Index for 32A01T_API_Matlab_Yang_Modified

Matlab files in this directory:

CRC	Accurate implementations of the 16-bit CRC-CCITT, used with a look up table.
api_calibration	API : calibrate to correct IMU gyro static drift (third setp!)
api_countingReadAng	API: counting read Euler angle data
api_countingReadQuat	API: counting read quaternion data
api_countingReadSensor	API: counting read sensor data
api_createSerial	API: serial object creation given parameters (first step!).
api_get96bitID	API: get 96-bit IMU ID.
api_getBasicInfo	API: get IMU basic information.
api_getSoftwareVersion	API: get software version.
api_initialization	API: initialize equipment (second setp!)
api_loadCommandList	API: load whole command list
api_serialReadAng	API: serial read Euler angle data
api_serialReadAngGyroAccMag	API: serial read Euler angle, gyroscope, accelerometer, and magnetometer
api_serialReadDCM	API: serial read direct cosine matrix data
api_serialReadQuat	API: serial read quaternion
api_serialReadQuatAcc	API: serial read quaternion and accelerometer data
api_serialReadQuatAccMag	API: serial read quaternion accelerometer and magnetometer data
api_serialReadQuatMag	API: serial read quaternion and magnetometer data
api_serialReadSensor	API: serial read sensor data
api_setEquipID	API: set equipment ID (1-80)(default = 1)
api_setOutputRate	API: set IMU output frequecy division (1-255)(default = 10)
api_setSamplingRate	API: set sampling rate (1-500) [Hz](default = 500)
api_setSensitivity	API: set IMU sensitivity (0.04 - 0.4)(default = 0.12)
callback_countingReadAng	callback: counting read Euler angle data
callback_countingReadQuat	callback: counting read quaternion
callback_countingReadSensor	callback: counting read sensor data
callback_serialReadAng	callback: serial read Euler angle data
callback_serialReadAngGyroAccMag	callback: serial read angle, gyroscope, accelerometer and magnetometer
callback_serialReadDCM	callback: serial read direct cosine matrix
callback_serialReadQuat	callback: serial read quaternion data
callback_serialReadQuatAcc	callback: serial read quaternion and accelerometer data
callback_serialReadQuatAccMag	callback: serial read quaternion, accelerometer and magnetometer data

callback_serialReadQuatMag	callback: serial read quaternion and magnetometer data
callback_serialReadSensor	callback: serial read sensor data
hexsingle2num	Convert single precision IEEE hexadecimal string to number.
main	32A01T commmand test

---

main

PURPOSE

32A01T commmand test

SYNOPSIS

This is a script file.

DESCRIPTION

```
32A01T commmand test
```

CROSS-REFERENCE INFORMATION

This function calls:

- api_calibration API : calibrate to correct IMU gyro static drift (third setp!)
- api_countingReadAng API: counting read Euler angle data
- api_countingReadQuat API: counting read quaternion data
- api_countingReadSensor API: counting read sensor data
- api_createSerial API: serial object creation given parameters (first step!).
- api_get96bitID API: get 96-bit IMU ID.
- api_getBasicInfo API: get IMU basic information.
- api_getSoftwareVersion API: get software version.
- api_initialization API: initialize equipment (second setp!)
- api_loadCommandList API: load whole command list
- api_serialReadAng API: serial read Euler angle data
- api_serialReadAngGyroAccMag API: serial read Euler angle, gyroscope, accelerometer, and magnetometer
- api_serialReadDCM API: serial read direct cosine matrix data
- api_serialReadQuat API: serial read quaternion
- api_serialReadQuatAcc API: serial read quaternion and accelerometer data
- api_serialReadQuatAccMag API: serial read quaternion accelerometer and magnetometer data
- api_serialReadQuatMag API: serial read quaternion and magnetometer data
- api_serialReadSensor API: serial read sensor data
- api_setEquipID API: set equipment ID (1-80)(default = 1)
- api_setOutputRate API: set IMU output frequecy division (1-255)(default = 10)
- api_setSamplingRate API: set sampling rate (1-500) [Hz](default = 500)
- api_setSensitivity API: set IMU sensitivity (0.04 - 0.4)(default = 0.12)

This function is called by:

