

OOPI

Generated by Doxygen 1.8.15

1 Hierarchical Index	1
1.1 Class Hierarchy	1
2 Data Structure Index	3
2.1 Data Structures	3
3 File Index	5
3.1 File List	5
4 Data Structure Documentation	7
4.1 Communicative Class Reference	7
4.1.1 Detailed Description	7
4.1.2 Constructor & Destructor Documentation	7
4.1.2.1 Communicative()	8
4.1.2.2 ~Communicative()	8
4.1.3 Member Function Documentation	8
4.1.3.1 isPeripheralConnected()	8
4.1.3.2 requestData()	9
4.1.3.3 requestIdentity()	10
4.1.3.4 requestReply()	10
4.2 Data Struct Reference	11
4.2.1 Detailed Description	12
4.2.2 Member Function Documentation	12
4.2.2.1 operator=() [1/2]	12
4.2.2.2 operator=() [2/2]	12
4.2.3 Field Documentation	12
4.2.3.1 DataPoints	12
4.2.3.2 NumColumns	13
4.2.3.3 NumRows	13
4.2.3.4 RowHeadings	13
4.2.3.5 rowUnits	13
4.3 DataSource Class Reference	13
4.3.1 Detailed Description	14
4.3.2 Constructor & Destructor Documentation	14
4.3.2.1 DataSource()	14
4.3.3 Member Function Documentation	15
4.3.3.1 getDataArray()	15
4.3.3.2 getDataVector()	15
4.3.3.3 getNumberOfDataColumns()	16
4.3.3.4 getNumberOfDataRows()	16
4.3.3.5 getRowHeadings()	16
4.3.3.6 getRowUnits()	17
4.3.3.7 getValueOne()	17

4.3.3.8	getValueThree()	18
4.3.3.9	getValueTwo()	18
4.3.3.10	getVectorHeading()	18
4.3.3.11	getVectorLength()	19
4.3.3.12	getVectorUnits()	19
4.3.3.13	isThereData()	19
4.3.3.14	loadData()	20
4.4	Identifiable Class Reference	21
4.4.1	Detailed Description	21
4.4.2	Constructor & Destructor Documentation	21
4.4.2.1	Identifiable()	21
4.4.3	Member Function Documentation	22
4.4.3.1	getIDNumber()	22
4.4.3.2	getSensorName()	22
4.4.3.3	hasIdentityChanged()	23
4.4.3.4	updateIdentity()	23
4.5	Identity Struct Reference	24
4.5.1	Detailed Description	24
4.5.2	Member Function Documentation	24
4.5.2.1	operator=() [1/2]	25
4.5.2.2	operator=() [2/2]	25
4.5.3	Field Documentation	25
4.5.3.1	sensorChipSelect	25
4.5.3.2	sensorID	25
4.5.3.3	SensorName	25
4.6	Instructable Class Reference	26
4.6.1	Detailed Description	26
4.6.2	Constructor & Destructor Documentation	26
4.6.2.1	Instructable()	26
4.6.3	Member Function Documentation	27
4.6.3.1	areYouConnected()	27
4.6.3.2	issueCommand() [1/4]	27
4.6.3.3	issueCommand() [2/4]	28
4.6.3.4	issueCommand() [3/4]	29
4.6.3.5	issueCommand() [4/4]	30
4.7	Instructor Class Reference	30
4.7.1	Detailed Description	31
4.7.2	Constructor & Destructor Documentation	31
4.7.2.1	Instructor()	32
4.7.3	Member Function Documentation	32
4.7.3.1	getCurrentCommandFloat()	32
4.7.3.2	getCurrentCommandInstruction()	32

4.7.3.3 <code>getCurrentCommandInt()</code>	32
4.7.3.4 <code>getCurrentCommandString()</code>	33
4.7.3.5 <code>howLongShouldIWait()</code>	33
4.7.3.6 <code>howManyInstructions()</code>	33
4.7.3.7 <code>loadNextCommand()</code>	34
4.8 Master Class Reference	34
4.8.1 Detailed Description	36
4.8.2 Constructor & Destructor Documentation	36
4.8.2.1 <code>Master()</code> [1/3]	37
4.8.2.2 <code>Master()</code> [2/3]	37
4.8.2.3 <code>~Master()</code>	37
4.8.2.4 <code>Master()</code> [3/3]	37
4.8.3 Member Function Documentation	37
4.8.3.1 <code>beginMeasurement()</code>	37
4.8.3.2 <code>ClearMeasurementVector()</code>	38
4.8.3.3 <code>getCurrentInstruction()</code>	38
4.8.3.4 <code>getCurrentInstructionFloatParameter()</code>	38
4.8.3.5 <code>getCurrentInstructionIntParameter()</code>	38
4.8.3.6 <code>getCurrentInstructionNumber()</code>	38
4.8.3.7 <code>Handshake()</code>	39
4.8.3.8 <code>isThereData()</code>	39
4.8.3.9 <code>loadRequest()</code>	39
4.8.3.10 <code>operator=()</code> [1/2]	40
4.8.3.11 <code>operator=()</code> [2/2]	40
4.8.3.12 <code>PopMeasurementVector()</code>	40
4.8.3.13 <code>PushMeasurementVector()</code>	41
4.8.3.14 <code>resendCurrentUserInstruction()</code>	41
4.8.3.15 <code>restartUserInstructionCycle()</code>	41
4.8.3.16 <code>sendData()</code>	42
4.8.3.17 <code>sendIdentity()</code>	42
4.8.3.18 <code>sendNextUserInstruction()</code>	42
4.8.3.19 <code>sendReply()</code> [1/9]	43
4.8.3.20 <code>sendReply()</code> [2/9]	43
4.8.3.21 <code>sendReply()</code> [3/9]	43
4.8.3.22 <code>sendReply()</code> [4/9]	44
4.8.3.23 <code>sendReply()</code> [5/9]	44
4.8.3.24 <code>sendReply()</code> [6/9]	44
4.8.3.25 <code>sendReply()</code> [7/9]	44
4.8.3.26 <code>sendReply()</code> [8/9]	44
4.8.3.27 <code>sendReply()</code> [9/9]	45
4.8.3.28 <code>sendTotalNumOfInstructions()</code>	45
4.8.3.29 <code>setMeasurementVectorHeading()</code>	45

4.8.3.30 setMeasurementVectorUnits()	46
4.8.3.31 SETUP()	46
4.8.3.32 shallStart()	47
4.8.3.33 SPISetup()	47
4.9 mCmd Struct Reference	48
4.9.1 Detailed Description	48
4.9.2 Constructor & Destructor Documentation	49
4.9.2.1 mCmd() [1/3]	49
4.9.2.2 mCmd() [2/3]	49
4.9.2.3 mCmd() [3/3]	49
4.9.3 Member Function Documentation	49
4.9.3.1 operator=() [1/2]	49
4.9.3.2 operator=() [2/2]	49
4.9.4 Field Documentation	49
4.9.4.1 fParam	50
4.9.4.2 Instruction	50
4.9.4.3 iParam	50
4.10 sCmd Struct Reference	50
4.10.1 Detailed Description	51
4.10.2 Member Function Documentation	51
4.10.2.1 operator=() [1/2]	51
4.10.2.2 operator=() [2/2]	51
4.10.3 Field Documentation	51
4.10.3.1 fParam	51
4.10.3.2 Instruction	51
4.10.3.3 iParam	52
4.10.3.4 sParam	52
4.11 Sensor Class Reference	52
4.11.1 Detailed Description	53
4.11.2 Constructor & Destructor Documentation	53
4.11.2.1 Sensor()	53
4.11.3 Member Function Documentation	53
4.11.3.1 PauseMeasurementForMillis()	53
4.11.3.2 RestartMeasurement()	55
4.11.3.3 StartMeasurement()	56
4.12 UserInstructions Struct Reference	56
4.12.1 Member Function Documentation	56
4.12.1.1 operator=() [1/2]	57
4.12.1.2 operator=() [2/2]	57
4.12.2 Field Documentation	57
4.12.2.1 fParams	57
4.12.2.2 InstructionCounter	57

4.12.2.3 InstructionSet	57
4.12.2.4 iParams	57
4.12.2.5 MasterInstructionSet	57
4.12.2.6 NumOfInstructions	57
5 File Documentation	59
5.1 Communicative.cpp File Reference	59
5.1.1 Variable Documentation	59
5.1.1.1 REQUEST_DELAY_MICROS	60
5.2 Communicative.h File Reference	60
5.3 DataSource.cpp File Reference	61
5.4 DataSource.h File Reference	61
5.5 Identifiable.cpp File Reference	63
5.6 Identifiable.h File Reference	63
5.7 Instructable.cpp File Reference	64
5.8 Instructable.h File Reference	65
5.9 Instructor.cpp File Reference	66
5.10 Instructor.h File Reference	67
5.11 Master.cpp File Reference	69
5.11.1 Function Documentation	69
5.11.1.1 SPI_IRQ()	69
5.11.2 Variable Documentation	70
5.11.2.1 SensorMaster	70
5.12 Master.h File Reference	70
5.12.1 Macro Definition Documentation	71
5.12.1.1 SPI1_NSS_PIN	71
5.12.2 Function Documentation	72
5.12.2.1 RequestHandler()	72
5.12.2.2 SPI_IRQ()	72
5.12.3 Variable Documentation	73
5.12.3.1 MAX_USER_INSTRUCTION_NUMBER	73
5.12.3.2 SensorMaster	73
5.13 Sensor.cpp File Reference	73
5.14 Sensor.h File Reference	74
5.15 SPI_InstructionSet.h File Reference	75
5.15.1 Enumeration Type Documentation	76
5.15.1.1 MeasurementVectors	76
5.15.1.2 mInstruct	76
5.15.1.3 sInstruct	77
5.15.2 Variable Documentation	77
5.15.2.1 DATA_ROW_LENGTH	77
5.15.2.2 IDENTITY_SENSOR_NAME_LENGTH	77

5.15.2.3 NUMBER_OF_DATA_ROWS	78
5.15.2.4 ROW_HEADING_LENGTH	78
5.15.2.5 ROW_UNIT_LENGTH	78
5.15.2.6 SLAVE_COMMMAND_STRING_LENGTH	78

Index	79
--------------	-----------

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Communicative	7
Data	11
DataSource	13
Sensor	52
Identifiable	21
Sensor	52
Identity	24
Instructable	26
Sensor	52
Instructor	30
Sensor	52
Master	34
mCmd	48
sCmd	50
UserInstructions	56

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Communicative	A class to manage communication with slave module	7
Data	Type used to encapsulate the data collected by the slave	11
DataSource	A class which models a Sensor/peripheral as an entity which is a source of data	13
Identifiable	A class which models a Sensor/peripheral as an identifiable entity	21
Identity	Type used to convey the Slave identity	24
Instructable	A class which models a Sensor/peripheral as entity which can receive commands	26
Instructor	A class which models a Sensor/peripheral as entity which can issue instructions to the master	30
Master	A monolithic class to encapsulate and abstract the slave's communication with the master	34
mCmd	Type used by master to send requests to slave	48
sCmd	Type used by slave to send reply to master,	50
Sensor	A class which models a Sensor/peripheral	52
UserInstructions	56

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Communicative.cpp	59
Communicative.h	60
DataSource.cpp	61
DataSource.h	61
Identifiable.cpp	63
Identifiable.h	63
Instructable.cpp	64
Instructable.h	65
Instructor.cpp	66
Instructor.h	67
Master.cpp	69
Master.h	70
Sensor.cpp	73
Sensor.h	74
SPI_InstructionSet.h	75

Chapter 4

Data Structure Documentation

4.1 Communicative Class Reference

A class to manage communication with slave module.

```
#include <Communicative.h>
```

Public Member Functions

- **Communicative** (const int CS)
Constructor.
- **~Communicative** (void)
Destructor.
- bool **isPeripheralConnected** (void)
Checks whether Slave is connected.
- **sCmd RequestReply** (const **mCmd**)
*Performs a complete transaction; expects Slave to Reply with **sCmd** object.*
- **Identity RequestIdentity** (const **mCmd**)
*Performs a complete transaction; expects Slave to Reply with **Identity** object.*
- **Data RequestData** (const **mCmd**)
*Performs a complete transaction; expects Slave to Reply with a **Data** object.*

4.1.1 Detailed Description

A class to manage communication with slave module.

This class is designed for SPI communication with a slave device. The class responsibilities include both SPI initialisation and fundamental transactions. The transaction protocol implemented follows the following flow: Clear SS -> Send '?' to slave -> receive 'ACK' (0x06) from slave -> send request **mCmd** -> receive sCmd/Data/Identity as expected -> set SS. Where **mCmd**, **sCmd**, **Data** and **Identity** are structures defined as types.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 Communicative()

```
Communicative (
    const int CS )
```

Constructor.

Constructor initialises the SS pin to be used in communications and initialises SPI as [Master](#).

Parameters

CS	is the Slave Select pin designation; most commonly PA4.
----	---

See also

[SPISetup\(\)](#)

4.1.2.2 ~Communicative()

```
~Communicative (
    void )
```

Destructor.

Executes SPI.end();

4.1.3 Member Function Documentation

4.1.3.1 isPeripheralConnected()

```
bool isPeripheralConnected (
    void )
```

Checks whether Slave is connected.

Executes a nop transaction.

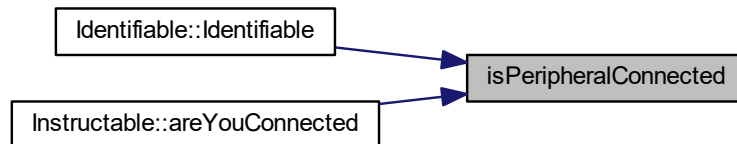
Returns

true if handshake is successful and Slave responds to '?' with 'ACK'

See also

[areYouAlive\(\)](#)

Here is the caller graph for this function:



4.1.3.2 RequestData()

```
Data RequestData (
    const mCmd Request )
```

Performs a complete transaction; expects Slave to Reply with a [Data](#) object.

Executes a complete transaction: Clear SS -> Send '?' to slave -> recieve 'ACK' (0x06) from slave -> send request [mCmd](#) -> recieve [Data](#) -> set SS.

Parameters

mCmd	is the mCmd object which constitutes the Request made to the slave.
----------------------	---

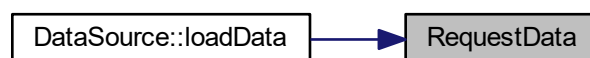
Returns

The [Data](#) object generated by the Slave containing a two dimensional array of data points and the length, headings and units of the data array rows.

