                                          Error,
                                          ErrorText,
                                          256);
            ...Process Error...
            }
```

3. Read Serial Numbers

```
LONG ITC_GetSerialNumbers(    HANDLE      DeviceHandle,
                              void*       HostSerialNumber,
                              void*       MasterSerialNumber,
                              void*       SecondarySerialNumber);
```

This function reads serial numbers embedded in the hardware.
Please note:
ITC16 does not support MasterBoxSerialNumber serial numbers. PCI16 does.
ITC18 reads MasterBoxSerialNumber and PCI18 serial number.
Reading MasterSerialNumber of ITC18 will cause function unloading (reset)
All devices must be initialized before serial numbers can be read.

Serial Number is a 32 bit word.
Any Serial Number arguments may point to NULL in order to discard this information.

Example:

```
      HANDLE DeviceHandle;
      unsigned long hsn, msn, ssn;
      long Error;
      char ErrorText[256];

      Error = ITC_GetSerialNumbers(DeviceHandle, &hsn, &msn, &ssn);
      if(Error != ACQ_SUCCESS)
            {
            Error = ITC_GetStatusText(    DeviceHandle,
                                          Error,
                                          ErrorText,
                                          256);
            ...Process Error...
            }
```

4. Get Status Text

```
LONG ITC_GetStatusText( HANDLE   DeviceHandle,
                        LONG     Status,
                        char*    Text,
                        unsigned long   MaxCharacters);
```

Fills the Text buffer up to MaxCharacters with description of Error (**Status**).

Example:

```
      HANDLE DeviceHandle;
      char Text[256];
      long ErrorCode, NewError;
      ...
      NewError = ITC_GetStatusText(DeviceHandle, ErrorCode, Text, 256);
      switch(NewError)
            {
            // If Error > 0 (status for ITC16/18 only)
            case Error_Open: “Device is not opened”
                  break;
            case Error_DeviceIsNotSupported: “Invalid error code”
                  break;
            // For all devices:
            case  Error_MemoryError: “Text buffer overrun”
            }
```

5. Set User Signature

```
LONG ITC_SetSoftKey(HANDLE         DeviceHandle,
                    unsigned long  SoftKey);
```

This function allows application programs to set a unique signature.
This function is only valid after a successful ITC_InitDevice() call.

Signature Format:
MSW – Software Vendor ID
LSW – Application Program ID

Predefined Vendor IDs (Software Keys **MSW**)

```
#define     PaulKey                 0x5053
#define     HekaKey                 0x4845
#define     UicKey                  0x5543
#define     InstruKey               0x4954
#define     AlexKey                 0x4142
```

Predefined Program IDs (Software Keys **LSW**)

```
#define     EcellKey                0x4142
#define     SampleKey               0x5470
#define     TestKey                 0x4444
#define     TestSuiteKey            0x5453
#define     DemoKey                 0x4445
#define     IgorKey                 0x4947
```

Example:

```
      HANDLE DeviceHandle;
      long Error;
      char ErrorText[256];

      Error = ITC_SetSoftKey(DeviceHandle, (InstruKey << 16) | DemoKey);
      if(Error != ACQ_SUCCESS)
            {
            Error = ITC_GetStatusText(    DeviceHandle,
                                          Error,
                                          ErrorText,
                                          256);
            ...Process Error...
            }
```

C. Dynamic Information Functions

1. Get Device State

```
LONG ITC_GetState(HANDLE      DeviceHandle,
                  void*       ITCStatus);
```

ITCStatus

```
typedef struct
      {
      LONG CommandStatus;
      LONG RunningMode;
      LONG Overflow;
      LONG Clipping;

      LONG State;
      LONG Reserved0;
      LONG Reserved1;
      LONG Reserved2;

      double TotalSeconds;
      double RunSeconds;
      }
ITCStatus;
```

CommandStatus (bitwise command to read data):

```
#define     READ_TOTALTIME                      0x01
#define     READ_RUNTIME                        0x02
#define     READ_ERRORS                         0x04
#define     READ_RUNNINGMODE                    0x08
#define     READ_OVERFLOW                       0x10
#define     READ_CLIPPING                       0x20
```

RunningMode codes:

```
#define     RUN_STATE                           0x10
#define     ERROR_STATE                         0x80000000
#define     DEAD_STATE                          0x00
#define     EMPTY_INPUT                         0x01
#define     EMPTY_OUTPUT                        0x02
```

Overflow codes:

```
#define     ITC_READ_OVERFLOW_H                 0x01
#define     ITC_WRITE_UNDERRUN_H                0x02
#define     ITC_READ_OVERFLOW_S                 0x10
#define     ITC_WRITE_UNDERRUN_S                0x20
```

Clipping codes:

```
Bit 3..0 – clipping bits of Rack 0
Bit 19..16 – clipping bits of Rack 1
```

State codes (used by READ_ERRORS flag):

```
#define     RACKLCAISALIVE                      0x80000000
#define     PLLERRORINDICATOR                   0x08000000
```

```
#define     RACK0MODEMASK                      0x70000000
#define     RACK0MODEMASK                      0x07000000
#define     RACK0IDERROR                       0x00020000
#define     RACK1IDERROR                       0x00010000
#define     RACK0CRCERRORMASK                  0x0000FF00
#define     RACK1CRCERRORMASK                  0x000000FF
```

Example:

```
      HANDLE DeviceHandle;
      long Error;
      ITCStatus MyStatus;
      char ErrorText[256];

      ZeroMemory(MyStatus, sizeof(MyStatus));
      MyStatus.CommandStatus = READ_ERRORS | READ_OVERFLOW

      Error = ITC_GetState(DeviceHandle, &MyStatus);
      if(Error != ACQ_SUCCESS)
            {
            Error = ITC_GetStatusText(    DeviceHandle,
                                          Error,
                                          ErrorText,
                                          256);
            ...Process Error...
            }
      //Analyze MyStatus.Overflow and NyStatus.State
```

2. Set Device State

```
LONG ITC_SetState(HANDLE   DeviceHandle,
                  void*    sParam);
```

Not yet implemented in current version.

Future versions may reset error/overflow/underrun bits, etc.

3. Get FIFO Information

```
LONG ITC_GetFIFOInformation(  HANDLE             DeviceHandle,
                              unsigned long      NumberOfChannels,
                              void               *FIFOData);
```

Get FIFO information for specified channels

NumberOfChannels:

specify how many channels to process (size of FIFOData array)

FIFOData:

