KSequence Protocol Handler

Overview

This component provides serial and ethernet KSequence services to WinPLC applications. To minimize complexity, the API is intentionally minimal - all implementation details are neatly hidden. As always – keep it simple.

The handler uses a worker thread for protocol operation to minimize interaction with the host application. Blocking and synchronization are provided to prevent external changes to protocol data during logic processing. To improve read performance during lengthy logic solves, the handler maintains two copies of protocol data. While the master copy is locked from external changes, external reads continue to be serviced from the internal cache copy.

If desired, backplane I/O can be read through the protocol, however, direct I/O writes are not supported. If direct I/O control is desired, it must be added to the control program. This limitation is intentional and forces control of the I/O to the control application. It is the author's firm belief that direct access to the I/O from an asynchronous comm protocol invites bad things to happen.

Basic emulation of a DL450 is provided to fool devices that require it. The selection of the DL450 was based on the fact that it is the only CPU that natively supports the V memory ranges that were chosen for mapping WX, WY, DWX, and DWY. Obviously a 205 CPU would have been preferable, but none supported the desired range.

Operation

Operation of the KSequence handler is simple but specific – there aren't many calls to make, but they do need to happen a specific way. The accompanying example, KSeqDemo, correctly implements the required calls and might clear any confusion. The basic steps are:

1) Create a new handler my calling CreateKSeqHandler. Store the new object's handle, you'll need it. At this point the handler object has been constructed, but it is not yet functional.

2) Initialize the new handler for serial or ethernet operation by calling InitAsSerialHandler or InitAsEthernetHandler. This creates the worker thread, sets up the specified port, and starts running the protocol. The handler is now functional and should be responding to external requests.

3) At the beginning of each scan, the master copy of the protocol data must be locked from external changes by calling LockExternalAccess. Failure to do so could result in corrupted data.

4) The master data can now be changed by the logic. There are 2 possible ways to go about this: direct access and mirrored.

   - For applications with no formal I/O driver interface – most user apps – direct access to the protocol data is usually easiest. For these apps GetV, SetV, GetC, and SetC are the way to go.

   - For applications with a formal I/O driver interface – most software vendors' control engines – it's probably better to mirror the protocol's memory and interface your control engine to the mirror copy. To do so, you should read all of the protocol data into a local copy, which can then be manipulated directly by program logic. Then, when all logic is solved, write back all local data to the master data. The functions for block access of the master data are ReadWordMemory, WriteWordMemory, ReadBitMemory, and WriteBitMemory.

   - In either case, it's totally up to the developer. Either way will work fine. GetV/SetV/GetC/SetC will result is more overhead, but is sometimes easier to use. For user apps the overhead is usually negligible, but for a control engine it might be an issue.

5) At the bottom of each scan, after all changes to the master copy have been written, the master copy must be unlocked by calling UnlockExternalAccess. Failure to do so will result in the protocol handler waiting endlessly to access the master copy when a write transaction occurs. Additionally, the handler's internal cache data is updated during the call to UnlockExternalAccess.

6) After calling UnlockExternalAccess, but prior to starting the next scan, allow some idle time for the protocol engine to work. The amount you chose is arbitrary, the more you allow, the more responsive the comm will be.

7) When the handler is no longer required, shut it down by calling ExitHandler. This terminates the protocol thread, but does not free memory.

8) After calling ExitHandler, the handler can safely be freed by calling DestroyKSeqHandler .

Notes

- This component supports multiple instances. If you run multiple instances, each port must be unique – you can't run more than one instance on COM1 or IP 0x7676.

- The data is tied to the instance. In other words, writing V0 through a serial instance on COM1 will not update the value of V0 on another instance running on COM2. Unlike in a traditional PLC where there is one and only one copy of the data, there is one copy per protocol instance. If you wish to share values between more than instance, it is up to the control program to move data between them. If anyone would like to do this, but doesn't understand how, send us an email and we'll send you a sample app. It's isn't hard, but to eliminate confusion in KSeqDemo, I opted to leave out multiples for now.

Supported Task Codes

Task code	Function	Notes
0x40	Monitor data	See supported ranges for details
0x46	Write V memory	See supported ranges for details
0x50	Read scratch pad	Minimal support. Supported only for PLC emulation.
0x51	Write scratch pad	Minimal support. Supported only for PLC emulation.
0x2A	Read program	Minimal support. Supported only for PLC emulation.
0x44	Force bit on	See supported ranges for details
0x45	Force bit off	See supported ranges for details

Supported Ranges

Data Range	Notes
V0 – V7777	User V memory range. General purpose registers.
V40400 – V40477	V mapping of backplane inputs (X). Supported when I/O is enabled.
V40500 – V40577	V mapping of backplane outputs (Y). Supported when I/O is enabled.
V40600 – V40777	V mapping of C memory.
X0 – X1777	Backplane inputs. Supported when I/O is enabled.
Y0 – Y1777	Backplane outputs. Supported when I/O is enabled.
C0 – C3777	C memory. General purpose bit memory.
V30000 – V32000	V mapping of backplane word and dword, inputs and outputs. Ksequence has no direct support for word or dword I/O since Koyo CPUs don't support them. The WinPLC does support WX, WY, DWX, and DWY. Beginning at V30000, WX, WY, DWX, and DWYs are mapped by slot, for each I/O device that supports them.