See also

[RequestReply\(\)](#), [RequestIdentity\(\)](#)

Here is the caller graph for this function:



4.1.3.3 RequestIdentity()

```
Identity RequestIdentity (
    const mCmd Request )
```

Performs a complete transaction; expects Slave to Reply with [Identity](#) object.

Executes a complete transaction: Clear SS -> Send '?' to slave -> recieve 'ACK' (0x06) from slave -> send request [mCmd](#) -> recieve [Identity](#) -> set SS.

Parameters

mCmd	is the mCmd object which constitutes the Request made to the slave.
----------------------	---

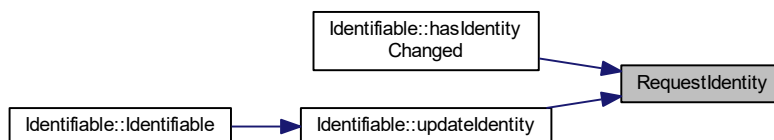
Returns

The Identity object generated by the Slave containing the Slave ID and name.

See also

[RequestReply\(\)](#), [RequestData\(\)](#)

Here is the caller graph for this function:



4.1.3.4 RequestReply()

```
sCmd RequestReply (
    const mCmd Request )
```

Performs a complete transaction; expects Slave to Reply with [sCmd](#) object.

Executes a complete transaction: Clear SS -> Send '?' to slave -> recieve 'ACK' (0x06) from slave -> send request [mCmd](#) -> recieve [sCmd](#) -> set SS.

Parameters

[mCmd](#) is the [mCmd](#) object which constitutes the Request made to the slave.

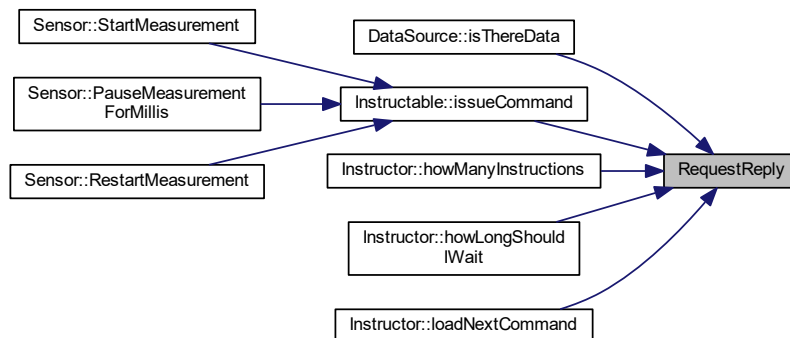
Returns

The [sCmd](#) object generated by the Slave as the reply to the request.

See also

[RequestIdentity\(\)](#), [RequestData\(\)](#)

Here is the caller graph for this function:



The documentation for this class was generated from the following files:

- [Communicative.h](#)
- [Communicative.cpp](#)

4.2 Data Struct Reference

Type used to encapsulate the data collected by the slave.

```
#include <SPI_InstructionSet.h>
```

Public Member Functions

- `Data & operator= (const volatile Data &rhs) volatile`
- `volatile Data & operator= (const Data &rhs) volatile`

Data Fields

- int `NumColumns` [`NUMBER_OF_DATA_ROWS`]
Number of data points currently stored in each row.
- int `NumRows`
Number of rows. Defined at compilation.
- char `RowHeadings` [`NUMBER_OF_DATA_ROWS`][`ROW_HEADING_LENGTH`]
String headings to describe the data in each row.
- char `rowUnits` [`NUMBER_OF_DATA_ROWS`][`ROW_UNIT_LENGTH`]
String units to qualify the data in each row.
- float `DataPoints` [`NUMBER_OF_DATA_ROWS`][`DATA_ROW_LENGTH`]
Two dimensional array of data. Each row generally treated as an independent vector.

4.2.1 Detailed Description

Type used to encapsulate the data collected by the slave.

`Data` is contained in a two dimensional array but generally modelled as a collection of 'vectors' or rows of data. Each row is allowed a variable number of data points, a string heading and a string unit.

4.2.2 Member Function Documentation

4.2.2.1 `operator=()` [1/2]

```
Data& operator= (
    const volatile Data & rhs ) volatile [inline]
```

4.2.2.2 `operator=()` [2/2]

```
volatile Data& operator= (
    const Data & rhs ) volatile [inline]
```

4.2.3 Field Documentation

4.2.3.1 `DataPoints`

```
float DataPoints[NUMBER_OF_DATA_ROWS][DATA_ROW_LENGTH]
```

Two dimensional array of data. Each row generally treated as an independent vector.

4.2.3.2 NumColumns

```
int NumColumns[NUMBER_OF_DATA_ROWS]
```

Number of data points currently stored in each row.

4.2.3.3 NumRows

```
int NumRows
```

Number of rows. Defined at compilation.

4.2.3.4 RowHeadings

```
char RowHeadings[NUMBER_OF_DATA_ROWS][ROW_HEADING_LENGTH]
```

String headings to describe the data in each row.

4.2.3.5 rowUnits

```
char rowUnits[NUMBER_OF_DATA_ROWS][ROW_UNIT_LENGTH]
```

String units to qualify the data in each row.

The documentation for this struct was generated from the following file:

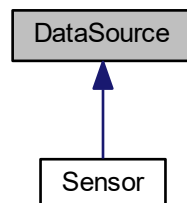
- [SPI_InstructionSet.h](#)

4.3 DataSource Class Reference

A class which models a Sensor/peripheral as an entity which is a source of data.

```
#include <DataSource.h>
```

Inheritance diagram for DataSource:



Public Member Functions

- [DataSource](#) (const int ChipSelect)
Constructor.
- bool [isThereData](#) (void)
Asks Sensor/peripheral whether there is [Data](#) ready to be collected.
- [Data loadData](#) (void)
Loads [Data](#) from the [Sensor](#).
- int [getNumberOfDataColumns](#) (const [MeasurementVectors](#) VectorNumber)
Gets the number of data points along a particular row of the data array.
- int [getNumberOfDataRows](#) (void)
Gets the number of rows used in the data array.
- void [getRowHeadings](#) (char[[NUMBER_OF_DATA_ROWS](#)][[ROW_HEADING_LENGTH](#)])
Gets string headings of all the vectors in the data array.
- void [getRowUnits](#) (char[[NUMBER_OF_DATA_ROWS](#)][[ROW_UNIT_LENGTH](#)])
Gets string units of all the vectors in the data array.
- void [getDataArray](#) (float[[NUMBER_OF_DATA_ROWS](#)][[DATA_ROW_LENGTH](#)])
Gets the entire data array.
- void [getDataVector](#) (const [MeasurementVectors](#) VectorNumber, float[[DATA_ROW_LENGTH](#)])
Gets the indicated data vector.
- int [getVectorLength](#) (const [MeasurementVectors](#) VectorNumber)
Gets the indicated data vector length.
- void [getVectorHeading](#) (const [MeasurementVectors](#) VectorNumber, char[[ROW_HEADING_LENGTH](#)])
Gets the string heading for the vector in question.
- void [getVectorUnits](#) (const [MeasurementVectors](#) VectorNumber, char[[ROW_UNIT_LENGTH](#)])
Gets the string units for the vector in question.
- float [getValueOne](#) (void)
Gets the first data point in the first vector.
- float [getValueTwo](#) (void)
Gets the second data point in the first vector.
- float [getValueThree](#) (void)
Gets the third data point in the first vector.

4.3.1 Detailed Description

A class which models a Sensor/peripheral as an entity which is a source of data.

This class models a [Sensor](#) as a source of data. The convention in use is that any [Sensor](#) can store data such that it occupies a two dimensional float array with maximum dimensions [NUMBER_OF_DATA_ROWS](#) x [DATA_ROW_LENGTH](#). The data can either be treated as a square array, the dimensions of which can be requested, or as a series of 'vectors', the length of which can be requested.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 DataSource()

```
DataSource (
    const int ChipSelect )
```

Constructor.

Parameters

<i>ChipSelect</i>	is the Slave Select pin of the SPI peripheral in question.
-------------------	--

4.3.3 Member Function Documentation

4.3.3.1 getDataArray()

```
void getDataArray (
    float DataVals[NUMBER_OF_DATA_ROWS][DATA_ROW_LENGTH] )
```

Gets the entire data array.

Retrieves the entire two dimensional data array, irrespective of which elements/vectors are actually in use.

Parameters

<i>float</i>	is the array into which the data is written.
--------------	--

See also

[Data](#)

4.3.3.2 getDataVector()

```
void getDataVector (
    const MeasurementVectors VectorNumber,
    float DataVect[DATA_ROW_LENGTH] )
```

Gets the indicated data vector.

Retrieves a singel row in the two dimesnional data array.

Parameters

<i>VectorNumber</i>	is the row in the data array to be retrieved.
<i>float</i>	is the floating point array into which the data points will be written.

See also

[Data](#), [MeasurementVectors](#)

4.3.3.3 `getNumberOfDataColumns()`

```
int getNumberOfDataColumns (
    const MeasurementVectors VectorNumber )
```

Gets the number of data points along a particular row of the data array.

Parameters

<i>VectorNumber</i>	is an enumerated type referring to the row in the two dimensional data array.
---------------------	---

Returns

The number of data points along a particular row/vector.

See also

[MeasurementVectors](#), [Data](#)

4.3.3.4 `getNumberOfDataRows()`

```
int getNumberOfDataRows (
    void )
```

Gets the number of rows used in the data array.

Returns the number of 'vectors' (rows) which the sensor has used to store data. Ideally, one should utilise the result of this function to iterate through the vectors.

Returns

The number of vectors in use.

See also

[Data](#)

4.3.3.5 `getRowHeadings()`

```
void getRowHeadings (
    char Headings[NUMBER_OF_DATA_ROWS][ROW_HEADING_LENGTH] )
```

Gets string headings of all the vectors in the data array.

Each vector/row is assigned a heading to describe the nature of the data contained within that vector. Such as "Ambient Temperature".

Parameters

<i>char</i>	is the array of character arrays into which the headings are loaded.
-------------	--

See also[Data](#)**4.3.3.6 getRowUnits()**

```
void getRowUnits (
    char Units[NUMBER_OF_DATA_ROWS][ROW_UNIT_LENGTH] )
```

Gets string units of all the vectors in the data array.

Each vector/row is assigned a Units string to define the units of the data contained within that vector. Such as "V" or "Amperes".

Parameters

<i>char</i>	is the array of character arrays into which the units are loaded.
-------------	---

See also[Data](#)**4.3.3.7 getValueOne()**

```
float getValueOne (
    void )
```

Gets the first data point in the first vector.

Returns

The first data point in the first vector. dataArray[0][0].

See also[Data](#), [MeasurementVectors](#)

4.3.3.8 `getValueThree()`

```
float getValueThree (
    void )
```

Gets the third data point in the first vector.

Returns

The third data point in the first vector. `dataArray[2][0]`.

See also

[Data](#), [MeasurementVectors](#)

4.3.3.9 `getValueTwo()`

```
float getValueTwo (
    void )
```

Gets the second data point in the first vector.

Returns

The second data point in the first vector. `dataArray[1][0]`.

See also

[Data](#), [MeasurementVectors](#)

4.3.3.10 `getVectorHeading()`

```
void getVectorHeading (
    const MeasurementVectors VectorNumber,
    char Heading[ROW_HEADING_LENGTH] )
```

Gets the string heading for the vector in question.

Parameters

<i>VectorNumber</i>	is the row in the data array to which the heading corresponds.
---------------------	--

See also

[getRowHeadings\(\)](#), [Data](#), [MeasurementVectors](#)

4.3.3.11 getVectorLength()

```
int getVectorLength (
    const MeasurementVectors VectorNumber )
```

Gets the indicated data vector length.

The data vectors (rows) have a max length of DATA_ROW_LENGTH and the [Sensor](#) will push data points into said vector. As the [Sensor](#) may not utilise the entire width of the data array, the length indicates the number of values which the [Sensor](#) has pushed into the vector in question.

Parameters

<i>VectorNumber</i>	is the row in the data array.
---------------------	-------------------------------

See also

[Data](#), [MeasurementVectors](#)

4.3.3.12 getVectorUnits()

```
void getVectorUnits (
    const MeasurementVectors VectorNumber,
    char Units[ROW_UNIT_LENGTH] )
```

Gets the string units for the vector in question.

Parameters

<i>VectorNumber</i>	is the row in the data array to which the heading corresponds.
---------------------	--

See also

[getRowUnits\(\)](#), [Data](#), [MeasurementVectors](#)

4.3.3.13 isThereData()

```
bool isThereData (
    void )
```

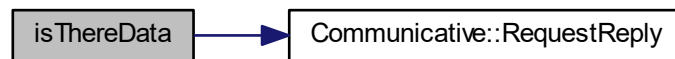
Asks Sensor/peripheral whether there is [Data](#) ready to be collected.

Asks the sensor whether the data is ready to be retrieved by the master. Slave's are, however, required to instantiate a [Data](#) object and so premature loads thereof will not fail.

Returns

True if the data is ready to be collected from the [Sensor](#).

Here is the call graph for this function:



4.3.3.14 loadData()

```
Data loadData (
    void )
```

Loads [Data](#) from the [Sensor](#).

Loads the [Data](#) object from the [Sensor](#) into local memory.

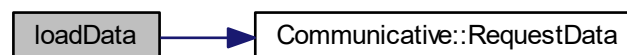
Returns

The [Data](#) object loaded into local memory. User of accessors preferred.

See also

[getNumberOfDataColumns\(\)](#), [getNumberOfDataRows\(\)](#), [getRowHeadings\(\)](#), [getRowUnits\(\)](#), [getDataArray\(\)](#), [getDataVector\(\)](#), [getVectorLength\(\)](#), [getVectorHeading\(\)](#), [getVectorUnits\(\)](#), [getValueOne\(\)](#), [getValueTwo\(\)](#), [getValueThree\(\)](#)

Here is the call graph for this function:



The documentation for this class was generated from the following files:

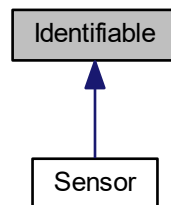
- [DataSource.h](#)
- [DataSource.cpp](#)

4.4 Identifiable Class Reference

A class which models a Sensor/peripheral as an identifiable entity.

```
#include <Identifiable.h>
```

Inheritance diagram for Identifiable:



Public Member Functions

- `Identifiable` (const int ChipSelect)
A constructor.
- bool `hasIdentityChanged` (void)
Checks to see whether the `Identity` in local memory is different to the `Identity` advertised by peripheral.
- void `updateIdentity` (void)
Loads the `Identity` advertised by the peripheral into local memory.
- int `getIdNumber` (void)
Gets the identity number of the attached peripheral.
- void `getSensorName` (char name[IDENTITY_SENSOR_NAME_LENGTH])
Gets the sensor name of the attached peripheral.