```
typedef struct
{
unsigned short ChannelType;          //Channel Type
unsigned short Command;              //Command
unsigned short ChannelNumber;        //Channel Number
unsigned short Status;               //Status
unsigned long  Value;                //Number of points OR Data Value
void* DataPointer;                         //Data
}
ITCChannelDataEx;
```

ChannelType:

```
#define D2H                                 0x00        //Input
#define H2D                                 0x01        //Output
#define DIGITAL_INPUT                       0x02        //Digital Input
#define DIGITAL_OUTPUT                      0x03        //Digital Output
#define AUX_INPUT                           0x04        //Aux Input
#define AUX_OUTPUT                          0x05        //Aux Output
```

Value:

size of FIFO in points

DataPointer:

pointer to FIFO buffer

This function is a sub-function of the **ITC_GetChannels()** function.

Example:

```
      HANDLE DeviceHandle;
      long Error;
      unsigned long NumberOfChannels = 3;
      char ErrorText[256];
      ITCChannelDataEx myFIFOInfo[3];
      ZeroMemory(myFIFOInfo, sizeof(myFIFOInfo));
      myFIFOInfo[0].ChannelType = D2H;
      myFIFOInfo[0].ChannelNumber = 2;
      myFIFOInfo[1].ChannelType = D2H;
      myFIFOInfo[1].ChannelNumber = 5;
      myFIFOInfo[2].ChannelType = H2D;
      myFIFOInfo[2].ChannelNumber = 1;
      Error = ITC_GetFIFOInformation (DeviceHandle, myFIFOInfo);
      if(Error != ACQ_SUCCESS)
            {
            Error = ITC_GetStatusText(    DeviceHandle,
                                          Error,
                                          ErrorText,
                                          256);
            ...Process Error...
            }
```

4. Get TIME Information

```
LONG ITC_GetTime( HANDLE            DeviceHandle,
                  double            *Seconds);
```

```
This function returns the following in second:
for ITC16/18 – the OS counter since power up.
for ITC1600 – the DSP timer value since DSP initialization.
```

Example:

```
      HANDLE DeviceHandle;
      long Error;
      char ErrorText[256];
      double Seconds;
      Error = ITC_GetTime( DeviceHandle, &Seconds);
      if(Error != ACQ_SUCCESS)
            {
            Error = ITC_GetStatusText(    DeviceHandle,
                                          Error,
                                          ErrorText,
                                          256);
            ...Process Error...
            }
```

D. Configuration Functions

1. Configure Device

```
LONG ITC_ConfigDevice(HANDLE   DeviceHandle,
                      void*    ITCPublicConfig);
```

Set modes and parameters.

ITCPublicConfig:

```
typedef struct
{
LONG DigitalInputMode;        // Bit 0: Latch Enable, Bit 1: Invert
LONG ExternalTriggerMode;     // Bit 0: Transition, Bit 1: Invert
LONG ExternalTrigger;         // Enable

LONG EnableExternalClock;     // Enable
LONG DACShiftValue;           // For ITC18 Only. Needs special LCA

LONG InputRange;              // AD0...AD7

LONG TriggerOutPosition;
LONG OutputEnable;            // For ITC1600: Separate enable for each channel

LONG  SequenceLengthOut;      // If 0, driver will convert from SetChannels()
LONG* SequenceOut;
LONG  SequenceLengthIn;       // Only for ITC1600
LONG* SequenceIn;             // Only for ITC1600

LONG ResetFIFOFlag;           // For ITC16/18 -> Reset FIFO
LONG ControlLight;

double SamplingInterval;  //(Sec.) Note: may be calculated from channel setting
}
ITCPublicConfig;
```

NOTE: -1 represents "do not change mode."

DigitalInputMode:

```
for ITC18: Bit 0: Latch Enable, Bit 1: Invert
for ITC1600:
Bit 15: Digital Port 2 active low
Bit 14: Digital Port 2 latching mode
Bit 13: Trigger In active low
Bit 12: Trigger In latching mode
Bit 11: Digital Port 0 (bits 2,3) active low
Bit 10: Digital Port 0 (bits 2,3)latching mode
Bit  9: Digital Port 0 (bits 0,1)active low
Bit  8: Digital Port 0 (bits 0,1)latching mode
Bit  7: Digital Port 1 (bits 12..15)active low
Bit  6: Digital Port 1 (bits 12..15)latching mode
Bit  5: Digital Port 1 (bits 8..11)active low
Bit  4: Digital Port 1 (bits 8..11)latching mode
Bit  3: Digital Port 1 (bits 4..7)active low
Bit  2: Digital Port 1 (bits 4..7)latching mode
Bit  1: Digital Port 1 (bits 0..3)active low
Bit  0: Digital Port 1 (bits 0..3)latching mode
```

ExternalTriggerMode:
for ITC18: Bit 0: Transition, Bit 1: Invert

ExternalTrigger:

```
for ITC16/18: Bit 0: Enable trigger.
for ITC1600:
Bit 0:      Enable trigger.
Bit 1:      Use trigger from PCI1600
Bit 2:      Use timer trigger
Bit 3:      Use Rack 0 TrigIn BNC
Bit 4:      Use Rack 0 Digital Input 0 BNC
Bit 5:      Use Rack 0 Digital Input 1 BNC
Bit 6:      Use Rack 0 Digital Input 2 BNC
Bit 7:      Use Rack 0 Digital Input 3 BNC

Bit 8:      Use trigger from PCI1600, but with Rack Reload function, for better
            synch.
```

```
#define USE_TRIG_IN           0x01  //Enable trigger.
#define USE_TRIG_IN_HOST      0x02  //Use trigger form PCI1600
#define USE_TRIG_IN_TIMER     0x04  //Use timer trigger
#define USE_TRIG_IN_RACK      0x08  //Use Rack 0 TrigIn BNC
#define USE_TRIG_IN_FDI0      0x10  //Use Rack 0 Digital Input 0 BNC
#define USE_TRIG_IN_FDI1      0x20  //Use Rack 0 Digital Input 1 BNC
#define USE_TRIG_IN_FDI2      0x40  //Use Rack 0 Digital Input 2 BNC
#define USE_TRIG_IN_FDI3      0x80  //Use Rack 0 Digital Input 3 BNC
#define TRIG_IN_MASK          0xFF

#define USE_HARD_TRIG_IN      0x100
```

EnableExternalClock:
for ITC18: Use external sampling clock if external clock option installed

DACShiftValue:
For ITC18: when phaseshifter function is loaded.

InputRange:

```
For ITC18: set ADGain for each channel
bit 0..1 – ADC0
bit 2..3 – ADC1
bit 4..5 – ADC2
bit 6..7 – ADC3
bit 8..9 – ADC4
bit 10..11 – ADC5
bit 12..13 – ADC6
bit 14..15 – ADC7

gain selection:
00    - 1X Gain (default)
01    - 2X Gain
10    - 5X Gain
```

11   - 10X Gain

TriggerOutPosition:
For ITC18/1600: Send trigger out in the specified sequence RAM entry

OutputEnable:
Enable output channels

Next 4 parameters are to be used for backward compatibility to older Instrutech drivers.
SequenceLengthOut:
number of sequence entries
SequenceOut:
sequence according to hardware manual
SequenceLengthIn:
For ITC1600 only
SequenceIn:
For ITC1600 only

ResetFIFOFlag:
Reset FIFO

ControlLight:
for ITC18 – control ready light
for ITC1600

bit 12 – Status 0 LED	Rack 0	
bit 13 – AD 8-15 LED	Rack 0	
bit 14 – Ready LED	Rack 0	(default)
bit 28 – Status 0 LED	Rack 1	
bit 29 – AD 8-15 LED	Rack 1	(default)
bit 30 – Ready LED	Rack 1	(default)

```
#define     READY_LIGHT                  0x4000
#define     STATUS_1                     0x2000
#define     STATUS_0                     0x1000
#define     LEDMASK                      0x7000
```

Next parameter is to be used for backward compatibility to older Instrutech drivers.
SamplingInterval:
In seconds

Example:

```
      HANDLE DeviceHandle;
      long Error;
      ITC18PublicConfig Itc18Param;
      char ErrorText[256];
      ZeroMemory(&Itc18Param,sizeof(Itc18Param));

      Itc18Param. DigitalInputMode = 1;  //Latch enable
      Itc18Param.InputRange    = 0x5555;  //Gain 2X for all channels
      Itc18Param.ControlLight  = 0x1;     //Ready light on

      Error = ITC_ConfigDevice( DeviceHandle, &Itc18Param);
      if(Error != ACQ_SUCCESS)
            {
            Error = ITC_GetStatusText(    DeviceHandle,
                                          Error,
                                          ErrorText,
                                          256);
            ...Process Error...
            }
```

2. Reset Channels Selection

```
LONG ITC_ResetChannels(HANDLE   DeviceHandle)
```

This function resets all channel settings.

Note: This function does not change current selection until the software executes the **ITC_UpdateChannels()** function.

Example:

```
      HANDLE DeviceHandle;
      char ErrorText[256];
      Error = ITC_ResetChannels(DeviceHandle);
      if(Error != ACQ_SUCCESS)
            {
            Error = ITC_GetStatusText(    DeviceHandle,
                                          Error,
                                          ErrorText,
                                          256);
            ...Process Error...
            }
```

3. Configure Channels

```
LONG ITC_SetChannels(   HANDLE          DeviceHandle,
                        unsigned long   NumberOfChannels,
                        void*           sChannels);
```

This function sets individual channel settings.
NumberOfChannels:
specify number of channels to process (size of sChannels array)

sChannels:

```
typedef struct
{
unsigned long ModeNumberOfPoints;
unsigned long ChannelType;          - Type of channel (Input or Output)
unsigned long ChannelNumber;        - Channel Number (Starts from 0 for each
                                        channel type)
unsigned long Reserved0;
unsigned long ErrorMode;            - What type of error, if one occurs
unsigned long ErrorState;           - Error indicator
void* FIFOPointer;                  - FIFO for this channel
unsigned long FIFONumberOfPoints;   - if FIFOPointer is not specified:
                                        Recommended FIFO Size (0=don't care)
                                        (FIFO Size will be aligned to
                                        the nearest power of 2.)
unsigned long ModeOfOperation;      - How to process data
unsigned long SizeOfModeParameters; - Size of next structure
void* ModeParameters;               - parameters for data processing
unsigned long SamplingIntervalFlag; - See below

double SamplingRate;                - Sampling Rate in Hz or in Sec or in Ticks
double StartOffset;                 - Start offset in seconds
double Gain;                        - Gain
double Offset;                      - Offset in volts

unsigned long ExternalDecimation;   - Decimate base sampling rate on the host
                                        (“SMART_MODE” only)
unsigned long Reserved1;
unsigned long Reserved2;
unsigned long Reserved3;
}
ITCChannelInfo;
```

Note: This function does not change the current selection until the **ITC_UpdateChannels()** function is executed.

ModeNumberOfPoints:
Specify total number of points user wants to send out. “0”: no limitation.

ChannelType:

```
#define D2H                                 0x00        //Input
#define H2D                                 0x01        //Output
#define DIGITAL_INPUT                       0x02        //Digital Input
#define DIGITAL_OUTPUT                      0x03        //Digital Output
#define AUX_INPUT                           0x04        //Aux Input
#define AUX_OUTPUT                          0x05        //Aux Output
```

ChannelNumber:
Up to “Max Channel Number” (individual for each device type)

```
//ITC18 CHANNELS
#define     ITC18_NUMBEROFCHANNELS        16
#define     ITC18_NUMBEROFOUTPUTS          7
#define     ITC18_NUMBEROFINPUTS           9
#define     ITC18_NUMBEROFADCINPUTS        8
#define     ITC18_NUMBEROFDACOUTPUTS       4
#define     ITC18_NUMBEROFDIGINPUTS        1
#define     ITC18_NUMBEROFDIGOUTPUTS       2
#define     ITC18_NUMBEROFAUXINPUTS        0
#define     ITC18_NUMBEROFAUXOUTPUTS       1

#define     ITC18_DA0                                 0
#define     ITC18_DA1                                 1
#define     ITC18_DA2                                 2
#define     ITC18_DA3                                 3
#define     ITC18_DO0                                 4
#define     ITC18_DO1                                 5
#define     ITC18_AUX                                 6

#define     ITC18_AD0                                 0
#define     ITC18_AD1                                 1
#define     ITC18_AD2                                 2
#define     ITC18_AD3                                 3
#define     ITC18_AD4                                 4
#define     ITC18_AD5                                 5
#define     ITC18_AD6                                 6
#define     ITC18_AD7                                 7
#define     ITC18_DI                                  8

#define ITC18_DA_CH_MASK                  0x3               //4 DA Channels
#define ITC18_DO0_CH                      0x4               //DO0
#define ITC18_DO1_CH                      0x5               //DO1
#define ITC18_AUX_CH                      0x6               //AUX

