MannaSim Documentation

# SensorNode Class

## METHODS

• void SensorNode::selfTest() – performs a sensor node self test verifying its proper functioning.

• void SensorNode::sleep() – turns the sensor node oﬀ when necessary. Stop all of its applications.

• void SensorNode::wakeUp() – wakes the sensor node up after a sleepy period. Start all of its applications.

• double SensorNode::sensingPower() – gets sensor node sensing energy consumption.

• double SensorNode::processingPower() – gets sensor node processing energy consumption.

• int SensorNode::instructionsPerSecond() – gets sensor node instructions per second executed by its processor.

## FIELDS

• AppList apps – the list of WSN applications witch the sensor node is associated to. The AppList type is simple an STL1 list of AppData objects.

• int instructionsPerSecond –the number of instructions the nodes processor can execute in a second. This ﬁeld has its value deﬁned in TCL simulation script.

• double processingPower – the energy consumption of the sensor node in processing activities. This ﬁeld has its value expressed in watts (joules/second) and is deﬁned in TCL simulation script.

• double sensingPower – the energy consumption of the sensor node in sensing activities. This ﬁeld has its value expressed in watts (joules/second) and is deﬁned in TCL simulation script.

• int sensorUseState – the sensor state. Indicates whether the node’s sensing activity is in use or not.

• int processorUseState – the processor state. Indicates whether the node’s processing activity is in use or not.

• int transceptorUseState – the transceptor state. Indicates whether the node’s dissemination activity is in use or not.

# Battery Class

## METHODS

• void Battery::DecrSensingEnergy(double sensing time, double sensing power) – computes the energy spent by a sensing task and reduce energy stock by this amount.

• void Battery::DecrProcessingEnergy(int number instructions, double instructions per second, double processing power) – computes the energy spent by a processing task and reduce energy stock by this amount. .

• void Battery::setNodeOn() – turns sensor node on.

• void Battery::setNodeOff() – turns sensor node oﬀ.

• void Battery::sleep() – puts node battery in sleep mode.

• void Battery::wakeUp() – wakes up the sleepy node battery.

## FIELDS

All ﬁelds are inherited from NS-2 EnergyModel.

# DataGenerator Class

## METHODS

• void DataGenerator::generateData() – simulates the sensing activity, gets data from collect() method and deliver it to sensor processing module.

• virtual AppData\* DataGenerator::collect() – the truly data generator function. Creates speciﬁc synthetic data, encapsulates it and return to generateData() method. It’s a virtual method in DataGenerator class, so it must be overloaded.

• void DataGenerator::insertInterference() – inserts interference in data gathered by the sensor. This method hasn’t been implemented yet.

• void DataGenerator::start() – schedules the ﬁrst sensing activity if the network has periodic sensing.

• void DataGenerator::stop() – stops the sensing activity and drops all scheduled sensing events.

• void DataGenerator::setSensingInterval(double si) – sets the sensing interval for a periodic sensing network. Actually all kinds of sensing should set the sensing interval, in continuous sensing for example, this value should be set as small as possible.

• double DataGenerator::getSensingInterval() – informs the sensing interval of the network. This ﬁeld has its value deﬁned in TCL simulation script.

## FIELDS

• AppDataType type – the identity of the data generator in terms of application data type.

• SensorAppList app – the list of sensor applications associated with the data generator. The SensorAppList type is simple an STL list of SensorBaseApp objects.

• double sensing interval – the sensing interval for data gathering.

• int sensing type –the way the sensor node, and consequently the DataGenerator class, is oriented to gather data. Possible values are PROGRAMMED, CONTINUOUS, ON DEMAND and EVENT DRIVEN representing respectively periodic, continuous, on demand and event driven data sensing.

# TemperatureDataGenerator

## METHODS

• virtual AppData\* DataGenerator::collect() – the truly temperature data generator method. Creates temperature synthetic data, encapsulates it in a TemperatureAppData object and deliver this new object for further processing.

• double TemperatureDataGenerator::getAvgMeasure()–returnstheaveragemeasurefor temperature data generation.

• void TemperatureDataGenerator::setAvgMeasure(double avg measure) – adjusts the average measure for temperature data generation.

• TemperatureAppData\* TemperatureDataGenerator::getMaximumAllowedValue()–return the maximum allowed temperature value that can be generated.

## FIELDS

• RNG\* rand – random number used to generate synthetic temperature data.

• double avg measure – average measure for synthetic temperature data generation.

• double std deviation – standard deviation measure for synthetic temperature data generation.