SOURCE CODE

```
% 32A01T commmand test

%%%%%%%%%%%%%%%%%%%%%%%% 32A01T commmand test %%%%%%%%%%%%%%%%%%%%%%%%

% Author: Hongsheng He
% Mender: Yang Yang (yang.yang@dunanusa.com)
% Version: 1.0  |  Date: 2017.07.24
% Reference:
% 1. https://www.mathworks.com/help/matlab/matlab_external/getting-started-with-serial-i-o.html#f61191
% 2. https://www.mathworks.com/matlabcentral/fileexchange/6927-hexsingle2num
% 3. https://www.mathworks.com/matlabcentral/fileexchange/47682-crc-16-ccitt-m
% 4. https://www.mathworks.com/help/matlab/ref/save.html#zmw57dd0e895968

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

close all;                                                                  % close all figures
clear;                                                                      % clear all variables
clc;                                                                        % clear the command terminal

IMU = struct;

%% 0. Load command list;

[IMU.cmd_name, IMU.cmd_strings] = api_loadCommandList();


%% 1. Create and onfigure a serial port

s1 = api_createSerial();


%% 2. Initialization

api_initialization(s1);


%% 3. Calibration

api_calibration(s1);


%% 5. Command test

command_pool = [ 1 2 3 ...                                                  % Get IMU information
                 7 8 9 10 ...                                               % Set IMU parameters
                 13 15 17 ...                                               % Read counting data
                 14 16 18 20 21 22 23 24 ];                                 % Read serial data


disp('Command test starts: ');


for ii = 1 : length(command_pool)

    r = command_pool(ii);
    % r = command_pool(randi([1,length(command_pool)]));                    % Select a random number from the command pool


    disp(['Testing started: ', int2str(r), '-th command: ', ...
        native2unicode(IMU.cmd_name{r})]);                                  % Command number under test (1-18)

    switch r
        case 1
            IMU.softwareVersion = api_getSoftwareVersion(s1);
            % disp(['Embedded Software Version: ', IMU.softwareVersion]);

        case 2
            IMU.identification = api_get96bitID(s1);
            % disp(['Embedded Software Version: ', IMU.identification]);

        case 3
```

```
0072             IMU.basicInfo = api_getBasicInfo(s1);
0073             % disp(IMU.basicInfo);
0074
0075         case 7
0076             IMU.equipID = api_setEquipID(s1, 1);                            % Only set ID =
1 now
0077             % disp(['Equipment ID is: ', num2str(IMU.equipID)]);
0078
0079         case 8
0080             IMU.samplingRate = api_setSamplingRate(s1, 500);
0081             % disp(IMU.samplingRate);
0082
0083         case 9
0084             IMU.outputRate = api_setOutputRate(s1, 10);
0085             % disp(IMU.outputRate);
0086
0087         case 10
0088             IMU.sensitivity = api_setSensitivity(s1, 0.12);
0089             % disp(['New sentivity setting: ', num2str(IMU.sensitivity)]);
0090
0091         case 13
0092             countingReadSensor = api_countingReadSensor(s1, 32);
0093
0094         case 15
0095             countingReadQuat = api_countingReadQuat(s1, 32);
0096
0097         case 17
0098             countingReadAng = api_countingReadAng(s1, 32);
0099
0100         case 14
0101             serialReadSensor = api_serialReadSensor(s1);
0102
0103         case 16
0104             serialReadQuat = api_serialReadQuat(s1);
0105
0106         case 18
0107             serialReadAng = api_serialReadAng(s1);
0108
0109         case 20
0110             serialReadQuatMag = api_serialReadQuatMag(s1);
0111
0112         case 21
0113             serialReadQuatAcc = api_serialReadQuatAcc(s1);
0114
0115         case 22
0116             serialReadQuatAccMag = api_serialReadQuatAccMag(s1);
0117
0118         case 23
0119             serialReadDCM = api_serialReadDCM(s1);
0120
0121         case 24
0122             serialReadAngGyroAccMag = api_serialReadAngGyroAccMag(s1);
0123
0124         otherwise
0125             disp('Error, please re-run the program.')
0126     end
0127
0128     disp(['Testing done: ',int2str(r), '-th command:', IMU.cmd_name{r}]);
0129     ii = ii + 1;
0130
0131 end
0132
0133 disp('Command tests finish and success!');
0134 % delete(instrfindall);                                                     % All the api in
the command_pool can be called if the serial port stays connected
```

api_loadCommandList

PURPOSE

API: load whole command list

SYNOPSIS

function [cmd_name, cmd_strings] = api_loadCommandList()

DESCRIPTION

```
API: load whole command list
cmd_name is the command name list
cmd_strings is the commmand strings list
Example:
      [ cmd_name, cmd_strings ] = api_loadCommandList();
```

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

- api_calibration API : calibrate to correct IMU gyro static drift (third setp!)
- api_countingReadAng API: counting read Euler angle data
- api_countingReadQuat API: counting read quaternion data
- api_countingReadSensor API: counting read sensor data
- api_get96bitID API: get 96-bit IMU ID.
- api_getBasicInfo API: get IMU basic information.
- api_getSoftwareVersion API: get software version.
- api_initialization API: initialize equipment (second setp!)
- api_serialReadAng API: serial read Euler angle data
- api_serialReadAngGyroAccMag API: serial read Euler angle, gyroscope, accelerometer, and magnetometer
- api_serialReadDCM API: serial read direct cosine matrix data
- api_serialReadQuat API: serial read quaternion
- api_serialReadQuatAcc API: serial read quaternion and accelerometer data
- api_serialReadQuatAccMag API: serial read quaternion accelerometer and magnetometer data
- api_serialReadQuatMag API: serial read quaternion and magnetometer data
- api_serialReadSensor API: serial read sensor data
- api_setEquipID API: set equipment ID (1-80)(default = 1)
- api_setOutputRate API: set IMU output frequecy division (1-255)(default = 10)
- api_setSamplingRate API: set sampling rate (1-500) [Hz](default = 500)
- api_setSensitivity API: set IMU sensitivity (0.04 - 0.4)(default = 0.12)
- main 32A01T commmand test