//ITC16 CHANNELS
#define     ITC16_NUMBEROFCHANNELS        14
#define     ITC16_NUMBEROFOUTPUTS          5
#define     ITC16_NUMBEROFINPUTS           9
#define     ITC16_DO_CH                    4

#define     ITC16_NUMBEROFADCINPUTS        8
#define     ITC16_NUMBEROFDACOUTPUTS       4
#define     ITC16_NUMBEROFDIGINPUTS        1
#define     ITC16_NUMBEROFDIGOUTPUTS       1
#define     ITC16_NUMBEROFAUXINPUTS        0
#define     ITC16_NUMBEROFAUXOUTPUTS       0

#define     ITC16_DA0                                 0
#define     ITC16_DA1                                 1
#define     ITC16_DA2                                 2
#define     ITC16_DA3                                 3
#define     ITC16_DO                                  4
```

```
#define     ITC16_AD0                                 0
#define     ITC16_AD1                                 1
#define     ITC16_AD2                                 2
#define     ITC16_AD3                                 3
#define     ITC16_AD4                                 4
#define     ITC16_AD5                                 5
#define     ITC16_AD6                                 6
#define     ITC16_AD7                                 7
#define     ITC16_DI                                  8

//ITC1600 CHANNELS
#define     ITC1600_NUMBEROFCHANNELS      47
#define     ITC1600_NUMBEROFINPUTS        32
#define     ITC1600_NUMBEROFOUTPUTS       15

#define     ITC1600_NUMBEROFADCINPUTS     16
#define     ITC1600_NUMBEROFDACOUTPUTS    8
#define     ITC1600_NUMBEROFDIGINPUTS     6
#define     ITC1600_NUMBEROFDIGOUTPUTS    6
#define     ITC1600_NUMBEROFAUXINPUTS     8
#define     ITC1600_NUMBEROFAUXOUTPUTS    1
#define     ITC1600_NUMBEROFINPUTGROUPS   11
#define     ITC1600_NUMBEROFOUTPUTGROUPS  5

//DACs
#define     ITC1600_DA0                               0           //RACK0
#define     ITC1600_DA1                               1
#define     ITC1600_DA2                               2
#define     ITC1600_DA3                               3
#define     ITC1600_DA4                               4           //RACK1
#define     ITC1600_DA5                               5
#define     ITC1600_DA6                               6
#define     ITC1600_DA7                               7
//Digital outputs
#define     ITC1600_DOF0                              8           //RACK0
#define     ITC1600_DOS00                             9
#define     ITC1600_DOS01                             10
#define     ITC1600_DOF1                              11          //RACK1
#define     ITC1600_DOS10                             12
#define     ITC1600_DOS11                             13
#define     ITC1600_HOST                              14
//ADCs
#define     ITC1600_AD0                               0           //RACK0
#define     ITC1600_AD1                               1
#define     ITC1600_AD2                               2
#define     ITC1600_AD3                               3
#define     ITC1600_AD4                               4
#define     ITC1600_AD5                               5
#define     ITC1600_AD6                               6
#define     ITC1600_AD7                               7
#define     ITC1600_AD8                               8           //RACK1
#define     ITC1600_AD9                               9
#define     ITC1600_AD10                              10
#define     ITC1600_AD11                              11
#define     ITC1600_AD12                              12
#define     ITC1600_AD13                              13
#define     ITC1600_AD14                              14
```

```
#define     ITC1600_AD15                              15
//Asynchronous ADCs
#define     ITC1600_SAD0                              16          //RACK0
#define     ITC1600_SAD1                              17
#define     ITC1600_SAD2                              18
#define     ITC1600_SAD3                              19
#define     ITC1600_SAD4                              20          //RACK1
#define     ITC1600_SAD5                              21
#define     ITC1600_SAD6                              22
#define     ITC1600_SAD7                              23
//Digital Inputs
#define     ITC1600_DIF0                              26          //RACK0
#define     ITC1600_DIS00                             27
#define     ITC1600_DIS01                             28
#define     ITC1600_DIF1                              29          //RACK1
#define     ITC1600_DIS10                             30
#define     ITC1600_DIS11                             31
```

ErrorMode:

Set Error mode

```
#define     ITC_STOP_CH_ON_OVERFLOW       0x0001   // Stop One Channel
#define     ITC_STOP_CH_ON_UNDERRUN       0x0002
#define     ITC_STOP_DR_ON_OVERFLOW       0x0100   // Stop One Direction
#define     ITC_STOP_DR_ON_UNDERRUN       0x0200
#define     ITC_STOP_ALL_ON_OVERFLOW      0x1000   // Stop System(Hardware
STOP)
#define     ITC_STOP_ALL_ON_UNDERRUN      0x2000
```

ErrorState:

Read Error indicator

```
#define     ITC_STOP_CH_ON_OVERFLOW       0x0001   // Stop One Channel
#define     ITC_STOP_CH_ON_UNDERRUN       0x0002
#define     ITC_STOP_DR_ON_OVERFLOW       0x0100   // Stop One Direction
#define     ITC_STOP_DR_ON_UNDERRUN       0x0200
#define     ITC_STOP_ALL_ON_OVERFLOW      0x1000   // Stop System(Hardware
STOP)
#define     ITC_STOP_ALL_ON_UNDERRUN      0x2000
```

SamplingIntervalFlag:

Bit 0: USE_FREQUENCY / USE_TIME

```
#define    USE_FREQUENCY      0x0
#define    USE_TIME           0x1
```

Bit 1: USE_TICKS

```
#define    USE_TICKS          0x2
```

Bit 2-3:

```
#define     NO_SCALE          0x0   
#define     MS_SCALE          0x4   // Milliseconds
#define     US_SCALE          0x8   // Microseconds
#define     NS_SCALE          0xC   // Nanoseconds
#define     SCALE_MASK        0x0C
```

Bit 4:

```
#define    ADJUST_RATE        0x10  //Adjust to closest available
```

Bit 5:

```
#define    DONTIGNORE_SCAN    0x20  //Use StartOffset to set Scan Number
```

FIFOPointer:
FIFONumberOfPoints:
If **FIFOPointer** NULL, the driver will use internal driver FIF0 for this channel otherwise, it will use specified buffer.

StartOffset (in seconds):
For ITC1600: If sampling interval more than 5us data acquisition could be offset from start time. This parameter must be in multiples of 5us.

Gain:
May be 1x, 2x, 5x, or 10x (both 0 and 1 are interpreted as 1x)
ITC18 implements ADC hardware gain
Offset:
Driver will calculate an offset for ITC16/18,
DSP – for ITC1600

Note: Currently ITC16 Gain and Offset is not implemented.
ITC1600 uses internal gain/offset for each unit.

ExternalDecimation:
Decimate data.

Example:

