



Standalone Basic for the Xpert2/9210B

A means to run Basic programs when the Xpert application is not running or recording is turned off.

Version 3.2.0.8

Prepared by:
R&D
July, 2009

Sutron Corporation
22400 Davis Drive
Sterling, Virginia 20164
TEL: (703) 406-2800
FAX: (703) 406-2801
WEB: <http://www.sutron.com/>



Introduction

Standalone Basic is a separate program (StandaloneBasic.exe) which may be loaded in to an Xpert2 or 9210B in order to run Basic programs outside the Xpert application. This allows Basic programs to be created which can run when recording is turned off or the Xpert application is completely shut down and hence is appropriate for creating communications drivers. The language and syntax is nearly identical to Xpert Basic except for capabilities which cannot be supported outside the Xpert application and some new capabilities which have been added.

The Xpert Basic manual covers most of the functionality of Standalone Basic, so this application note will focus on the features which are different, new, or unsupported.

To obtain a copy of the standalone basic application, please contact Sutron Customer Service.

Running Standalone Basic

Standalone Basic is not part of the standard build and must be transferred to the Xpert2/9210B before it can be used. It may be run interactively from Remote's Flash Disk prompt or it can be run automatically on boot by editing Autoexec.bat file. By default Standalone Basic will start running any file in the Flash Disk folder with the extension .SAB., but what is run and where it's run from may be changed by passing a path on the command line. The extension .SAB is used by default to prevent a conflict with Xpert Basic as both may be used in the same system. Status and error messages are passed on to Remote and can be viewed using the "Report" command, but unlike messages from Xpert Basic they are not stored in the System log.

Example (starting Standalone Basic from a custom folder)

```
\Flash Disk> Report High
Reporting enabled and filter is set
\Flash Disk> StandaloneBasic "\My Folder\*.sab"
\Flash Disk>07/09/2009 14:09:19 (Stat) - Standalone Basic ver 3.2.0.8 (C)2009
Sutron Corp is Running (\My Folder\*.sab)
07/09/2009 14:09:20 (Stat) - Reloading Basic Programs
07/09/2009 14:09:20 (Stat) - Standalone Basic has Stopped (\My Folder\*.sab)
```



In the above example, we turned message reporting on and then started Standalone Basic running programs from a custom folder named “My Folder”. The program ran and stopped. You can verify if any Standalone Basic programs are still running with the “Tasks” command, and can terminate a specific instance with the /STOP option or all instances with the /STOPALL option.

Example (stopping Standalone Basic from a custom folder)

```
\Flash Disk>StandaloneBasic "\My Folder\*.sab" /STOP
07/09/2009 14:18:11 (Stat) - Stopping Standalone Basic Programs (\My Folder\*.SAB)
07/09/2009 14:18:16 (Stat) - Standalone Basic has Stopped (\My Folder\*.SAB)
```

If you wish to just stop any and all instances of Standalone Basic, then you may use the /STOPALL option.

Example (stopping all instances of Standalone Basic)

```
\Flash Disk>StandaloneBasic /STOPALL
07/09/2009 14:32:42 (Stat) - Standalone Basic has Stopped (\Flash Disk\*.SAB)
07/09/2009 14:32:45 (Stat) - Standalone Basic has Stopped (\My Folder\*.SAB)
```

One major difference with Standalone Basic is how it runs and when it stops. The later is simple. Standalone Basic will stop running when the program no longer has any work to do. This includes the main body of the program and any tasks created by the program. Tasks are a new feature and replace the functionality of scheduled subroutines.

Supported Features from Xpert Basic

Many of the features of Xpert Basic are supported in Standalone Basic, this includes SSP functions, Serial I/O functions, File I/O, Socket I/O, Web Server, The Tag() function, StatusMsg, WarningMsg, ErrorMessage, StatusMsg, GetScheduledTime.

The combination of support for Serial, SSP, and TCP/IP allows for “bridge” applications to be created. For instance commands can be processed from a communications port using the Serial I/O functions and then issued to Remote’s telnet command prompt using the Socket I/O functions. Data can be retrieved from the Xpert Application and injected in to the application by using the Tag() function.

Unsupported Features

Some of the features of Xpert Basic that could not be supported by Standalone Basic include Basic Sensors, Basic Blocks, Basic Schedules (use Tasks instead), Custom Command Line Parsers, Log Files, Turn LAN option but other devices are supported, Alarm Handling, Sensor Measurements



The inability to log data is a major limitation, but it's possible for a Standalone Basic program to append data to a text file, or perhaps to pass data via tags to an Xpert Basic program which would then log the contents.

