Using FieldServers with Modbus/TCP

Presented by:

Gordon MacLachlan (Mac)

Presentation Highlights

Presentation Summary

The Modbus Standard
- •Function codes
- •Dual addressing system

Modbus Variations
- •Variations caused by transport medium
- •Variations caused by integer restriction
- •JBUS

Modbus TCP
- •Modbus TCP vs Modbus RTU
- •Considerations resulting from the introduction of Ethernet
- •Troubleshooting Ethernet protocols.

Configuration tips
- •Dealing with poll lengths
- •Dealing with Function codes
- •About the “Could not create cache block” error
- •Remote devices that use fixed node addressing.

Question and Answer Session

The Modbus Standard

General Comments

- Modbus is a truly open standard. The specification can be downloaded from the web and implemented by anybody. There is no charge for the specification
- Modbus is based on a Master/Slave poll and response topology.
- Modbus is an integer protocol, i.e: floating point is not supported by the protocol (variations of modbus exist that support floating point)
- Modbus supports 4 basic data types: Analog Inputs, Analog Outputs (Registers), Digital Inputs, Digital Outputs (Coils/Flags)
- Modbus supports 9999 addresses per data type (a variation, JBUS, supports 65535 per data type)
- Modbus builds NO intelligence into what it is sending, i.e: it does not care what the data in the registers represent.
- Modbus is used worldwide, and can probably boast that it is the most supported protocol in the world.

Modbus Memory Map

Address Format 1	Data Type	Address Format 2
00001 09999	Flags (Read/Write)	Read: Function Code 1, Offsets 0-9998 Write Single: Use Function Code 5 Write Multiple: Use function Code 15
10001 19999	Digital Inputs (Read Only)	Read: Function Code 2, Offsets 0-9998 Write Single: N/A Write Multiple: N/A
30001 39999	Analog Inputs (Read Only)	Read: Function Code 4, Offsets 0-9998 Write Single: N/A Write Multiple: N/A
40001 49999	Registers (Read/Write)	Read: Function Code 3, Offsets 0-9998 Write Single: Use Function Code 6 Write Multiple: Use function Code 16

Modbus Variations

Variations resulting from a difference in connection medium:

- Modbus RTU: This is the most common Modbus variation. The medium for communication is RS-232 or RS-485. Data is transferred in hexadecimal format. Only one master allowed.
- Modbus ASCII: Also uses RS-232 or RS-485. Data is transferred in ASCII format. Only one master allowed.
- Modbus TCP: Medium is Ethernet. Data is transferred in hexadecimal format using the TCP/IP transport protocol. Allows multiple masters
- Modbus Plus: Uses a proprietary medium. Special hardware needed. Allows multiple masters.
- All of the above protocols support Modbus in its true format.

Modbus Variations

Variations resulting from protocol restrictions:

The following FieldServer drivers are modbus variants:

- Modbus Tekair
- Modbus Daniels
- Modbus OmniFlow

These drivers exist due to the need for the vendor to compensate for Modbus being a 16 bit integer protocol by providing a method for sending 32 bit values. Many other vendors do this too.

