

# SynqNet Objects

## Introduction

A **SynqNet** object manages a single SynqNet network connected to a motion controller. It represents the physical network. It contains information about the network state, number of nodes, and network status.

A SynqNet network can have one or more [SqNode](#) objects associated with it and each SqNode object can have one or more motors and/or drive interfaces. The SynqNet network provides read/write access to each node. During network initialization, the SynqNet nodes are discovered and mapped to the SynqNet object. The number of motors per SqNode object are determined and mapped to the controller's motor objects.

| [Error Messages](#) |

## Methods

### Create, Delete, Validate Methods

[meiSynqNetCreate](#)

[meiSynqNetDelete](#)

[meiSynqNetValidate](#)

### Configuration and Information Methods

[meiSynqNetCableNumToNodePort](#)

[meiSynqNetConfigGet](#)

[meiSynqNetConfigSet](#)

[meiSynqNetFlashConfigGet](#)

[meiSynqNetFlashConfigSet](#)

[meiSynqNetIdleCableListGet](#)

[meiSynqNetIdleCableStatus](#)

[meiSynqNetInfo](#)

[meiSynqNetPacketFlashConfigGet](#)

[meiSynqNetPacketFlashConfigSet](#)

[meiSynqNetPortToCableNum](#)

[meiSynqNetPacketConfigGet](#)

[meiSynqNetPacketConfigSet](#)

[meiSynqNetStatus](#)

[meiSynqNetTiming](#)

## Action Methods

[meiSynqNetFlashTopologyClear](#)

[meiSynqNetFlashTopologySave](#)

[meiSynqNetInit](#)

[meiSynqNetNetworkObjectNext](#)

[meiSynqNetShutdown](#)

## Event Methods

[meiSynqNetEventNotifyGet](#)

[meiSynqNetEventNotifySet](#)

[meiSynqNetEventReset](#)

## Relational Methods

[meiSynqNetControl](#)

[meiSynqNetNumber](#)

## Data Types

[MEISynqNetCableList](#)

[MEISynqNetCableLength](#)

[MEISynqNetCableStatus](#)

[MEISynqNetConfig](#)

[MEISynqNetFailedNodeMask](#)

[MEISynqNetInfo](#)

[MEISynqNetMessage](#)

[MEISynqNetRecoveryMode](#)

[MEISynqNetResourceCommand](#)

[MEISynqNetResourceIoBits](#)

[MEISynqNetResourceMonitor](#)

[MEISynqNetPacketCfg](#)

[MEISynqNetPacketCfgEncoder](#)

[MEISynqNetPacketCfgIo](#)

[MEISynqNetPacketCfgMotor](#)

[MEISynqNetPacketCfgNode](#)

[MEISynqNetPacketCfgProbe](#)

[MEISynqNetResourceCfgProbeDepth](#)

[MEISynqNetState](#)

[MEISynqNetStatus](#)

[MEISynqNetStatusCrcError](#)

[MEISynqNetTiming](#)

[MEISynqNetTrace](#)

## Constants

[MEISynqNetMaxCableHOP\\_COUNT](#)

[MEISynqNetMaxMotorFEEDBACK\\_PRIMARY\\_COUNT](#)

[MEISynqNetMaxMotorFEEDBACK\\_SECONDARY\\_COUNT](#)

[MEISynqNetMaxMotorCAPTURE\\_COUNT](#)

[MEISynqNetMaxMotorCOMPARE\\_COUNT](#)

[MEISynqNetMaxMotorENCODER\\_COUNT](#)

[MEISynqNetMaxMotorPULSE\\_ENGINE\\_COUNT](#)

[MEISynqNetMaxMotors](#)

[MEISynqNetMaxNodeMOTORS](#)

[MEISynqNetMaxNODE\\_COUNT](#)

[MEISynqNetNodeMaskELEMENTS](#)

[MEISynqNetPacketCfgIo](#)

[MEISynqNetPacketCfgMotor](#)

[MEISynqNetPacketCfgNode](#)

# meiSynqNetCreate

## Declaration

```
MEISynqNet meiSynqNetCreate(MPIControl control,
                             long number)
```

**Required Header:** stdmei.h

## Description

**meiSynqNetCreate** creates a SynqNet object identified by **number**, which is associated with a **control** object. SynqNetCreate is the equivalent of a C++ constructor.

<b>control</b>	a handle to a Control object
<b>number</b>	An index to the SynqNet network. First network = 0 Second network = 1 Third network = 2, etc.

### Return Values

<b>handle</b>	to a SynqNet object. After creating a SynqNet object it must be validated using <code>meiSynqNetValidate(...)</code> .
<b>MPIHandleVOID</b>	if the object could not be created

## See Also

[meiSynqNetDelete](#) | [meiSynqNetValidate](#)

# meiSynqNetDelete

## Declaration

```
long meiSynqNetDelete(MEISynqNet synqNet);
```

**Required Header:** stdmei.h

## Description

**meiSynqNetDelete** deletes a SynqNet object and invalidates its handle. SynqNetDelete is the equivalent of a C++ destructor. All objects that are created must be deleted in the reverse order to avoid memory leaks.

<b>synqNet</b>	a handle to a SynqNet object.
----------------	-------------------------------

### Return Values

[MPIMessageOK](#)

## See Also

[meiSynqNetCreate](#) | [meiSynqNetValidate](#)

# meiSynqNetValidate

## Declaration

```
long meiSynqNetValidate( MEISynqNet  synqNet ) ;
```

**Required Header:** stdmei.h

## Description

**meiSynqNetValidate** validates the SynqNet object and its handle. SynqNetValidate should be called immediately after an object is created.

<b>synqNet</b>	handle to a valid SynqNet object.
----------------	-----------------------------------

### Return Values

[MPIMessageOK](#)

[MEISynqNetMessageINTERFACE\\_NOT\\_FOUND](#)

## See Also

[meiSynqNetCreate](#) | [meiSynqNetDelete](#)

# meiSynqNetCableNumToNodePort

## Declaration

```
long meiSynqNetCableNumToNodePort ( MEISynqNet      synqNet ,
                                     long              cableNumber ,
                                     long              *nodeNumber ,
                                     MEINetworkPort *port ) ;
```

**Required Header:** stdmei.h

## Description

**meiSqnNodeConfigGet** converts a cable number on a SynqNet network into a node number and port. It reads the node number and writes it into a long pointed to by **nodeNumber** and reads the port and writes it to the value pointed to by **port**.

Network connections can be identified by a cable number OR a node number and port. For simple network topologies, it is easier to identify network connections by a cable number. For complex network topologies, it is easier to identify network connections by a node number and port.

<b>synqNet</b>	a handle to the SynqNet object
<b>cableNumber</b>	the number of the cable
<b>*nodeNumber</b>	a pointer to a node number
<b>*port</b>	a pointer to an enumerated port value

### Return Values

[MPIMessageOK](#)

## See Also

[meiSynqNetNodePortToCableNum](#) | [MEISynqNetCableList](#)

# meiSynqNetNodePortToCableNum

## Declaration

```
long meiSynqNetNodePortToCableNum( MEISynqNet      synqNet ,
                                     long              nodeNumber ,
                                     MEINetworkPort port ,
                                     long              *cableNumber ) ;
```

**Required Header:** stdmei.h

## Description

**meiSynqNetNodePortToCableNum** converts a node number and port on a SynqNet network into a cable number. It reads the cable number and writes it into a long pointed to by ***cableNumber***.

Network connections can be identified by a cable number OR a node number and port. For simple network topologies, it is easier to identify network connections by a cable number. For complex network topologies, it is easier to identify network connections by a node number and port.

<b>synqNet</b>	a handle to the SynqNet object
<b>nodeNumber</b>	the number of the node
<b>port</b>	an enumerated port value
<b>*cableNumber</b>	a pointer to a cable number

### Return Values

[MPIMessageOK](#)

## See Also

[MEISynqNetCableList](#)



# meiSynqNetConfigGet

## Declaration

```
long meiSynqNetConfigGet(MEISynqNet      synqNet ,
                        MEISynqNetConfig *config);
```

**Required Header:** stdmei.h

## Description

**meiSynqNetConfigGet** reads a SynqNet network (**synqNet**) configuration and writes it into the structure pointed to by **config**.

<b>synqNet</b>	a handle to a SynqNet object
<b>*config</b>	a pointer to a SynqNet config structure.

### Return Values

[MPIMessageOK](#)

## See Also

[meiSynqNetInfo](#) | [meiSqNodeConfigGet](#) | [meiSynqNetConfigSet](#)



# meiSynqNetInfo

## Declaration

```
long meiSynqNetInfo(MEISynqNet      synqNet,
                   MEISynqNetInfo  *info);
```

**Required Header:** stdmei.h

## Description

**meiSynqNetInfo** reads static information about the network associated with the **SynqNet** object and writes it into the structure pointed to by **info**. The SynqNet info structure contains read only data that was determined during network initialization.

<b>synqNet</b>	a handle to a SynqNet object
<b>*info</b>	pointer to a SynqNet network information structure

### Return Values

[MPIMessageOK](#)

[MPIMessageARG\\_INVALID](#)

## Sample Code

```
/*
   Get the number of nodes on the SynqNet network
   Returns -1 if there is an error
*/
int NumberOfNodes(MEISynqNet synqNet)
{
    MEISynqNetInfo info;
    long returnValue;
    int numNodes = -1;

    returnValue = meiSynqNetInfo(synqNet, &info);
    if(returnValue == MPIMessageOK)
    {
        if(info.nodeCount >= 0 && info.nodeCount <= MEIXmpMaxSynqNetBlocks)
        {
            numNodes = info.nodeCount;
        }
    }
}
```

```
return numNodes;  
}
```

## See Also

[meiSynqNetIdleCableListGet](#) | [MEISynqNetState](#)

# meiSynqNetConfigSet

## Declaration

```
long meiSynqNetConfigSet(MEISynqNet      synqNet ,
                        MEISynqNetConfig  *config);
```

**Required Header:** stdmei.h

## Description

**meiSynqNetConfigSet** writes a SynqNet network (**synqNet**) configuration from the structure pointed to by **config**.

<b>synqNet</b>	a handle to a SynqNet object
<b>*config</b>	a pointer to a SynqNet config structure.

### Return Values

[MPIMessageOK](#)

[MPIMessageARG\\_INVALID](#)

[MPIMessagePARAM\\_INVALID](#)

[MPISynqNetMessageRING\\_ONLY](#)

## See Also

[meiSynqNetInfo](#) | [meiSqNodeConfigSet](#) | [meiSynqNetConfigGet](#)

# meiSynqNetFlashConfigGet

## Declaration

```
long meiSynqNetFlashConfigGet(MEISynqNet      synqNet ,
                               void              *flash ,
                               MEISynqNetConfig *config) ;
```

**Required Header:** stdmei.h

## Description

**meiSynqNetFlashConfigGet** gets a SynqNet object's flash configuration and writes it in the structure pointed to by ***config***.

<b>synqNet</b>	a handle to a SynqNet object
<b>*config</b>	pointer to a locally instantiated MEISynqNetConfig structure.
<b>*flash</b>	<i>flash</i> is either an MEIFlash handle or MPIHandleVOID. If <i>flash</i> is MPIHandleVOID, an MEIFlash object will be created and deleted internally.

### Return Values

[MPIMessageOK](#)

## See Also

[meiSynqNetFlashConfigSet](#) | [MEISynqNetConfig](#) | [MEIFlash](#)

# meiSynqNetFlashConfigSet

## Declaration

```
long meiSynqNetFlashConfigSet(MEISynqNet      synqNet ,
                              MEISynqNetConfig *flash ,
                              MEIFlash      *config );
```

**Required Header:** stdmei.h

## Description

**meiSynqNetFlashConfigSet** gets a SynqNet object's flash configuration and writes it in the structure pointed to by **config**.

**NOTE:** The network topology must first be saved before changing MEISynqNetConfig.cableLength[n] values in Flash memory. These values will also be cleared when network topology is cleared using meiSynqNetFlashTopologyClear(...).