Unsupported Statements and Functions

A number of statements and functions cannot be supported by Standalone Basic. Most of these have to do with measuring sensors and special features such as support for Basic Blocks, Basic Sensors, etc. These include AddGroup, Log, ClearAlarm, ClearAlert, ConfigAd, SetCounter, SetDigital, DisableAlert, EnableAlert, Measure, StartTag, StopTag, Power (except for "USB" and "LAN" options), PowerAd, RaiseAlarm, RaiseAlert, SetOutputData, SetOutputDigits, SetOutputName, SetOutputAlarm, SetOutputQuality, SetOutputUnits, Turn "LAN", Ad, Ad420, AdAC, AdGain, AdRato, AdTherm, Counter, Digital, Frequency, GetInputAlarm, GetInputData, GetInputQuality, InAlarm, InAlert, SDI, SdiCollect, Systat(1, 4, 5, 6, 9, 10, 26),

The primary means to read a sensor or any value that Xpert Basic can access but Standalone Basic cannot is by using the Tag() function. Since Basic tags can be created which run code when the value is retrieved, it's possible to create a tag which will measure a sensor when it's requested.

New Statements and Functions

Standalone Basic introduces a number of new statements and functions which primarily revolve around adding the ability to create tasks, manage tasks, and synchronize tasks. The new features will likely propagate to Xpert Basic in the near future and once that happens these new functions will also be able to signal or synchronize tasks across the two versions of Basic.

Standalone Basic introduces two new special subroutines: START_PROGRAM, and STOP_PROGRAM. START_PROGRAM is called after the main body of all the program have completed running, and STOP_PROGRAM is run when there's a request to stop the program or if all tasks complete. You define these in your program as "Public Sub" with no parameters.

Example:

```
TimeToStop = 0

Public Sub START_PROGRAM
    StatusMsg "Start Program"
    TimeToStop = 0
    REM Run the main processing loop every 5 seconds
    StartTask "ProcessingLoop", 0, TimeSerial(0, 0, 0), TimeSerial(0, 0, 5)
End Sub

Public Sub STOP_PROGRAM
    StatusMsg "Stop Program"
    TimeToStop = -1
    StopTask "ProcessingLoop"
End Sub
```



Here is a description of the new statements and functions:

StartTask Statement

Syntax:

```
StartTask subroutine, parameter, [offset, interval, [startinterval]]
```

StartTask runs a subroutine in a separate thread of execution and is typically used to perform background processing. The subroutine argument is actually a string which contains the name of a public subroutine, parameter is a value that may be passed in to the subroutine. If no other arguments are specified then the task is started immediately and run just once. If an offset and an interval are specified then the task will be automatically run on a scheduled basis. A startinterval may also be specified and this will be used instead of interval for just the first execution. These values are time variables and can be formed using the TimeSerial() function. A scheduled tasks' local variables will persist across calls if the Dim statement is used to declare them.

A task may only be started once. If you try to start a task that's already running or scheduled to run the error BE_TASK_ALREADY_RUNNING will occur.

Example:

```
Public Sub AveragingTask(Channel)
    Dim Count
    Dim Sum
    Sum = Sum + Ad(1, Channel)
    Count = Count + 1
    If Count >= 10 Then
        StatusMsg "Average for channel " & Channel & " is " & Sum/Count
        Count = 0
        Sum = 0
    End If
End Sub

REM Starting on the hour, constantly average A/D channel 1 every minute
StartTask "AveragingTask", 1, TimeSerial(0,0,0),
        TimeSerial(0,1,0), TimeSerial(1,0,0)
```

Errors:

```
44: BE_TASK_ALREADY_RUNNING
```

StopTask Statement

Syntax:

```
StopTask subroutine
```

StopTask will stop a task that has been scheduled with the StartTask statement, but cannot stop a task that is currently running. StopTask only requests a task to stop and hence returns immediately. The best way to signal a running task to stop would be to use an event and add code to the task to check the event on a regular basis.



If the task has already stopped then the error BE_TASK_NOT_RUNNING will occur. This can be used to tell when a task has actually stopped.