API Reference

CreateKSeqHandler

Creates an uninitialized KSequence handler. Prior to using the handler it must be initialized as an Ethernet handler or serial handler using the InitAsXXXXHandler functions.

```
KSEQ_API HKSEQ CreateKSeqHandler
(
BOOL ExposeIO          //  TRUE to enable exposure of IO
);
```

Parameters:

ExposeIO
Boolean flag to enable the exposure of PLC IO types X, Y, WX, WY, DWX, DWY.

Return Values:

Returns a handle to the newly created handler or NULL on failure. When the handler is no longer required it should be destroyed using DestroyKSeqHandler.

DestroyKSeqHandler

Destroys a previously created KSeq handler.

```
KSEQ_API void DestroyKSeqHandler
(
HKSEQ hKSeq            // Handle of KSeq handler to destroy
);
```

Parameters:

hKSeq
Handle of KSeq handler to destroy. Following destruction the handle is invalid and shouldn't be used.

Return Values:

None

InitAsSerialHandler

Initializes a newly created Kseq handler for use with a serial port. Either this or InitAsEthernetHandler must be called before the handler is operational. The InitAsXXXHandler functions must not be called more than once on a single handle. When the handler in no longer required, call ExitHandler prior to destroying.

```
KSEQ_API BOOL InitAsSerialHandler
(
HKSEQ hKSeq,                // Handle of KSeq
WCHAR *pPort,               // Port identifier
DWORD BaudRate,    // Baud rate
BYTE ByteSize,              // Number of data bits
BYTE Parity,                // Type of parity
BYTE StopBits               // Number of stop bits
);
```

Parameters:

hKSeq
Handle of KSeq to initialize. Must be initialized, but not more than once.

pPort
Pointer to wide string containing identifier of port to open. Example L"COM1:". For COM10, use L"COM0:"

BaudRate
Baud rate of port. Can use the Win32 CBR_XXX defines or the raw baud rate value.

ByteSize
Size of data byte. Normally 8.

Parity
Parity of port. Uses standard Win32 defines: NOPARITY, ODDPARITY, EVENPARITY, MARKPARITY, SPACEPARITY.

StopBits
Number of stop bits for port. Uses standard Win32 defines: ONESTOPBIT, ONE5STOPBITS, TWOSTOPBITS.

Return Value:

Non-zero on success.

Notes:

To emulate a DL250 programming port, call with the following values:
InitAsSerialHandler(hKSeq, L"COM1:", CBR_9600, 8, ODDPARITY, ONESTOPBIT);

InitAsEthernetHandler

Initializes a newly created KSeq handler for use with an ethernet port. Either this or InitAsSerialHandler must be called before the handler is operational. The InitAsXXXHandler functions must not be called more than once on a single handle. When the handler in no longer required, call ExitHandler prior to destroying.

```
KSEQ_API BOOL InitAsEthernetHandler
(
HKSEQ hKSeq,          // Handle of KSeq
WORD Port   // IP port value
);
```

Parameters:

hKSeq
Handle of uninitialized KSeq. Must be called, but only once.

Port
IP port value of installed handler. Do not use Host default value of 0x7070. Other than that, anything above decimal 5000 is fine. The client software must use the value selected here.

Return Value:

Non-zero on success.

ExitHandler

Terminates an active KSeq handler. Should be called once on every initialized handler prior to calling DestroyKSeqHandler.

```
KSEQ_API BOOL ExitHandler
(
HKSEQ hKSeq          // Handle of KSeq
);
```

Parameters:

hKSeq
Handle of KSeq to terminate.

Return Value:

Non-zero on success.

LockExternalAccess

Locks master copy of protocol memory to prevent external changes. This should be called prior to calls to ReadXXXXMemory or WriteXXXMemory to prevent corruption from protocol thread.

```
KSEQ_API void LockExternalAccess
(
HKSEQ hKSeq          // Handle of KSeq
);
```

Parameters:

hKSeq
Handle of KSeq to lock.

Return Value:

None

Notes:

Must be unlocked with UnlockExternalAccess to allow external changes.

UnlockExternalAccess

Unlocks master copy of protocol memory previously locked with LockExternalAccess. Also updates the protocol handler's internal read cache.

```
KSEQ_API void UnlockExternalAccess
(
HKSEQ hKSeq          // Handle of KSeq
);
```

Parameters:

hKSeq
Handle of KSeq.