<b>synqNet</b>	a handle to a SynqNet object
<b>*flash</b>	<i>flash</i> is either an MEIFlash handle or MPIHandleVOID. If <i>flash</i> is MPIHandleVOID, an MEIFlash object will be created and deleted internally.  If <i>flash</i> is a valid MEIFlash handle, then the MEIFlash object cache will be updated but the actual write to controller flash will not occur. Use <a href="#">meiFlashMemoryFromFileType(...)</a> to prompt the actual write to flash.
<b>*config</b>	pointer to a locally instantiated <a href="#">MEISynqNetConfig</a> structure that has been initialized by performing a call to <a href="#">meiSynqNetFlashConfigGet(...)</a> or <a href="#">meiSynqNetConfigGet(...)</a> .

### Return Values

[MPIMessageOK](#)

[MPIMessageARG\\_INVALID](#)

[MPIMessagePARAM\\_INVALID](#)

[MEISynqNetMessageRING\\_ONLY](#)

[MEIFlashMessageNETWORK\\_TOPOLOGY\\_ERROR](#)

## See Also

[MEISynqNetConfig](#) | [MEIFlash](#) | [meiSynqNetFlashConfigGet](#) | [meiSynqNetConfigGet](#) | [meiSynqNetFlashTopologySave](#) | [meiSynqNetFlashTopologyClear](#)



# meiSynqNetFlashTopologySave

## Declaration

```
long meiSynqNetFlashTopologySave(MEISynqNet    synqNet ,
                                MEIFlash      flash ) ;
```

**Required Header:** stdmei.h

## Description

**meiSynqNetFlashTopologySave** will save the current SynqNet topology to the controller's flash memory.

After a power-on or network shutdown/init, the controller/MPI Library will compare the discovered network topology with the topology saved in flash memory. If the topology information matches, the controller will automatically transition the network to SYNQ (cyclic) mode and configured service commands are sent to the nodes.

**WARNING:** This is a low-level network configuration routine that will reset the SynqNet network.

The meiSynqNetFlashTopologySave method will:

1. Shutdown the SynqNet network.
2. Reset SqNode and Motor configurations to their defaults. See [MPI Object Configurations that use Service Commands](#).
3. Clear network topology information from Flash and Dynamic memory.
4. Initialize the SynqNet network.

<b>synqNet</b>	a handle to an MEISynqNet object whose network topology is to be saved.
<b>flash</b>	is either a valid MEIFlash handle or MPIHandleVOID. If flash is MPIHandleVOID, an MEIFlash object will be created and deleted internally.

### Return Values

[MPIMessageOK](#)

[MPISynqNetMessageTOPOLOGY\\_SAVED](#)

## See Also

[Save/Clear Topology to Flash](#) | [meiSynqNetFlashTopologyClear](#)

# meiSynqNetFlashTopologyClear

## Declaration

```
long meiSynqNetFlashTopologyClear(MEISynqNet    synqNet ,
                                  MEIFlash      flash ) ;
```

**Required Header:** stdmei.h

## Description

**meiSynqNetFlashTopologyClear** will clear the topology information that is saved in the controller's flash memory by meiSynqNetTopologySave().

Executing this method will disable the controller from comparing the discovered network topology with the topology saved in flash memory. But, during network initialization, the MPI Library still compares the topology stored in the controller's dynamic memory with the discovered topology. In both cases, the default service commands are sent to the nodes if the network initialization transitions to SYNQ (cyclic) mode.

**WARNING:** This is a low-level network configuration routine that will clear any SqNode or Motor configurations that use service commands.

The meiSynqNetFlashTopologyClear method will:

1. Shutdown the SynqNet network.
2. Reset SqNode and Motor configurations to their defaults. See [MPI Object Configurations that use Service Commands](#).
3. Clear network topology information from Flash and Dynamic memory.
4. Initialize the SynqNet network.

<b>synqNet</b>	a handle to an MEISynqNet object whose network topology is to be cleared.
<b>flash</b>	is either a valid MEIFlash handle or MPIHandleVOID. If flash is MPIHandleVOID, an MEIFlash object will be created and deleted internally.

### Return Values

[MPIMessageOK](#)

[MEIFlashMessageNETWORK\\_TOPOLOGY\\_ERROR](#)

## See Also

[Save/Clear Topology to Flash](#) | [meiSynqNetFlashTopologySave](#)

# meiSynqNetIdleCableListGet

## Declaration

```
long meiSynqNetIdleCableListGet(MEISynqNet synqNet,
                                MEISynqNetCableList *idleCable);
```

**Required Header:** stdmei.h

## Description

**meiSynqNetIdleCableListGet** reads a SynqNet network's list of idle cables and writes them into the structure pointed to by *idleCable*. An idle cable has no data traffic. It is the redundant network connection available in ring topologies only. A single ring topology has only one idle cable.

After network initialization, the cable from the last node to the controller is idle by default. If a network fault occurs and the network is configured for fault recovery, the network traffic will be redirected around the faulty connection, through the idle cable, and the faulty connection will become the new idle cable. To test the idle cable, use the `meiSynqNetIdleCableStatus(...)` method.

<b>synqNet</b>	a handle to a SynqNet object
<b>*idleCable</b>	a pointer to a SynqNet cable list structure. The cable list structure contains the number of idle cables and their identifying numbers.

### Return Values

[MPIMessageOK](#)

## See Also

[meiSynqNetIdleCableStatus](#) | [SynqNet Topologies](#)

# meiSynqNetIdleCableStatus

## Declaration

```
long meiSynqNetIdleCableStatus(MEISynqNet      synqNet ,
                               long              cableNumber ,
                               MEISynqNetCableStatus *cableStatus ) ;
```

**Required Header:** stdmei.h

## Description

**meiSynqNetIdleCableStatus** reads an idle cable's status and writes it into the structure pointed to by **cableStatus**. Normally, the idle cable has no data traffic. `meiSynqNetIdleCableStatus(...)` sends a special test packet across the specified idle cable and then waits for a valid response, then it sends another test packet in the opposite direction and waits for valid response.

The idle cable number for a network can be found using `meiSynqNetIdleCableListGet(...)`. `SynqNetIdleCableStatus` is not allowed for non-idle cables or when the network is recovering from a fault. During fault recovery (`SynqNetState = SYNQ_RECOVERING`), the network traffic is redirected around the faulty connection and the idle cable is reassigned. After recovery is complete (`SynqNetState = SYNQ`), the new idle cable can be tested with `SynqNetIdleCableStatus`. Use `meiSynqNetStatus(...)` to determine the `SynqNet` state.

<b>synqNet</b>	a handle to a SynqNet object
<b>cableNumber</b>	the number of the cable to be tested
<b>*cableStatus</b>	a pointer to a SynqNet cable status enumerated value.

### Return Values

[MPIMessageOK](#)

## See Also

[meiSynqNetIdleCableListGet](#) | [MEISynqNetState](#)

# meiSynqNetPacketFlashConfigGet

## Declaration

```
long  meiSynqNetPacketFlashConfigGet(MEISynqNet      synqNet ,
                                     void              *flash ,
                                     MEISynqNetPacketCfg *config );
```

**Required Header:** stdmei.h

## Description

**meiSynqNetPacketFlashConfigGet** is currently unsupported and is reserved for future use.

<b>synqNet</b>	a handle to a SynqNet object
<b>*flash</b>	<i>flash</i> is either an MEIFlash handle or MPIHandleVOID. If <i>flash</i> is MPIHandleVOID, an MEIFlash object will be created and deleted internally.  If <i>flash</i> is a valid MEIFlash handle, then the MEIFlash object cache will be updated, but the actual write to controller flash will not occur. Use <a href="#">meiFlashMemoryFromFileType(...)</a> to prompt the actual write to flash.
<b>*config</b>	pointer to configuration structure defined by <a href="#">MEISynqNetPacketCfg</a> .

### Return Values

[MPIMessageUNSUPPORTED](#)

## See Also

[meiSynqNetPacketFlashConfigSet](#) | [MEISynqNetPacketCfgNode](#)

# meiSynqNetPacketFlashConfigSet

## Declaration

```
long  meiSynqNetPacketFlashConfigSet(MEISynqNet      synqNet ,
                                     void             *flash ,
                                     MEISynqNetPacketCfg *config );
```

**Required Header:** stdmei.h

## Description

**meiSynqNetPacketFlashConfigSet** is currently unsupported and is reserved for future use.

<b>synqNet</b>	a handle to a SynqNet object
<b>*flash</b>	<i>flash</i> is either an MEIFlash handle or MPIHandleVOID. If <i>flash</i> is MPIHandleVOID, an MEIFlash object will be created and deleted internally.  If <i>flash</i> is a valid MEIFlash handle, then the MEIFlash object cache will be updated but the actual write to controller flash will not occur. Use meiFlashMemoryFromFileType() to prompt the actual write to flash
<b>*config</b>	pointer to configuration structure defined by MEISynqNetPacketCfg.

### Return Values

[MPIMessageUNSUPPORTED](#)

## See Also

[meiSynqNetPacketFlashConfigGet](#) | [MEISynqNetPacketCfgNode](#)



# meiSynqNetPacketConfigGet

## Declaration

```

/* WARNING: meiSynqNetPacketConfigSet(...) and
 * meiSynqNetPacketFlashConfigSet(...) are low-level network
 * routines that must clear other controller object configurations.
 * This method should be run before configuring most MPI objects.
 * Please refer to the online documentation for more information.
 */

long  meiSynqNetPacketConfigGet(MEISynqNet          synqNet ,
                               MEISynqNetPacketCfg *config);

```

**Required Header:** stdmei.h

## Description

**meiSynqNetPacketConfigGet** reads the current network packet configuration for all nodes found on the network to the location pointed to by **config**.

This method is useful for viewing the current network packed data being sent across the network. It is used in conjunction with `meiSynqNetPacketConfigSet(...)`. This method is also useful for optimizing network traffic (bandwidth).

Only configurable packet data fields are configured by this method. Fixed packet fields are not application configurable.

<b>synqNet</b>	a handle to a SynqNet object
<b>*config</b>	pointer to configuration structure defined by <code>MEISynqNetPacketCfg</code> .

### Return Values

[MPIMessageOK](#)

[MPIMessageARG\\_INVALID](#)

[MPIMessageUNSUPPORTED](#)

## See Also

[meiSynqNetPacketConfigSet](#) | [MEISynqNetPacketCfgNode](#)



# meiSynqNetPacketConfigSet

## Declaration

```

/* WARNING: meiSynqNetPacketConfigSet(...) and
 * meiSynqNetPacketFlashConfigSet(...) are low-level network
 * routines that must clear other controller object configurations.
 * This method should be run before configuring most MPI objects.
 * Please refer to the online documentation for more information.
 */

long meiSynqNetPacketConfigSet(MEISynqNet          synqNet ,
                               MEISynqNetPacketCfg *config );

```

**Required Header:** stdmei.h

## Description

**meiSynqNetPacketConfigSet** sets the network packet configuration to the configuration defined in the location pointed to by **config**.

**WARNING:** `meiSynqNetPacketConfigSet(...)` is a low-level network routine that will clear other controller object configurations and reset the SynqNet network. This method should be executed in your application before configuring any other MPI objects. However, any control object configurations that force a network re-initialization must be performed before this routine is executed – please see [mpiControlConfigSet\(...\)](#) routine for more information.

This method is useful for optimizing network traffic (bandwidth).

Only configurable packet data fields are configured by this method. Fixed packet fields are not application configurable.

<b>synqNet</b>	a handle to a SynqNet object
<b>*config</b>	pointer to configuration structure defined by <code>MEISynqNetPacketCfg</code> .