4.4.1 Detailed Description

A class which models a Sensor/peripheral as an identifiable entity.

This class models a peripheral as an identifiable entity with ID number and string name. The class allows for the identity to be loaded from the peripheral and interrogated.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 Identifiable()

```
Identifiable (
    const int ChipSelect )
```

A constructor.

Constructor for class which loads the identity of any connected SPI peripheral into local memory.

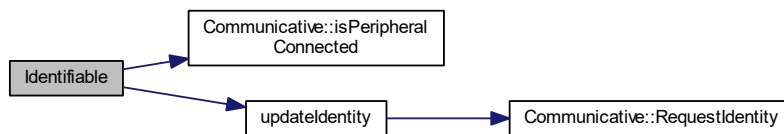
Parameters

<i>The</i>	Slave Select pin of the SPI peripheral in question.
------------	---

See also

[updateIdentity](#).

Here is the call graph for this function:

**4.4.3 Member Function Documentation****4.4.3.1 getIDNumber()**

```
int getIDNumber (
    void )
```

Gets the identity number of the attached peripheral.

Reports the identity number of the peripheral currently stored in local memory.

Returns

The sensor ID number.

4.4.3.2 getSensorName()

```
void getSensorName (
    char name[IDENTITY_SENSOR_NAME_LENGTH] )
```

Gets the sensor name of the attached peripheral.

Reports the sensor name of the peripheral currently stored in local memory.

Parameters

<i>name</i>	is the character array into which the sensor name is loaded.
-------------	--

4.4.3.3 hasIdentityChanged()

```
bool hasIdentityChanged (
    void )
```

Checks to see whether the [Identity](#) in local memory is different to the [Identity](#) advertised by peripheral.

Returns

True if the stored [Identity](#) is different than the [Identity](#) advertised by the peripheral.

See also

[Identity](#)

Here is the call graph for this function:

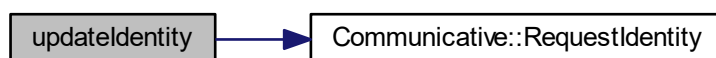


4.4.3.4 updateIdentity()

```
void updateIdentity (
    void )
```

Loads the [Identity](#) advertised by the peripheral into local memory.

Here is the call graph for this function:



Here is the caller graph for this function:



The documentation for this class was generated from the following files:

- [Identifiable.h](#)
- [Identifiable.cpp](#)

4.5 Identity Struct Reference

Type used to convey the Slave identity.

```
#include <SPI_InstructionSet.h>
```

Public Member Functions

- [Identity](#) & [operator=](#) (const volatile [Identity](#) &rhs) volatile
- volatile [Identity](#) & [operator=](#) (const [Identity](#) &rhs) volatile

Data Fields

- char [SensorName](#) [[IDENTITY_SENSOR_NAME_LENGTH](#)]
String name of the slave. Used for informative reporting to user.
- int [sensorID](#)
Single byte identification number.
- int [sensorChipSelect](#)
SPI chip select of the peripheral in question. Used by slave, but used by [Master](#).

4.5.1 Detailed Description

Type used to convey the Slave identity.

4.5.2 Member Function Documentation

4.5.2.1 operator=() [1/2]

```
Identity& operator= (
    const volatile Identity & rhs ) volatile [inline]
```

4.5.2.2 operator=() [2/2]

```
volatile Identity& operator= (
    const Identity & rhs ) volatile [inline]
```

4.5.3 Field Documentation

4.5.3.1 sensorChipSelect

```
int sensorChipSelect
```

SPI chip select of the peripheral in question. Used by slave, but used by [Master](#).

4.5.3.2 sensorID

```
int sensorID
```

Single byte identification number.

4.5.3.3 SensorName

```
char SensorName[IDENTITY_SENSOR_NAME_LENGTH]
```

String name of the slave. Used for informative reporting to user.

The documentation for this struct was generated from the following file:

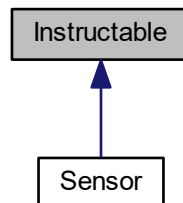
- [SPI_InstructionSet.h](#)

4.6 Instructable Class Reference

A class which models a Sensor/peripheral as entity which can receive commands.

```
#include <Instructable.h>
```

Inheritance diagram for Instructable:



Public Member Functions

- `Instructable` (const int ChipSelect)
A constructor.
- bool `issueCommand` (mInstruct)
Issues a command to the peripheral.
- bool `issueCommand` (mInstruct, int)
Issues a command to the peripheral.
- bool `issueCommand` (mInstruct, float)
Issues a command to the peripheral.
- bool `issueCommand` (mInstruct, int, float)
Issues a command to the peripheral.
- bool `areYouConnected` (void)
Checks to see whether the peripheral is connected.

4.6.1 Detailed Description

A class which models a Sensor/peripheral as entity which can receive commands.

This class models a peripheral as an entity which can be issued commands. The commands issued are elements of the set defined by the mInstruct type. Each instruction can be accompanied by an integer and/or float as required to act as parameters to qualify the command. For example, if the peripheral is commanded to pause for an interval, the integer parameter is used by the peripheral to determine the length of time for which to pause.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 Instructable()

```
Instructable (
    const int ChipSelect )
```

A constructor.

Parameters

<i>The</i>	Slave Select pin of the SPI peripheral in question.
------------	---

4.6.3 Member Function Documentation

4.6.3.1 `areYouConnected()`

```
bool areYouConnected (
    void )
```

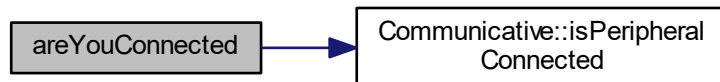
Checks to see whether the peripheral is connected.

Initiates handshake and nop transaction with the sensor to ensure that it is connected and responding appropriately.

Returns

True if the peripheral is connected and communicating effectively.

Here is the call graph for this function:



4.6.3.2 `issueCommand()` [1/4]

```
bool issueCommand (
    mInstruct Command )
```

Issues a command to the peripheral.

Sends a command which is an element of the `mInstruct` type.

Parameters

<i>mInstruct</i>	is the command issued to the peripheral.
------------------	--

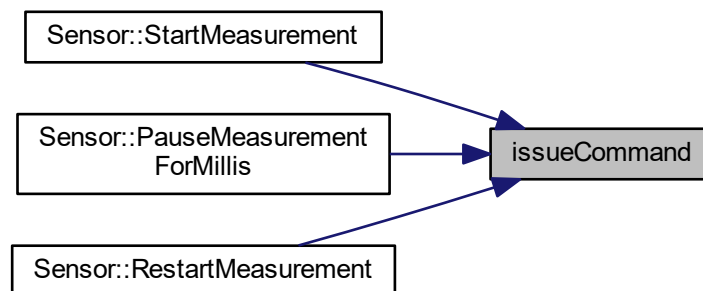
Returns

True if the peripheral acknowledges the command.

Here is the call graph for this function:



Here is the caller graph for this function:

**4.6.3.3 issueCommand()** [2/4]

```

bool issueCommand (
    mInstruct Command,
    int intParam )
  
```

Issues a command to the peripheral.

Sends a command which is an element of the `mInstruct` type qualified by an integer parameter (generally used to instruct on wait time or similar).

Parameters

<i>mInstruct</i>	is the command issued to the peripheral and int is the integer qualifier.
------------------	---

Returns

True if the peripheral acknowledges the command.

Here is the call graph for this function:

**4.6.3.4 issueCommand()** [3/4]

```
bool issueCommand (
    mInstruct Command,
    float floatParam )
```

Issues a command to the peripheral.

Sends a command which is an element of the `mInstruct` type qualified by a float parameter.

Parameters

<i>mInstruct</i>	is the command issued to the peripheral and float is the floating point qualifier.
------------------	--

Returns

True if the peripheral acknowledges the command.

Here is the call graph for this function:



4.6.3.5 `issueCommand()` [4 / 4]

```
bool issueCommand (
    mInstruct Command,
    int intParam,
    float floatParam )
```

Issues a command to the peripheral.

Sends a command which is an element of the `mInstruct` type qualified by an integer parameter (generally used to instruct on wait time or similar) and a floating point parameter.

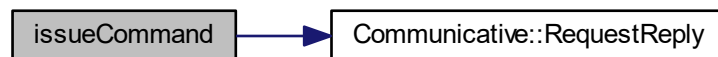
Parameters

<i>mInstruct</i>	is the command issued to the peripheral, <i>int</i> is the integer qualifier and <i>float</i> is the floating point qualifier.
------------------	--

Returns

True if the peripheral acknowledges the command.

Here is the call graph for this function:



The documentation for this class was generated from the following files:

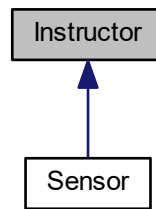
- [Instructable.h](#)
- [Instructable.cpp](#)

4.7 Instructor Class Reference

A class which models a Sensor/peripheral as entity which can issue instructions to the master.

```
#include <Instructor.h>
```

Inheritance diagram for Instructor:



Public Member Functions

- **Instructor** (const int ChipSelect)
Constructor.
- int **howManyInstructions** (void)
Asks the [Sensor](#) how many instructions there are in a measurement cycle.
- int **howLongShouldIWait** (void)
Asks the sensor how long the master should pause for. Deprecated.
- void **loadNextCommand** (void)
Fetches the next instruction issued by the sensor in its instruction cycle.
- void **getCurrentCommandString** (char[SLAVE_COMMMAND_STRING_LENGTH])
Returns the character array which qualifies the instruction currently loaded into local memory as issued by the [Sensor](#).
- int **getCurrentCommandInt** (void)
Returns the integer which qualifies the instruction currently loaded into local memory as issued by the [Sensor](#).
- float **getCurrentCommandFloat** (void)
Returns the float which qualifies the instruction currently loaded into local memory as issued by the [Sensor](#).
- **sinstruct** **getCurrentCommandInstruction** (void)
Returns the sinstruct object which defines the instruction currently loaded into local memory as issued by the [Sensor](#).

4.7.1 Detailed Description

A class which models a Sensor/peripheral as entity which can issue instructions to the master.

This class models a peripheral as an entity which can issue commands to the master. The premise is that a sensor peripheral will conduct a measurement by cycling through a number of steps. At each step the sensor may wish for the [Master](#) to perform certain actions, such as display a message to the user, pause for a certain period of time or wait until the user has acknowledged an instruction by button press. In general, the master is expected to iterate through the instruction set, loading an instruction each iteration and repsonding appropriately. i.e. follow the procedure: [howManyInstructions\(\)](#)->start loop->[loadNextCommand\(\)](#)->React to command->repeat until all instructions have been processed.

4.7.2 Constructor & Destructor Documentation

4.7.2.1 Instructor()

```
Instructor (
    const int ChipSelect )
```

Constructor.

Parameters

<i>ChipSelect</i>	is the Slave Select pin of the SPI peripheral in question.
-------------------	--

4.7.3 Member Function Documentation

4.7.3.1 getCurrentCommandFloat()

```
float getCurrentCommandFloat (
    void )
```

Returns the float which qualifies the instruction currently loaded into local memory as issued by the [Sensor](#).

The float which qualifies the instruction issued by the sensor is generally used to either augment the information displayed to the user to instruct the master as to how it should carry out the request of the slave.

4.7.3.2 getCurrentCommandInstruction()

```
sInstruct getCurrentCommandInstruction (
    void )
```

Returns the sInstruct object which defines the instruction currently loaded into local memory as issued by the [Sensor](#).

Elements of the sInstruct type define all the potential instructions which can be issued by a Slave.

Returns

The instruction issued by the slave.

4.7.3.3 getCurrentCommandInt()

```
int getCurrentCommandInt (
    void )
```

Returns the integer which qualifies the instruction currently loaded into local memory as issued by the [Sensor](#).

The integer which qualifies the instruction issued by the sensor is generally used to either augment the information displayed to the user to instruct the master as to how it should carry out the request of the slave, such as how long to pasue for.

Returns

The integer which qualifies the instruction.

4.7.3.4 getCurrentCommandString()

```
void getCurrentCommandString (
    char Instruction[SLAVE_COMMAND_STRING_LENGTH] )
```

Returns the character array which qualifies the instruction currently loaded into local memory as issued by the [Sensor](#).

The character array (string) issued by the [Sensor](#) is generally intended to be displayed to the user, to update the user on the progress of the measurement procedure or instruct the user on the next step in the measurement procedure, such as inserting the probe into the measurement environment.

Parameters

<i>char</i>	is the character array into which the instruction string is loaded.
-------------	---

4.7.3.5 howLongShouldIWait()

```
int howLongShouldIWait (
    void )
```

Asks the sensor how long the master should pause for. Deprecated.

Returns

The duration, in milliseconds, the master should pause for.

Here is the call graph for this function:



4.7.3.6 howManyInstructions()

```
int howManyInstructions (
    void )
```

Asks the [Sensor](#) how many instructions there are in a measurement cycle.

In general, the master is expected to iterate through the instruction set.

Returns

The number of instructions in a measurement cycle.

Here is the call graph for this function:

**4.7.3.7 loadNextCommand()**

```
void loadNextCommand (
    void )
```

Fetches the next instruction issued by the sensor in its instruction cycle.