Example:

```
REM Try for up to a minute to stop the averaging tasks
On Error Goto Done
For i = 1 To 600
    StopTask "AveragingTask"
    Sleep 0.1
Next i
Done:
On Error Goto 0
```

Errors:

```
45: BE_TASK_ALREADY_RUNNING
```

TriggerTask Statement

Syntax:

```
TriggerTask subroutine
```

TriggerTask will cause a scheduled task to start running immediately ignoring its execution interval.

If the task has already stopped then the error BE_TASK_NOT_RUNNING will occur.

Example:

```
TriggerTask "AveragingTask"
```

Errors:

```
45: BE_TASK_ALREADY_RUNNING
```

Lock Statement

Syntax:

```
Lock [timeout-seconds, [semaphore, [name]]]
```

Locks the global basic semaphore. If another program has locked the semaphore, then the program will sleep for up to the specified timeout. Be sure to call [UnLock](#) when done. An Err code is set if the semaphore could not be obtained within the timeout (which may have been 0).

In addition a global variable may be passed in to be used as a semaphore and optionally a name may be given to create a named semaphore which can be used across processes to serialize access to a shared resource.

Example:

```
Static GlobalData
Lock
GlobalData = GlobalData + 1
```



```
UnLock
```

```
Static MySemaphore
```

```
Lock 60, MySemaphore, "Com2Lock"  
Call TalkToCom2  
UnLock MySemaphore
```

Errors:

```
33: BE_LOCK_TIMEOUT
```

UnLock Statement

Syntax:

```
UnLock [semaphore]
```

Unlocks the Global Basic Semaphore allowing other programs to obtain it (see [Lock Statement](#)). A user created semaphore (named or unnamed) may also be UnLock'd.

Rnd Function

Syntax:

```
result = Rnd([LowLimit], HighLimit)
```

Returns a random number between LowLimit and HighLimit. If LowLimit is not specified then it's assumed to be 0.0. If HighLimit is not specified then it is assumed to be 1.0.

Example:

```
Function SimulateAirTemp  
    SimulateAirTemp = Rnd(-40, 70)  
End Function
```

WaitEvent Function

Syntax:

```
Result = WaitEvent(timeout-sec, Event1, [Event2, ...])
```

Waits up to timeout-sec seconds for an event to be signaled (see SetEvent). One or more events can be waited upon but only one of the events needs to be signaled for the wait to stop. The function returns 0 if a timeout occurs or 1 if Event1 was signaled, 2 if Event2 was signaled, etc. The events must have been initialized with either SetEvent or ResetEvent before they can be passed to WaitEvent or else a BE_INVALID_ARGUMENT error will occur. Events are a powerful means to synchronize and signal tasks to perform work or a special function. If more than one event is signaled at the same time, then the lowered numbered event will be returned.

Example:

```
Dim DoSomeWork  
ResetEvent DoSomeWork  
  
REM CallDoSomeWork until the StopLoop event has been raised
```



```
Do While WaitEvent(0, StopLoop) = 0
    Call DoSomeWork
End Loop
```

Errors:

```
29: BE_INVALID_ARGUMENT
```

ResetEvent Statement

Syntax:

```
ResetEvent event, [name]
```

ResetEvent converts a regular variable in to an event variable (if it isn't already) and clears it. A name may optionally be specified to create a named event that can be used across processes. The event variable must be declared or initialized before ResetEvent is used, and ResetEvent or SetEvent must be called before the variable may be used by WaitEvent.

Resetting an event variable will prevent tasks which have called WaitEvent on the event variable from waking up.

The main advantage of using events rather than a simple boolean variable and a loop is that there is no CPU consumed when WaitEvent is used to wait for an event to be triggered.

Example:

```
REM Create an event and make it initially cleared
Dim MyEvent
ResetEvent MyEvent
```

SetEvent Statement

Syntax:

```
SetEvent event, [name]
```

SetEvent converts a regular variable in to an event variable (if it isn't already) and sets it. A name may optionally be specified to create a named event that can be used across processes. The event variable must be declared or initialized before SetEvent is used, and ResetEvent or SetEvent must be called before the variable may be used by WaitEvent.

Setting an event variable will wake-up any tasks which have called WaitEvent on the event variable.

The main advantage of using events rather than a simple boolean variable and a loop is that there is no CPU consumed when WaitEvent is used to wait for an event to be triggered.

Example:

```
REM Trigger an event
SetEvent MyEvent
```