**Return Values**[MPIMessageOK](#)[MPIMessageARG\\_INVALID](#)[MEISynqNetMessageINCOMPLETE\\_MOTOR](#)[MEISynqNetMessageINVALID\\_AUX\\_ENC\\_COUNT](#)[MEISynqNetMessageINVALID\\_MOTOR\\_COUNT](#)[MEISynqNetMessageINVALID\\_COMMAND\\_CFG](#)[MEISynqNetMessageINVALID\\_ENCODER\\_COUNT](#)[MEISynqNetMessageINVALID\\_CAPTURE\\_COUNT](#)[MEISynqNetMessageINVALID\\_COMPARE\\_COUNT](#)[MEISynqNetMessageINVALID\\_INPUT\\_COUNT](#)[MEISynqNetMessageINVALID\\_OUTPUT\\_COUNT](#)[MEISynqNetMessageINVALID\\_MONITOR\\_CFG](#)[MPIMessageUNSUPPORTED](#)**See Also**

[meiSynqNetPacketConfigGet](#) | [MEISynqNetPacketCfgNode](#) | [MEISynqNetPacketCfg](#) | [mpiControlConfigSet](#)

# meiSynqNetStatus

## Declaration

```
long meiSynqNetStatus(MEISynqNet      synqNet ,
                     MEISynqNetStatus *status) ;
```

**Required Header:** stdmei.h

## Description

**meiSynqNetStatus** reads status from the network associated with the SynqNet object and writes it into the structure pointed to by status. The SynqNet status structure contains network operation state and error counters. This data is updated every controller sample.

<b>synqNet</b>	a handle to a SynqNet object
<b>*status</b>	pointer to a SynqNet status structure

### Return Values

[MPIMessageOK](#)

[MPIMessageARG\\_INVALID](#)

## See Also

[meiSqNodeStatus](#) | [meiSynqNetInfo](#)

# meiSynqNetTiming

## Declaration

```
long meiSynqNetTiming(MEISynqNet      synqNet ,
                     MEISynqNetTiming *timing) ;
```

**Required Header:** stdmei.h

## Description

**meiSynqNetTiming** returns the network timing values to the location pointed to by *\*timing* for the current operating network. This method can only be called in MEISynqNetStatus.state >= MEISynqNetStateSYNQ.

<b>synqNet</b>	handle to a valid SynqNet object.
<b>*timing</b>	a pointer to a MEISynqNetTiming structure.

### Return Values

[MPIMessageOK](#)

[MPIMessageARG\\_INVALID](#)

## See Also

[MEISynqNetTiming](#) | [MEISynqNetState](#)

# meiSynqNetInit

## Declaration

```
long meiSynqNetInit(MEISynqNet synqNet);
```

**Required Header:** stdmei.h

## Description

**meiSynqNetInit** initializes a SynqNet network. This method performs the same network initialization that is automatically done with mpiControllInit(...).

<b>synqNet</b>	a handle to a SynqNet object
----------------	------------------------------

### Return Values

<a href="#">MPIMessageOK</a>	
<a href="#">MPIMessageARG_INVALID</a>	

## See Also

[meiSynqNetInfo](#) | [meiSynqNetStatus](#)

# meiSynqNetNetworkObjectNext

## Declaration

```
long meiSynqNetNetworkObjectNext ( MEISynqNet          synqNet ,
                                   MEINetworkPort       port ,
                                   MEINetworkObjectInfo *info ) ;
```

**Required Header:** stdmei.h

**Change History:** Added in the 03.03.00

## Description

**meiSynqNetNetworkObjectNext** gets the information for the neighboring network object (device) connected to the specified port and writes the information into the structure pointed to by *info*.

**NOTE:** This info.type value may be MEINetworkObjectTypeNONE if there is nothing connected to the given port.

<b>synqNet</b>	a handle to a SynqNet object
<b>port</b>	specifies the node's IN or OUT port.
<b>*info</b>	a pointer to the next object's info structure.

### Return Values

[MPIMessageOK](#)

[MPIMessageARG\\_INVALID](#)

## See Also

[meiSqNodeNetworkObjectNext](#) | [MEINetworkObjectInfo](#)

[Version Utility](#)



# meiSynqNetShutdown

## Declaration

```
long  meiSynqNetShutdown ( MEISynqNet      synqNet ) ;
```

**Required Header:** stdmei.h

## Description

**meiSynqNetShutdown** disables a SynqNet network by shutting off cyclic data transmission from the controller to the nodes. This will cause the network state to transition to MEISynqNetStateDISCOVERY and nodes will go into a SynqLost state and disable their outputs.

<b>synqNet</b>	a handle to a SynqNet object
----------------	------------------------------

### Return Values

[MPIMessageOK](#)

## See Also

[meiSynqNetInit](#) | [mpiControlReset](#) | [mpiControlInit](#) | [meiSynqNetStatus](#)

# meiSynqNetEventNotifyGet

## Declaration

```
long meiSynqNetEventNotifyGet ( MEISynqNet      synqNet ,
                               MPIEventMask   *eventMask ,
                               void                *external ) ;
```

**Required Header:** stdmei.h

## Description

**meiSynqNetEventNotifyGet** reads the event mask (that specifies the event types for which host notification has been requested) to the location pointed to by `eventMask`, and also writes it into the implementation specific location pointed to by `external`. (if `external` is not NULL).

Use the event mask macros `mpiEventMaskGET(...)`, `mpiEventMaskBitGET(...)`, etc. to decode the `eventMask`.

The event notification data in `external` is in addition to the event notification data in `eventMask`. If either `eventMask` is NULL or `external` is NULL (not both), then the event notification data will not be copied to the NULL pointer.

## Remarks

**external** either points to a structure of type `MEIEventNotifyData` or is NULL.

The `MEIEventNotifyData` structure is an array of controller addresses, whose contents are placed into the `MEIEventStatusInfo` structure (of all events generated by this object).

<b>synqNet</b>	a handle to a SynqNet object
<b>*eventMask</b>	pointer to an event mask, whose bits are defined by the MPI/MEIEventType enumerations.
<b>*external</b>	pointer to external

### Return Values

[MPIMessageOK](#)

[MPIMessageARG\\_INVALID](#)

## See Also

[MEI/MPIEventType](#) | [MEIEventNotifyData](#) | [MEIEventStatusInfo](#)

# meiSynqNetEventNotifySet

## Declaration

```
long meiSynqNetEventNotifySet ( MEISynqNet      synqNet ,
                               MPIEventMask   eventMask ,
                               void                *external ) ;
```

**Required Header:** stdmei.h

## Description

**meiSynqNetEventNotifySet** requests host notification of the event(s) that are generated by **SynqNet** and specified by **eventMask**, and also specified by the implementation specific location pointed to by **external** (if external is not NULL).

Use the event mask macros `meiEventMaskSYNQNET(...)`, `mpiEventMaskSET(...)`, `mpiEventMaskBitSET(...)`, `mpiEventMaskCLEAR(...)`, etc. to create the eventMask.

The event notification data in **external** is in addition to the event notification data in **eventMask**. If either **eventMask** is NULL or **external** is NULL (not both), then the event notification data will not be copied to the NULL pointer.

## Remarks

**external** either points to a structure of type `MEIEventNotifyData` or is NULL.

The `MEIEventNotifyData` structure is an array of controller addresses, whose contents are placed into the `MEIEventStatusInfo` structure (of all events generated by this object).

<b>synqNet</b>	a handle to a SynqNet object
<b>eventMask</b>	pointer to an event mask, whose bits are defined by the MPI/MEIEventType enumerations.
<b>*external</b>	pointer to external

**Return Values**[MPIMessageOK](#)[MPIMessageARG\\_INVALID](#)**See Also**[MEI/MPIEventType](#) | [MEIEventNotifyData](#) | [MEIEventStatusInfo](#)

# meiSynqNetEventReset

## Declaration

```
long meiSynqNetEventReset( MEISynqNet      synqNet ,
                           MEIEventMask    eventMask ) ;
```

**Required Header:** stdmei.h

## Description

**meiSynqNetEventReset** resets the status/event bits that are specified in the eventMask and generated by the synqNet object. After a SynqNet event occurs, SynqNetEventReset should be called to reset the latched status/event bits so the specified event(s) can be generated again.

<b>synqNet</b>	a handle to a SynqNet object
<b>eventMask</b>	An array that defines the event mask bits. The array is defined as:  <pre>typedef MPIEventMaskELEMENT_TYPE         MPIEventMask[ MPIEventMaskELEMENTS ]</pre> <p>The bits are defined by the MPI/MEIEventType enumerations.</p>

### Return Values

[MPIMessageOK](#)

## See Also

[meiSynqNetStatus](#) | [meiSqNodeEventReset](#) | [meiSqNodeStatus](#) |  
[mpiControlEventReset](#) | [mpiMotionEventReset](#) | [mpiMotorEventReset](#) |  
[mpiRecorderEventReset](#) | [mpiSequenceEventReset](#) | [meiSqNodeEventReset](#) |  
[mpiAxisEventReset](#)

[Event Notification Methods](#)

# meiSynqNetControl

## Declaration

```
MPIControl meiSynqNetControl(MEISynqNet synqNet);
```

**Required Header:** stdmei.h

## Description

**meiSynqNetControl** returns a handle to the control object associated with the *SynqNet* object.

<b>synqNet</b>	a handle to a SynqNet object
----------------	------------------------------

### Return Values

<b>MPIControl</b>	a handle to a control object.
-------------------	-------------------------------

<b>MPIHandleVOID</b>	if synqNet is not valid.
----------------------	--------------------------

## See Also

[meiSynqNetCreate](#) | [mpiControlCreate](#)

# meiSynqNetNumber

## Declaration

```
long meiSynqNetNumber(MEISynqNet    synqNet ,
                      long          *number ) ;
```

**Required Header:** stdmei.h

## Description

**meiSynqNetNumber** reads the index of a SynqNet network and writes it into the contents of a long pointed to by number. Each SynqNet network associated with a controller is indexed by an identification number (0, 1, 2, etc.).

<b>synqNet</b>	a handle to a SynqNet object.
<b>*number</b>	a pointer to the index of a SynqNet network.

### Return Values

[MPIMessageOK](#)

## See Also

[meiSynqNetInfo](#)



# MEISynqNetCableLength

## Definition

```
typedef struct MEISynqNetCableLength { /* lengths in meters */
    long minimum;
    long nominal;
    long maximum;
} MEISynqNetCableLength;
```

## Description

**MEISynqNetCableLength** contains the nominal (average), minimum and maximum length in meters for a given network cable.

To disable the cable length checking feature, set the **minimum**, **nominal**, and **maximum** values to zero.

<b>minimum</b>	Minimum cable length in meters.  A value less than zero or greater than the nominal value is invalid.
<b>nominal</b>	Nominal cable length in meters.  A value less than zero is invalid.
<b>maximum</b>	Maximum cable length in meters.  A value less than the nominal value is invalid.

## See Also

[MEISynqNetConfig](#) | [meiSynqNetConfigSet](#) | [meiSynqNetConfigSet](#) | [meiSynqNetInfo](#)

# MEISynqNetCableList

## Definition

```
typedef struct MEISynqNetCableList {
    long    count;
    long    cableNumber[MEISynqNetCableHOP_COUNT];
} MEISynqNetCableList;
```

## Description

**MEISynqNetCableList** contains the number of cables in the SynqNet network and their identifying numbers. The idle cables can be identified with the `meiSynqNetIdleCableListGet(...)` method.

A cable can also be identified by the node number and port. Use [meiSynqNetCableNumToNodePort\(...\)](#) to translate the cable number to a node number and a port.

<b>count</b>	The number of cables and number of valid elements in the <code>cableNumber[]</code> array.. Range is 0 to <code>MEISynqNetCableHOP_COUNT</code> .
<b>cableNumber</b>	The cables identified by their number. Cables are numbered in the order they are discovered during network initialization.

## See Also

[meiSynqNetIdleCableListGet](#) | [meiSynqNetIdleCableStatus](#) | [meiSynqNetCableNumToNodePort](#)

# MEISynqNetCableStatus

## Definition

```
typedef enum MEISynqNetCableStatus {
    MEISynqNetCableStatusGOOD,
    MEISynqNetCableStatusBAD_UPSTREAM,
    MEISynqNetCableStatusBAD_DOWNSTREAM,
    MEISynqNetCableStatusBAD,
} MEISynqNetCableStatus;
```

## Description

**MEISynqNetCableStatus** is an enumeration of a network connection's operating condition. Data transmission is verified in both directions. If the network hardware does not support separate upstream and downstream data path verification, it will report either GOOD or BAD.