Fetches the next instruction from the [Sensor](#) and loads it into local memory. Instructions are issued in the [sCmd](#) type and are therefore constitute an element of the `sInstruct` instruction set, qualified by a character array (string), integer and float. Here is the call graph for this function:



The documentation for this class was generated from the following files:

- [Instructor.h](#)
- [Instructor.cpp](#)

4.8 Master Class Reference

A monolithic class to encapsulate and abstract the slave's communication with the master.

```
#include <Master.h>
```

Public Member Functions

- **Master** (const int SensorIDNumber, const char SensorName[], const char InstructionSet[][SLAVE_COMMAND_STRING_LENGTH], const int NumberOfInstructions, const **sInstruct** MasterInstructionSet[], const int intParams[], const float floatParams[])
- **Master** (void)
- **~Master** (void)
- **Master** (volatile const **Master** &)
- volatile **Master** & **operator=** (const **Master** &rhs) volatile
- volatile **Master** & **operator=** (volatile const **Master** &rhs) volatile
- void **SETUP** (const int SensorIDNumber, volatile char SensorName[], volatile char InstructionSet[][SLAVE_COMMAND_STRING_LENGTH], const int NumberOfInstructions, volatile **sInstruct** MasterInstructionSet[], volatile int intParams[], volatile float floatParams[]) volatile
Set up of the communication mechanism.
- void **SPISetup** (void) volatile
Sets up SPI and attaches interrupt.
- bool **Handshake** (void) volatile
Manages the handshake component of any transaction.
- **mCmd loadRequest** (void) volatile
Retrieve the request sent by the master.
- **mInstruct getCurrentInstruction** (void) volatile
Returns the mInstruct component of the most recently loaded request sent by the master.
- int **getCurrentInstructionIntParameter** (void) volatile
Returns the integer parameter of the most recently loaded request sent by the master.
- float **getCurrentInstructionFloatParameter** (void) volatile
Returns the floating point parameter of the most recently loaded request sent by the master.
- void **sendReply** (const **sCmd** Reply) volatile
Send an sCmd object in reply to the request recieved from Master.
- void **sendReply** (const **sInstruct** Instruction) volatile
Send a reply to the request recieved from master.
- void **sendReply** (const **sInstruct** Instruction, volatile char InstructionString[SLAVE_COMMAND_STRING_LENGTH]) volatile
Send a reply to the request recieved from master.
- void **sendReply** (const **sInstruct** Instruction, const int iParam) volatile
Send a reply to the request recieved from master.
- void **sendReply** (const **sInstruct** Instruction, const float fParam) volatile
Send a reply to the request recieved from master.
- void **sendReply** (const **sInstruct** Instruction, const int iParam, const int fParam) volatile
Send a reply to the request recieved from master.
- void **sendReply** (**sInstruct** Instruction, int iParam, volatile char InstructionString[SLAVE_COMMAND_STRING_LENGTH]) volatile
Send a reply to the request recieved from master.
- void **sendReply** (**sInstruct** Instruction, float fParam, volatile char InstructionString[SLAVE_COMMAND_STRING_LENGTH]) volatile
Send a reply to the request recieved from master.
- void **sendReply** (**sInstruct** Instruction, int iParam, float fParam, volatile char InstructionString[SLAVE_COMMAND_STRING_LENGTH]) volatile
Send a reply to the request recieved from master.
- void **sendData** (void) volatile
Sends the local Data object to the master in reply to appropriate request.
- void **sendIdentity** (void) volatile
Sends slave Identity object to the master in reply to appropriate request.

- bool `PushMeasurementVector` (const `MeasurementVectors` VectorNumber, const float Measurement) volatile
Pushes a data point onto one of the data vectors.
- bool `PopMeasurementVector` (`MeasurementVectors` VectorNumber) volatile
Pops a data point from the tail end of a designated vector.
- void `ClearMeasurementVector` (`MeasurementVectors` VectorNumber) volatile
Clears all data points from a particular vector.
- void `setMeasurementVectorHeading` (`MeasurementVectors` VectorNumber, volatile char Heading[`ROW_HEADING_LENGTH`]) volatile
Sets the string heading assigned to a particular data vector.
- void `setMeasurementVectorUnits` (`MeasurementVectors` VectorNumber, volatile char Units[`ROW_UNIT_LENGTH`]) volatile
Sets the string Units assigned to a particular data vector.
- bool `isThereData` (void) volatile
Checks to see whether any data has been pushed to any of the data vectors locally.
- void `sendTotalNumOfInstructions` (void) volatile
Sends a reply to `Master` specifying the total number of instructions in a measurement procedure.
- int `getCurrentInstructionNumber` (void) volatile
Gets the value of the counter which tracks the current instruction number throughout the measurement procedure.
- bool `sendNextUserInstruction` (void) volatile
Sends the next instruction in the measurement cycle to the master.
- void `resendCurrentUserInstruction` (void) volatile
Resends the current instruction in the measurement procedure to the master.
- void `restartUserInstructionCycle` (void) volatile
Restarts the measurement procedure.
- void `beginMeasurement` (void) volatile
Updates state to indicate that `Master` has requested the initiation of the measurement procedure.
- bool `shallStart` (void) volatile
Checks whether the `beginMeasurement()` method has been called.

4.8.1 Detailed Description

A monolithic class to encapsulate and abstract the slave's communication with the master.

A single transaction is characterised by the following flow of control: Clear SS -> Enter IRQ -> Recieve '?' from master -> send 'ACK' (0x06) to master -> recieve request `mCmd` -> send `sCmd`/Data/Identity as expected -> exit IRQ. Where `mCmd`, `sCmd`, `Data` and `Identity` are structures defined as types. The initial Recieve '?' -> send 'ACK' is known as the handshake. This class' responsibilities include initialising and handling the SPI, attaching the interrupt and providing the IRQ, managing the handshake and encapsulating the response mechanism. Note the use of a volatile interface to allow for safe use of the interrupt.

4.8.2 Constructor & Destructor Documentation

4.8.2.1 Master() [1/3]

```
Master (
    const int SensorIDNumber,
    const char SensorName[],
    const char InstructionSet[][SLAVE_COMMAND_STRING_LENGTH],
    const int NumberOfInstructions,
    const sInstruct MasterInstructionSet[],
    const int intParams[],
    const float floatParams[] )
```

4.8.2.2 Master() [2/3]

```
Master (
    void )
```

4.8.2.3 ~Master()

```
~Master (
    void )
```

4.8.2.4 Master() [3/3]

```
Master (
    volatile const Master & rhs )
```

4.8.3 Member Function Documentation

4.8.3.1 beginMeasurement()

```
void beginMeasurement (
    void ) volatile
```

Updates state to indicate that **Master** has requested the initiation of the measurement procedure.

Designed to allow the IRQ to update the slave state so that the procedural code in `main()` can initiate the measurement procedure. Here is the call graph for this function:



4.8.3.2 ClearMeasurementVector()

```
void ClearMeasurementVector (
    MeasurementVectors VectorNumber ) volatile
```

Clears all data points from a particular vector.

Parameters

<i>VectorNumber</i>	is the enumerated reference to the row/vector in the data array being accessed.
---------------------	---

4.8.3.3 getCurrentInstruction()

```
mInstruct getCurrentInstruction (
    void ) volatile
```

Returns the mInstruct component of the most recently loaded request sent by the master.

4.8.3.4 getCurrentInstructionFloatParameter()

```
float getCurrentInstructionFloatParameter (
    void ) volatile
```

Returns the floating point parameter of the most recently loaded request sent by the master.

4.8.3.5 getCurrentInstructionIntParameter()

```
int getCurrentInstructionIntParameter (
    void ) volatile
```

Returns the integer parameter of the most recently loaded request sent by the master.

4.8.3.6 getCurrentInstructionNumber()

```
int getCurrentInstructionNumber (
    void ) volatile
```

Gets the value of the counter which tracks the current instruction number throughout the measurement procedure.

4.8.3.7 Handshake()

```
bool Handshake (
    void ) volatile
```

Manages the handshake component of any transaction.

Manages the handshake between master and slave; defined by: Recieve '?' -> send 'ACK' (0x06).

Returns

True if the handshake was successful.

Here is the caller graph for this function:



4.8.3.8 isThereData()

```
bool isThereData (
    void ) volatile
```

Checks to see whether any data has been pushed to any of the data vectors locally.

Returns

True if any data points exist in the data vectors locally.

4.8.3.9 loadRequest()

```
mCmd loadRequest (
    void ) volatile
```

Retrieve the request sent by the master.

After each handshake, the master will proceed to send a request, defined by an `mCmd` object. This function must run after each handshake. This function reassembles the `mCmd` request sent by the master, byte by byte and stores it in local memory.

Returns

the `mCmd` object sent by the master.

Here is the caller graph for this function:

**4.8.3.10 operator=()** [1/2]

```
volatile Master& operator= (
    const Master & rhs ) volatile [inline]
```

4.8.3.11 operator=() [2/2]

```
volatile Master& operator= (
    volatile const Master & rhs ) volatile [inline]
```

4.8.3.12 PopMeasurementVector()

```
bool PopMeasurementVector (
    MeasurementVectors VectorNumber ) volatile
```

Pops a data point from the tail end of a designated vector.

Removes the most recent data point from the vector in question by marking said data slot writeable to be the next `PushMeasurementVector()`.

Parameters

<i>VectorNumber</i>	is the enumerated reference to the row/vector in the data array being accessed.
---------------------	---

Returns

False if the vector/row in question is empty.

4.8.3.13 PushMeasurementVector()

```
bool PushMeasurementVector (
    const MeasurementVectors VectorNumber,
    const float Measurement ) volatile
```

Pushes a data point onto one of the data vectors.

Adds a data point to the next available slot in a particular vector until the vector (row) in the data array is full.

Parameters

<i>VectorNumber</i>	is the enumerated reference to the row/vector in the data array being accessed.
<i>Measurement</i>	is the data point to be stored.

Returns

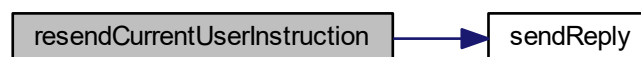
False if the vector/row in question is full. (See DATA_ROW_LENGTH).

4.8.3.14 resendCurrentUserInstruction()

```
void resendCurrentUserInstruction (
    void ) volatile
```

Resends the current instruction in the measurement procedure to the master.

This method will NOT advance the measurement cycle. Here is the call graph for this function:



4.8.3.15 restartUserInstructionCycle()

```
void restartUserInstructionCycle (
    void ) volatile
```

Restarts the measurement procedure.

Restarts the measurement procedure such that the next instruction sent will be the first instruction in the measurement cycle.

4.8.3.16 sendData()

```
void sendData (
    void ) volatile
```

Sends the local [Data](#) object to the master in reply to appropriate request.

Sends [Data](#) object to the master in reponse to a request made by the master for [Data](#). NOTE: This must be used as the response to the appropriate mInstruct request. In paricular, (mInstruct)SendDataPlease

4.8.3.17 sendIdentity()

```
void sendIdentity (
    void ) volatile
```

Sends slave [Identity](#) object to the master in reply to appropriate request.

Sends the slave [Identity](#) object in response to an appropriate request made by master. NOTE: This must be used as the response to the appropriate mInstruct request. In paricular, (mInstruct)WhoAreYou

4.8.3.18 sendNextUserInstruction()

```
bool sendNextUserInstruction (
    void ) volatile
```

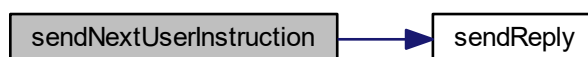
Sends the next instruction in the measurement cycle to the master.

Sends the next instruction in the measurement cycle array in reponse to the appropriate request from master. Specifically, (mInstruct)NextCommandPlease. This method will auto-advance the measurement cycle to the next instruction and re-define the 'current instruction'.

Returns

False if the final instruction in the measurement procedure has already been sent and the measurement procedure is now complete.

Here is the call graph for this function:

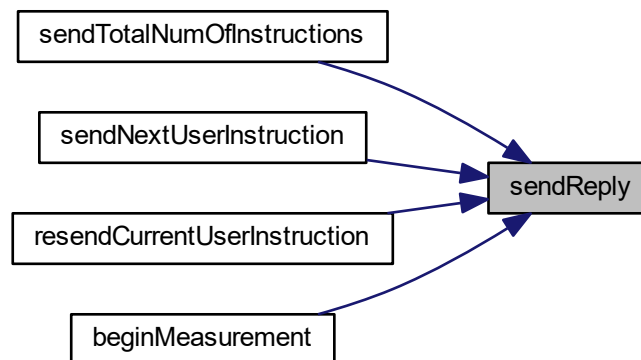


4.8.3.19 sendReply() [1/9]

```
void sendReply (
    const sCmd Reply ) volatile
```

Send an `sCmd` object in reply to the request recieved from `Master`.

Here is the caller graph for this function:



4.8.3.20 sendReply() [2/9]

```
void sendReply (
    const sInstruct Instruction ) volatile
```

Send a reply to the request recieved from master.

Assembles the `sCmd` object from the supplied parameters. Overloaded.

4.8.3.21 sendReply() [3/9]

```
void sendReply (
    const sInstruct Instruction,
    volatile char InstructionString[SLAVE_COMMAND_STRING_LENGTH] ) volatile
```

Send a reply to the request recieved from master.

Assembles the `sCmd` object from the supplied parameters. Overloaded. Note the need for the `char*` to be defined locally as `volatile char[]`. Cannot pass string literals.

4.8.3.22 sendReply() [4/9]

```
void sendReply (
    const sInstruct Instruction,
    const int iParam ) volatile
```

Send a reply to the request recieved from master.

Assembles the sCmd object from the supplied parameters. Overloaded.

4.8.3.23 sendReply() [5/9]

```
void sendReply (
    const sInstruct Instruction,
    const float fParam ) volatile
```

Send a reply to the request recieved from master.

Assembles the sCmd object from the supplied parameters. Overloaded.

4.8.3.24 sendReply() [6/9]

```
void sendReply (
    const sInstruct Instruction,
    const int iParam,
    const int fParam ) volatile
```

Send a reply to the request recieved from master.

Assembles the sCmd object from the supplied parameters. Overloaded.

4.8.3.25 sendReply() [7/9]

```
void sendReply (
    sInstruct Instruction,
    int iParam,
    volatile char InstructionString[SLAVE_COMMAND_STRING_LENGTH] ) volatile
```

Send a reply to the request recieved from master.

Assembles the sCmd object from the supplied parameters. Overloaded. Note the need for the char* to be defined locally as volatile char[]. Cannot pass string literals.

4.8.3.26 sendReply() [8/9]

```
void sendReply (
    sInstruct Instruction,
    float fParam,
    volatile char InstructionString[SLAVE_COMMAND_STRING_LENGTH] ) volatile
```

Send a reply to the request recieved from master.

Assembles the sCmd object from the supplied parameters. Overloaded. Note the need for the char* to be defined locally as volatile char[]. Cannot pass string literals.

4.8.3.27 sendReply() [9/9]

```
void sendReply (
    sInstruct Instruction,
    int iParam,
    float fParam,
    volatile char InstructionString[SLAVE_COMMAND_STRING_LENGTH] ) volatile
```

Sends a reply to the request recieved from master.

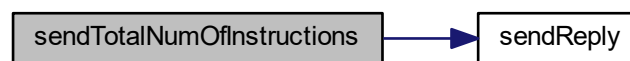
Assembles the `sCmd` object from the supplied parameters. Overloaded. Note the need for the `char*` to be defined locally as `volatile char[]`. Cannot pass string literals.

4.8.3.28 sendTotalNumOfInstructions()

```
void sendTotalNumOfInstructions (
    void ) volatile
```

Sends a reply to `Master` specifying the total number of instructions in a measurement procedure.

Here is the call graph for this function:

**4.8.3.29 setMeasurementVectorHeading()**

```
void setMeasurementVectorHeading (
    MeasurementVectors VectorNumber,
    volatile char Heading[ROW_HEADING_LENGTH] ) volatile
```

Sets the string heading assigned to a paritcular data vector.