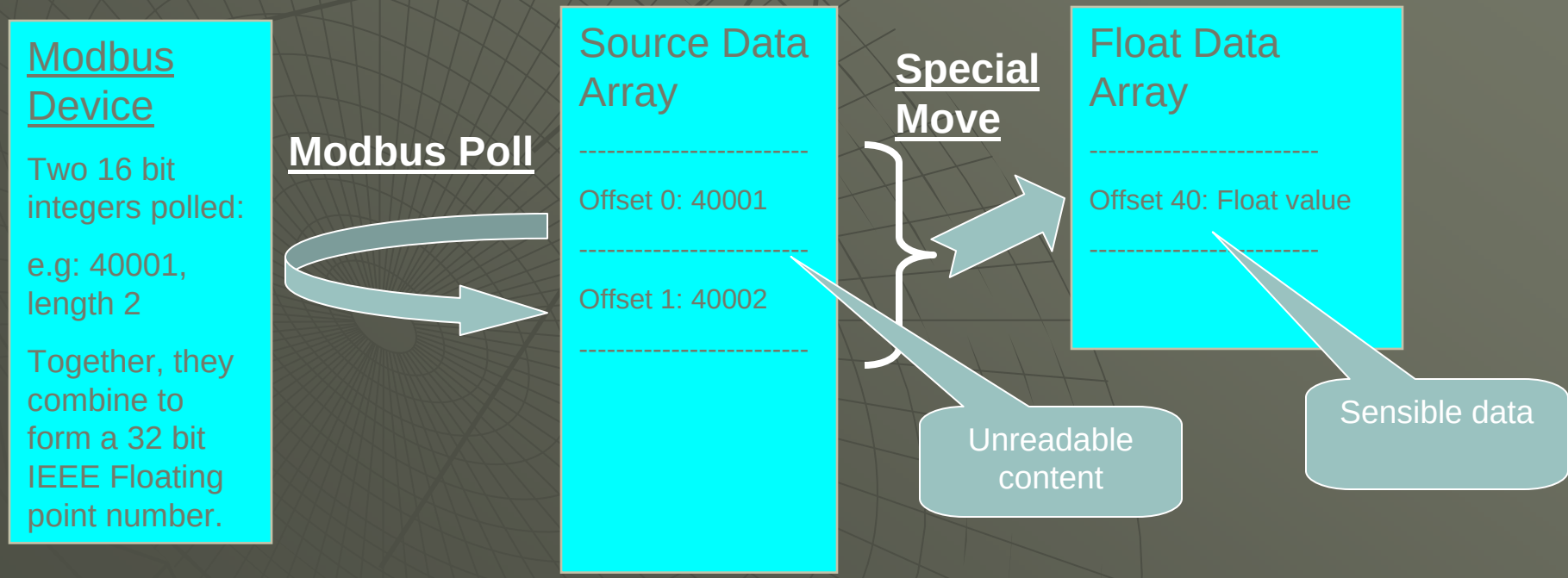
Additionally, FieldServer has a special move function that can assist in decoding these 32 bit variants for use by other protocols.

Modbus Variations

Supporting 32 bit values with Modbus

- FieldServer concept for manipulating "modbus floating point" (Concept is the same for 32 bit integers):

```
Moves
Source_Data_Array ,Source_Offset ,Target_Data_Array ,Target_Offset ,Length ,Function
Source_DA         ,0             ,Target_DA        ,40            ,10     ,2.i16-1.float-sw
```



Modbus Variations

Variations resulting from protocol restrictions (JBUS):

Background:

- Modbus stores the offset (address) as an integer in the poll message
- However, the legal value for offset can only range from 0-9998.
- An unsigned integer can represent any integer from 0-65535. So why the limit?
- JBUS takes care of this and allows for the offset to range from 0-65535
- JBUS therefore supports 65536 addresses per data type.

Notes

1. You can poll a JBUS device with Modbus, but you will not have access to offsets above 9998
2. JBUS is sometimes called “Extended Modbus”

Modbus TCP

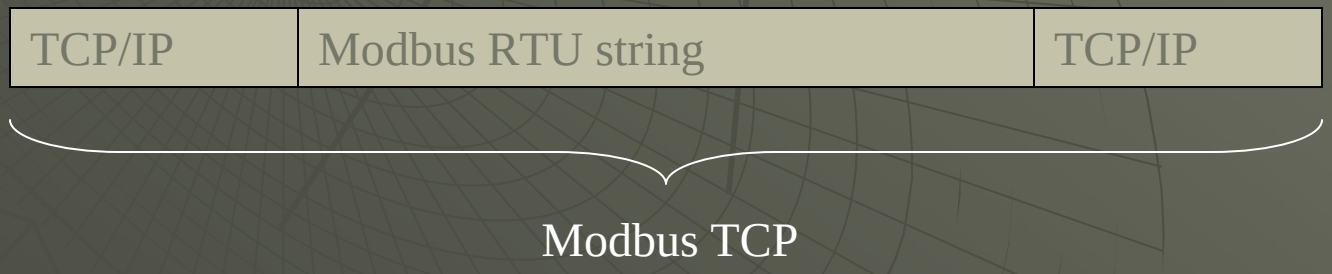
Modbus RTU vs Modbus TCP:

Modbus RTU:

- Serial protocol using RS-232 or RS-485 as a medium
- Devices are addressed by Station Address (Node_ID)

Modbus TCP:

- Uses Ethernet.
- Modbus RTU + TCP/IP Layer = Modbus TCP
- Devices are addressed by IP address and Station address.



Modbus TCP

Dealing with Ethernet and Modbus TCP:

- User needs to become familiar with subnetting and the IP addressing system
- Using Modbus TCP across subnets requires the use of IP gateways, so the Gateway address setting in IP configuration becomes important
- Modbus TCP uses port 502 to communicate. You'll need to know this if you are to punch a hole in any firewall for communications.
- Modbus TCP makes use of TCP/IP connection management to allow multiple communications connections to occur. This makes multiple masters on a Modbus TCP Network possible.