<b>MEISynqNetCableStatusGOOD</b>	Network communication across the cable is working properly in both directions.
<b>MEISynqNetCableStatusBAD_UPSTREAM</b>	Network communication across the cable failed in the upstream direction (from node to controller).
<b>MEISynqNetCableStatusBAD_DOWNSTREAM</b>	Network communication across the cable failed in the downstream direction (from the controller to the node).
<b>MEISynqNetCableStatusBAD</b>	Network communication across the cable failed in both directions.

## See Also

[meiSynqNetIdleCableStatus](#) | [meiSynqNetIdleCableListGet](#)

# MEISynqNetConfig

## Definition

```
typedef struct MEISynqNetConfig {
    MEISynqNetRecoveryMode    recoveryMode;
    MEISynqNetCableLength    cableLength[MEISynqNetCableHOP\_COUNT];
} MEISynqNetConfig;
```

## Description

**MEISynqNetConfig** contains configurations for the network's fault recovery mode and the network's cable lengths. The SynqNet configuration can be read with `meiSynqNetConfigGet(...)` and can be written with `meiSynqNetConfigSet(...)`.

<b>recoveryMode</b>	A enumerated value representing the network's fault recovery response mode.
<b>cableLength</b>	<p>An array of cable lengths. Range for cable lengths is 0 to 100 meters.</p> <p>This structure is provided to allow the user application to specify minimum, maximum, and nominal values to use network scheduling equations. Setting a minimum and maximum value will also enable cable length checking at network initialization time. If measured cable lengths fall outside the range defined by the minimum and maximum values, network initialization will fail with a return value of <a href="#">MEISynqNetMessageCABLE_LENGTH_MISMATCH</a>.</p> <p>By default, the values in this structure are zero. Most applications will not need to modify these defaults.</p> <p><b>NOTE:</b> The network topology must be saved before changing these values in Flash memory. These values will also be cleared when network topology is cleared using <code>meiSynqNetFlashTopologyClear(...)</code>.</p>

## See Also

[meiSynqNetConfigGet](#) | [meiSynqNetConfigSet](#) | [meiSynqNetInfo](#) |  
[meiSynqNetFlashConfigGet](#) | [meiSynqNetFlashConfigSet](#) |  
[meiSynqNetFlashTopologyClear](#)

[Cable Length](#)

# MEISynqNetFailedNodeMask

## Definition

```
#define MEISynqNetNodeMaskELEMENTS    (1)

typedef long MEISynqNetFailedNodeMask  [MEISynqNetNodeMaskELEMENTS];
```

## Description

**MEISynqNetFailedNodeMask** is an array of longs with a length of `MEISynqNetNodeMaskELEMENTS`. Each bit in the failed node mask represents a failed node (0x1 = node 0, 0x2 = node 1, 0x4 = node 2, 0x8 = node 3, etc.). A node failure occurs when the packet error rate counters exceed the packet error rate fail limit.

## See Also

[MEISynqNetStatus](#) | [meiSynqNetStatus](#) | [meiSqNodeStatus](#) | [MEISqNodeConfig](#)

# MEISynqNetInfo

## Definition

```
typedef struct MEISynqNetInfo {
    MEINetworkType      networkType;
    long                 nodeCount;
    long                 nodeOffset;
    MEISynqNetCableLength cableLength[MEISynqNetCableHOP\_COUNT];
                                     /* measured length */
} MEISynqNetInfo;
```

## Description

**MEISynqNetInfo** contains static data that is determined during network initialization. It identifies the network type, number of nodes on the network, starting number (offset) of the first node, and an rough estimate of measured cable lengths.

<b>networkType</b>	contains currently discovered network topology type.
<b>nodeCount</b>	Contains the number of nodes currently discovered on the network.
<b>nodeOffset</b>	Starting node number for nodes on this network. Nodes are currently numbered sequentially across all networks on a controller.
<b>cableLength</b>	<p>An array of measured network cable lengths. These values are estimated values.</p> <p>At network discovery, where network topology has not been saved to flash, the controller will send network packets to measure each cable length and automatically fill out these values for each cable found.</p> <p>These values are estimated values and are the values used in the network scheduling calculations.</p>

## Sample Code

```
/*  
    Get the number of nodes on the SynqNet network  
    Returns -1 if there is an error  
*/  
int NumberOfNodes(MEISynqNet synqNet)  
{  
    MEISynqNetInfo info;  
    long returnValue;  
    int numNodes = -1;  
  
    returnValue = meiSynqNetInfo(synqNet, &info);  
  
    if(returnValue == MPIMessageOK)  
    {  
        if(info.nodeCount >= 0 && info.nodeCount <= MEIXmpMaxSynqNetBlocks)  
        {  
            numNodes = info.nodeCount;  
        }  
    }  
  
    return numNodes;  
}
```

## See Also

[meiSynqNetInfo](#) | [MEISynqNetConfig](#)

# MEISynqNetMessage

## Definition

```
typedef enum {
    MEISynqNetMessageSYNQNET_INVALID,
    MEISynqNetMessageMAX_NODE_ERROR,
    MEISynqNetMessageSTATE_ERROR,

    MEISynqNetMessageCOMM_ERROR,
    MEISynqNetMessageCOMM_ERROR_CRC,
    MEISynqNetMessageCOMM_ERROR_RX,
    MEISynqNetMessageCOMM_ERROR_RX_LEN,
    MEISynqNetMessageCOMM_ERROR_RX_FIFO,
    MEISynqNetMessageCOMM_ERROR_RX_DRIBBLE,
    MEISynqNetMessageCOMM_ERROR_RX_CRC,

    MEISynqNetMessageINTERFACE_NOT_FOUND,
    MEISynqNetMessageTOPOLOGY_MISMATCH,
    MEISynqNetMessageTOPOLOGY_MISMATCH_FLASH,

    MEISynqNetMessageRESET_REQ_TIMEOUT,
    MEISynqNetMessageRESET_ACK_TIMEOUT,
    MEISynqNetMessageDISCOVERY_TIMEOUT,
    MEISynqNetMessageNO_NODES_FOUND,
    MEISynqNetMessageNO_TIMING_DATA_AVAIL,

    MEISynqNetMessageINTERNAL_BUFFER_OVERFLOW,
    MEISynqNetMessageINVALID_MOTOR_COUNT,
    MEISynqNetMessageINVALID_AUX_ENC_COUNT,
    MEISynqNetMessageINCOMPLETE_MOTOR,
    MEISynqNetMessageINVALID_COMMAND_CFG,
    MEISynqNetMessageINVALID_PULSE_ENGINE_COUNT,
    MEISynqNetMessageINVALID_ENCODER_COUNT,
    MEISynqNetMessageINVALID_CAPTURE_COUNT,
    MEISynqNetMessageINVALID_COMPARE_COUNT,
    MEISynqNetMessageINVALID_INPUT_COUNT,
    MEISynqNetMessageINVALID_OUTPUT_COUNT,
    MEISynqNetMessageINVALID_MONITOR_CFG,
    MEISynqNetMessageINVALID_ANALOG_IN_COUNT,
    MEISynqNetMessageINVALID_DIGITAL_IN_COUNT,
    MEISynqNetMessageINVALID_DIGITAL_OUT_COUNT,
    MEISynqNetMessageINVALID_ANALOG_OUT_COUNT

    MEISynqNetMessageLINK_NOT_IDLE,
    MEISynqNetMessageIDLE_LINK_UNKNOWN,
    MEISynqNetMessageRING_ONLY,
    MEISynqNetMessageRECOVERING,

    MEISynqNetMessageCABLE_LENGTH_UNSUPPORTED,
```



```

MEISynqNetMessageCABLE_LENGTH_MISMATCH,
MEISynqNetMessageCABLE_LENGTH_INVALID_NOMINAL,
MEISynqNetMessageCABLE_LENGTH_INVALID_MIN,
MEISynqNetMessageCABLE_LENGTH_INVALID_MAX,

MEISynqNetMessageNODE_FPGA_VERSION,
MEISynqNetMessageMAX_MOTOR_ERROR,
MEISynqNetMessagePLL_ERROR,
MEISynqNetMessageNODE_INIT_FAIL,

MEISynqNetMessageTOPOLOGY_CLEAR,
MEISynqNetMessageTOPOLOGY_SAVED,
MEISynqNetMessageTOPOLOGY_AMPS_ENABLED,

MEISynqNetMessageNODE_MAC_VERSION,
MEISynqNetMessageADC_SAMPLE_FAILURE,

MEISynqNetMessageSCHEDULING_ERROR,

MEISynqNetMessageINVALID_PROBE_CFG,
MEISynqNetMessageINVALID_PROBE_DEPTH,
} MEISynqNetMessage;

```

**Required Header:** stdmei.h

**Change History:** Modified in the 03.03.00. Modified in the 03.02.00.

## Description

**MEISynqNetMessage** is an enumeration of SynqNet error messages that can be returned by the MPI library.

### MEISynqNetMessageSYNQNET\_INVALID

The SynqNet number is out of range. This message code is returned by [meiSynqNetCreate\(...\)](#) if the SynqNet network number is less than zero or greater than or equal to MEIXmpMaxSynqNets.

### MEISynqNetMessageMAX\_NODE\_ERROR

The SynqNet node number is out of range. This message code is returned by a SynqNet method if the specified node number is greater than or equal to MEIXmpMaxSynqNetBlocks.

### MEISynqNetMessageSTATE\_ERROR

The SynqNet network state is not valid. This message code is returned by any method that initializes a SynqNet network if the controller's network state is not a member of the MEIXmpSynqNetState or MEIXmpSynqNetInternalState enumeration. The most commonly used methods that initialize the SynqNet network are: [mpiControlInit\(...\)](#), [mpiControlReset\(...\)](#) and [meiSynqNetInit\(...\)](#). This message code indicates a failure in the controller's initialization sequence. To correct this problem, call [mpiControlReset\(...\)](#).

### MEISynqNetMessageCOMM\_ERROR

The SynqNet network communication failed. This message code is returned by MPI methods that fail a service command transaction due to a network shutdown. This message indicates a node or network cable failure. To correct this problem, check your network wiring and node condition.

#### **MEISynqNetMessageCOMM\_ERROR\_CRC**

The SynqNet network communication failed due to excessive CRC errors. This message code is returned by MPI methods that fail a service command transaction due to a network shutdown. This message indicates a node or network cable failure. To correct this problem, check your network wiring and node condition.

#### **MEISynqNetMessageCOMM\_ERROR\_RX**

The SynqNet network communication failed due to a Rincon receive error. This message code is returned by MPI methods that fail a service command transaction due to a network shutdown. This message indicates a node or network cable failure. To correct this problem, check your network wiring and node condition.

#### **MEISynqNetMessageCOMM\_ERROR\_RX\_LEN**

The SynqNet network communication failed due to a Rincon receive length error. This message code is returned by MPI methods that fail a service command transaction due to a network shutdown. This message indicates a node or network cable failure. To correct this problem, check your network wiring and node condition.

#### **MEISynqNetMessageCOMM\_ERROR\_RX\_FIFO**

The SynqNet network communication failed due to a Rincon receive buffer error. This message code is returned by MPI methods that fail a service command transaction due to a network shutdown. This message indicates a node or network cable failure. To correct this problem, check your network wiring and node condition.

#### **MEISynqNetMessageCOMM\_ERROR\_RX\_DRIBBLE**

The SynqNet network communication failed due to a Rincon receive dribble error. This message code is returned by MPI methods that fail a service command transaction due to a network shutdown. This message indicates a node or network cable failure. To correct this problem, check your network wiring and node condition.