Return Value:

None

ReadWordMemory

Reads from master copy of word memory (V memory). Should be locked with LockExternalAccess prior to reading. This function is provided for block reading of word memory, not single point access. For single point reads of word memory, see GetV.

```
KSEQ_API void ReadWordMemory
(
HKSEQ hKSeq,        // Handle of KSeq
DWORD Offset,       // Starting word to read from
int Count,          // Number of words to read
void *pData         // Buffer to receive data
);
```

Parameters:

hKSeq
Handle of KSeq.

Offset
Beginning word location to read from.

Count
Number of words of word memory to read.

pData
Buffer to receive word memory.

Return Value:

None

WriteWordMemory

Writes to master copy of word memory. Should be locked with LockExternalAccess prior to writing. This function is provided for block writing of word memory, not single point access. For single point writes of word memory, see SetV.

```
KSEQ_API void WriteWordMemory
(
HKSEQ hKSeq,        // Handle of KSeq
DWORD Offset,       // Starting word to write to
int Count,          // Number of words to write
void *pData         // Source data
);
```

Parameters:

hKSeq
Handle of KSeq.

Offset
Beginning word location to write.

Count
Number of words of word memory to write.

pData
Buffer to provide word memory.

Return Value:

None

ReadBitMemory

Reads from master copy of bit memory (C memory). Should be locked with LockExternalAccess prior to reading. This function is provided for block reading of bit memory, not for single point access. For single point reads of bit memory, see GetC.

```
KSEQ_API void ReadBitMemory
(
HKSEQ hKSeq,        // Handle of KSeq
DWORD Offset,       // Starting byte to read from
int Length,         // Bytes to read
void *pData         // Buffer to receive data
);
```

Parameters:

hKSeq
Handle of KSeq.

Offset
Beginning byte offset to read. Unlike GetC, this is a byte offset, not a bit address.

Count
Number of bytes of bit memory to read.

pData
Buffer to receive bit memory.

Return Value:

None

WriteBitMemory

Writes to master copy of bit memory (C memory). Should be locked with LockExternalAccess prior to writing. This function is provided for block writing of bit memory, not for single point access. For single point writes of bit memory, see SetC.

```
KSEQ_API void WriteBitMemory
(
HKSEQ hKSeq,        // Handle of KSeq
DWORD Offset,       // Beginning byte to write
int Length,         // Number of bytes to write
void *pData         // Source data
);
```

Parameters:

hKSeq
Handle of KSeq.

Offset
Beginning byte offset to write. Unlike SetC, this is a byte offset, not a bit address.

Count
Number of bytes of bit memory to write.

pData
Source data to write.

Return Value:

None.

GetV

Reads a single location from master copy of word memory (V memory). Provided for use in applications that do not wish to mirror protocol memory using block accesses. For block reads of word memory, see ReadWordMemory.

```
KSEQ_API WORD GetV
(
HKSEQ hKSeq,          // Handle of KSeq
int Address           // Address of V location to read
);
```

Parameters:

hKSeq
Handle of KSeq.

Address
V memory location to read.

Return Value:

Value of specified V location.

SetV

Writes a single location to the master copy of word memory (V memory). Provided for use in applications that do not wish to mirror protocol memory using block accesses. For block writes of word memory, see WriteWordMemory.

```
KSEQ_API void SetV
(
HKSEQ hKSeq,        // Handle of KSeq
int Address,        // Address of V location to write
WORD Val            // New value to write
);
```

Parameters:

hKSeq
Handle of Kseq.

Address
V memory location to write

Val
New value to write.

Return Value

None.

GetC

Reads a single bit location from the master copy of bit memory (C memory). Provided for use in applications that do not wish to mirror protocol memory using block accesses. For block reads of bit memory, see ReadBitMemory.

```
KSEQ_API BOOL GetC
(
HKSEQ hKSeq,          // Handle of KSeq
int Address           // Bit address of C location to read
);
```

Parameters

hKSeq
Handle of Kseq.

Address
C memory location to read. Unlike ReadBitMemory, this is a bit address.

Return Value

Boolean value of specified location.

SetC

Writes a single bit location to the master copy of bit memory (C memory). Provided for use in applications that do not wish to mirror protocol memory using block accesses. For block writes of bit memory, see WriteBitMemory.

```
KSEQ_API void SetC
(
HKSEQ hKSeq,        // Handle of KSeq
int Address,        // Bit address of C location to write.
BOOL Val            // New value of C location
);
```

Parameters

hKSeq
Handle of KSeq.

Address
C memory location to write. Unlike WriteBitMemory, this is a bit address.

Val
New value of C location

Return Value

None.