NOTE: The Heading parameter must be declared locally as `volatile char[]`. Literal strings cannot be passed to this function.

Parameters

<i>VectorNumber</i>	is the enumerated reference to the row/vector in the data array in question.
<i>Heading[]</i>	is the character array containing the string heading.

4.8.3.30 setMeasurementVectorUnits()

```
void setMeasurementVectorUnits (
    MeasurementVectors VectorNumber,
    volatile char Units[ROW_UNIT_LENGTH] ) volatile
```

Sets the string Units assigned to a paritcular data vector.

NOTE: The Units parameter must be declared locally as volatile char[]. Literal strings cannot be passed to this function.

Parameters

<i>VectorNumber</i>	is the enumerated reference to the row/vector in the data array in question.
<i>Units[]</i>	is the character array containing the string heading.

4.8.3.31 SETUP()

```
void SETUP (
    const int SensorIDNumber,
    volatile char SensorName[],
    volatile char InstructionSet[][SLAVE_COMMAND_STRING_LENGTH],
    const int NumberOfInstructions,
    volatile sInstruct MasterInstructionSet[],
    volatile int intParams[],
    volatile float floatParams[] ) volatile
```

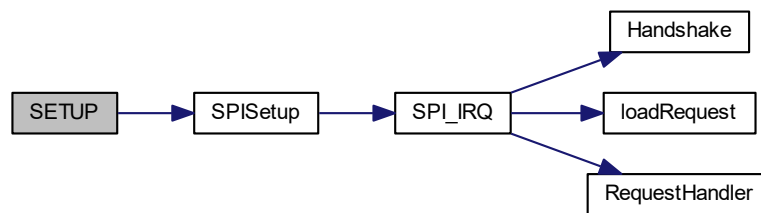
Set up of the commuication mechanism.

Intialised the auto-instantiated **Master** object, attaches the IRQ and sets up SPI communications.

Parameters

<i>SensorIDNumber</i>	is the identity number of th slave.
<i>SensorName</i>	is the string name of the slave.
<i>InstructionSet</i>	is the array of strings associated with each instruction in the measuremnt cycle. Generally to be displayed to the user.
<i>NumberOfInstructions</i>	is the number of instructions in a single measurement cycle/procedure.
<i>MasterInstructionSet</i>	is the array of sInstruct objects associated with each instruction in the measurement cycle. Defines the required action by the master for each step of the measurement procedure.
<i>IntParams</i>	is the array of integer parameter associated with each instruction.
<i>FloatParams</i>	is the array of floating point parameters accosiated with each instruction.

Here is the call graph for this function:



4.8.3.32 shallIStart()

```
bool shallIStart (  
    void ) volatile
```

Checks whether the [beginMeasurement\(\)](#) method has been called.

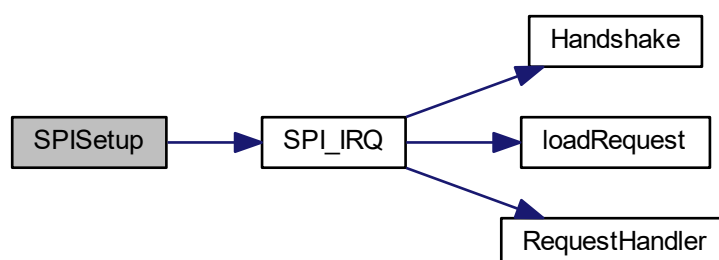
Allows procedural code in `main()` to determine whether the system state has changed during an interrupt in response to a request by the master to initiate the measurement procedure.

4.8.3.33 SPISetup()

```
void SPISetup (  
    void ) volatile
```

Sets up SPI and attaches interrupt.

Here is the call graph for this function:



Here is the caller graph for this function:



The documentation for this class was generated from the following files:

- [Master.h](#)
- [Master.cpp](#)

4.9 mCmd Struct Reference

Type used by master to send requests to slave.

```
#include <SPI_InstructionSet.h>
```

Public Member Functions

- [mCmd](#) ([mInstruct](#) Instruct, int i, float f)
- [mCmd](#) ()
- [mCmd](#) (volatile [mCmd](#) &rhs)
- [mCmd](#) & [operator=](#) (const volatile [mCmd](#) &rhs) volatile
- volatile [mCmd](#) & [operator=](#) (const [mCmd](#) &rhs) volatile

Data Fields

- [mInstruct](#) [Instruction](#)
Instruction to slave.
- int [iParam](#)
Integer paramter which qualifies the instruction.
- float [fParam](#)
Floating point parameter which wualifies the instruction.

4.9.1 Detailed Description

Type used by master to send requests to slave.

Each transaction, following the initial handshake, the master will send a request and the slave will send a reply. Requests made by the master will always take the form of an [mCmd](#) object, which contains a parameterised instance of the instruction set, [mInstruct](#), which defines the class of repsonse expected by the slave.

4.9.2 Constructor & Destructor Documentation

4.9.2.1 mCmd() [1/3]

```
mCmd (
    mInstruct Instruct,
    int i,
    float f ) [inline]
```

4.9.2.2 mCmd() [2/3]

```
mCmd ( ) [inline]
```

4.9.2.3 mCmd() [3/3]

```
mCmd (
    volatile mCmd & rhs ) [inline]
```

4.9.3 Member Function Documentation

4.9.3.1 operator=() [1/2]

```
mCmd& operator= (
    const volatile mCmd & rhs ) volatile [inline]
```

4.9.3.2 operator=() [2/2]

```
volatile mCmd& operator= (
    const mCmd & rhs ) volatile [inline]
```

4.9.4 Field Documentation

4.9.4.1 fParam

```
float fParam
```

Floating point parameter which wualifies the instruction.

4.9.4.2 Instruction

```
mInstruct Instruction
```

Instruction to slave.

Defines the request made of the slave by the master during any transacion. mInstruct defines a finite set of requests which the master can make of the slave.

4.9.4.3 iParam

```
int iParam
```

Integer paramter which qualifies the instruction.

The documentation for this struct was generated from the following file:

- [SPI_InstructionSet.h](#)

4.10 sCmd Struct Reference

Type used by slave to send reply to master,.

```
#include <SPI_InstructionSet.h>
```

Public Member Functions

- [sCmd](#) & [operator=](#) (const volatile [sCmd](#) &rhs) volatile
- volatile [sCmd](#) & [operator=](#) (const [sCmd](#) &rhs) volatile

Data Fields

- [sInstruct](#) [Instruction](#)
Instruction to master.
- char [sParam](#) [[SLAVE_COMMMAND_STRING_LENGTH](#)]
String parameter which qualifies the instruction. Often used to convey instructions which are to be displayed to the user.
- int [iParam](#)
Integer parameter which qualifies the instruction.
- float [fParam](#)
Floating point parameter which qualifies the instruction.

4.10.1 Detailed Description

Type used by slave to send reply to master,.

Each transaction, following the initial handshake, the master will send a request and the slave will send a reply. Replied made by the slave are generally in the form of an [sCmd](#) object, which contains a parameterised instance of the slave instruction set, [sInstruct](#), which defines the action which the slave requires the master to carry out. Other acceptable replies to particular requests from the master are [Data](#) and [Identity](#) objects.

4.10.2 Member Function Documentation

4.10.2.1 [operator=\(\)](#) [1/2]

```
sCmd& operator= (
    const volatile sCmd & rhs ) volatile [inline]
```

4.10.2.2 [operator=\(\)](#) [2/2]

```
volatile sCmd& operator= (
    const sCmd & rhs ) volatile [inline]
```

4.10.3 Field Documentation

4.10.3.1 [fParam](#)

```
float fParam
```

Floating point parameter which qualifies the instruction.

4.10.3.2 [Instruction](#)

```
sInstruct Instruction
```

Instruction to master.

Defines the reply made by the slave in repsonse to the request posed by the master during a single transaction. Used to confirm commands issued by the master or issue commands to the master.

4.10.3.3 iParam

```
int iParam
```

Integer parameter which qualifies the instruction.

4.10.3.4 sParam

```
char sParam[SLAVE_COMMAND_STRING_LENGTH]
```

String parameter which qualifies the instruction. Often used to convey instructions which are to be displayed to the user.

The documentation for this struct was generated from the following file:

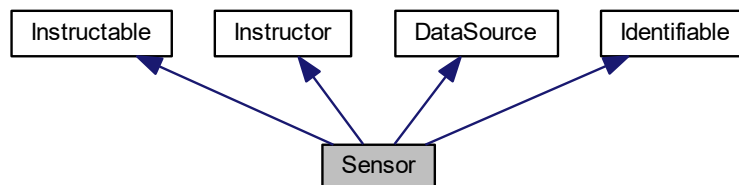
- [SPI_InstructionSet.h](#)

4.11 Sensor Class Reference

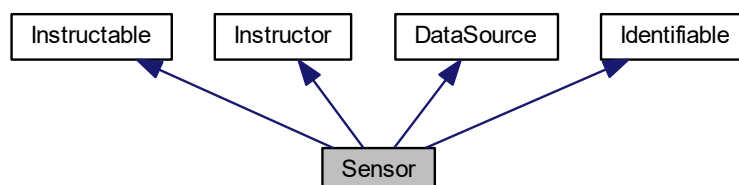
A class which models a Sensor/peripheral.

```
#include <Sensor.h>
```

Inheritance diagram for Sensor:



Collaboration diagram for Sensor:



Public Member Functions

- [Sensor](#) (const int ChipSelect)
Constructor.
- int [StartMeasurement](#) (void)
Instruct the sensor to initiate the measurement procedure.
- int [PauseMeasurementForMillis](#) (int)
Instruct the sensor to pause the measurement procedure for a brief period.
- int [RestartMeasurement](#) (void)
Instruct the sensor to restart the measurement procedure.

4.11.1 Detailed Description

A class which models a Sensor/peripheral.

This class models a sensor peripheral as an entity which has a queriable [Identity](#), is a source of [Data](#), can be issued commands and can issue a series of instructions in turn. In particular, a sensor is considered to perform a measurement procedure consisting of a series of steps; at each step, the sensor will issue instructions to the master to be acted upon and/or displayed to the user. The sensor may also require feedback from the user such as confirmation of the completion of an instruction, before proceeding to the next instruction in the measurement procedure. While the sensor dictates the flow of the measurement procedure, a master reserves the right to initiate the procedure, pause the procedure and restart the procedure.

4.11.2 Constructor & Destructor Documentation

4.11.2.1 Sensor()

```
Sensor (
    const int ChipSelect )
```

Constructor.

Parameters

<i>ChipSelect</i>	is the Slave Select pin of the SPI peripheral in question.
-------------------	--

4.11.3 Member Function Documentation

4.11.3.1 PauseMeasurementForMillis()

```
int PauseMeasurementForMillis (
    int PauseTime )
```

Instruct the sensor to pause the measurement procedure for a brief period.

Instructs the sensor to temporarily pause the measurement procedure for a period defined in milliseconds.

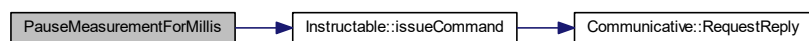
Parameters

<i>int</i>	is the number of milliseconds for which the sensor is to pause.
------------	---

Returns

True if the sensor acknowledges the request.

Here is the call graph for this function:

**4.11.3.2 RestartMeasurement()**

```
int RestartMeasurement (
    void )
```

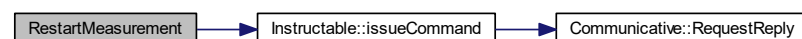
Instruct the sensor to restart the measurement procedure.

Instructs the sensor to restart the measurement procedure. The sensor will wait revert to its initial state, waiting for a [StartMeasurement\(\)](#) command before proceeding to the first instruction in the procedure.

Returns

True if the sensor acknowledges the request.

Here is the call graph for this function:



4.11.3.3 StartMeasurement()

```
int StartMeasurement (
    void )
```

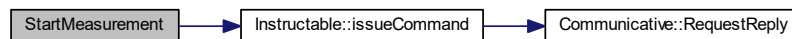
Instruct the sensor to initiate the measurement procedure.

Inform the sensor to start the measurement procedure and proceed to the first instruction step.

Returns

True if the sensor acknowledges the request.

Here is the call graph for this function:



The documentation for this class was generated from the following files:

- [Sensor.h](#)
- [Sensor.cpp](#)

4.12 UserInstructions Struct Reference

```
#include <Master.h>
```

Public Member Functions

- [UserInstructions](#) & `operator=` (const volatile [UserInstructions](#) &rhs) volatile
- volatile [UserInstructions](#) & `operator=` (const [UserInstructions](#) &rhs) volatile

Data Fields

- int [NumOfInstructions](#)
- int [InstructionCounter](#)
- char [InstructionSet](#) [MAX_USER_INSTRUCTION_NUMBER][SLAVE_COMMMAND_STRING_LENGTH]
- [sInstruct](#) [MasterInstructionSet](#) [MAX_USER_INSTRUCTION_NUMBER]
- int [iParams](#) [MAX_USER_INSTRUCTION_NUMBER]
- float [fParams](#) [MAX_USER_INSTRUCTION_NUMBER]

4.12.1 Member Function Documentation

4.12.1.1 operator=() [1/2]

```
UserInstructions& operator= (  
    const volatile UserInstructions & rhs ) volatile [inline]
```

4.12.1.2 operator=() [2/2]

```
volatile UserInstructions& operator= (  
    const UserInstructions & rhs ) volatile [inline]
```

4.12.2 Field Documentation

4.12.2.1 fParams

```
float fParams[MAX_USER_INSTRUCTION_NUMBER]
```

4.12.2.2 InstructionCounter

```
int InstructionCounter
```

4.12.2.3 InstructionSet

```
char InstructionSet[MAX_USER_INSTRUCTION_NUMBER][SLAVE_COMMAND_STRING_LENGTH]
```

4.12.2.4 iParams

```
int iParams[MAX_USER_INSTRUCTION_NUMBER]
```

4.12.2.5 MasterInstructionSet

```
sInstruct MasterInstructionSet[MAX_USER_INSTRUCTION_NUMBER]
```

4.12.2.6 NumOfInstructions

```
int NumOfInstructions
```

The documentation for this struct was generated from the following file:

- [Master.h](#)