#### **MEISynqNetMessageCOMM\_ERROR\_RX\_CRC**

The SynqNet network communication failed due to a Rincon receive CRC error. This message code is returned by MPI methods that fail a service command transaction due to a network shutdown. This message indicates a node or network cable failure. To correct this problem, check your network wiring and node condition.

#### **MEISynqNetMessageINTERFACE\_NOT\_FOUND**

The controller does not support a SynqNet interface. This message code is returned by [meiSynqNetValidate\(...\)](#), [meiSynqNetFlashTopologySave\(...\)](#), [meiSynqNetFlashTopologyClear\(...\)](#), [mpiControlInit\(...\)](#), and [mpiControlReset\(...\)](#) if the controller does not have a SynqNet hardware interface. To correct this problem, use a controller that supports SynqNet.

#### **MEISynqNetMessageTOPOLOGY\_MISMATCH**

The network topology does not match the expected network topology. This message code is returned by [mpiControlInit\(...\)](#) or [meiSynqNetInit\(...\)](#) if the discovered network topology does not match the controller's expected network topology (stored in dynamic memory). During the first network initialization the controller stores node identification information (manufacturer, product, and unique values) into its dynamic memory. This message code indicates the number of nodes, the node order, or types of nodes have changed since the initial network initialization. To correct this problem, either change the network topology to the original configuration or clear the controller's memory with [mpiControlReset\(...\)](#).

#### **MEISynqNetMessageTOPOLOGY\_MISMATCH\_FLASH**

The network topology does not match the expected network topology. This message code is returned by [mpiControlInit\(...\)](#) or [meiSynqNetInit\(...\)](#) if the discovered network topology does not match the controller's expected network topology (stored in flash memory). During the first network initialization the controller stores node identification information (manufacturer, product, and unique values) into its dynamic memory. Later, when [meiSynqNetFlashTopologySave\(...\)](#) is called, the topology information is stored into flash memory. This message indicates the number of nodes, the node order, or types of nodes have changed since the topology information was stored in flash. To correct this problem, either change the network topology to the saved configuration or clear the controller's flash topology with [meiSynqNetFlashTopologyClear\(...\)](#).

#### **MEISynqNetMessageRESET\_REQ\_TIMEOUT**

The network reset request packet exceeded the timeout. This message code is returned by [mpiControlInit\(...\)](#), [mpiControlReset\(...\)](#), or [meiSynqNetInit\(...\)](#) if the reset request packet fails to traverse the network in the allotted time. This message code indicates a node or network cable failure. To correct this problem, check your network wiring and node condition.

#### **MEISynqNetMessageRESET\_ACK\_TIMEOUT**

The network reset complete packet exceeded the timeout. This message code is returned by [mpiControlInit\(...\)](#), [mpiControlInit\(...\)](#), or [meiSynqNetInit\(...\)](#) if the reset complete packet fails to traverse the network in the allotted time. This message code indicates a node or network cable failure. To correct this problem, check your network wiring and node condition.

#### **MEISynqNetMessageDISCOVERY\_TIMEOUT**

The network topology discovery exceeded the timeout. This message code is returned by [mpiControlInit\(...\)](#), [mpiControlInit\(...\)](#), or [meiSynqNetInit\(...\)](#) if the controller failed to discover the network topology in the allotted time. This message code indicates a node or network cable failure. To correct this problem, check your network wiring and node condition.

#### **MEISynqNetMessageNO\_NODES\_FOUND**

The controller did not find network nodes. This message code is returned by [mpiControlInit\(...\)](#), [meiSynqNetPacketConfigSet\(...\)](#), or [meiSynqNetInit\(...\)](#) if the controller failed to discover any nodes during network initialization. This message code indicates the first node has failed or the network connection from the controller to the first node is faulty. To correct this problem, check your network wiring and node condition.

#### **MEISynqNetMessageNO\_TIMING\_DATA\_AVAIL**

The corresponding SynqNet node module does not contain timing data, so the network cannot be initialized. Contact MEI or drive manufacturer for an updated node module.

#### **MEISynqNetMessageINTERNAL\_BUFFER\_OVERFLOW**

The controller's SynqNet buffer size was exceeded. This message code is returned by [mpiControlInit\(...\)](#) or [mpiControlReset\(...\)](#) if the controller's SynqNet buffer could not be allocated due to overflow. This message code indicates the controller does not have enough memory to initialize the network topology. To correct the problem, either reduce the network nodes or use a different controller model.

#### **MEISynqNetMessageINVALID\_MOTOR\_COUNT**

This message is returned by [meiSynqNetPacketConfigSet\(...\)](#), when the packet configuration contains more motors than supported by the node.

#### **MEISynqNetMessageINVALID\_AUX\_ENC\_COUNT**

This message is returned by [meiSynqNetPacketConfigSet\(...\)](#), when the packet configuration contains more secondary encoders than supported by the node.

#### **MEISynqNetMessageINCOMPLETE\_MOTOR**

This message is returned by [meiSynqNetPacketConfigSet\(...\)](#), when the packet configuration contains a motor configuration that is missing feedback or command fields.

#### **MEISynqNetMessageINVALID\_COMMAND\_CFG**

This message is returned by [meiSynqNetPacketConfigSet\(...\)](#), when the packet configuration contains a different command configuration than is supported by the node.

#### **MEISynqNetMessageINVALID\_PULSE\_ENGINE\_COUNT**

#### **MEISynqNetMessageINVALID\_ENCODER\_COUNT**

This message is returned by [meiSynqNetPacketConfigSet\(...\)](#), when the packet configuration contains more encoders for feedback than are supported by the node.

#### **MEISynqNetMessageINVALID\_CAPTURE\_COUNT**

This message is returned by [meiSynqNetPacketConfigSet\(...\)](#), when the packet configuration contains a larger capture count than is supported by the node.

#### **MEISynqNetMessageINVALID\_COMPARE\_COUNT**

This message is returned by [meiSynqNetPacketConfigSet\(...\)](#), when the packet configuration contains a larger compare count than is supported by the node.

#### **MEISynqNetMessageINVALID\_INPUT\_COUNT**

This message is returned by [meiSynqNetPacketConfigSet\(...\)](#), when the packet configuration contains a larger ioInput count than is supported by the node.

#### **MEISynqNetMessageINVALID\_OUTPUT\_COUNT**

This message is returned by [meiSynqNetPacketConfigSet\(...\)](#), when the packet configuration contains a larger ioOutput count than is supported by the node.

#### **MEISynqNetMessageINVALID\_MONITOR\_CFG**

This message is returned by [meiSynqNetPacketConfigSet\(...\)](#), when the packet configuration contains more monitor fields than the node can support.

#### **MEISynqNetMessageINVALID\_ANALOG\_IN\_COUNT**

This message is returned by [meiSynqNetPacketConfigSet\(...\)](#), when the packet configuration contains a larger analogIn count than is supported by the node.

#### **MEISynqNetMessageINVALID\_DIGITAL\_IN\_COUNT**

This message is returned by [meiSynqNetPacketConfigSet\(...\)](#), when the packet configuration contains a larger digitalIn count than is supported by the node.

#### **MEISynqNetMessageINVALID\_DIGITAL\_OUT\_COUNT**

This message is returned by [meiSynqNetPacketConfigSet\(...\)](#), when the packet configuration contains a larger digitalOut count than is supported by the node.

#### **MEISynqNetMessageINVALID\_ANALOG\_OUT\_COUNT**

This message is returned by [meiSynqNetPacketConfigSet\(...\)](#), when the packet configuration contains a larger analogOut count than is supported by the node.

#### **MEISynqNetMessageLINK\_NOT\_IDLE**

This message is returned by [meiSynqNetIdleCableStatus\(...\)](#) when the cable number supplied is not the idle link. The status check can only be run on an idle cable. See [meiSynqNetIdleCableListGet\(...\)](#).

#### **MEISynqNetMessageIDLE\_LINK\_UNKNOWN**

This message is returned by [meiSynqNetIdleCableListGet\(...\)](#) when an idle cable number in a ring topology cannot be determined. This is due to one or more failed nodes on a ring topology. Be sure to check the status of the nodes.

#### **MEISynqNetMessageRING\_ONLY**

This message is returned by [meiSynqNetIdleCableListGet\(...\)](#) and [meiSynqNetIdleCableStatus\(...\)](#) because idle cables exist on ring topologies only. It is also returned by [meiSynqNetConfigSet\(...\)](#) when attempting to enable SynqNet recovery mode, which is only supported for ring topologies.

**MEISynqNetMessageRECOVERING**

The network is recovering from a fault condition. This message is returned by [meiSynqNetIdleCableStatus\(...\)](#) during network fault recovery, because the network traffic is being redirected around the faulted cable and a new idle cable is in the process of being assigned. If you receive this message, wait for recovery to complete and then check the identity of the idle cable with [meiSynqNetIdleCableListGet\(...\)](#).

**MEISynqNetMessageCABLE\_LENGTH\_UNSUPPORTED**

This message is returned when a cable on the network is too long.

**MEISynqNetMessageCABLE\_LENGTH\_MISMATCH**

This message is returned at network initialization when topology has been saved to flash and a cable length is detected that lies outside the `MEISynqNetConfig.cableLength[n].minimum` and `maximum`. Check cable length (if necessary) and save new cable length values to flash.

**MEISynqNetMessageCABLE\_LENGTH\_INVALID\_NOMINAL**

The nominal cable length is too long.

**MEISynqNetMessageCABLE\_LENGTH\_INVALID\_MIN**

The minimum cable length is too long or exceeds nominal value.

**MEISynqNetMessageCABLE\_LENGTH\_INVALID\_MAX**

The maximum cable length is too long or is less than nominal value.

**MEISynqNetMessageNODE\_FPGA\_VERSION**

The node resource FPGA image is out of date. This is only a warning message. Your network will still reach cyclic (SYNQ) mode, but certain node resources may not be available or may not operate as expected. Only ignore this warning if you are aware of FPGA changes between this version and the version built with the MPI or use the `sqNodeFlash.exe` utility to load the most current FPGA version to the node.

**MEISynqNetMessageMAX\_MOTOR\_ERROR**

The number of motors on the network exceeds the number set by [MEISynqNetMaxMOTORS](#).

**MEISynqNetMessagePLL\_ERROR**

The node PLL is unable to lock with drive.

**MEISynqNetMessageNODE\_INIT\_FAIL**

A node specific initialization routine was unable to complete successfully. Verify node FPGA is the default version for your MPI version. See [MEISqNodeInfoFpga.defaultVersion](#).

**MEISynqNetMessageTOPOLOGY\_CLEAR**

An attempt to clear the saved network topology was made when no network topology has been saved.

### MEISynqNetMessageTOPOLOGY\_SAVED

An attempt to save network topology was made when the network topology has already been saved. Clear the network topology before attempting to save another topology to flash.

### MEISynqNetMessageTOPOLOGY\_AMPS\_ENABLED

An [meiSynqNetFlashTopologySave /Clear](#) routine has been called while one or more motor amplifiers are enabled. Disables all motor amplifiers ([mpiMotorAmpEnableSet\(...\)](#)) before calling these low-level routines. To avoid this error message, disable the motor(s) amp enable before calling [meiSynqNetFlashTopologySave/Clear](#).

### MEISynqNetMessageNODE\_MAC\_VERSION

The node MAC FPGA image is out of date. This is an error message. Your network will probably never reach cyclic (SYNQ) mode. Use the [sqNodeFlash.exe](#) utility to load the most current FPGA version to the node.

### MEISynqNetMessageADC\_SAMPLE\_FAILURE

A check to verify that all analog inputs can be sampled during the given controller sample rate has failed. Either reduce the number of analog inputs on your network (see [meiSynqNetPacketConfigGet / meiSynqNetPacketConfigSet](#)) or decrease your controller sample rate (see [mpiControlConfigGet / mpiControlConfigSet](#)).