SOURCE CODE

```
0001 function [ cmd_name, cmd_strings ] = api_loadCommandList()
0002 % API: load whole command list
0003 %
0004 % cmd_name is the command name list
0005 %
0006 % cmd_strings is the commmand strings list
0007 %
0008 % Example:
0009 %       [ cmd_name, cmd_strings ] = api_loadCommandList();
0010 %
0011
0012 cmd_name={};
0013 cmd_strings={};
0014
0015 %% Action command
0016
0017 cmd_name={cmd_name{:}, 'Embedded Software Version, N, 0x0000, Read IMU embedded vision
informaion'};
0018 % 49 4D 55 43 01 02 00 00 9C 14 0D 0A
0019 cmd_strings={cmd_strings{:},uint8(hex2dec(['49'; '4D'; '55'; '43'; '01'; '02'; '00'; '00';
'9C'; '14'; '0D'; '0A';]))};
0020
0021 cmd_name={cmd_name{:}, 'IMU 96-Bit ID N 0x0001 Read IMU 96-bit identity number'};
0022 % 49 4D 55 43 01 02 00 01 8C 35 0D 0A
0023 cmd_strings={cmd_strings{:},uint8(hex2dec(['49'; '4D'; '55'; '43'; '01'; '02'; '00'; '01';
'8C'; '35'; '0D'; '0A';]))};
0024
0025 cmd_name={cmd_name{:}, 'IMU Information N 0x0002 Read IMU basic information'};
0026 % 49 4D 55 43 01 02 00 02 BC 56 0D 0A
0027 cmd_strings={cmd_strings{:},uint8(hex2dec(['49'; '4D'; '55'; '43'; '01'; '02'; '00'; '02';
'BC'; '56'; '0D'; '0A';]))};
0028
0029 cmd_name={cmd_name{:}, 'Error History N 0x0003 Read IMU error history'};
0030 % 49 4D 55 43 01 02 00 03 AC 77 0D 0A
0031 cmd_strings={cmd_strings{:},uint8(hex2dec(['49'; '4D'; '55'; '43'; '01'; '02'; '00'; '03';
'AC'; '77'; '0D'; '0A';]))};
0032
0033 cmd_name={cmd_name{:}, 'Initializes Equipment N 0x02 Initizlize equipment'};
0034 % 49 4D 55 43 01 01 02 E8 DF 0D 0A
0035 cmd_strings={cmd_strings{:},uint8(hex2dec(['49'; '4D'; '55'; '43'; '01'; '01'; '02'; 'E8';
'DF'; '0D'; '0A';]))};
0036
0037 cmd_name={cmd_name{:}, 'Stop Command N 0x03 Stop data transmission'};
0038 % 49 4D 55 43 01 01 03 F8 FE 0D 0A
0039 cmd_strings={cmd_strings{:},uint8(hex2dec(['49'; '4D'; '55'; '43'; '01'; '01'; '03'; 'F8';
'FE'; '0D'; '0A';]))};
0040
0041 cmd_name={cmd_name{:}, 'Set Equipment ID (1-80) Y 0x04 Set euqipment ID'};
0042 % 49 4D 55 43 01 02 04 01 40 F1 0D 0A
0043 cmd_strings={cmd_strings{:},uint8(hex2dec(['49'; '4D'; '55'; '43'; '01'; '02'; '04'; '01';
'40'; 'F1'; '0D'; '0A';]))};
0044
0045 cmd_name={cmd_name{:}, 'Set Sampling Time Y 0x05 Set IMU computational rate(1-500Hz)'};
0046 % 49 4D 55 43 01 03 05 01 F4 57 DB 0D 0A?500Hz?
0047 cmd_strings={cmd_strings{:},uint8(hex2dec(['49'; '4D'; '55'; '43'; '01'; '03'; '05'; '01';
'F4'; '57'; 'DB'; '0D'; '0A';]))};
0048
0049 cmd_name={cmd_name{:}, 'Set Message Output Rate Y 0x06 Set data output rate(1-255)'};
0050 % 49 4D 55 43 01 02 06 0A 97 F8 0D 0A?500Hz????10??=50Hz?????
0051 cmd_strings={cmd_strings{:},uint8(hex2dec(['49'; '4D'; '55'; '43'; '01'; '02'; '06'; '0A';
'97'; 'F8'; '0D'; '0A';]))};
0052
0053 cmd_name={cmd_name{:}, 'Set Sensitivity Gain Y 0x08 Set IMU sensitivity'};
0054 % 49 4D 55 43 01 05 08 3D F5 C2 8F 05 1F 0D 0A?0.12?
0055 cmd_strings={cmd_strings{:},uint8(hex2dec(['49'; '4D'; '55'; '43'; '01'; '05'; '08'; '3D';
'F5'; 'C2'; '8F'; '05'; '1F'; '0D'; '0A';]))};
0056
0057 cmd_name={cmd_name{:}, 'Set Geomagnetism Calibration Y 0x09 Set geomagnetism calibration
matrix'};
0058 % 49 4D 55 43 01 31 09 3F 85 28 2A 00 00 00 00 00 00 00 00 00 00 00 00 3F 7C 1D 8A 00 00 00
00 00 00 00 00 00 00 00 00 3F 83 B6 3D 00 00 00 00 00 00 00 00 00 00 00 00 C6 F7 0D 0A
0059 cmd_strings={cmd_strings{:},uint8(hex2dec(['49'; '4D'; '55'; '43'; '01'; '31'; '09'; '3F';
'85'; '28'; '2A'; '00'; '00'; '00'; '00'; '00'; '00'; '00'; '00'; '00'; '00'; '00'; '00'; '3F';
'7C'; '1D'; '8A'; '00'; '00'; '00'; '00'; '00'; '00'; '00'; '00'; '00'; '00'; '00'; '00'; '3F';
'83'; 'B6'; '3D'; '00'; '00'; '00'; '00'; '00'; '00'; '00'; '00'; '00'; '00'; '00'; '00'; 'C6';
```

```
'F7'; '0D'; '0A';]))};
0060
0061 cmd_name={cmd_name{:}, 'Calibration N 0x0F IMU calibration command'};