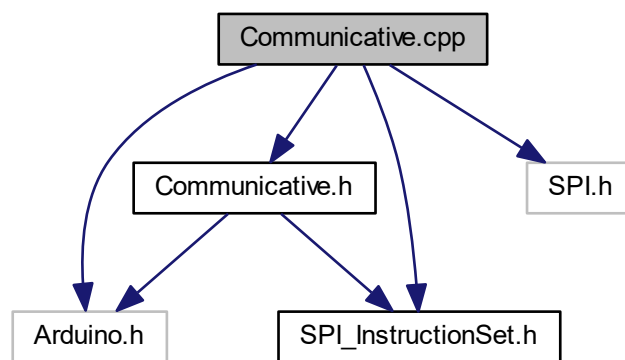
Chapter 5

File Documentation

5.1 Communicative.cpp File Reference

```
#include <Arduino.h>
#include "Communicative.h"
#include "SPI_InstructionSet.h"
#include <SPI.h>
```

Include dependency graph for Communicative.cpp:



Variables

- `const int REQUEST_DELAY_MICROS = 100`

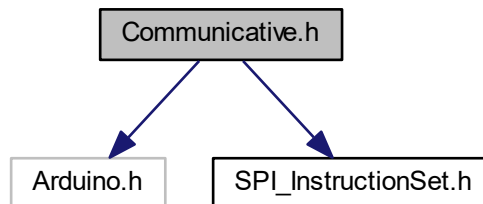
5.1.1 Variable Documentation

5.1.1.1 REQUEST_DELAY_MICROS

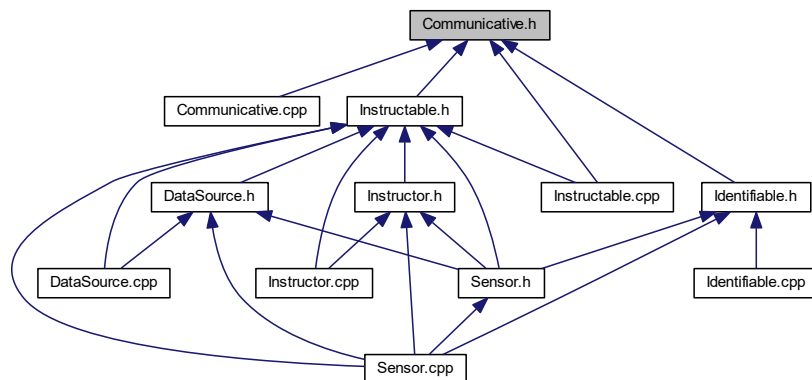
```
const int REQUEST_DELAY_MICROS = 100
```

5.2 Communicative.h File Reference

```
#include <Arduino.h>
#include "SPI_InstructionSet.h"
Include dependency graph for Communicative.h:
```



This graph shows which files directly or indirectly include this file:



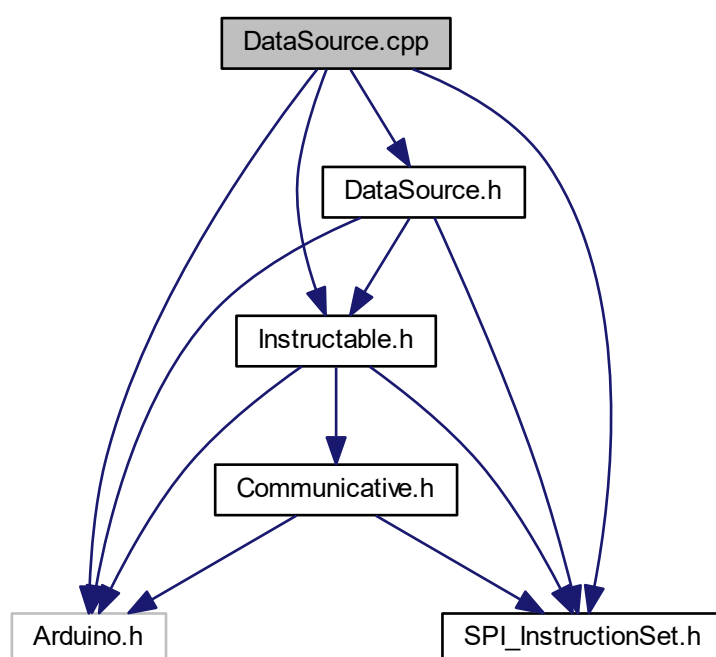
Data Structures

- class [Communicative](#)

A class to manage communication with slave module.

5.3 DataSource.cpp File Reference

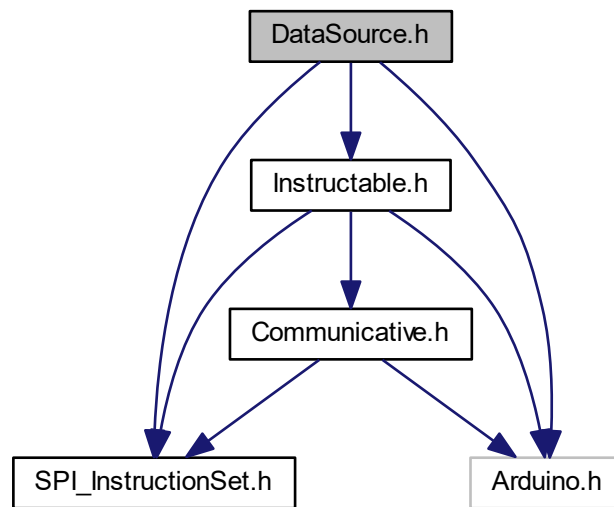
```
#include <Arduino.h>
#include "Instructable.h"
#include "DataSource.h"
#include "SPI_InstructionSet.h"
Include dependency graph for DataSource.cpp:
```



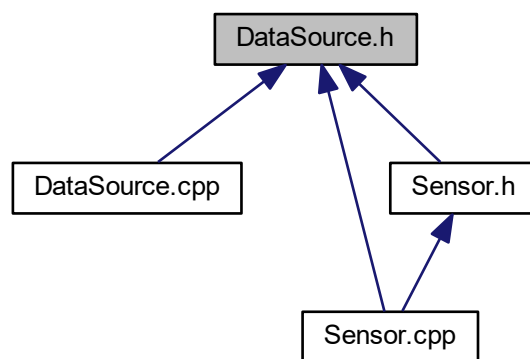
5.4 DataSource.h File Reference

```
#include <Arduino.h>
#include "SPI_InstructionSet.h"
#include "Instructable.h"
```

Include dependency graph for DataSource.h:



This graph shows which files directly or indirectly include this file:



Data Structures

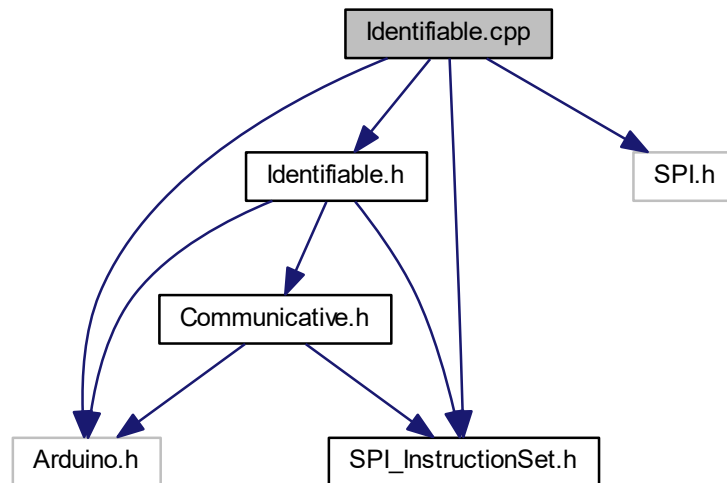
- class [DataSource](#)

A class which models a Sensor/peripheral as an entity which is a source of data.

5.5 Identifiable.cpp File Reference

```
#include <Arduino.h>
#include "Identifiable.h"
#include "SPI_InstructionSet.h"
#include <SPI.h>
```

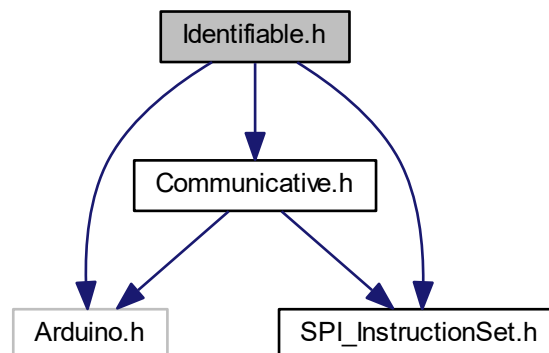
Include dependency graph for Identifiable.cpp:



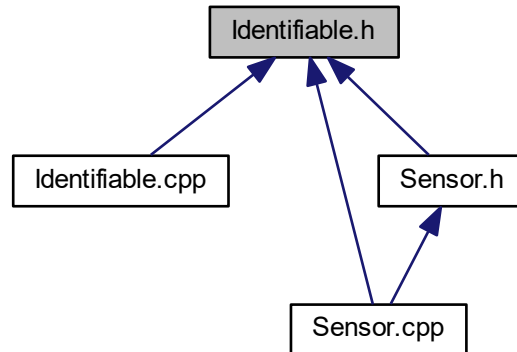
5.6 Identifiable.h File Reference

```
#include <Arduino.h>
#include "SPI_InstructionSet.h"
#include "Communicative.h"
```

Include dependency graph for Identifiable.h:



This graph shows which files directly or indirectly include this file:



Data Structures

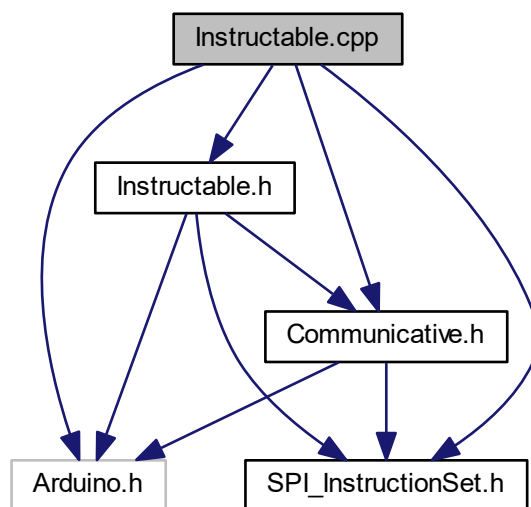
- class [Identifiable](#)

A class which models a Sensor/peripheral as an identifiable entity.

5.7 Instructable.cpp File Reference

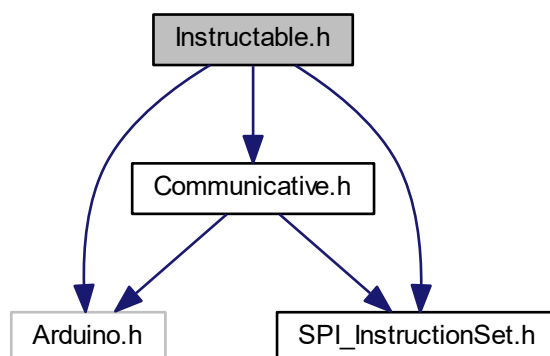
```
#include <Arduino.h>
#include "Communicative.h"
#include "Instructable.h"
#include "SPI_InstructionSet.h"
```


Include dependency graph for Instructable.cpp:

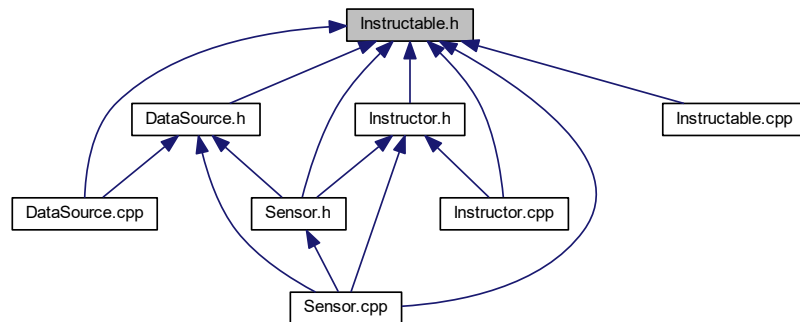


5.8 Instructable.h File Reference

```
#include <Arduino.h>
#include "SPI_InstructionSet.h"
#include "Communicative.h"
Include dependency graph for Instructable.h:
```



This graph shows which files directly or indirectly include this file:



Data Structures

- class [Instructable](#)

A class which models a Sensor/peripheral as entity which can receive commands.

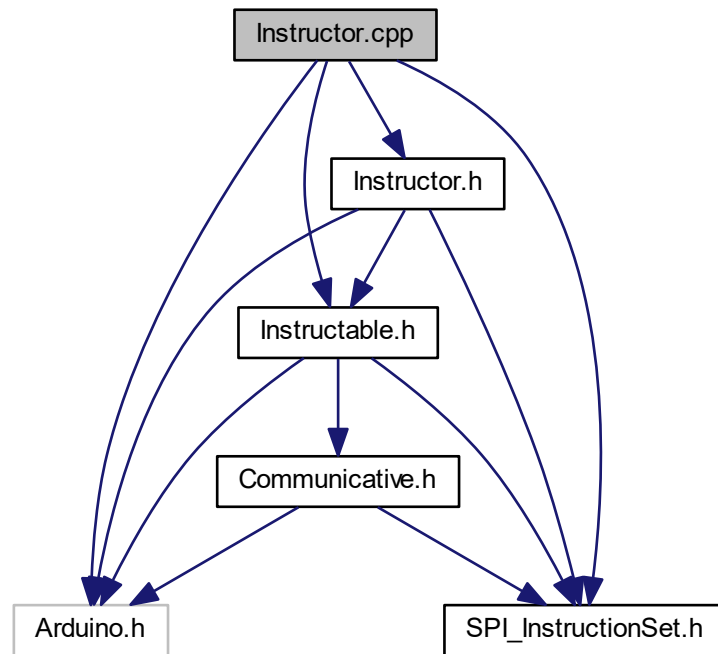
5.9 Instructor.cpp File Reference

```

#include <Arduino.h>
#include "Instructable.h"
#include "Instructor.h"
#include "SPI_InstructionSet.h"

```

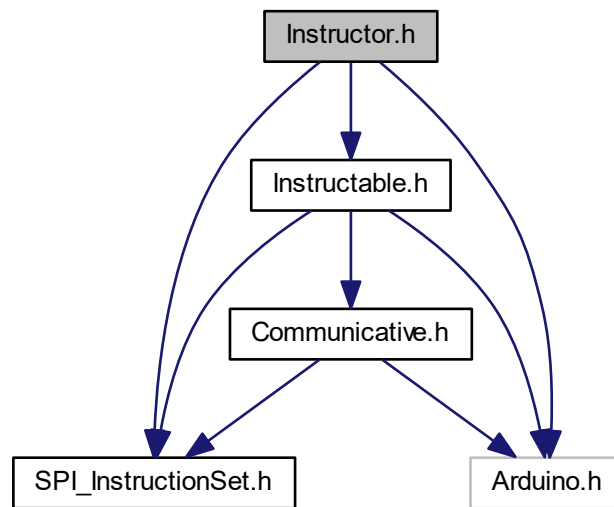
Include dependency graph for Instructor.cpp:



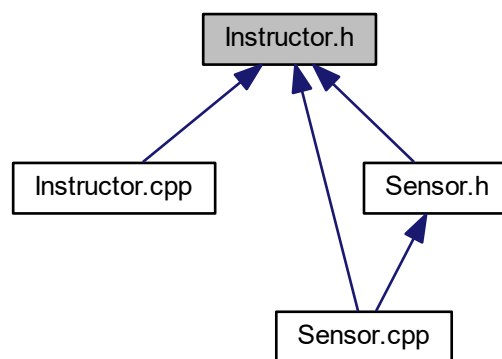
5.10 Instructor.h File Reference

```
#include <Arduino.h>
#include "SPI_InstructionSet.h"
#include "Instructable.h"
```

Include dependency graph for Instructor.h:



This graph shows which files directly or indirectly include this file:



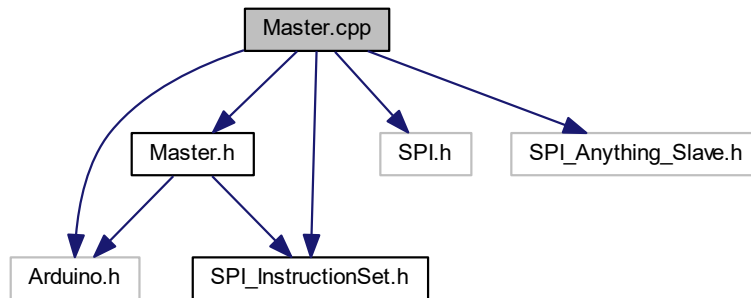
Data Structures

- class [Instructor](#)

A class which models a Sensor/peripheral as entity which can issue instructions to the master.