### MEISynqNetMessageSCHEDULING\_ERROR

This is a generic return value to indicate that a problem has occurred while calculating the network scheduling. For more specific information about the error, run your application with timing assignment tracing turned on, using the trace mask:

```
0x00100000    SynqNet: Display timing assignments
```

### MEISynqNetMessageINVALID\_PROBE\_CFG

This message is returned by [meiSynqNetPacketConfigSet\(...\)](#), when the [MEISynqNetPacketCfgProbe](#).count contains a larger count than is supported by the node.

### MEISynqNetMessageINVALID\_PROBE\_DEPTH

This message is returned by [meiSynqNetPacketConfigSet\(...\)](#), when the [MEISynqNetPacketCfgProbe](#).depth contains a larger count than is supported by the node.

## See Also

# MEISynqNetResourceCommand

## Definition

```
typedef enum MEISynqNetResourceCommand {
    MEISynqNetResourceCommandINVALID,
    MEISynqNetResourceCommandNONE,
    MEISynqNetResourceCommandDAC_ONE,
    MEISynqNetResourceCommandDAC_TWO,
    MEISynqNetResourceCommandDEMAND_A,
    MEISynqNetResourceCommandDEMAND_AB,
    MEISynqNetResourceCommandDEMAND_ABC,
} MEISynqNetResourceCommand;
```

**Change History:** Modified in the 03.03.00

## Description

**MEISynqNetResourceCommand** enumeration is a member of the **MEISynqNetPacketCfgMotor** configuration structure. The values are used to configure the number of 32-bit command data fields sent to drive a motor. Valid values range from greater or equal to **MEISynqNetResourceCommandFIRST** and less than **MEISynqNetResourceCommandLast**, but may be further limited by the available resources on the node.

**NOTE:** A motor is considered incomplete if it has a command value of **NONE**.

<b>MEISynqNetResourceCommandINVALID</b>	Non-vaild command value was detected.
<b>MEISynqNetResourceCommandNONE</b>	No command data will be sent to this motor. NOTE: this disables the motor and will require the application to remove all other resources for this motor.
<b>MEISynqNetResourceCommandDAC_ONE</b>	Configures a 32bit data field of DAC data to besent to the motor.
<b>MEISynqNetResourceCommandDAC_TWO</b>	Configures two 32bit data fields of DAC data to besent to the motor.
<b>MEISynqNetResourceCommandDEMAND_A</b>	Configures one 16bit data field of Demand data to be sent to the motor.
<b>MEISynqNetResourceCommandDEMAND_AB</b>	Configures two 16bit data fields of Demand data to be sent to the motor.



**MEISynqNetResourceCommandDEMAND\_ABC**

Configures three 16bit data fields of Demand data to be sent to the motor.

## See Also

[meiSynqNetPacketConfigGet](#) | [meiSynqNetPacketConfigSet](#) |  
[MEISynqNetPacketCfgMotor](#) | [MEISynqNetPacketCfg](#) | [MEISynqNetPacketCfgNode](#)

# MEISynqNetResourceIoBits

## Definition

```
typedef enum MEISynqNetResourceIoBits {
    MEISynqNetResourceIoBitsINVALID,
    MEISynqNetResourceIoBitsNONE,
    MEISynqNetResourceIoBits16,
    MEISynqNetResourceIoBits48,
} MEISynqNetResourceIoBits;
```

## Description

The **MEISynqNetResourceIoBits** enumeration is a member of the **MEISynqNetPacketCfgMotor** configuration structure. The values are used to configure the number of I/O data fields sent to a motor. Valid values range from greater or equal to **MEISynqNetResourceIoBitsFIRST** and less than **MEISynqNetResourceIoBitsLAST**, but may be further limited by the available resources on the node. This is a generic enumeration that is used to configure the resource count of different types of motor and node I/O.

**NOTE:** (for Motor I/O only) Enabled motors always contain one fixed 16bit data field of dedicated I/O. This is not application configurable.

<b>MEISynqNetResourceIoBitsINVALID</b>	Non-valid bit count was detected.
<b>MEISynqNetResourceIoBitsNONE</b>	No I/O bits will be sent/received.
<b>MEISynqNetResourceIoBits16</b>	One data field of 16 I/O bits will be sent/received.
<b>MEISynqNetResourceIoBits48</b>	One data field of 48 I/O bits will be sent/received.

## See Also

[meiSynqNetPacketConfigGet](#) | [meiSynqNetPacketConfigSet](#) | [MEISynqNetPacketCfgMotor](#) | [MEISynqNetPacketCfg](#) | [MEISynqNetPacketCfgNode](#)

# MEISynqNetResourceMonitor

## Definition

```
typedef enum MEISynqNetResourceMonitor {
    MEISynqNetResourceMonitorINVALID,
    MEISynqNetResourceMonitorNONE,
    MEISynqNetResourceMonitorAB,
    MEISynqNetResourceMonitorABCD,
} MEISynqNetResourceMonitor;
```

## Description

The **MEISynqNetResourceMonitor** enumeration is a member of the **MEISynqNetPacketCfgMotor** configuration structure. The values are used to configure the number of Monitor data fields received from a motor. Valid values range from greater or equal to **MEISynqNetResourceMonitorFIRST** and less than **MEISynqNetResourceMonitorLAST**, but may be further limited by the available resources on the node.

<b>MEISynqNetResourceMonitorINVALID</b>	Non-valid bit count was detected.
<b>MEISynqNetResourceMonitorNONE</b>	No monitor data fields will be sent/received for this motor.
<b>MEISynqNetResourceMonitorAB</b>	Two 16bit fields of monitor data will be sent/received for this motor.
<b>MEISynqNetResourceMonitorABCD</b>	Four 16bit fields of monitor data will be sent/received for this motor.

## See Also

# MEISynqNetPacketCfg

## Definition

```
typedef struct MEISynqNetPacketCfg {  
    MEISynqNetPacketCfgNode node[MEISynqNetMaxNODE_COUNT];  
} MEISynqNetPacketCfg;
```

## Description

**MEISynqNetPacketCfg** structure is used as a parameter to the `meiSynqNetPacketConfigGet/Set` methods. It contains the network packet configuration for all nodes found on the network. Only configurable packet data is represented in this structure.

**NOTE:** Fixed packet fields are NOT application configurable.

### **node**

is an array of configurable packet structures for node data. Array indices 0 through `MEISynqNetInfo.nodeCount` are valid, with a maximum number of motors per node being defined by `MEISynqNetMaxNODE_COUNT`.

## See Also

[meiSynqNetPacketConfigSet](#) | [meiSynqNetPacketConfigGet](#) |  
[MEISynqNetPacketCfgNode](#) | [MEISynqNetPacketCfgMotor](#) |  
[MEISynqNetPacketCfgEncoder](#)

# MEISynqNetPacketCfgEncoder

## Definition

```
typedef struct MEISynqNetPacketCfgEncoder {
    long feedbackCount;    /* per encoder */
    long captureCount;    /* per encoder */
    long compareCount;    /* per encoder */
} MEISynqNetPacketCfgEncoder;
```

## Description

**MEISynqNetPacketCfgEncoder** structure is a member of the **MEISynqNetPacketCfgMotor** configuration structure. It contains the network packet configuration for a single encoder found on a particular motor. Only configurable packet data is represented in this structure. Fixed packet fields are not application configurable.

Setting all structure members to zero (0x0) values will effectively disable the encoder on the motor. Disabling an encoder will NOT affect encoder numbering, however, encoders with higher index numbers must be disabled before using encoders with lower index numbers.

**NOTE:** A motor is considered incomplete if it has no encoders enabled.

<b>feedbackCount</b>	<p>configures the number of 32-bit feedback data fields to be sent for an encoder. Valid numbers are zero thru <b>MEISynqNetMaxEncoderFEEDBACK_COUNT</b> , but may be further limited by the available resources on the node.</p> <p><b>NOTE:</b> Setting this count to a value of zero (0x0) will affectively disable the encoder. All other encoder resources must also be set to NONE or zero (0x0).</p>
<b>captureCount</b>	<p>configures the number of 32-bit capture data fields to be sent for an encoder. Valid numbers are zero thru <b>MEISynqNetMaxMotorCAPTURE_COUNT</b>, but may be further limited by the available resources on the node. If <b>feedbackCount</b> is zero (0x0), then this value must also be zero.</p> <p><b>NOTE:</b> all capture/compare resources on a motor rely on 2 fixed 32bit data fields for command and status information. These fixed fields are currently not configurable.</p>

**compareCount**

configures the number of 32-bit compare data fields to be sent for an encoder. Valid numbers are zero thru MEISynqNetMaxMotorCOMPARE\_COUNT, but may be further limited by the available resources on the node.

If feedbackCount is zero (0x0), then this value must also be zero.

**NOTE:** all capture/compare resources on a motor rely on 2 fixed 32bit data fields for command and status information. These fixed fields are currently not configurable.

## See Also

[meiSynqNetPacketConfigGet](#) | [meiSynqNetPacketConfigSet](#) |  
[MEISynqNetPacketCfgMotor](#) | [MEISynqNetPacketCfg](#) | [MEISynqNetPacketCfgNode](#)

# MEISynqNetPacketCfgIo

## Definition

```
typedef struct MEISynqNetPacketCfgIo {
    long    digitalInCount ;
    long    digitalOutCount ;
    long    analogInCount ;
    long    analogOutCount ;
} MEISynqNetPacketCfgIo ;
```

## Description

**MEISynqNetPacketCfgIo** structure configures the network data packets that contain general purpose I/O.

<b>digitalInCount</b>	Selects the number of 32bit words of digital input data to receive over the SynqNet network.
<b>digitalOutCount</b>	Selects the number of 32bit words of digital output data to send over the SynqNet network.
<b>analogInCount</b>	Selects the number of 32bit words of analog input data to receive over the SynqNet network.
<b>analogOutCount</b>	Selects the number of 32bit words of analog output data to send over the SynqNet network.

## See Also

[meiSynqNetPacketConfigSet](#) | [meiSynqNetPacketConfigGet](#) | [MEISynqNetPacketCfgNode](#)

# MEISynqNetPacketCfgMotor

## Definition

```
typedef struct MEISynqNetPacketCfgMotor {
    MEISynqNetResourceCommand    command;
    long                          pulseEngineCount;
    long                          feedbackPrimaryCount;
    long                          captureCount;
    long                          compareCount;
    MEISynqNetResourceIoBits    ioInput;
    MEISynqNetResourceIoBits    ioOutput;
    MEISynqNetResourceMonitor    monitor;
    MEISynqNetPacketCfgProbe    probe;
} MEISynqNetPacketCfgMotor;
```

## Description

**MEISynqNetPacketCfgMotor** structure is a member of the **MEISynqNetPacketCfgNode** configuration structure. It contains the network packet configuration for a single motor found on a particular node. Only configurable packet data is represented in this structure. Fixed packet fields are not application configurable.

**NOTE:** Setting all structure members to a NONE or Zero (0x0) value will effectively disable the motor on the network. This will cause the controller to renumber the subsequent motors on the network.

<b>command</b>	Selects the command data type and count. Please see <a href="#">MEISynqNetResourceCommand</a> .
<b>pulseEngineCount</b>	Configures the number of pulse engine to be enable for a motor. Valid numbers are zero thru <b>MEISynqNetMaxMotorPULSE_ENGINE_COUNT</b> , but may be further limited by the available resources on the node.  Each pulse engine requires a 64-bit upstream data field and a 32-bit downstream data field.
<b>feedbackPrimaryCount</b>	Configures the number of 32-bit feedback data fields to be sent for a motor. Valid numbers are zero thru <b>MEISynqNetMaxEncoderFEEDBACK_COUNT</b> , but may be further limited by the available resources on the node.