0062 % 49 4D 55 43 01 01 0F 39 72 0D 0A
0063 cmd_strings={cmd_strings{:},uint8(hex2dec(['49'; '4D'; '55'; '43'; '01'; '01'; '0F'; '39';
'72'; '0D'; '0A';]))};
0064
0065
0066 %% Data output command
0067
0068 cmd_name={cmd_name{:}, 'Counting Read Sensor Data Y 0x0100 Read n sensor data'};
0069 % 49 4D 55 43 01 04 01 00 00 20 90 60 0D 0A
0070 cmd_strings={cmd_strings{:},uint8(hex2dec(['49'; '4D'; '55'; '43'; '01'; '04'; '01'; '00';
'00'; '20'; '90'; '60'; '0D'; '0A';]))};
0071
0072 cmd_name={cmd_name{:}, 'Read Sensor Data N 0x0101 Continuously read sensor data'};
0073 % 49 4D 55 43 01 02 01 01 BF 04 0D 0A
0074 cmd_strings={cmd_strings{:},uint8(hex2dec(['49'; '4D'; '55'; '43'; '01'; '02'; '01'; '01';
'BF'; '04'; '0D'; '0A';]))};
0075
0076 % Counting Read Quaternion Data Y 0x0102 ????n??????
0077 cmd_name={cmd_name{:}, 'Counting Read Quaternion Data Y 0x102 Read n quaterinon data'};
0078 % 49 4D 55 43 01 04 01 02 00 20 FE 00 0D 0A
0079 cmd_strings={cmd_strings{:},uint8(hex2dec(['49'; '4D'; '55'; '43'; '01'; '04'; '01'; '02';
'00'; '20'; 'FE'; '00'; '0D'; '0A';]))};
0080
0081 cmd_name={cmd_name{:}, 'Read Quaternion Data N 0x0103 Continuously read quaternion data'};
0082 % 49 4D 55 43 01 02 01 03 9F 46 0D 0A
0083 cmd_strings={cmd_strings{:},uint8(hex2dec(['49'; '4D'; '55'; '43'; '01'; '02'; '01'; '03';
'9F'; '46'; '0D'; '0A';]))};
0084
0085 cmd_name={cmd_name{:}, 'Counting Read Angle Data Y 0x0104 Read n Euler angle data'};
0086 %49 4D 55 43 01 04 01 04 00 20 4C A0 0D 0A?32????
0087 cmd_strings={cmd_strings{:},uint8(hex2dec(['49'; '4D'; '55'; '43'; '01'; '04'; '01'; '04';
'00'; '20'; '4C'; 'A0'; '0D'; '0A';]))};
0088
0089 cmd_name={cmd_name{:}, 'Read Angle Data N 0x0105 Continuously read Euler angle data'};
0090 % 49 4D 55 43 01 02 01 05 FF 80 0D 0A
0091 cmd_strings={cmd_strings{:},uint8(hex2dec(['49'; '4D'; '55'; '43'; '01'; '02'; '01'; '05';
'FF'; '80'; '0D'; '0A';]))};
0092
0093 cmd_name={cmd_name{:}, 'Read Magnetic Data N 0x0107 Continuously read magnetic data'};
0094 % 49 4D 55 43 01 02 01 07 DF C2 0D 0A
0095 cmd_strings={cmd_strings{:},uint8(hex2dec(['49'; '4D'; '55'; '43'; '01'; '02'; '01'; '07';
'DF'; 'C2'; '0D'; '0A';]))};
0096
0097 cmd_name={cmd_name{:}, 'Read Quaternion and Magnetic Data N 0x0108 Continuously read
quaterinon and magnetic data'};
0098 % 49 4D 55 43 01 02 01 08 2E 2D 0D 0A
0099 cmd_strings={cmd_strings{:},uint8(hex2dec(['49'; '4D'; '55'; '43'; '01'; '02'; '01'; '08';
'2E'; '2D'; '0D'; '0A';]))};
0100
0101 cmd_name={cmd_name{:}, 'Read Quaternion and Acceleration Data N 0x0109 Continuously read
quaternion and acceleration data'};
0102 % 49 4D 55 43 01 02 01 09 3E 0C 0D 0A
0103 cmd_strings={cmd_strings{:},uint8(hex2dec(['49'; '4D'; '55'; '43'; '01'; '02'; '01'; '09';
'3E'; '0C'; '0D'; '0A';]))};
0104
0105 cmd_name={cmd_name{:}, 'Read Quaternion, Acceleration and Magnetic Data N 0x010A
Continuously read quaternion, acceleration and magnetic data'};
0106 % 49 4D 55 43 01 02 01 0A 0E 6F 0D 0A
0107 cmd_strings={cmd_strings{:},uint8(hex2dec(['49'; '4D'; '55'; '43'; '01'; '02'; '01'; '0A';
'0E'; '6F'; '0D'; '0A';]))};
0108
0109 cmd_name={cmd_name{:}, 'Read Direction Cosine Matrix Data N 0x010B Continuously read
direction cosine matrix data'};
0110 % 49 4D 55 43 01 02 01 0B 1E 4E 0D 0A
0111 cmd_strings={cmd_strings{:},uint8(hex2dec(['49'; '4D'; '55'; '43'; '01'; '02'; '01'; '0B';
'1E'; '4E'; '0D'; '0A';]))};
0112
0113 cmd_name={cmd_name{:}, 'Read Angle, Angular Rate, Acceleration and Magnetic Data N 0x010C
Continuously read kangle, angular rate acceleration and magnetic data'};
0114 % 49 4D 55 43 01 02 01 0C 6E A9 0D 0A
0115 cmd_strings={cmd_strings{:},uint8(hex2dec(['49'; '4D'; '55'; '43'; '01'; '02'; '01'; '0C';
'6E'; 'A9'; '0D'; '0A';]))};
0116
0117
0118 end
0119
```