5.11 Master.cpp File Reference

```
#include <Arduino.h>
#include "Master.h"
#include "SPI_InstructionSet.h"
#include <SPI.h>
#include "SPI_Anything_Slave.h"
Include dependency graph for Master.cpp:
```



Functions

- void [SPI_IRQ](#) (void)
Innacesible IRQ, called on SS falling.

Variables

- volatile [Master SensorMaster](#)
Auto-instantiated instance of [Master](#) class, called during the privately implemented IRQ routine and accessible externally by main code.

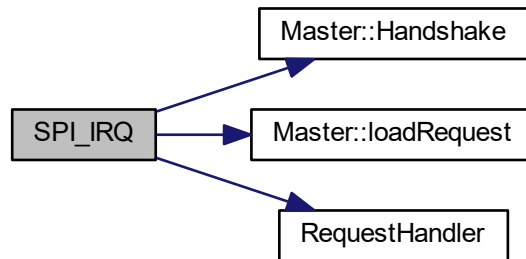
5.11.1 Function Documentation

5.11.1.1 SPI_IRQ()

```
void SPI_IRQ (
    void )
```

Innacesible IRQ, called on SS falling.

Here is the call graph for this function:



5.11.2 Variable Documentation

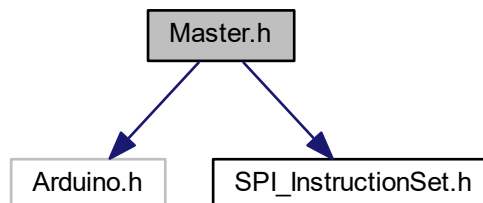
5.11.2.1 SensorMaster

```
volatile Master SensorMaster
```

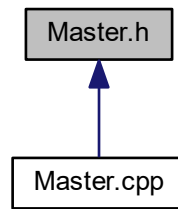
Auto-instantiated instance of `Master` class, called during the privately implemented IRQ routine and accessible externally by main code.

5.12 Master.h File Reference

```
#include <Arduino.h>
#include "SPI_InstructionSet.h"
Include dependency graph for Master.h:
```



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [UserInstructions](#)
- class [Master](#)

A monolithic class to encapsulate and abstract the slave's communication with the master.

Macros

- `#define` [SPI1_NSS_PIN](#) PA4

Functions

- void [SPI_IRQ](#) (void)
Innaccessible IRQ, called on SS falling.
- void [RequestHandler](#) ([mCmd](#) &Request)
Function prototype for user-implemented IRQ method; called after handshake and the reconstruction of the request from the master.

Variables

- const int [MAX_USER_INSTRUCTION_NUMBER](#) = 5
- volatile [Master](#) [SensorMaster](#)
Auto-instantiated instance of [Master](#) class, called during the privately implemented IRQ routine and accessible externally by main code.

5.12.1 Macro Definition Documentation

5.12.1.1 SPI1_NSS_PIN

```
#define SPI1_NSS_PIN PA4
```

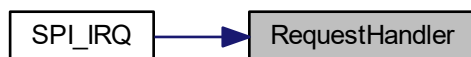
5.12.2 Function Documentation

5.12.2.1 RequestHandler()

```
void RequestHandler (
    mCmd & Request )
```

Function prototype for user-implemented IRQ method; called after handshake and the reconstruction of the request from the master.

Here is the caller graph for this function:

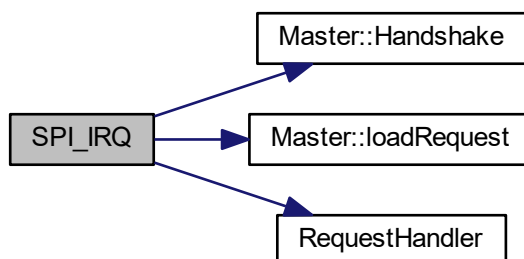


5.12.2.2 SPI_IRQ()

```
void SPI_IRQ (
    void )
```

Innacesible IRQ, called on SS falling.

Here is the call graph for this function:



5.12.3 Variable Documentation

5.12.3.1 MAX_USER_INSTRUCTION_NUMBER

```
const int MAX_USER_INSTRUCTION_NUMBER = 5
```

5.12.3.2 SensorMaster

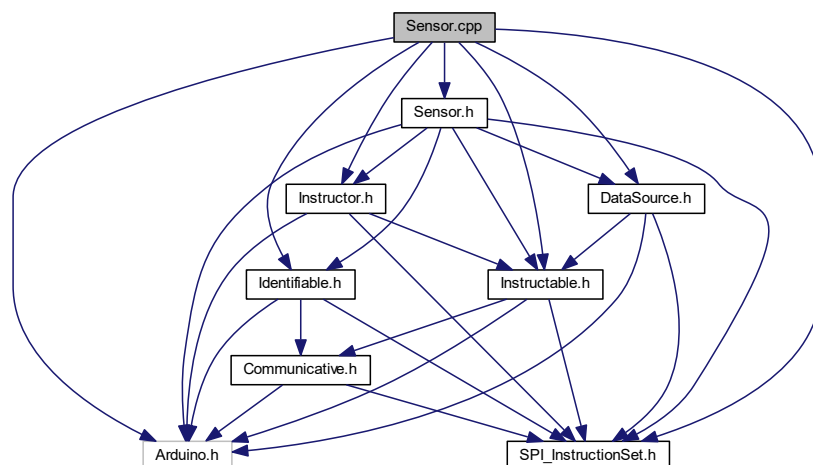
```
volatile Master SensorMaster
```

Auto-instantiated instance of [Master](#) class, called during the privately implemented IRQ routine and accessible externally by main code.

5.13 Sensor.cpp File Reference

```
#include <Arduino.h>
#include "DataSource.h"
#include "Instructor.h"
#include "Instructable.h"
#include "Identifiable.h"
#include "Sensor.h"
#include "SPI_InstructionSet.h"
```

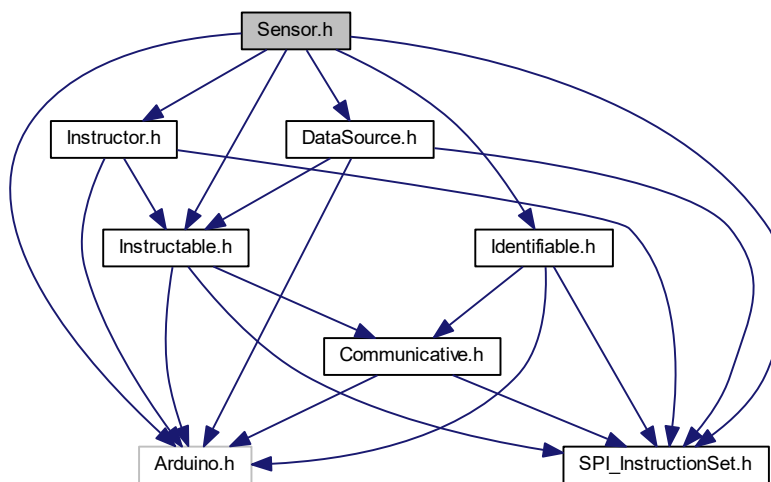
Include dependency graph for Sensor.cpp:



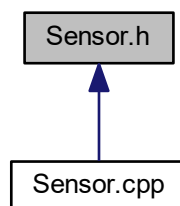
5.14 Sensor.h File Reference

```
#include <Arduino.h>
#include "SPI_InstructionSet.h"
#include "Instructor.h"
#include "DataSource.h"
#include "Instructable.h"
#include "Identifiable.h"
```

Include dependency graph for Sensor.h:



This graph shows which files directly or indirectly include this file:



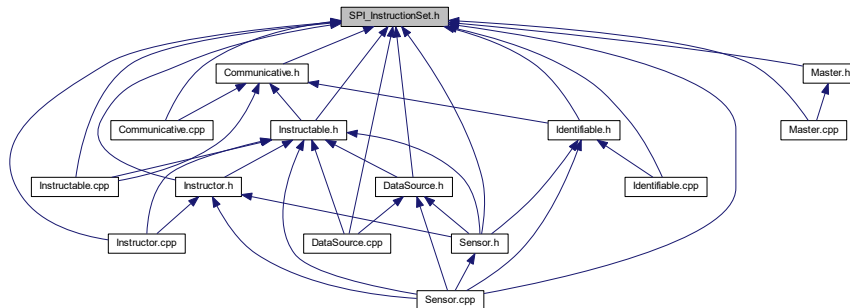
Data Structures

- class [Sensor](#)

A class which models a Sensor/peripheral.

5.15 SPI_InstructionSet.h File Reference

This graph shows which files directly or indirectly include this file:



Data Structures

- struct [sCmd](#)
Type used by slave to send reply to master,.
- struct [mCmd](#)
Type used by master to send requests to slave.
- struct [Identity](#)
Type used to convey the Slave identity.
- struct [Data](#)
Type used to encapsulate the data collected by the slave.

Enumerations

- enum [sInstruct](#) {
[DisplayInstructionAndWait](#), [DisplayInstructionAndWaitForUser](#), [DontDisplayAndWait](#), [DontDisplayAndContinue](#),
[ACK](#), [Yes](#), [No](#), [NAK](#),
[ReferToInt](#), [ReferToFloat](#), [ReferToString](#) }
Instruction set used by slave to instruct master.
- enum [mInstruct](#) {
[PauseMeasurementForiParam](#), [RestartMeasurementProcedure](#), [ResetDevice](#), [HowManyInstructions](#),
[NextCommandPlease](#), [IsThereData](#), [SendDataPlease](#), [WhoAreYou](#),
[HowLongShouldIWait](#), [BeginMeasurement](#), [SitRep](#) }
Instruction set used by [Master](#) to instruct/request responses from [Slave](#).
- enum [MeasurementVectors](#) { [First](#), [Second](#), [Third](#) }
Enumeration to provide human-readable references to different rows in the [Data](#) array.

Variables

- const int [SLAVE_COMMAND_STRING_LENGTH](#) = 40
- const int [IDENTITY_SENSOR_NAME_LENGTH](#) = 25
- const int [NUMBER_OF_DATA_ROWS](#) = 3
- const int [ROW_HEADING_LENGTH](#) = 10
- const int [ROW_UNIT_LENGTH](#) = 5
- const int [DATA_ROW_LENGTH](#) = 64

5.15.1 Enumeration Type Documentation

5.15.1.1 MeasurementVectors

enum `MeasurementVectors`

Enumeration to provide human-readable references to different rows in the `Data` array.

Enumerator

First	
Second	
Third	

5.15.1.2 mInstruct

enum `mInstruct`

Instruction set used by `Master` to instruct/request responses from Slave.

During a single transaction, the `Master` will send a request, characterised by an `mCmd` object, which contains an `mInstruct` object, integer and float. The `mInstruct` object will determine how the request is processed by the slave and will define the object type which the master must expect in reply. In general, the master will expect replies in the form of `sCmd` objects. However, the slave may also send `Data` and `Identity` objects in response to specific `mInstruct` instances; in particular: `SendDataPlease` and `WhoAreYou`.

Enumerator

<code>PauseMeasurementForiParam</code>	Require Slave to pause for a duration specified by the integer parameter of <code>mCmd</code> . Expects <code>sCmd</code> {ACK or NAK} in response.
<code>RestartMeasurementProcedure</code>	Require Slave to restart the measurement procedure from the first instruction. Expects <code>sCmd</code> {ACK or NAK} in response.
<code>ResetDevice</code>	Require Slave to reset. Expects <code>sCmd</code> {ACK or NAK} in response.
<code>HowManyInstructions</code>	Request the slave to confirm the number of instructions in a measurement cycle. Expects <code>sCmd</code> {ReferToInt, int NumberOfInstructions} in response.
<code>NextCommandPlease</code>	Request the next instruction in the measurement cycle from the slave. Expects <code>sCmd</code> {sInstruct SomeInstruction, int PotentialIntParam, float PotentialFloatParam, char* PotentialStringParam} or <code>sCmd</code> {No} in response.
<code>IsThereData</code>	Requests slave to confirm that <code>Data</code> is ready for collection by master. Expects <code>sCmd</code> {Yes or No}.
<code>SendDataPlease</code>	Require slave to send the data object. Expects <code>Data</code> object in response.
<code>WhoAreYou</code>	Require the slave to send its <code>Identity</code> object; contains char* Name and int SensorID. Expects <code>Identity</code> object in response.
<code>HowLongShouldIWait</code>	Ask the slave whether the <code>Master</code> should pause before calling for the next instruction. Unused. Expects <code>sCmd</code> {ReferToInt, int PauseDuration} in reply.
<code>BeginMeasurement</code>	Instruct the slave to initiate its measurement cycle and expect the first instruction to be called for. Expects <code>sCmd</code> {ACK or NAK}.
<code>SitRep</code>	Request good/bad status from slave. Unused. Expects <code>sCmd</code> {No} in reply.