<b>captureCount</b>	<p>Configures the number of 32-bit capture data fields to be sent for a motor. Valid numbers are zero thru MEISynqNetMaxMotorCAPTURE_COUNT, but may be further limited by the available resources on the node. If feedbackCount is zero (0x0), then this value must also be zero.</p> <p><b>NOTE:</b> All capture/compare resources on a motor rely on two fixed 32bit data fields for command and status information. These fixed fields are currently not configurable.</p>
<b>compareCount</b>	<p>Configures the number of 32-bit compare data fields to be sent for a motor. Valid numbers are zero thru MEISynqNetMaxMotorCOMPARE_COUNT, but may be further limited by the available resources on the node.</p> <p><b>NOTE:</b> all capture/compare resources on a motor rely on two fixed 32bit data fields for command and status information. These fixed fields are currently not configurable.</p>
<b>ioInput</b>	<p>Selects the number of input bit data received from this motor. Please see <a href="#">MEISynqNetResourceIoBits</a>.</p>
<b>ioOutput</b>	<p>Selects the number of output bit data sent to this motor. Please see <a href="#">MEISynqNetResourceIoBits</a>.</p>
<b>monitor</b>	<p>Selects the number of montior data fields received from this motor. Please see <a href="#">MEISynqNetResourceMonitor</a>.</p>
<b>probe</b>	<p>A structure that specifies the probe count and register depth.</p>

## See Also

[meiSynqNetPacketConfigGet](#) | [meiSynqNetPacketConfigSet](#) | [MEISynqNetPacketCfg](#)  
[MEISynqNetPacketCfgNode](#) | [MEISynqNetResourceCommand](#) |  
[MEISynqNetResourceIoBits](#) | [MEISynqNetResourceMonitor](#) |  
[MEISynqNetPacketCfgProbe](#)

# MEISynqNetPacketCfgNode

## Definition

```
typedef struct MEISynqNetPacketCfgNode {
    MEISynqNetPacketCfgMotor    motor [ MEISynqNetMaxNodeMOTORS ]
    MEISynqNetPacketCfgIo      io;
    long                        feedbackSecondaryCount;
} MEISynqNetPacketCfgNode;
```

## Description

**MEISynqNetPacketCfgNode** is a member of the **MEISynqNetPacketCfg** configuration structure. It contains the network packet configuration for all motors found on a particular node. Only configurable packet data is represented in this structure. Fixed packet fields are not application configurable.

<b>motor</b>	An array of configurable packet structures for motor data. Array indices 0 through <b>MEISqNodeInfo.motorCount</b> are valid, with a maximum number of motors per node being defined by <a href="#">MEISynqNetMaxNodeMOTORS</a> .
<b>io</b>	Configures the network data packets general purpose node I/O.
<b>feedbackSecondaryCount</b>	Configures the number of 32-bit secondary feedback data fields to be sent for a node. Valid numbers are zero thru <a href="#">MEISynqNetMaxMotorFEEDBACK_SECONDARY_COUNT</a> , but may be further limited by the available resources on the node.

## See Also

[meiSynqNetPacketConfigGet](#) | [meiSynqNetPacketConfigSet](#) | [MEISynqNetPacketCfg](#)

# MEISynqNetPacketCfgProbe

## Definition

```
typedef struct MEISynqNetPacketCfgProbe {
    long count;
    MEISynqNetResourceProbeDepth depth[MEISynqNetMaxMotorPROBE\_COUNT];
} MEISynqNetPacketCfgProbe;
```

## Description

**MEISynqNetPacketCfgProbe** specifies the network packet configuration for all the Probes found on a particular motor. Only configurable packet data is represented in this structure. Fixed packet fields are not application configurable.

<b>count</b>	the number of Probe engines per motor. Each Probe engine requires cyclic packet data for the status fields.
<b>depth</b>	an array of enumerated values representing the Probe data depth. Each probe engine can support up to 16 register fields for the probe data. The length of the array is specified by the Probe count.

## See Also

[MEISynqNetPacketCfgMotor](#) | [MEISynqNetResourceProbeDepth](#) | [MPIProbeStatus](#)

# MEISynqNetRecoveryMode

## Definition

```
typedef enum MEISynqNetRecoveryMode {  
    MEISynqNetRecoveryModeDISABLED,  
    MEISynqNetRecoveryModeSINGLE_SHOT,  
    MEISynqNetRecoveryModeAUTO_ARM,  
} MEISynqNetRecoveryMode;
```

## Description

**MEISynqNetRecoveryMode** is an enumeration of a network's fault recovery mode. Networks with ring topologies can be configured to recover from a fault condition. A fault condition occurs when a packet error rate counter exceeds its packet error rate fault limit. If fault recovery is enabled and a fault occurs, the node hardware and/or controller will detect the location of the fault and switch the direction of network traffic for nodes downstream from the faulted connection. During fault recovery, the network state is `MEISynqNetStateSYNQ_RECOVERING`. After the upstream and downstream packet error rate counters decrement to zero, the recovery is completed, and the network state returns to `MEISynqNetStateSYNQ`.

When fault recovery occurs, the controller will generate a `MEIEventTypeSYNQNET_RECOVERY` status/event. An application should notify the user about the fault and fix the broken cable/hardware as soon as possible. An application can determine the location of the fault using `meiSynqNetIdleCableListGet(...)`. A network with a ring topology has one idle cable, which has no data traffic. The operation of the idle cable can be tested with `meiSynqNetIdleCableStatus(...)`.

**WARNING:** If the idle cable is broken and a second fault occurs, then fault recovery will not be able to fully recover from the fault. SynqNet will try to recover, but some nodes will be stranded. Presently, SynqNet does not support an event to notify the application if an idle cable fails. In the meantime, an application can periodically poll the idle cable with `meiSynqNetIdleCableStatus(...)` to test the cable.

<b>MEISynqNetRecoveryModeDISABLED</b>	The network will not attempt to recover from a fault condition. This is the default mode for string topologies.
<b>MEISynqNetRecoveryModeSINGLE_SHOT</b>	The network will only attempt to recover from a fault condition one time. A second fault will be ignored.
<b>MEISynqNetRecoveryModeAUTO_ARM</b>	The network will attempt to recover from a fault condition. After the fault recovery is complete, the network will automatically be re-armed to respond to another fault condition. This is the default mode for ring topologies.

## See Also

[meiSynqNetConfigGet](#) | [meiSynqNetConfigSet](#) | [meiSynqNetStatus](#) | [meiSynqNetInfo](#)  
| [meiSynqNetIdleCableListGet](#) | [meiSynqNetIdleCableStatus](#)

# MEISynqNetResourceCfgProbeDepth

## Definition

```
typedef enum MEISynqNetResourceProbeDepth {
    MEISynqNetResourceProbeDepthNONE,
    MEISynqNetResourceProbeDepthTWO,
    MEISynqNetResourceProbeDepthFOUR,
    MEISynqNetResourceProbeDepthEIGHT,
    MEISynqNetResourceProbeDepthSIXTEEN,
} MEISynqNetResourceProbeDepth;
```

## Description

**MEISynqNetResourceCfgProbeDepth** is an enumeration of the possible Probe register depths. Each Probe register is 16 bits. The packet data size is 32 bits. Each Probe engine can support up to 16 register fields for the Probe data.

<b>MEISynqNetResourceProbeDepthNONE</b>	zero Probe data registers will be transmitted upstream
<b>MEISynqNetResourceProbeDepthTWO</b>	2 Probe data registers will be transmitted upstream
<b>MEISynqNetResourceProbeDepthFOUR</b>	4 Probe data registers will be transmitted upstream
<b>MEISynqNetResourceProbeDepthEIGHT</b>	8 Probe data registers will be transmitted upstream
<b>MEISynqNetResourceProbeDepthSIXTEEN</b>	16 Probe data registers will be transmitted upstream

## See Also

# MEISynqNetState

## Definition

```
typedef enum MEISynqNetState {
    MEISynqNetStateINVALID,
    MEISynqNetStateDISCOVERY,
    MEISynqNetStateASYNQ,
    MEISynqNetStateSYNQ,
    MEISynqNetStateSYNQ_RECOVERING,
} MEISynqNetState;
```

## Description

**MEISynqNetState** is an enumeration of the SynqNet network states. Each state shows the present operation mode for the network. Network data traffic occurs in two modes: asynchronous and synchronous.

During **asynchronous** communication, the MPI or controller sends one packet to one node and the node responds with one packet. There are no timing restrictions. Asynchronous communication is used during network initialization, reset, discovery, and node binary download.

During **synchronous** communication, the controller sends packets to the nodes and the nodes respond with packets. All data traffic is scheduled in fixed timeslots to avoid collisions. Packet data updates are cyclic and synchronized to the controller's sample rate. The network state can be read with `meiSynqNetStatus(...)`.

<b>MEISynqNetStateDISCOVERY</b>	This is the initial state before the network is initialized. During this phase, the controller checks the network integrity, determines the network topology, resets the nodes, identifies, initializes, and addresses the nodes. Data packets are sent/received asynchronously.
<b>MEISynqNetStateASYNQ</b>	This state is used for off-line operations, like node binary download. Data packets are sent/received asynchronously.
<b>MEISynqNetStateSYNQ</b>	This is the normal operating state. Data packets are sent/received synchronously.

**MEISynqNetStateSYNQ\_RECOVERING**

During this state, a network fault condition was detected and the network is being reconfigured to route data packets around the fault. After the reconfiguration is complete and the packet error rate counters have decremented to zero, the network will return to the SYNQ state.

**See Also**

[MEISynqNetStatus](#) | [meiSynqNetStatus](#)



# MEISynqNetStatus

## Definition

```
typedef struct MEISynqNetStatus {
    MEISynqNetState          state;
    MPI_BOOL                topologySaved; /* TRUE/FALSE */
    MEISynqNetStatusCrcError  crcError;
    MPIEventMask              eventMask;
    MEISynqNetFailedNodeMask failedNodeMask;
} MEISynqNetStatus;
```

**Change History:** Modified in the 03.03.00

## Description

**MEISynqNetStatus** contains network state information, CRC counters, eventMask, and the failedNodeMask for a SynqNet network. The network status can be read with the meiSynqNetStatus(...) method.

<b>state</b>	Present operation mode for the network.
<b>topologySaved</b>	<p>Contains the status of the network topology.</p> <p>FALSE means the topology is dynamically discovered at initialization.</p> <p>TRUE means that the topology is verified against an expected topology at initialization.</p> <p>See <a href="#">Saving Current Topology To Flash</a> for more information.</p>
<b>crcError</b>	CRC error counters for the controller's network ports. The CRC value increments when received data is corrupt. Range is from 0 to 255. Counter saturates at 255.
<b>eventMask</b>	<p>An array that defines the status for the event mask bits. The array is defined as:</p> <pre>typedef MPIEventMaskELEMENT_TYPE         MPIEventMask [ MPIEventMaskELEMENTS ]</pre> <p>The bits are defined by the MPI/MEIEventType enumerations.</p>

**failedNodeMask**

Array that defines the failed node mask bits. Each bit represents a failed node (0x1 = node 0, 0x2 = node 2, 0x4 = node 3, etc.). The array is defined as:

```
#define MEISynqNetNodeMaskELEMENTS (1)
typedef long MEISynqNetFailedNodeMask
    [MEISynqNetNodeMaskELEMENTS];
```

## See Also

[meiSynqNetStatus](#) | [meiSqNodeStatus](#) | [MEISqNodeStatus](#)

# MEISynqNetStatusCrcError

## Definition

```
typedef struct MEISynqNetStatusCrcError {  
    long port[MEINetworkPortLAST];  
} MEISynqNetStatusCrcError;
```

## Description

**MEISynqNetStatusCrcError** contains the CRC values for each network port on the controller. This structure is read via the `meiSynqNetStatus(...)` method.