• double maximumTemperatureValue – maximum temperature value allowed to be generated.

# TemperatureAppData Class

## METHODS

• AppData \* TemperatureAppData::copy() – creates a copy of temperature application data object.

• int TemperatureAppData::size() const – informs application data size in bytes.

• bool TemperatureAppData::checkEvent(AppData\* data )–checksiftheparametricdata represents an event considering the object data. For example, temperature value get greater? This method is used by event driven WSN.

• bool TemperatureAppData::compareData(AppData\* data, int operation)–compares parametric data according to speciﬁed operation. This method is used by on demand WSN.

• double TemperatureAppData::data() – returns temperature data from application data.

• void TemperatureAppData::setData(double data) – adjusts temperature data of application data.

• double TemperatureAppData::time() – returns timestamp of application data.

• void TemperatureAppData::setTime(double time) – adjusts timestamp of application data.

• int & TemperatureAppData::getPriority() – return application data priority (Usefull for temperature data delivery).

• void TemperatureAppData::setPriority(int p) – set application data priority (Usefull for temperature data delivery).

## FIELDS

• double data – temperature data gathered in sensing task.

• double time – timestamp for temperature data.

• int priority – temperature data priority (Usefull for temperature data deliver).

# Processing Class

## METHODS

• void AggregateProcessing::recvData(AppData\* data ) – receives data from various sources, process it and return to the application for dissemination. This method must be overloaded.

• AppData \* Processing::processRequest(AppData\* data) – simulates processing activity. Gets the raw data from sensing activity, process it, computes energy spent and returns processed data. This method should be overloaded in extensions of the Processing class.

• AppData\* Processing::processSensedData(AppData\* data , AppData\* eventData )– simulates processing activity for event driven WSN.

• AppData\* Processing::process request data(OnDemandParameter\* parameter, int request type) – manages requests from an outsider observer in a on demand WSN2. This method only redirects the raw data to further specialized data processing.

• AppData\* Processing::process real request(AppData\* data , int operation)–process real requests from an on demand WSN. In real requests the sensor node drops all data from its buﬀer, gather new one, process and deliver it to the disseminating module.

• AppData\* Processing::process buffer request(AppData\* data , int operation)–process buﬀer requests from an on demand WSN. In buﬀer requests the sensor node process data from its buﬀer and give the results to the disseminating module.

• AppData \* CommonNodeApp::process average request(AppData\* data ,int operation) –processaveragerequestsfromanondemandWSN.Thismethodhasn’tbeingimplemented yet.

• void Processing::resetData() – clears the sensor processed data buﬀer.

• SensedData \* Processing::getProcessedData() – returns data generated by the processing activity.

## FIELDS

• SensedData\* info – processed data buﬀer, data stored here is ready for dissemination.

• SensorBaseApp \* app – application attached to the sensor node. Usefull in on demand WSN.

• SensorNode\* sensor node – sensor node where the processing task takes place. Used for energy contability proposes.

# SensedData Class

## METHODS

• AppData \* SensedData::copy() – creates a copy of the sensed data object.

• bool SensedData::existsData() – informs if there is processed data stored in data buﬀer.

• AppDataList SensedData::getData() – returns processed data stored in data buﬀer.

• int & SensedData::msgType() – returns a reference to sensed data message type.

• int & SensedData::node id() – returns a reference to source node identiﬁcation.

• int & SensedData::eventType() – returns a reference to sensed data event type. This method is designed to be used in an event driven application.

• double & SensedData::timeStamp() – returns a reference to sensed data timestamp;

• int SensedData::priority() – returns sensed data priority. Useful during disseminating tasks.

• void SensedData::set priority(int p) – adjusts sensed data priority.

• void SensedData::insertNewData(AppData\* data) – inserts new processed data into data buﬀer.

## FIELDS

• int node id – source node identiﬁcation for sensed data dissemination purposes.

• int msgType – type of the message to be disseminated to the network.

• int priority – priority of sensed data message.

• int eventType – type of the event that caused sensed data processing.

• double timestamp – sensed data timestamp.

• AppDataList infoRepository – sensed data buﬀer. Storages all data processed by the sensor node.

# OnDemandData Class

## METHODS

• AppData \* OnDemandData::copy() – creates a copy of the current OnDemandData object.

• int OnDemandData::size() const – returns OnDemandData object size in bytes.

• int & OnDemandData::requestType() – returns a reference for the request type ﬁeld.

## FIELDS

• int request type – message request type for a on demand WSN. Possible values include REAL, BUFFER and AVERAGE.