api_createSerial

PURPOSE

API: serial object creation given parameters (first step!).

SYNOPSIS

function s = api_createSerial(comPort, baudrate)

DESCRIPTION

```
API: serial object creation given parameters (first step!).
Creats a new serial port to communicate with a IMU sensor. The default
baudrate is 115200 followingthe factory standard. Once connected all
asynchronous outputs will be turned off.

Example:
      s1 = api_createSerial();
      s2 = api_createSerial('COM8');
      s3 = api_createSerial('COM8', 115200);

See also SERIAL/INSTRFIND.

Copyright DunAn Precision, Inc.

Revision history:

v0.0 Initial release by Hongsheng He.
v0.1 Revised by Yang Yang @07/26/2017
```

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

- main 32A01T commmand test

SOURCE CODE

```
0001 function s = api_createSerial( comPort, baudrate )
0002 % API: serial object creation given parameters (first step!).
0003 % Creats a new serial port to communicate with a IMU sensor. The default
0004 % baudrate is 115200 followingthe factory standard. Once connected all
0005 % asynchronous outputs will be turned off.
0006 %
0007 % Example:
0008 %       s1 = api_createSerial();
0009 %       s2 = api_createSerial('COM8');
0010 %       s3 = api_createSerial('COM8', 115200);
0011 %
0012 % See also SERIAL/INSTRFIND.
0013 %
0014 % Copyright DunAn Precision, Inc.
0015 %
0016 % Revision history:
0017 %
0018 % v0.0 Initial release by Hongsheng He.
0019 % v0.1 Revised by Yang Yang @07/26/2017
```

```
0020 %
0021
0022 %%
0023 % Validate input arguments
0024 if nargin < 1
0025     if ismac || isunix
0026         % Code to run on Mac and Linux plaform
0027         comPort = inputdlg('Please input the serial port number ...', '',1,
{'/dev/tty.usbmodem1411'});
0028     elseif ispc
0029         % Code to run on Windows platform
0030         comPort = inputdlg('Please input the serial port number ...', '',1,
{char(seriallist)});
0031     else
0032         disp('Platform not supported')
0033     end
0034 end
0035 if ~exist('baudrate','var')
0036     baudrate = 115200;
0037 end
0038
0039
0040 %%
0041 % Release the COM object first before reopen a port
0042 delete(instrfindall);
0043
0044
0045 %%
0046 % Create a serial port
0047 s = serial(comPort, 'BaudRate', baudrate);
0048
0049 % Open serial object
0050 fopen(s);
0051
0052 end
0053
```

api_initialization

PURPOSE

API: initialize equipment (second setp!)