<b>port</b>	an array of CRC values, one for each network port.
-------------	--

## See Also

[MEINetworkPort](#) | [MEISynqNetStatus](#) | [meiSynqNetStatus](#) | [CRC Error Counters](#)

# MEISynqNetTiming

## Definition

```
typedef struct MEISynqNetTiming {
    double controllerFreq;      /* kHz */
    double controllerPeriod;   /* uS  */

    long   txTime;              /* %   */
    double calculationLimit;   /* uS  */

    double driveUpdateFreq;    /* kHz */
    double driveUpdatePeriod; /* uS  */

    long   bandwidthUsage;     /* %   */

    struct {
        double calculationTime; /* uS  */
        double calculationSlack; /* uS  */
        double demandLatency;   /* uS  */
        double feedbackLatency; /* uS  */

        double total;           /* uS  */
        double latencySlack;   /* uS  */
    } controlLatency;

    struct {
        double synq;           /* uS  - synq packet + spacing */
        double demand;        /* uS  - sum demand packets + spacing */
        double control;       /* uS  - sum control packets + spacing */

        double total;         /* uS  */
    } downstream;

    struct {
        double feedback;      /* uS  - sum feedback packets + spacing */
        double status;       /* uS  - sum status packets + spacing */

        double total;         /* uS  */
    } upstream;
} MEISynqNetTiming;
```

## Description

**MEISynqNetTiming** contains static data that is determined during network initialization. It identifies timing statistics about the current network. Most values are just reported based on network timing calculations. The values that can be configured by the MPI are stated below as configurable.

<b>controllerFreq</b>	<p>The sample rate calculation of the SynqNet controller board, in kHz. It is also the rate at which SynqNet packets are sent. The controllerFreq must be configured so that the driveUpdateFreq is an integer multiple of the controllerFreq.</p> <p>This value is configurable. See <a href="#">MPIControlConfig</a>.</p>
<b>controllerPeriod</b>	<p>The sample period calculation of the SynqNet controller board, in uS. Derived from controllerFreq.</p>
<b>txTime</b>	<p>The scheduled time for SynqNet packets to be sent, in % (of the controllerPeriod). Defaults to 75%. Must be set to a value greater than the foreground calculation time, and less than 100%.</p> <p>This value is configurable. See <a href="#">MEIControlConfig</a>.</p>
<b>calculationLimit</b>	<p>The maximum allowed foreground calculation time, in uS. Derived from txTime * controllerPeriod.</p>
<b>driveUpdateFreq</b>	<p>The cyclic communication rate of a drive processor on a SynqNet node, in kHz. Typically fixed to 16 kHz, but may vary depending on drive type. May be configurable on some drives. This rate often matches the drive PWM rate.</p>
<b>driveUpdatePeriod</b>	<p>The cyclic communication period of a drive processor on a SynqNet node, in uS. Derived from driveUpdateFreq.</p>
<b>bandwidthUsage</b>	<p>The amount of SynqNet bandwidth used, in %, for this SynqNet configuration. The actual packet payload configured is divided by the maximum available SynqNet bandwidth, for both upstream and downstream packets. The greater of the two is reported.</p>
<b>calculationTime</b>	<p>The foreground calculation time of the controller, in uS.</p>
<b>calculationSlack</b>	<p>The available slack time between the foreground calculation time, and the scheduled txTime, in uS.</p>
<b>demandLatency</b>	<p>The time to send SynqNet demand data downstream from controller to nodes, in uS. Does not include drive demand delays.</p>
<b>feedbackLatency</b>	<p>The time to send SynqNet feedback data upstream from nodes to controller, in uS. Rounded up to make the total SynqNet latency an integer multiple of the driveUpdatePeriod. Does not include drive feedback delays.</p>

<b>latencySlack</b>	The slack time from the configured feedbackLatency (which is rounded up to an integer multiple of the driveUpdatePeriod), to the minimum possible upstream delay, in uS.
<b>controlLatency</b>	The overall SynqNet system control latency, in uS. Always rounded up to an integer multiple of driveUpdatePeriod. Control latency begins when position feedback is sampled on the node, includes upstream packet delays, controller foreground calculation, downstream packet delays, and ends when demands are strobed on the nodes. Note drive feedback and demand delays are NOT included. Control latency can be broken down into the four components listed below: feedbackLatency, calculationTime, calculationSlack, and demandLatency.
<b>synq</b>	Downstream SYNQ packet payload, in uS. Includes spacing between packets.
<b>demand</b>	Downstream DEMAND packet payload, in uS. Includes spacing between packets.
<b>control</b>	Downstream CONTROL packet payload, in uS. Includes spacing between packets.
<b>downstream</b>	Total downstream (controller to nodes) packet payload, in uS. Includes spacing between packets. Breaks down into the following three packet types.
<b>feedback</b>	Upstream FEEDBACK packet payload, in uS. Includes spacing between packets.
<b>status</b>	Upstream STATUS packet payload, in uS. Includes spacing between packets.
<b>upstream</b>	Total upstream (nodes to controller) packet payload, in uS. Includes spacing between packets. Breaks down into the following two packet types.

## See Also

[SynqNet Timing Values](#)

# MEISynqNetTrace

## Definition

```
typedef enum {
    MEISynqNetTraceDYNA_ALLOC = MEISynqNetTraceFIRST << 0,
    MEISynqNetTraceDYNA_FREE = MEISynqNetTraceFIRST << 1,
    MEISynqNetTraceINIT      = MEISynqNetTraceFIRST << 2,
    MEISynqNetTraceRESET     = MEISynqNetTraceFIRST << 3,
    MEISynqNetTraceTIMING    = MEISynqNetTraceFIRST << 4,
    MEISynqNetTraceSERVICE_CMD = MEISynqNetTraceFIRST << 5,
    MEISynqNetTraceFLOWCTRL = MEISynqNetTraceFIRST << 6,
    MEISynqNetTraceMAP       = MEISynqNetTraceFIRST << 7,
    MEISynqNetTraceUNMAP     = MEISynqNetTraceFIRST << 8,
} MEISynqNetTrace;
```

## Description

**MEISynqNetTrace** is an enumeration of SynqNet trace bits that can be used to enable/disable library trace statement output for the SynqNet object.

<b>MEISynqNetTraceDYNA_ALLOC</b>	Enables trace statements for controller external memory allocation during SynqNet initialization.
<b>MEISynqNetTraceDYNA_FREE</b>	Enables trace statements for controller external memory de-allocation.
<b>MEISynqNetTraceINIT</b>	Enables trace statements for SynqNet network initialization.
<b>MEISynqNetTraceRESET</b>	Enables trace statements for SynqNet network reset.
<b>MEISynqNetTraceTIMING</b>	Enables trace statements for SynqNet network timing calculations.
<b>MEISynqNetTraceSERVICE_CMD</b>	Enables trace statements for SynqNet service command transactions between the controller and SynqNet nodes.
<b>MEISynqNetTraceFLOWCTRL</b>	This trace bit enables trace statements for the SynqNet service command handshake between the controller and SynqNet nodes.
<b>MEISynqNetTraceMAP</b>	This trace bit enables trace statements for the mapping of firmware pointers to the dynamic controller memory.

**MEISynqNetTraceUNMAP**

This trace bit enables trace statements for the unmapping of firmware pointers from the dynamic controller memory.

**See Also**

[Trace.exe utility](#)



# MEISynqNetMaxCableHOP\_COUNT

## Definition

```
#define MEISynqNetCableHOP_COUNT (MEISynqNetMaxNODE_COUNT + 1)
```

## Description

**MEISynqNetMaxCableHOP\_COUNT** is the maximum number of cables for a SynqNet network. Presently, string and ring topologies are supported. The maximum number of cables is based on the maximum number of nodes, plus one return cable for ring topologies.

## See Also

[MEISynqNetCableList](#) | [MEISynqNetConfig](#)

# MEISynqNetNodeMaskELEMENTS

## Definition

```
#define MEISynqNetNodeMaskELEMENTS (1)
typedef long MEISynqNetFailedNodeMask [MEISynqNetNodeMaskELEMENTS];
```

## Description

**MEISynqNetNodeMaskELEMENTS** defines the number of data elements in an [MEISynqNetFailedNodeMask](#).

## See Also

[MEISynqNetFailedNodeMask](#)

# MEISynqNetMaxMOTORS

## Definition

```
#define MEISynqNetMaxMOTORS (MEIXmpMAX_Motors) /* 32 */
```

## Description

**MEISynqNetMaxMOTORS** defines the maximum number of motors supported on a single SynqNet network.

## See Also

# MEISynqNetMaxNodeMOTORS

## Definition

```
#define MEISynqNetMaxNodeMOTORS ( MEISqNodeMaxMOTORS )
```

## Description

**MEISynqNetMaxNodeMOTORS** defines the maximum number of motor objects supported on a single SynqNet node.

## See Also

# MEISynqNetMaxMotorFEEDBACK\_SECONDARY\_COUNT

## Definition

```
#define MEISynqNetMaxMotorFEEDBACK_SECONDARY_COUNT  
    (MEISqNodeMaxFEEDBACK_SECONDARY)
```

## Description

**MEISynqNetMaxMotorFEEDBACK\_SECONDARY\_COUNT** defines the maximum number of secondary feedback resources per motor.

**NOTE:** Feedback count may be further limited by the available resources on the node.

## See Also

# MEISynqNetMaxMotorPROBE\_COUNT

## Definition

```
#define MEISynqNetMaxMotorPROBE_COUNT (4)
```

## Description

**MEISynqNetMaxMotorPROBE\_COUNT** defines the maximum number of probe resources per motor.

## See Also

[meiSynqNetPacketConfigGet](#) | [meiSynqNetPacketConfigSet](#)

# MEISynqNetMaxMotorFEEDBACK\_PRIMARY\_COUNT

## Definition

```
#define MEISynqNetMaxMotorFEEDBACK_PRIMARY_COUNT (1)
```

## Description

**MEISynqNetMaxMotorFEEDBACK\_PRIMARY\_COUNT** defines the maximum number of primary feedback resources per motor.

**NOTE:** Feedback count may be further limited by the available resources on the node.

## See Also

# MEISynqNetMaxMotorCAPTURE\_COUNT

## Definition

```
#define MEISynqNetMaxMotorCAPTURE_COUNT (MEIXmpMaxCapturesPerMotor) /* 2 */
```

## Description

**MEISynqNetMaxMotorCAPTURE\_COUNT** defines the maximum number of capture resources per motor.

**NOTE:** Capture count may be further limited by the available resources on the node.

This define should be used instead of the `MEIXmpMaxCapturesPerMotor` definition in `xmp.h`. It is recommended that applications avoid programming to defines or structures in `xmp.h`

## See Also



# MEISynqNetMaxMotorCOMPARE\_COUNT

## Definition

```
#define MEISynqNetMaxMotorCOMPARE_COUNT (MEIXmpMaxCapturesPerMotor) /* 2 */
```

## Description

**MEISynqNetMaxMotorCOMPARE\_COUNT** defines the maximum number of compare resources per motor.

**NOTE:** Compare count may be further limited by the available resources on the node.

This define should be used instead of the MEIXmpMaxComparesPerMotor definition in xmp.h. It is recommended that applications avoid programming to defines or structures in xmp.h

## See Also

# MEISynqNetMaxMotorENCODER\_COUNT

## Definition

```
#define MEISynqNetMaxMotorENCODER_COUNT (MEIXmpMotorEncoders) /* 2 */
```

## Description

**MEISynqNetMaxMotorENCODER\_COUNT** defines the maximum number of encoder resources per motor.

**NOTE:** The encoder count may be further limited by the available resources on the node.

This define should be used instead of the `MEIXmpMotorEncoders` definition in `xmp.h`. It is recommended that applications avoid programming to defines or structures in `xmp.h`.

## See Also

# MEISynqNetMaxMotorPULSE\_ENGINE\_COUNT

## Definition

```
#define MEISynqNetMaxMotorPULSE_ENGINE_COUNT (1)
```

## Description

**MEISynqNetMaxMotorPULSE\_ENGINE\_COUNT** defines the number of pulse engines per motor.

## See Also

# MEISynqNetMaxNODE\_COUNT

## Definition

```
#define MEISynqNetMaxNODE_COUNT (MEIXmpMaxSynqNetBlocks) /* 32 */
```

## Description

**MEISynqNetMaxNODE\_COUNT** defines the maximum number of nodes supported on a single SynqNet network. This define should be used instead of the **MEIXmpMaxSynqNetBlocks** definition in `xmp.h`. It is recommended that applications avoid programming to defines or structures in `xmp.h`

## See Also