5.15.1.3 sInstruct

enum `sInstruct`

Instruction set used by slave to instruct master.

During a single transaction, the master will request a reply, [Data](#) or [Identity](#). Replies are characterised by an [sCmd](#) object, which contains an `sInstruct` object, which defines the action which the slave requires of the master.

Enumerator

<code>DisplayInstructionAndWait</code>	Require the master to display the string contained within the sCmd object and pause for a duration defined by the integer parameter of the sCmd object
<code>DisplayInstructionAndWaitForUser</code>	Require the master to display the string contained within the sCmd object and pause until the user has confirmed adherence to the instruction.
<code>DontDisplayAndWait</code>	Require the master to pause for a duration defined by the integer parameter of the sCmd object
<code>DontDisplayAndContinue</code>	Require the master to take no action. Essentially a nop.
<code>ACK</code>	Slave acknowledges the master's command/response.
<code>Yes</code>	Respond affirmative.
<code>No</code>	Respond negative.
<code>NAK</code>	Slave unable to adhere to Master's command or understand it.
<code>ReferToInt</code>	Points master to the integer parameter of sCmd .
<code>ReferToFloat</code>	Points master to the float parameter of sCmd
<code>ReferToString</code>	Points master to the string parameter of sCmd

5.15.2 Variable Documentation

5.15.2.1 DATA_ROW_LENGTH

```
const int DATA_ROW_LENGTH = 64
```

5.15.2.2 IDENTITY_SENSOR_NAME_LENGTH

```
const int IDENTITY_SENSOR_NAME_LENGTH = 25
```

5.15.2.3 NUMBER_OF_DATA_ROWS

```
const int NUMBER_OF_DATA_ROWS = 3
```

5.15.2.4 ROW_HEADING_LENGTH

```
const int ROW_HEADING_LENGTH = 10
```

5.15.2.5 ROW_UNIT_LENGTH

```
const int ROW_UNIT_LENGTH = 5
```

5.15.2.6 SLAVE_COMMAND_STRING_LENGTH

```
const int SLAVE_COMMAND_STRING_LENGTH = 40
```

Index

- ~Communicative
 - Communicative, 8
- ~Master
 - Master, 37
- ACK
 - SPI_InstructionSet.h, 77
- areYouConnected
 - Instructable, 27
- BeginMeasurement
 - SPI_InstructionSet.h, 76
- beginMeasurement
 - Master, 37
- ClearMeasurementVector
 - Master, 37
- Communicative, 7
 - ~Communicative, 8
 - Communicative, 7
 - isPeripheralConnected, 8
 - RequestData, 9
 - RequestIdentity, 10
 - RequestReply, 10
- Communicative.cpp, 59
 - REQUEST_DELAY_MICROS, 59
- Communicative.h, 60
- Data, 11
 - DataPoints, 12
 - NumColumns, 12
 - NumRows, 13
 - operator=, 12
 - RowHeadings, 13
 - rowUnits, 13
- DATA_ROW_LENGTH
 - SPI_InstructionSet.h, 77
- DataPoints
 - Data, 12
- DataSource, 13
 - DataSource, 14
 - getDataArray, 15
 - getDataVector, 15
 - getNumberOfDataColumns, 15
 - getNumberOfDataRows, 16
 - getRowHeadings, 16
 - getRowUnits, 17
 - getValueOne, 17
 - getValueThree, 17
 - getValueTwo, 18
 - getVectorHeading, 18
 - getVectorLength, 19
 - getVectorUnits, 19
 - isThereData, 19
 - loadData, 20
- DataSource.cpp, 61
- DataSource.h, 61
- DisplayInstructionAndWait
 - SPI_InstructionSet.h, 77
- DisplayInstructionAndWaitForUser
 - SPI_InstructionSet.h, 77
- DontDisplayAndContinue
 - SPI_InstructionSet.h, 77
- DontDisplayAndWait
 - SPI_InstructionSet.h, 77
- First
 - SPI_InstructionSet.h, 76
- fParam
 - mCmd, 49
 - sCmd, 51
- fParams
 - UserInstructions, 57
- getCurrentCommandFloat
 - Instructor, 32
- getCurrentCommandInstruction
 - Instructor, 32
- getCurrentCommandInt
 - Instructor, 32
- getCurrentCommandString
 - Instructor, 32
- getCurrentInstruction
 - Master, 38
- getCurrentInstructionFloatParameter
 - Master, 38
- getCurrentInstructionIntParameter
 - Master, 38
- getCurrentInstructionNumber
 - Master, 38
- getDataArray
 - DataSource, 15
- getDataVector
 - DataSource, 15
- getIDNumber
 - Identifiable, 22
- getNumberOfDataColumns
 - DataSource, 15
- getNumberOfDataRows
 - DataSource, 16

- getRowHeadings
 - DataSource, 16
- getRowUnits
 - DataSource, 17
- getSensorName
 - Identifiable, 22
- getValueOne
 - DataSource, 17
- getValueThree
 - DataSource, 17
- getValueTwo
 - DataSource, 18
- getVectorHeading
 - DataSource, 18
- getVectorLength
 - DataSource, 19
- getVectorUnits
 - DataSource, 19
- Handshake
 - Master, 38
- hasIdentityChanged
 - Identifiable, 23
- HowLongShouldIWait
 - SPI_InstructionSet.h, 76
- howLongShouldIWait
 - Instructor, 33
- HowManyInstructions
 - SPI_InstructionSet.h, 76
- howManyInstructions
 - Instructor, 33
- Identifiable, 21
 - getIDNumber, 22
 - getSensorName, 22
 - hasIdentityChanged, 23
 - Identifiable, 21
 - updateIdentity, 23
- Identifiable.cpp, 63
- Identifiable.h, 63
- Identity, 24
 - operator=, 24, 25
 - sensorChipSelect, 25
 - sensorID, 25
 - SensorName, 25
- IDENTITY_SENSOR_NAME_LENGTH
 - SPI_InstructionSet.h, 77
- Instructable, 26
 - areYouConnected, 27
 - Instructable, 26
 - issueCommand, 27–29
- Instructable.cpp, 64
- Instructable.h, 65
- Instruction
 - mCmd, 50
 - sCmd, 51
- InstructionCounter
 - UserInstructions, 57
- InstructionSet
 - UserInstructions, 57
- Instructor, 30
 - getCurrentCommandFloat, 32
 - getCurrentCommandInstruction, 32
 - getCurrentCommandInt, 32
 - getCurrentCommandString, 32
 - howLongShouldIWait, 33
 - howManyInstructions, 33
 - Instructor, 31
 - loadNextCommand, 34
- Instructor.cpp, 66
- Instructor.h, 67
- iParam
 - mCmd, 50
 - sCmd, 51
- iParams
 - UserInstructions, 57
- isPeripheralConnected
 - Communicative, 8
- issueCommand
 - Instructable, 27–29
- IsThereData
 - SPI_InstructionSet.h, 76
- isThereData
 - DataSource, 19
 - Master, 39
- loadData
 - DataSource, 20
- loadNextCommand
 - Instructor, 34
- loadRequest
 - Master, 39
- Master, 34
 - ~Master, 37
 - beginMeasurement, 37
 - ClearMeasurementVector, 37
 - getCurrentInstruction, 38
 - getCurrentInstructionFloatParameter, 38
 - getCurrentInstructionIntParameter, 38
 - getCurrentInstructionNumber, 38
 - Handshake, 38
 - isThereData, 39
 - loadRequest, 39
 - Master, 36, 37
 - operator=, 40
 - PopMeasurementVector, 40
 - PushMeasurementVector, 40
 - resendCurrentUserInstruction, 41
 - restartUserInstructionCycle, 41
 - sendData, 41
 - sendIdentity, 42
 - sendNextUserInstruction, 42
 - sendReply, 42–44
 - sendTotalNumOfInstructions, 45
 - setMeasurementVectorHeading, 45
 - setMeasurementVectorUnits, 45
 - SETUP, 46

- shallStart, [47](#)
- SPISetup, [47](#)
- Master.cpp, [69](#)
 - SensorMaster, [70](#)
 - SPI_IRQ, [69](#)
- Master.h, [70](#)
 - MAX_USER_INSTRUCTION_NUMBER, [73](#)
 - RequestHandler, [72](#)
 - SensorMaster, [73](#)
 - SPI1_NSS_PIN, [71](#)
 - SPI_IRQ, [72](#)
- MasterInstructionSet
 - UserInstructions, [57](#)
- MAX_USER_INSTRUCTION_NUMBER
 - Master.h, [73](#)
- mCmd, [48](#)
 - fParam, [49](#)
 - Instruction, [50](#)
 - iParam, [50](#)
 - mCmd, [49](#)
 - operator=, [49](#)
- MeasurementVectors
 - SPI_InstructionSet.h, [76](#)
- mInstruct
 - SPI_InstructionSet.h, [76](#)
- NAK
 - SPI_InstructionSet.h, [77](#)
- NextCommandPlease
 - SPI_InstructionSet.h, [76](#)
- No
 - SPI_InstructionSet.h, [77](#)
- NUMBER_OF_DATA_ROWS
 - SPI_InstructionSet.h, [77](#)
- NumColumns
 - Data, [12](#)
- NumOfInstructions
 - UserInstructions, [57](#)
- NumRows
 - Data, [13](#)
- operator=
 - Data, [12](#)
 - Identity, [24](#), [25](#)
 - Master, [40](#)
 - mCmd, [49](#)
 - sCmd, [51](#)
 - UserInstructions, [56](#), [57](#)
- PauseMeasurementForiParam
 - SPI_InstructionSet.h, [76](#)
- PauseMeasurementForMillis
 - Sensor, [53](#)
- PopMeasurementVector
 - Master, [40](#)
- PushMeasurementVector
 - Master, [40](#)
- ReferToFloat
 - SPI_InstructionSet.h, [77](#)
- ReferToInt
 - SPI_InstructionSet.h, [77](#)
- ReferToString
 - SPI_InstructionSet.h, [77](#)
- REQUEST_DELAY_MICROS
 - Communicative.cpp, [59](#)
- RequestData
 - Communicative, [9](#)
- RequestHandler
 - Master.h, [72](#)
- RequestIdentity
 - Communicative, [10](#)
- RequestReply
 - Communicative, [10](#)
- resendCurrentUserInstruction
 - Master, [41](#)
- ResetDevice
 - SPI_InstructionSet.h, [76](#)
- RestartMeasurement
 - Sensor, [55](#)
- RestartMeasurementProcedure
 - SPI_InstructionSet.h, [76](#)
- restartUserInstructionCycle
 - Master, [41](#)
- ROW_HEADING_LENGTH
 - SPI_InstructionSet.h, [78](#)
- ROW_UNIT_LENGTH
 - SPI_InstructionSet.h, [78](#)
- RowHeadings
 - Data, [13](#)
- rowUnits
 - Data, [13](#)
- sCmd, [50](#)
 - fParam, [51](#)
 - Instruction, [51](#)
 - iParam, [51](#)
 - operator=, [51](#)
 - sParam, [52](#)
- Second
 - SPI_InstructionSet.h, [76](#)
- sendData
 - Master, [41](#)
- SendDataPlease
 - SPI_InstructionSet.h, [76](#)
- sendIdentity
 - Master, [42](#)
- sendNextUserInstruction
 - Master, [42](#)
- sendReply
 - Master, [42–44](#)
- sendTotalNumOfInstructions
 - Master, [45](#)
- Sensor, [52](#)
 - PauseMeasurementForMillis, [53](#)
 - RestartMeasurement, [55](#)
 - Sensor, [53](#)
 - StartMeasurement, [55](#)

- Sensor.cpp, [73](#)
- Sensor.h, [74](#)
- sensorChipSelect
 - Identity, [25](#)
- sensorID
 - Identity, [25](#)
- SensorMaster
 - Master.cpp, [70](#)
 - Master.h, [73](#)
- SensorName
 - Identity, [25](#)
- setMeasurementVectorHeading
 - Master, [45](#)
- setMeasurementVectorUnits
 - Master, [45](#)
- SETUP
 - Master, [46](#)
- shallIStart
 - Master, [47](#)
- sInstruct
 - SPI_InstructionSet.h, [77](#)
- SitRep
 - SPI_InstructionSet.h, [76](#)
- SLAVE_COMMAND_STRING_LENGTH
 - SPI_InstructionSet.h, [78](#)
- sParam
 - sCmd, [52](#)
- SPI1_NSS_PIN
 - Master.h, [71](#)
- SPI_InstructionSet.h, [75](#)
 - ACK, [77](#)
 - BeginMeasurement, [76](#)
 - DATA_ROW_LENGTH, [77](#)
 - DisplayInstructionAndWait, [77](#)
 - DisplayInstructionAndWaitForUser, [77](#)
 - DontDisplayAndContinue, [77](#)
 - DontDisplayAndWait, [77](#)
 - First, [76](#)
 - HowLongShouldIWait, [76](#)
 - HowManyInstructions, [76](#)
 - IDENTITY_SENSOR_NAME_LENGTH, [77](#)
 - IsThereData, [76](#)
 - MeasurementVectors, [76](#)
 - mInstruct, [76](#)
 - NAK, [77](#)
 - NextCommandPlease, [76](#)
 - No, [77](#)
 - NUMBER_OF_DATA_ROWS, [77](#)
 - PauseMeasurementForiParam, [76](#)
 - ReferToFloat, [77](#)
 - ReferToInt, [77](#)
 - ReferToString, [77](#)
 - ResetDevice, [76](#)
 - RestartMeasurementProcedure, [76](#)
 - ROW_HEADING_LENGTH, [78](#)
 - ROW_UNIT_LENGTH, [78](#)
 - Second, [76](#)
 - SendDataPlease, [76](#)
 - sInstruct, [77](#)
 - SitRep, [76](#)
 - SLAVE_COMMAND_STRING_LENGTH, [78](#)
 - Third, [76](#)
 - WhoAreYou, [76](#)
 - Yes, [77](#)
- SPI_IRQ
 - Master.cpp, [69](#)
 - Master.h, [72](#)
- SPISetup
 - Master, [47](#)
- StartMeasurement
 - Sensor, [55](#)
- Third
 - SPI_InstructionSet.h, [76](#)
- updateIdentity
 - Identifiable, [23](#)
- UserInstructions, [56](#)
 - fParams, [57](#)
 - InstructionCounter, [57](#)
 - InstructionSet, [57](#)
 - iParams, [57](#)
 - MasterInstructionSet, [57](#)
 - NumOfInstructions, [57](#)
 - operator=, [56, 57](#)
- WhoAreYou
 - SPI_InstructionSet.h, [76](#)
- Yes
 - SPI_InstructionSet.h, [77](#)