SYNOPSIS

function api_initialization(obj)

DESCRIPTION

```
API: initialize equipment (second setp!)

obj is the created serial port.

Example:
      api_initialization(s1);
```

CROSS-REFERENCE INFORMATION

This function calls:

- api_loadCommandList API: load whole command list

This function is called by:

- main 32A01T commmand test

SOURCE CODE

```
0001 function api_initialization(obj)
0002 % API: initialize equipment (second setp!)
0003 %
0004 % obj is the created serial port.
0005 %
0006 % Example:
0007 %       api_initialization(s1);
0008 %
0009
0010 %%
0011 % Validate input arguments
0012 switch (nargin)
0013     case 0
0014         error('Invalid input.');
0015     case 1
0016         if isvalid(obj)
0017
0018         else
0019             error('Invalid input.');
0020         end
0021 end
0022
0023
0024 %%
0025 % Load command list
0026 [ ~, cmd_strings ] = api_loadCommandList;
0027
0028
```

```
0029 %%
0030 % Reconfigure the serial port and start initialization
0031 disp('Initialization process starts: ');
0032
0033 fclose(obj);
0034 set(obj, 'BaudRate', 115200, 'DataBits', 8, 'Parity', 'none', 'StopBits', 1, ...
0035     'FlowControl', 'none', 'Timeout', 1, 'Terminator', 'CR/LF', ...
0036     'InputBufferSize', 2048, 'OutputBufferSize', 2048);
0037 obj.BytesAvailableFcnCount = 48;
0038 obj.BytesAvailableFcn = "";                                                    % No callback
function
0039 fopen(obj);
0040
0041 fwrite(obj, cmd_strings{:,5});
0042
0043 while obj.BytesAvailable <= 0
0044     pause(1);
0045 end
0046 res = fread(obj, obj.BytesAvailable);
0047 res = char(res)';
0048 info = [res(1:192),res(205:length(res))];
0049 disp(info);
0050
0051 if double(res(200)) == 1
0052     disp('Initialize successfully!');
0053 end
0054
0055 end
0056
0057
0058
0059
0060
0061
```

api_calibration

PURPOSE

API : calibrate to correct IMU gyro static drift (third setp!)

SYNOPSIS

function api_calibration(obj)

DESCRIPTION

```
API : calibrate to correct IMU gyro static drift (third setp!)
obj is the created serial port.
Example:
      api_calibration(s1);
```

CROSS-REFERENCE INFORMATION

This function calls:

- api_loadCommandList API: load whole command list

This function is called by:

- main 32A01T commmand test

SOURCE CODE

```
0001 function api_calibration(obj)
0002 % API : calibrate to correct IMU gyro static drift (third setp!)
0003 %
0004 % obj is the created serial port.
0005 %
0006 % Example:
0007 %       api_calibration(s1);
0008 %
0009
0010 %%
0011 % Validate input arguments
0012 switch (nargin)
0013     case 0
0014         error('Invalid input.');
0015     case 1
0016         if isvalid(obj)
0017
0018         else
0019             error('Invalid input.');
0020         end
0021 end
0022
0023
0024 %%
0025 % Load command list
0026 [ ~, cmd_strings ] = api_loadCommandList;
0027
0028
```

```
0029 %%
0030 % Reconfigure the serial port and start Calibration
0031 disp('Calibration process starts: ');
0032
0033 fclose(obj);
0034 set(obj, 'BaudRate', 115200, 'DataBits', 8, 'Parity', 'none', 'StopBits', 1, ...
0035     'FlowControl', 'none', 'Timeout', 1, 'Terminator', 'CR/LF', ...
0036     'InputBufferSize', 2048, 'OutputBufferSize', 2048);
0037 obj.BytesAvailableFcnCount = 48;
0038 obj.BytesAvailableFcn = "";                                                    % No callback
function
0039 fopen(obj);
0040
0041 fwrite(obj, cmd_strings{:,12});
0042
0043 while obj.BytesAvailable <= 0
0044     pause(1);
0045 end
0046 res = fread(obj, obj.BytesAvailable);
0047 res = char(res)';
0048 info = [res(1:29),res(42:length(res))];
0049 disp(info);
0050
0051 if double(res(37)) == 1
0052     disp('Calibrate successfully!');
0053 end
0054
0055 end
0056
0057
0058
0059
0060
0061
```

api_getSoftwareVersion

PURPOSE

API: get software version.

SYNOPSIS

function output = api_getSoftwareVersion(obj)

DESCRIPTION

```
API: get software version.
Read IMU embedded software version information. Read-only.

obj is the created serial port.

output is the IMU software version with 1x5 char format

Example:
      output = api_getSoftwareVersion(s1);
```

CROSS-REFERENCE INFORMATION

This function calls:

- api_loadCommandList API: load whole command list

This function is called by:

- main 32A01T commmand test

SOURCE CODE

```
0001 function output = api_getSoftwareVersion(obj)
0002 % API: get software version.
0003 % Read IMU embedded software version information. Read-only.
0004 %
0005 % obj is the created serial port.
0006 %
0007 % output is the IMU software version with 1x5 char format
0008 %
0009 % Example:
0010 %       output = api_getSoftwareVersion(s1);
0011 %
0012
0013 %%
0014 % Validate input arguments
0015 switch (nargin)
0016     case 0
0017         error('Invalid input.');
0018     case 1
0019         if isvalid(obj)
0020
0021         else
0022             error('Invalid input.');
0023         end
0024 end
0025
```

```
0026
0027 %%
0028 % Load command list
0029 [ ~, cmd_strings ] = api_loadCommandList;
0030
0031
0032 %%
0033 % Reconfigure the serial port and get software version
0034 fclose(obj);
0035 set(obj, 'BaudRate', 115200, 'DataBits', 8, 'Parity', 'none', 'StopBits', 1, ...
0036     'FlowControl', 'none', 'Timeout', 1, 'Terminator', 'CR/LF', ...
0037     'InputBufferSize', 2048, 'OutputBufferSize', 2048);
0038 obj.BytesAvailableFcnCount = 48;
0039 obj.BytesAvailableFcn = "";                                                 % No callback
function
0040 fopen(obj);
0041
0042 fwrite(obj, cmd_strings{:,1});                                              % Write the
'countingResdAng' command into the IMU
0043
0044 res = fscanf(obj);
0045 output = res(9:13);
0046
0047 end
```

---

api_get96bitID

PURPOSE

API: get 96-bit IMU ID.

SYNOPSIS

```
function output = api_get96bitID(obj)
```

DESCRIPTION

```
API: get 96-bit IMU ID.
Read IMU 96-bit global unique code. Read-only.

obj is the created serial port.

output is the 96-bit IMU ID with 1x96 char format

Example:
      output = api_get96bitID(s1);
```

CROSS-REFERENCE INFORMATION

This function calls:

- api_loadCommandList API: load whole command list

This function is called by:

- main 32A01T commmand test

SOURCE CODE

```
0001 function output = api_get96bitID(obj)
0002 % API: get 96-bit IMU ID.
0003 % Read IMU 96-bit global unique code. Read-only.
0004 %
0005 % obj is the created serial port.
0006 %
0007 % output is the 96-bit IMU ID with 1x96 char format
0008 %
0009 % Example:
0010 %       output = api_get96bitID(s1);
0011 %
0012
0013 %%
0014 % Validate input arguments
0015 switch (nargin)
0016     case 0
0017         error('Invalid input.');
0018     case 1
0019         if isvalid(obj)
0020
0021         else
0022             error('Invalid input.');
0023         end
0024 end
0025
```

```
0026
0027 %%
0028 % Load command list
0029 [ ~, cmd_strings ] = api_loadCommandList;
0030
0031
0032 %%
0033 % Reconfigure the serial port and get 96-bit ID
0034 fclose(obj);
0035 set(obj, 'BaudRate', 115200, 'DataBits', 8, 'Parity', 'none', 'StopBits', 1, ...
0036     'FlowControl', 'none', 'Timeout', 1, 'Terminator', 'CR/LF', ...
0037     'InputBufferSize', 2048, 'OutputBufferSize', 2048);
0038 obj.BytesAvailableFcnCount = 48;
0039 obj.BytesAvailableFcn = "";                                          % No callback
function
0040 fopen(obj);
0041
0042 fwrite(obj, cmd_strings{:,2});                                       % Write the
'countingResdAng' command into the IMU
0043
0044 res = fscanf(obj);
0045 output = reshape(dec2bin(res(9:20),8),1,96);
0046
0047 end
```

api_getBasicInfo

PURPOSE

API: get IMU basic information.

SYNOPSIS

function output = api_getBasicInfo(obj)

DESCRIPTION

```
API: get IMU basic information.
obj is the created serial port.
output is the basic IMU information with char format
Example:
      output = api_getBasicInfo(s1, cmd_strings);
```

CROSS-REFERENCE INFORMATION

This function calls:

- api_loadCommandList API: load whole command list

This function is called by:

- main 32A01T commmand test

SOURCE CODE

```
0001 function output = api_getBasicInfo(obj)
0002 % API: get IMU basic information.
0003 %
0004 % obj is the created serial port.
0005 %
0006 % output is the basic IMU information with char format
0007 %
0008 % Example:
0009 %       output = api_getBasicInfo(s1, cmd_strings);
0010 %
0011
0012 %%
0013 % Validate input arguments
0014 switch (nargin)
0015     case 0
0016         error('Invalid input.');
0017     case 1
0018         if isvalid(obj)
0019
0020         else
0021             error('Invalid input.');
0022         end
0023 end
0024
0025
0026 %%
```

```
0027 % Load command list
0028 [ ~, cmd_strings ] = api_loadCommandList;
0029
0030
0031 %%
0032 % Reconfigure the serial port and get IMU basic information
0033 fclose(obj);
0034 set(obj, 'BaudRate', 115200, 'DataBits', 8, 'Parity', 'none', 'StopBits', 1, ...
0035     'FlowControl', 'none', 'Timeout', 1, 'Terminator', 'CR/LF', ...
0036     'InputBufferSize', 2048, 'OutputBufferSize', 2048);
0037 obj.BytesAvailableFcnCount = 48;
0038 obj.BytesAvailableFcn = "";                                                    % No callback
function
0039 fopen(obj);
0040
0041 fwrite(obj, cmd_strings{:,3});                                                 % Write the
'countingResdAng' command into the IMU
0042
0043 while obj.BytesAvailable <= 0
0044     pause(1);
0045 end
0046
0047 res = char(fread(obj, obj.BytesAvailable))';
0048 output = res(1:length(res));
0049 end
```

api_setEquipID

PURPOSE

API: set equipment ID (1-80)(default = 1)