Modbus TCP

Troubleshooting on Ethernet:

- Arm yourself with a good Ethernet capture tool. One such package, Ethereal is available as freeware on the Web (go to www.ethereal.com)
- For testing, a copy of Modscan32 or similar is highly recommended (go to www.win-tech.com). Use this to emulate a Client on the network and test communications.
- Be acutely aware of the network layout, including all subnets involved, all Modbus TCP devices on the network, all routers, and all firewalls. All of these can influence communications. A detailed network layout diagram is essential.
- Some devices support IP addressing in conjunction with variable Unit identifier (Node_ID). Others will fix the unit identifier and work with IP addressing only. Know which variant you are dealing with, as the FieldServer will need to fix its Node_ID to the correct address if the latter is in use.

Configuration Tips

Dealing with Function Codes

- Usually, devices support function codes 1,2,3,4,5,6,15 and 16. This allows for reads and writes of all types.
- Some devices only support a subset of these function codes. This is legal, but it does mean that when communicating with these devices, the function code being used needs to be selectable.
- Managing Function Codes 1-4 is easily done by choice of address
- It is often necessary, however, to suppress the use of Function codes 15 or 16 and use 5 or 6 instead. To do this, FieldServer provides special map descriptor paramaters that allow the read or write type to be stipulated. See Enote018 for more details.