```
      // Setup 4 Channel data acquisition
      HANDLE DeviceHandle;
      long Error;
      char ErrorText[256];
      unsigned long NumberOfChannels = 4;
      ITCChannelInfo MyChannels[4];                   // 4 Channels
      ZeroMemory(MyChannels, sizeof(MyChannels));     // Clear Memory

      MyChannels[0].ChannelNumber = 0;                // First Channel
      MyChannels[0].ChannelType = H2D;                // Output
      MyChannels[0].SamplingRate = 100000.;           // 100kHz (In Hz)

      MyChannels[1].ChannelNumber = 4;                // Fifth Channel
      MyChannels[1].ChannelType = D2H;                // Input
      MyChannels[1].SamplingRate = 100000.;           // 100kHz (In Hz)

      MyChannels[2].ChannelNumber = 2;                // Third Channel
      MyChannels[2].ChannelType = D2H;                // Input
      MyChannels[2].SamplingRate = 100000.;           // 100kHz (In Hz)

      MyChannels[3].ChannelNumber = 1;                // Second Channel
      MyChannels[3].ChannelType = H2D;                // Output
      MyChannels[3].SamplingRate = 100000.;           // 100kHz (In Hz)

      Error = ITC_ResetChannels(DeviceHandle);
      if(Error != ACQ_SUCCESS)
            {
            Error = ITC_GetStatusText(    DeviceHandle,
                                          Error,
                                          ErrorText,
                                          256);
            ...Process Error...
            }

      Error = ITC_SetChannels(DeviceHandle,
                              NumberOfChannels,
                              MyChannels);
      if(Error != ACQ_SUCCESS)
            {
            Error = ITC_GetStatusText(    DeviceHandle,
                                          Error,
                                          ErrorText,
                                          256);
            ...Process Error...
            }


      Error = ITC_UpdateChannels(DeviceHandle)
      if(Error != ACQ_SUCCESS)
            {
            Error = ITC_GetStatusText(    DeviceHandle,
                                          Error,
                                          ErrorText,
                                          256);
            ...Process Error...
            }

      Error = ITC_GetChannels(DeviceHandle,
                              NumberOfChannels,
                              MyChannels);

      if(Error != ACQ_SUCCESS)
            {
            Error = ITC_GetStatusText(    DeviceHandle,
                                          Error,
                                          ErrorText,
                                          256);
            ...Process Error...
            }
```

4. Update Configure Channel Information

```
LONG ITC_UpdateChannels(HANDLE   DeviceHandle)
```

This function downloads all the channel information to the hardware, after performing all necessary optimization on the channel data.
After calling this function legitimate channel, information becomes available with the **ITC_GetChannels()** function.

Example:

See **ITC_SetChannels**()

5. Get Configure Channel Information

```
LONG ITC_GetChannels(HANDLE   DeviceHandle,
                     LONG     NumberOfChannels,
                     void*    sChannels);
```

NumberOfChannels:
specify number of channels to process (size of sChannels array)

See **ITCChannelInfo** Structure in **ITC_SetChannels()**.

NOTE: Software must call **ITC_UpdateChannels()** before calling **ITC_GetChannels()** to get the true values.

Example:
      See **ITC_SetChannels**()

E. Asynchronous Data Functions

1. Single Scan

```
LONG ITC_SingleScan(    HANDLE             DeviceHandle,
                        unsigned long      NumberOfChannels,
                        void*              scChannels);
```

This function performs limited acquisition with highest possible speed.

NumberOfChannels:
specify number of channels to process (size of scChannels array)

scChannels:

```
typedef struct
{
unsigned long ChannelType;
unsigned long ChannelNumber;
unsigned long Reserved0,
unsigned long Reserved1,
unsigned long Reserved2,
unsigned long NumberOfPoints,
unsigned long DecimateMode;
void* Data;
} ITCLimited;
```

ChannelType:

```
#define D2H                                   0x00        //Input
#define H2D                                   0x01        //Output
#define DIGITAL_INPUT                         0x02        //Digital Input
#define DIGITAL_OUTPUT                        0x03        //Digital Output
#define AUX_INPUT                             0x04        //Aux Input
#define AUX_OUTPUT                            0x05        //Aux Output
```

ChannelNumber:
Up to "Max Channel Number" (individual for each device type)

NumberOfPoints:
Number of output/input points

DecimateMode:
0, 1 – No decimation
2 – decimate by 2 (every other point)
...
(ITCXX_MaximumSingleScan – 1) – one point only

```
//SINGLE SCAN Limitations
#define ITC16_MaximumSingleScan                     16*1024
#define ITC18_MaximumSingleScan                     256*1024
#define ITC1600_MaximumSingleScan                   1024

#define     ITC16_MAX_SEQUENCE_LENGTH               16
#define     ITC18_MAX_SEQUENCE_LENGTH               16
#define     ITC1600_MAX_SEQUENCE_LENGTH             16
```

```
#define     ITC1600_NOP_CHANNEL_RACK0          0x80000000
#define     ITC1600_NOP_CHANNEL_RACK1          0x80000001
```

Example:

```
      HANDLE DeviceHandle;
      long Error;
      char ErrorText[256];
      ITCLimited ItcData[2];
      short InputData[128];
      short OutputData[64];

      ZeroMemory(ItcData, sizeof(ItcData));
      ItcData[0].ChannelType = DIGITAL_INPUT;
      ItcData[0].ChannelNumber = 0;
      ItcData[0].NumberOfPoints = 128;
      ItcData[0].Data = InputData;

      ItcData[1].ChannelType = DIGITAL_OUTPUT;
      ItcData[1].ChannelNumber = 0;
      ItcData[1].NumberOfPoints = 64;
      ItcData[1].Data = OutputData;
      for(int i = 0; i < 64; i++)
            OutputData[i] = i;

      Error = ITC_SingleScan (DeviceHandle, 2, ItcData);
      if(Error != ACQ_SUCCESS)
            {
            Error = ITC_GetStatusText(    DeviceHandle,
                                          Error,
                                          ErrorText,
                                          256);
            ...Process Error...
            }
```

2. Asynchronous Input / Output

```
LONG ITC_AsyncIO( HANDLE          DeviceHandle,
                  unsigned long   NumberOfChannels,
                  void*           saChannels);
```

Sets or reads single data point on specified channels

NumberOfChannels:
Specify number of channels to process (size of saChannels array)

saChannels:

```
typedef struct
{
unsigned short ChannelType;         //Channel Type
unsigned short Command;             //Command
unsigned short ChannelNumber;       //Channel Number
unsigned short Status;              //Status
unsigned long  Value;               //Number of points OR Data Value
void* DataPointer;                  //Data
}
ITCChannelDataEx;
```

ChannelType:
0 D2H – Input
1 H2D – Output
2 DIGITAL_INPUT – Digital Input
3 DIGITAL_OUTPUT – Digital Output
4 AUX_INPUT
5 AUX_OUTPUT

ChannelNumber:
Up to “Max Channel Number” (individual for each device type)

Value:
16 bit representation of output/input value

Example:

```
      HANDLE DeviceHandle;
      long Error;
      char ErrorText[256];
      ITCChannelDataEx ItcData[2];
      ZeroMemory(ItcData, sizeof(ItcData));
      ItcData[0].ChannelType = H2D;
      ItcData[0].ChannelNumber = 2;
      ItcData[0].Value = 0x1000;
      ItcData[1].ChannelType = DIGITAL_OUTPUT;
      ItcData[1].ChannelNumber = 0;
      ItcData[1].Value = 0x5555;
      Error = ITC_AsyncIO(DeviceHandle, 2, ItcData);
      if(Error != ACQ_SUCCESS)
            {
            Error = ITC_GetStatusText(    DeviceHandle,
                                          Error,
                                          ErrorText,
                                          256);
            ...Process Error...
            }
```

F. Synchronous Data Functions

1. Start Acquisition

```
LONG ITC_Start(HANDLE   DeviceHandle,
               void*    sStart);
```

Starts data acquisition. if **sStart** == NULL driver performs immediate start with either default settings or parameters set by ITC_ConfigDevice().

sStart

```
typedef struct
{
LONG ExternalTrigger;
LONG OutputEnable;
LONG StopOnOverflow;
LONG StopOnUnderrun;

LONG RunningOption;
LONG ResetFIFOs;
LONG NumberOf640usToRun;
LONG Reserved;

Double StartTime; // Seconds
Double StopTime;  // Seconds
}
ITCStartInfo;
```

if ExternalTrigger, OutputEnable, StopOnOverflow, StopOnUnderrun set to “-1” ITC_Start() will use with either default settings or parameters set by ITC_ConfigDevice().

ExternalTrigger:

for ITC16/18: Bit 0: Enable trigger.

for ITC1600:

Bit 0:	Enable trigger.
Bit 1:	Use trigger form PCI1600
Bit 2:	Use timer trigger
Bit 3:	Use Rack 0 TrigIn BNC
Bit 4:	Use Rack 0 Digital Input 0 BNC
Bit 5:	Use Rack 0 Digital Input 1 BNC
Bit 6:	Use Rack 0 Digital Input 2 BNC
Bit 7:	Use Rack 0 Digital Input 3 BNC

OutputEnable:

Enable output channels

StopOnOverflow:

Stop acquisition if input overflow condition occurs

StopOnUnderrun:

Stop acquisition if output underrun condition occurs

RunningOption:

Bit 0: DontUseTimerThread
If set the application program is responsible for calling **ITC_UpdateNow()** function, which transfer data from hardware FIFO to individual FIFOs.

Bit 1: FastPointersUpdate (for ITC1600 only)
FastPointersUpdate flag instruct driver to request DSP for precision data pointer. Otherwise, pointer increments by 1K sample steps.

Bit 1: ShortDataAcquisition
Specify that no new output points will be loaded to FIFO.

Bit 16..23: Timer Ticks in us
Timer ticks of thread which transfer data from hardware FIFO to individual FIFOs (not used if DontUseTimerThread is set).
if zero – default value is used

Bit 24..32: Timer update Interval in us
Timer update interval of thread, which transfer data from hardware FIFO to individual FIFOs (not used if DontUseTimerThread is set).
if zero – default value is used

```
//RunningOption
#define DontUseTimerThread          1
#define FastPointerUpdate           2
#define ShortDataAcquisition        4
#define TimerResolutionMask   0x00FF0000  //Timer ticks
#define TimerIntervalMask     0xFF000000  //Timer update Interval
#define TimerResolutionShift        16
#define TimerIntervalShift          24
```

ResetFIFOs:

Resets FIFO pointers before start
Please note use this for debugging purposes.

NumberOf640usToRun:

This parameter is for ITC1600. If specified (not 0) ITC600 will disable outputs after “NumberOf640usToRun” counter expired.
Use it in case of “short” acquisition cycles to reduce inter acquisition intervals.

StartTime:

Start time in seconds

StopTime:

Stop time in seconds

Example:

```
      HANDLE DeviceHandle;
      long Error;
      char ErrorText[256];
      Error = ITC_Start(DeviceHandle, NULL);
      if(Error != ACQ_SUCCESS)
            {
            Error = ITC_GetStatusText(    DeviceHandle,
                                          Error,
                                          ErrorText,
                                          256);
            ...Process Error...
            }
```

2. Stop Acquisition

```
LONG ITC_Stop(HANDLE   DeviceHandle,
              void*    sParam);
```

`sParam is not defined yet. May be NULL`
`It may contain` **`ExternalTrigger`**`,` **`StopTime`**`, etc.`

Example:

```
      HANDLE DeviceHandle;
      long Error;
      char ErrorText[256];
      Error = ITC_Stop(DeviceHandle, NULL);
      if(Error != ACQ_SUCCESS)
            {
            Error = ITC_GetStatusText(    DeviceHandle,
                                          Error,
                                          ErrorText,
                                          256);
            ...Process Error...
            }
```

---

3. Update Acquisition

```
LONG ITC_UpdateNow(HANDLE   DeviceHandle,
                   void*    sParam);
```

sParam is not defined yet

This function will update all FIFO data.
The most common use for this call is if the **DontUseTimerThread** flag is set in the **ITC_Start()** function or process data in faster rate
Use only in SMART_MODE.

4. Get Available Number of Points to Read/Write

```
LONG ITC_GetDataAvailable(HANDLE          DeviceHandle,
                          unsigned long   NumberOfChannels,
                          void*           sDataAvailable);
```

Returns available number of points for each channel (FIFO)

NumberOfChannels:
specify number of channels to process (size of sDataAvailable array)

sDataAvailable:

```
typedef struct
{
unsigned short ChannelType;         //Channel Type
unsigned short Command;             //Command
unsigned short ChannelNumber;       //Channel Number
unsigned short Status;              //Status
unsigned long  Value;               //Number of points OR Data Value
void* DataPointer;                  //Data
}
ITCChannelDataEx;
```

Example:

```
      HANDLE DeviceHandle;
      long Error;
      char ErrorText[256];
      unsigned long Channels = 2;
      ITCChannelDataEx xITCOneChannelData[2];
      ZeroMemory(Itc18Data, sizeof(xITCOneChannelData));

      xITCOneChannelData[0].ChannelType = D2H;
      xITCOneChannelData[0].ChannelNumber = 3;
      xITCOneChannelData[1].ChannelType = H2D;
      xITCOneChannelData[1].ChannelNumber = 2;
      Error = ITC_GetDataAvailable( DeviceHandle,
                                    Channels,
                                    xITCOneChannelData);
      if(Error != ACQ_SUCCESS)
            {
            Error = ITC_GetStatusText(    DeviceHandle,
                                          Error,
                                          ErrorText,
                                          256);
            ...Process Error...
            }
```

5. Read/Write Data for specified channels

```
LONG ITC_ReadWriteFIFO(HANDLE             DeviceHandle,
                       unsigned long      NumberOfChannels,
                       void*              sData);
```

Read or write data for each specified channel (FIFO)

NumberOfChannels:
specify number of channels to process (size of sData array)

sData:

```
typedef struct
{
unsigned short ChannelType;         //Channel Type
unsigned short Command;             //Command
unsigned short ChannelNumber;       //Channel Number
unsigned short Status;              //Status
unsigned long  Value;               //Number of points OR Data Value
void* DataPointer;                        //Data
}
ITCChannelDataEx;
```

ChannelType:
Combine ChannelType and Command parameters
ChannelType:

```
0 D2H – Input
1 H2D – Output
2 DIGITAL_INPUT  – Digital Input
3 DIGITAL_OUTPUT – Digital Output
4 AUX_INPUT
5 AUX_OUTPUT
```

Command:

```
      Reading from FIFO:
            FLUSH_FIFO_COMMAND_EX          -   Flush Hardware FIFO (for input)
            GET_LAST_POINTS_FIFO_COMMAND_EX –   Read Last NumberOfPoints
      Writing to FIFO:
            RESET_FIFO_COMMAND_EX          -   Reset Hardware FIFO (for output)
            PRELOAD_FIFO_COMMAND_EX        -   Preload Hardware FIFO
            LAST_FIFO_COMMAND_EX           –   Last data chunk to process.
                                               DA output disabled after this data
                                               chunk
```

```
#define RESET_FIFO_COMMAND_EX              0x0001
#define PRELOAD_FIFO_COMMAND_EX            0x0002
#define LAST_FIFO_COMMAND_EX               0x0004
#define FLUSH_FIFO_COMMAND_EX              0x0008
#define ITC_SET_SHORT_ACQUISITION_EX       0x0010
```

NumberOfPoints:
Number of points to process
Note: If **NumberOfPoints** is set to "–1", this function will read all available points.

Example:

```
HANDLE DeviceHandle;
long Error;
char ErrorText[256];
unsigned long Channels = 2;
ITCChannelDataEx xITCReadData[2];

short Data1[100], Data2[1000];

ZeroMemory(xITCReadData, sizeof(xITCReadData));
xITCReadData[0].ChannelType = D2H;
xITCReadData[0].ChannelNumber = 2;
xITCReadData[0].NumberOfPoints = 100;
xITCReadData[0].Data = (short*)Data1;
xITCReadData[1].ChannelType = D2H;
xITCReadData[1].Command = FLUSH_FIFO_COMMAND;
xITCReadData[1].ChannelNumber = 4;
xITCReadData[1].NumberOfPoints = 1000;
xITCReadData[1].Data = (short*)Data2;

Error = ITC_ReadWriteFIFO(    DeviceHandle,
                              Channels,
                              xITCReadData);
if(Error != ACQ_SUCCESS)
      {
      Error = ITC_GetStatusText(    DeviceHandle,
                                    Error,
                                    ErrorText,
                                    256);
      ...Process Error...
      }
```

6. Update Read Position

```
LONG ITC_UpdateFIFOPosition(HANDLE          DeviceHandle,
                            unsigned long   NumberOfChannels,
                            void*           sFIFOPosition);
```

Modify the position of FIFO pointers for each channel (FIFO) by specified number of points.

NumberOfChannels:
specify number of channels to process (size of sFIFOPosition array)

sFIFOPosition:

```
typedef struct
{
unsigned short ChannelType;         //Channel Type
unsigned short Command;             //Command
unsigned short ChannelNumber;       //Channel Number
unsigned short Status;              //Status
unsigned long  Value;               //Number of points OR Data Value
void* DataPointer;                  //Data
}
ITCChannelDataEx;
```

Note: If **PointsNumber == -1** this function will flush the FIFO buffer.

Only for "SMART" Mode

Example:

```
      HANDLE DeviceHandle;
      long Error;
      char ErrorText[256];
      unsigned long Channels = 2;
      ITCChannelDataEx xITCOneChannelPoints[2];

      ZeroMemory(xITCOneChannelPoints, sizeof(xITCOneChannelPoints));
      xITCOneChannelPoints[0].ChannelType = H2D;
      xITCOneChannelPoints[0].ChannelNumber = 1;
      xITCOneChannelPoints[0].Value = 64;
      xITCOneChannelPoints[1].ChannelType = DIGITAL_INPUT;
      xITCOneChannelPoints[1].ChannelNumber = 5;
      xITCOneChannelPoints[1].Value = -1;
      Error = ITC_UpdateFIFOPosition(     DeviceHandle,
                                          Channels,
                                          xITCOneChannelPoints);
      if(Error != ACQ_SUCCESS)
            {
            Error = ITC_GetStatusText(    DeviceHandle,
                                          Error,
                                          ErrorText,
                                          256);
            ...Process Error...
            }
```

7. Get FIFO Information

```
LONG ITC_GetFIFOInformation ( HANDLE             DeviceHandle,
                              unsigned long      NumberOfChannels,
                              ITCChannelDataEx*  sFIFOInfo);
```

This function returns the current **DataPointer** to the user and number of points available in the buffer until the end of this buffer

NumberOfChannels:
specify number of channels to process (size of sFIFOInfo array)

sFIFOInfo:

```
typedef struct
{
unsigned short ChannelType;          //Channel Type
unsigned short Command;              //Command
unsigned short ChannelNumber;        //Channel Number
unsigned short Status;               //Status
unsigned long  Value;                //Number of points OR Data Value
void* DataPointer;                   //Data
}
ITCChannelDataEx;
```

Command specifies requested information:

```
#define READ_FIFO_INFO                          0
#define READ_FIFO_READ_POINTER_COUNTER          1
#define READ_FIFO_WRITE_POINTER_COUNTER         2
```

If Command == “READ_FIFO_INFO”
Value - FIFO Size
DataPointer – FIFO Start Address

If Command == “READ_FIFO_READ_POINTER_COUNTER”
Value - FIFO Read Pointer
if(DataPointer != NULL) – Total Read Counter (64 bit).

If Command == “READ_FIFO_WRITE_POINTER_COUNTER”
Value - FIFO Write Pointer
if(DataPointer != NULL) – Total Write Counter (64 bit).

Only for "SMART" Mode

8. Get FIFO Pointers

```
LONG ITC_GetFIFOPointers (    HANDLE            DeviceHandle,
                              unsigned long     NumberOfChannels,
                              ITCChannelDataEx* sFIFOInfo);
```

This function returns the current **DataPointer** to the user and number of points available in the buffer until the end of this buffer

NumberOfChannels:
specify number of channels to process (size of sFIFOInfo array)

sFIFOInfo:

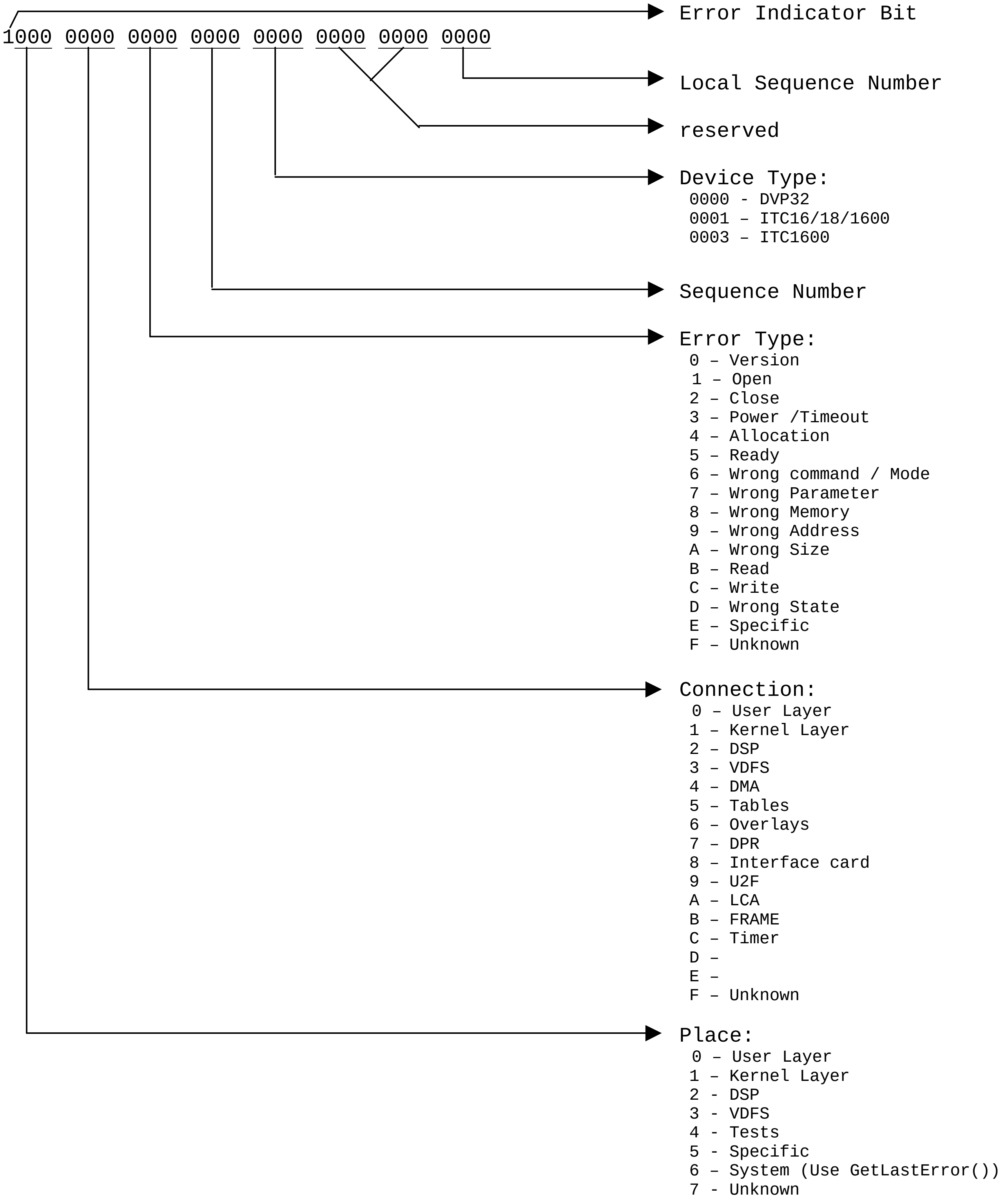
```
typedef struct
{
unsigned short ChannelType;         //Channel Type
unsigned short Command;             //Command
unsigned short ChannelNumber;       //Channel Number
unsigned short Status;              //Status
unsigned long  Value;               //Number of points OR Data Value
void* DataPointer;                  //Data
}
ITCChannelDataEx;
```

If (**Value == 0**), the function will calculate available number of points and set **Value** with that number, or how many points are left in "linear addressing", whichever is smaller.

Only for "SMART" Mode

G. Error definition

Errors are bit oriented:



1000 0000 0000 0000 0000 0000 0000 0000

Error Indicator Bit

Local Sequence Number

reserved

Device Type:
0000 - DVP32
0001 – ITC16/18/1600
0003 – ITC1600

Sequence Number

Error Type:
0 – Version
1 – Open
2 – Close
3 – Power /Timeout
4 – Allocation
5 – Ready
6 – Wrong command / Mode
7 – Wrong Parameter
8 – Wrong Memory
9 – Wrong Address
A – Wrong Size
B – Read
C – Write
D – Wrong State
E – Specific
F – Unknown

Connection:
0 – User Layer
1 – Kernel Layer
2 – DSP
3 – VDFS
4 – DMA
5 – Tables
6 – Overlays
7 – DPR
8 – Interface card
9 – U2F
A – LCA
B – FRAME
C – Timer
D –
E –
F – Unknown

Place:
0 – User Layer
1 – Kernel Layer
2 - DSP
3 - VDFS
4 - Tests
5 - Specific
6 – System (Use GetLastError())
7 - Unknown

H. Examples