# OnDemandParameter Class

## METHODS

• AppData \* OnDemandParameter :: copy() – creates a copy of the current OnDemandParameter object. • int OnDemandParameter::size() const – returns OnDemandParameter object size in bytes.

• AppData \* OnDemandParameter::data() – returns sample data used for request validation.

• int & OnDemandParameter :: operation() – returns a reference for the operation to be realized for request validation.

## FIELDS

• AppData\* data – sample data used for request validation.

• int operation – operation to be realized for request validation. Currently supported operations include GREATER THAN, LESS THAN and EQUAL.

# SensorBaseApp Class

## METHODS

• virtual void disseminateData() – disseminates sensed data into the network after its processing. This method is virtual so it must be overloaded.

• virtual void disseminateData(AppData\* data) – disseminates sensed data into the network after its processing. In this method disseminating data is passed as parameter and as the method is virtual it must be overloaded.

• virtual void recvSensedData(AppData\* data ) – receives data gathered by the node’s sensors, process it and disseminate the results. This method is virtual so it must be overloaded.

• virtual void recvSensedData(AppData\* data , AppData\* eventData )–receivesdata gathered by the node’s sensors, process it and if data corresponds to a valid event disseminate the results. This method is virtual so it must be overloaded.

• void SensorBaseApp::start() – schedules the ﬁrst sensing/disseminating tasks if the network application is periodic.

• void SensorBaseApp::stop() – drops all scheduled events, stopping also all sensing tasks.

• void SensorBaseApp::insertNewGenerator(DataGenerator\* gen) – inserts a new DataGenerator object (or one of its extensions) into data generators list of the application.

• DataGenList SensorBaseApp::getGenList() – returns the data genetators list of the application.

• SensorNode \* SensorBaseApp::sensor node() – returns a reference to the sensor node attached to the application.

## FIELDS

• SensorNode\* sensor node – sensor node where the applications is inserted.

• DataGenList gen – list of data generator objects (or one of its extensions). DataGenList type is simple a STL list data structure of data generator objects.

• Processing\* processing – processing module of the sensor node. Gets raw data from a data generator object and returns processed data.

• int disseminating type –the way the sensor node, and consequently the SensorBaseApp class, is oriented to disseminate data.. Possible values are PROGRAMMED, CONTINUOUS, ON DEMAND and EVENT DRIVEN representing respectively periodic, continuous, on demand and event driven dissemination.

• double disseminating interval – the application disseminating interval.

• DisseminatingTimer\* dissTimer – timer used .

• int destination id – node identiﬁcation for data dissemination.

# CommonNodeApp Class

## METHODS

• void CommonNodeApp::disseminateData() – disseminates sensed data into the network after its processing. Overloaded method.

• void CommonNodeApp::disseminateData(SensedData\* data )–disseminatessenseddata into the network after its processing. In this method disseminating data is passed as parameter. Overloaded method.

• void CommonNodeApp::recvSensedData(AppData\* data ) –receives data gatheredby the node’s sensors (via DataGenerator object), activates processing tasks and if necessary (continuous dissemination case) disseminate the results. Overloaded method.

• void CommonNodeApp::recvSensedData(AppData\* data , AppData\* eventData ) – receives data gathered by the node’s sensors (via DataGenerator object), activates processing tasks and if data corresponds to a valid event disseminate the results. Overloaded method.

• void CommonNodeApp::process data(int size, AppData\* data) – process on demand requests from outsider observer. This method overload NS-2 Process::process data() and is invoked in Agent::recv().

• inline bool CommonNodeApp::isDead() – informs whether or not the sensor node ran out of energy.

## FIELDS

All ﬁelds are inherited from SensorBaseApp class.

# ClusterHeadApp Class

## METHODS

• void ClusterHeadApp::disseminateData() – disseminates sensed data into the network after its processing. Overloaded method.

• void ClusterHeadApp::process data(int size, AppData\* data) – process on demand requests from outsider observer. This method overload NS-2 Process::process data() and is invoked in Agent::recv().

• void ClusterHeadApp::processRequest(AppData\* data) – forwards a request message for all nodes in cluster head children list.

• void ClusterHeadApp::insert child(int id) – insert node identiﬁcation into the cluster head children list.

• void ClusterHeadApp::remove child(int id) – remove parametric node identiﬁcation from the cluster head children list.

• bool ClusterHeadApp::search child(int id) – search the cluster head children list for a node whose identiﬁcation is the method parameter.

## FIELDS

• IdList child list – the cluster head children list. Storages identiﬁcation number of all nodes under the cluster head responsability.