Example

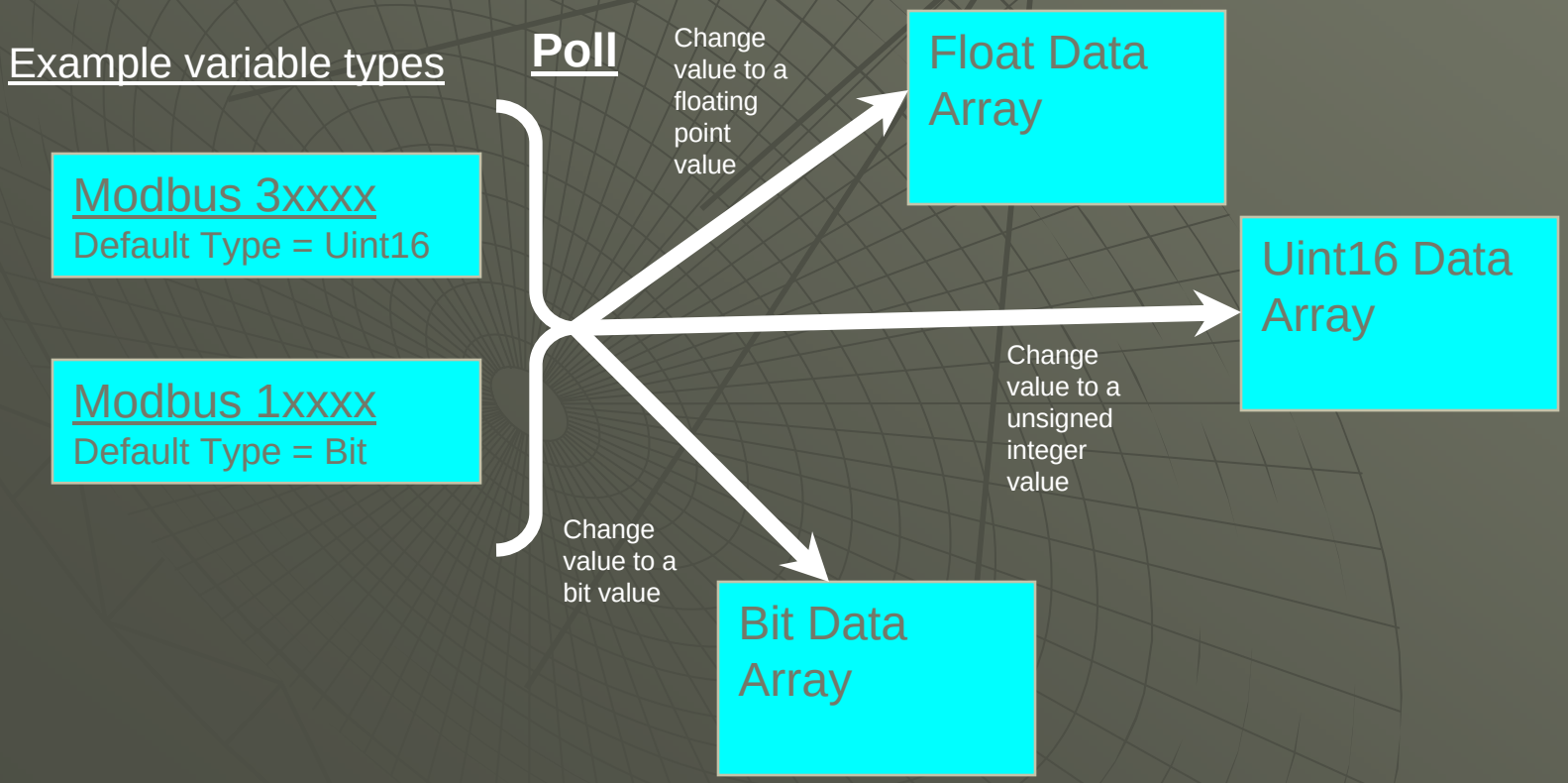
FC 6 = Write Single Register

1. Add a parameter to the Modbus client side map descriptor called data_type.
2. If you specify the data_type as Single_Register and the Function as WRBC or WRBX, then a modbus poll with FC 6 will be generated.
3. Of course Single_Register implies a length of one, and even if you try to set the length longer in the csv file, the length is limited to 1 in the driver.

Configuration Tips

Modbus and Typecasting (FieldServer does this by default)

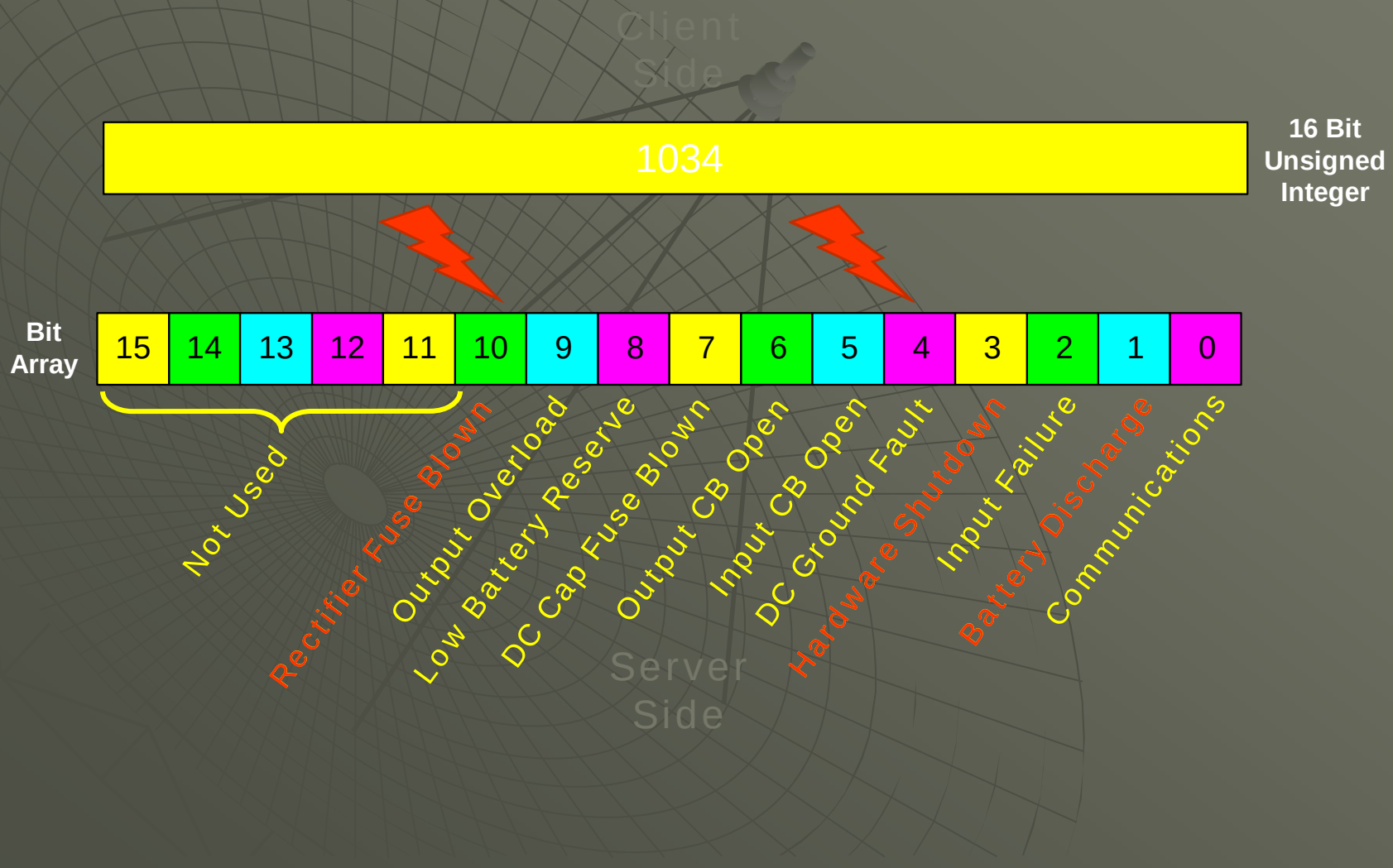
- Each protocol variable type is allocated a default Data type
- Moves between dissimilar data types result in type-casting
- To avoid type-casting, use matching data array types or special functions such as Floating point moves or Packed Bit Data Arrays



Configuration Tips

Bit Packing

Concept:

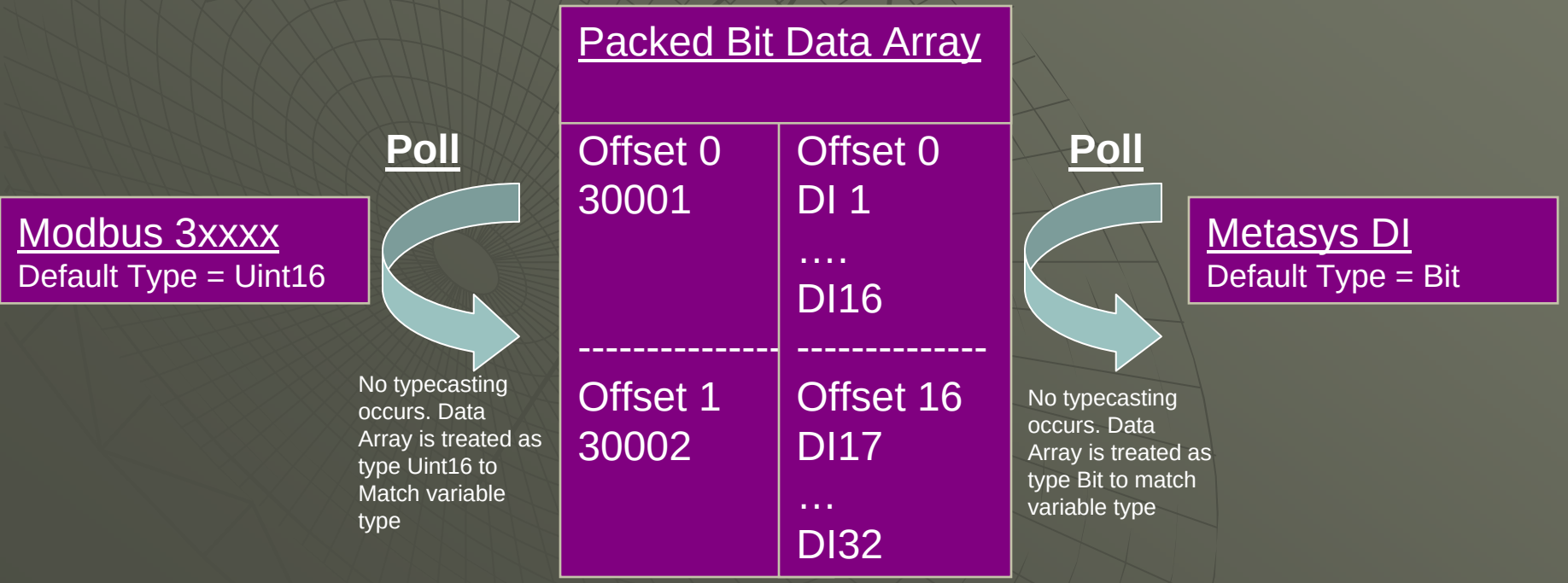


Configuration Tips

Bit Packing

Mechanics:

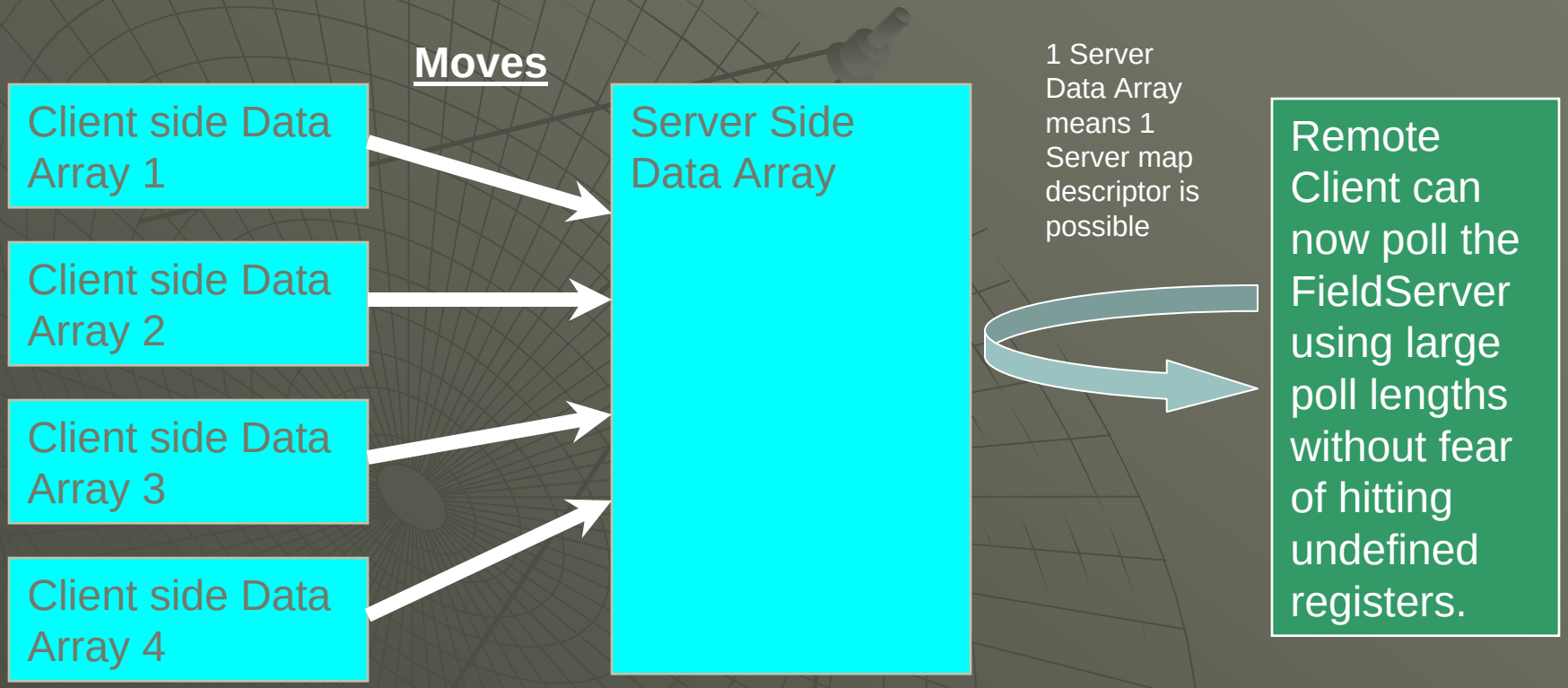
```
//===========================================================================
//
//    Data Arrays
//    Packed_Bit is enabled merely by declaring the data array to be
//    of type: Packed_Bit
Data_Arrays
Data_Array_Name ,Data_Format    ,Data_Array_Length
Packer          ,Packed_Bit     ,200
```



Configuration Tips

Modbus as a slave – Keep data map "large"contiguous

- Why? Because you don't know how a client will structure its poll, and if it polls for undefined points, the poll will fail.



Configuration Tips

Configuring Data Moves

```
//=============================================================================
//
//    Data Arrays
//
//    Declare Data Arrays for the data to be moved. In the real world example,
//    the Data Arrays may already be declared.
//

Data_Arrays
Data_Array_Name ,Data_Format    ,Data_Array_Length
Source_DA       ,Float          ,200
Target_DA       ,Float          ,200


//=============================================================================
//
//    Set up the moves to move the data.
//

Moves
Source_Data_Array ,Source_Offset ,Target_Data_Array ,Target_Offset ,Length ,Function
Source_DA         ,0             ,Target_DA         ,40            ,20     ,Move_Only
```

Configuration Tips

Help! I'm getting "could not create cache block"

- This error occurs frequently when the FieldServer is configured as a modbus slave
- It means that the FieldServer has received a poll for addresses that do not exist in the FieldServer under any one Server map descriptor.
- The content of the error message will tell what poll was received. The message can be intepreted as follows:

```
T02> MODBUS_TCP : Could not create cache block
T02>    Node:1 Addr:40001 Len:100
```

Server Node being polled

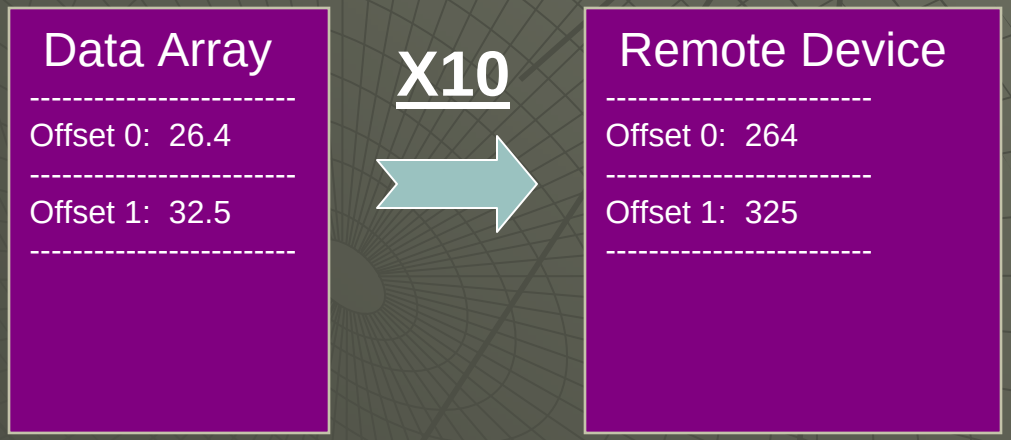
Modbus Address being polled

Length of poll

Configuration Tips

Managing Floating point values with Integers - Scaling

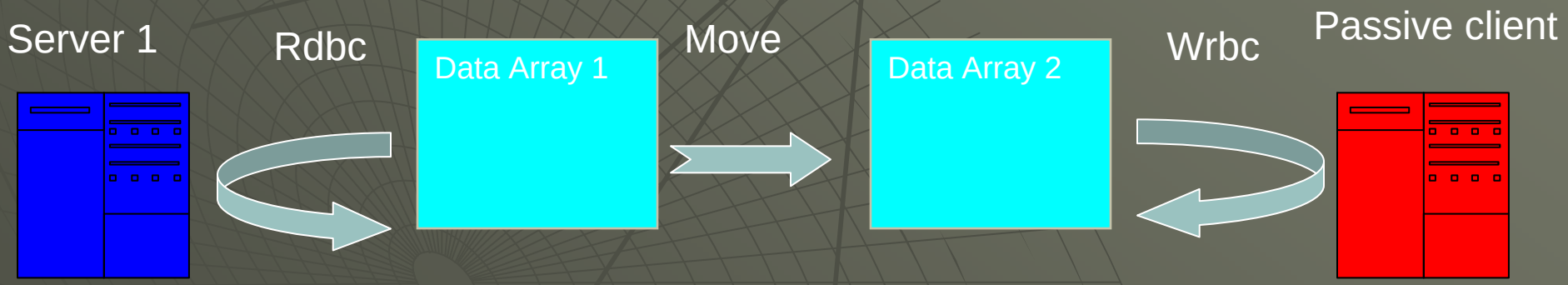
```
Map_Descriptors
Node_Name ,Address ,Length ,Data_Array_Low_Scale ,Data_Array _High_Scale ,Node_Low_Scale ,Node_High_Scale
MBP_Srv_1 ,30001   ,200    , 0                   ,10                      ,0              ,100
```



Configuration Tips

Making two modbus Slaves talk to each other using the FieldServer

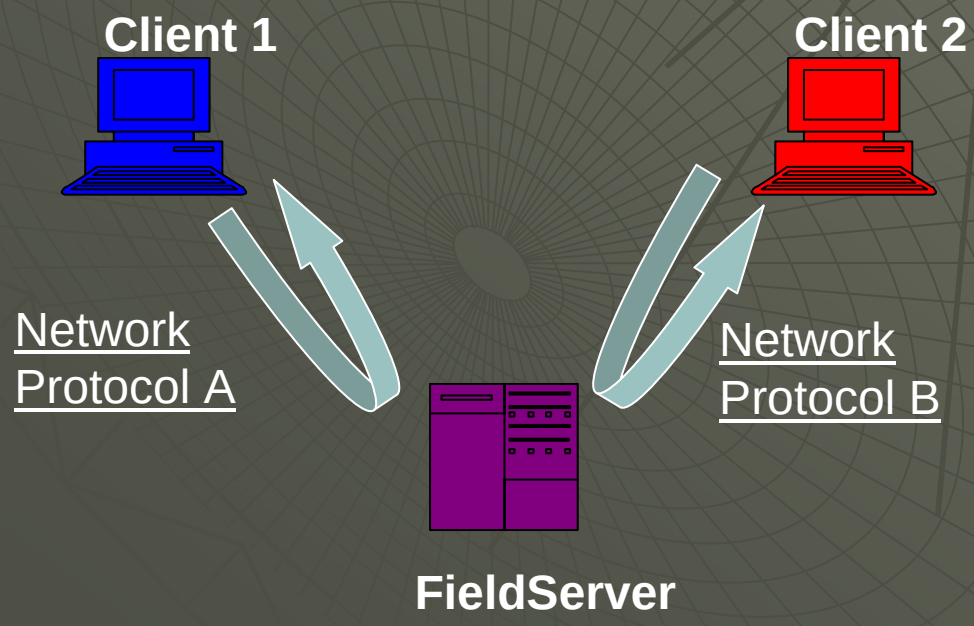
- Slaves are passive, so they cannot independantly request data from each other.
- Insert the FieldServer as a Master, read the data from one slave, and then write it to the other
- Wrbc is normally the best function for writing status data
- Moves may be necessary to separate responsible map descriptors



Configuration Tips

Helping Clients share data

- Passive Server map descriptors can reference the same data array. This means the Data Array can be used as a shared repository for multiple active clients.



- This “Data Sharing” configuration is useful in applications where clients on different networks need to share stored data
- The same setup can provide a passive server/passive client functionality too
- Map descriptor function used for both protocols A and B is “passive”
- Fieldserver is non-intrusive into both networks, and responds to queries and commands only.

Configuration Tips

Using the FieldServer to bring Modbus RTU into a Modbus TCP network

- Since the protocol is the same, port expansion can be used. This requires minimal configuration.
- Full Port expansion configuration example:

```
Connections
Port, Baud, Parity, Data_Bits, Stop_Bits, Protocol  , Handshaking, Poll_Delay,
R1  , 9600, None  , 8        , 1        , Modbus_RTU, None       , 0.100s    ,
R2  , 9600, None  , 8        , 1        , Modbus_RTU, None       , 0.100s    ,
P7  , 9600, None  , 8        , 1        , Modbus_RTU, None       , 0.100s    ,
P8  , 9600, None  , 8        , 1        , Modbus_RTU, None       , 0.100s    ,

Connections
Adapter, Protocol
N1     , Modbus/TCP

Nodes
Node_Name, Node_ID, Protocol  , Port,
PLC_07   , 7      , Modbus_RTU, P7  ,
PLC_08   , 8      , Modbus_RTU, P8  ,

PLC_11   , 11     , Modbus_RTU, R1  ,
PLC_12   , 12     , Modbus_RTU, R1  ,

PLC_21   , 21     , Modbus_RTU, R2  ,
PLC_22   , 22     , Modbus_RTU, R2  ,
```

Configuration Tips

Monitoring the status of Client side nodes (Servers) – Node Status

```
//=============================================================================
//
// Notes :  All that is needed to enable node status is the node status data array
//          declaration.  Note, though, that Node ID's in the config need to be
//          unique, otherwise the FieldServer will incorrectly report the status of
//          duplicate Node ID's
//
//=============================================================================



//=============================================================================,
//
//    Data Arrays
//
Data_Arrays
Data_Array_Name ,Data_Format ,Data_Array_Length ,Data_Array_Function
DA_Comm_Status  ,Bit         ,256               ,Node_Status

//=============================================================================,
//
//    Modbus TCP Server Map Descriptor Turns 10001 into Nde 1 Status, 10002=Node 2, etc.
//
Map_Descriptors
Map_Descriptor_Name, Data_Array_Name, Data_Array_Offset, Function, Node_Name , Address, Length,
Node_Status_Map    , DA_Comm_Status , 1                , Server  , MBP_Srv_11, 10001  , 200   ,
```

Resources

1) www.fieldserver.com
   - FieldServer Configuration manual
   - FieldServer Troubleshooting manual
   - Modbus TCP Manual
2) www.modbus.org
   - List of modbus certified vendors
   - Modbus TCP specification
   - Application guidelines
3) Vendor sites
   - Mapping for modbus devices
4) www.ethereal.com
   - Ethernet packet capture utility
5) www.win-tech.com
   - Modscan32

Questions?

Email Mac at:

sfint@comcast.net

THANK YOU!

.....for taking the time to attend this presentation.