SYNOPSIS

function output = api_setEquipID(obj, equipID)

DESCRIPTION

```
API: set equipment ID (1-80)(default = 1)

obj is the created serial port.

equipID is the user defined equipment ID number (1-80)

output is the updated equipment ID with double format

Example:
      output1 = api_setEquipID(s1);
      output2 = api_setEquipID(s1, 1);
```

CROSS-REFERENCE INFORMATION

This function calls:

- CRC Accurate implementations of the 16-bit CRC-CCITT, used with a look up table.
- api_loadCommandList API: load whole command list

This function is called by:

- main 32A01T commmand test

SOURCE CODE

```
0001 function output = api_setEquipID(obj, equipID)
0002 % API: set equipment ID (1-80)(default = 1)
0003 %
0004 % obj is the created serial port.
0005 %
0006 % equipID is the user defined equipment ID number (1-80)
0007 %
0008 % output is the updated equipment ID with double format
0009 %
0010 % Example:
0011 %       output1 = api_setEquipID(s1);
0012 %       output2 = api_setEquipID(s1, 1);
0013 %
0014
0015 %%
0016 % Validate input arguments
0017 switch (nargin)
0018     case 0
0019         error('Invalid input.');
0020     case 1
0021         if isvalid(obj)
0022             equipID = 1;
```

```
0023         end
0024     case 2
0025         if isvalid(obj) && (equipID == fix(equipID))...
0026                 && (equipID <=80) && (equipID >= 1)
0027
0028         else
0029             error('Invalid input.');
0030         end
0031 end
0032
0033
0034 %%
0035 % Load command list
0036 [ ~, cmd_strings ] = api_loadCommandList;
0037
0038
0039 %%
0040 % Reconfigure the port and set equipment ID
0041 fclose(obj);
0042 set(obj, 'BaudRate', 115200, 'DataBits', 8, 'Parity', 'none', 'StopBits', 1, ...
0043     'FlowControl', 'none', 'Timeout', 1, 'Terminator', 'CR/LF', ...
0044     'InputBufferSize', 2048, 'OutputBufferSize', 2048);
0045 obj.BytesAvailableFcnCount = 48;
0046 obj.BytesAvailableFcn = "";                                          % No callback
function
0047 fopen(obj);
0048
0049 command = dec2hex(equipID, 2);                                       % Modify the
equipment ID sending to the IMU
0050 command = uint8(hex2dec(command));
0051 cmd_strings{:, 7}(8) = command;
0052
0053 crc = CRC(cmd_strings{:, 7}(5:8));                                   % Update CRC
0054 cmd_strings{:, 7}(9) = uint8(hex2dec(crc(1:2)));
0055 cmd_strings{:, 7}(10) = uint8(hex2dec(crc(3:4)));
0056
0057 fwrite(obj, cmd_strings{:,7});                                       % Write the
'countingResdAng' command into the IMU
0058
0059 while obj.BytesAvailable <= 0
0060     pause(1);
0061 end
0062
0063 res = fread(obj, obj.BytesAvailable);
0064 output = res(8);
0065
0066 end
```

api_setSamplingRate

PURPOSE

API: set sampling rate (1-500) [Hz](default = 500)

SYNOPSIS

function output = api_setSamplingRate(obj, samplingRate)

DESCRIPTION

```
API: set sampling rate (1-500) [Hz](default = 500)
obj is the created serial port.
samplingRate is the user defined IMU sampling rate (1-500)[HZ]
output is the updated sampling rate message with char format
Example:
      output1 = api_setSamplingRate(s1)
      output2 = api_setSamplingRate(s1, 500);
```

CROSS-REFERENCE INFORMATION

This function calls:

- CRC Accurate implementations of the 16-bit CRC-CCITT, used with a look up table.
- api_loadCommandList API: load whole command list

This function is called by:

- main 32A01T commmand test

SOURCE CODE

```
0001 function output = api_setSamplingRate(obj, samplingRate)
0002 % API: set sampling rate (1-500) [Hz](default = 500)
0003 %
0004 % obj is the created serial port.
0005 %
0006 % samplingRate is the user defined IMU sampling rate (1-500)[HZ]
0007 %
0008 % output is the updated sampling rate message with char format
0009 %
0010 % Example:
0011 %
0012 %       output1 = api_setSamplingRate(s1)
0013 %       output2 = api_setSamplingRate(s1, 500);
0014 %
0015
0016 %%
0017 % Validate input arguments
0018 switch (nargin)
0019     case 0
0020         error('Invalid input.');
0021     case 1
```

```
0022         if isvalid(obj)
0023             samplingRate = 500;
0024         end
0025     case 2
0026         if isvalid(obj) && (samplingRate == fix(samplingRate))...
0027                 && (samplingRate <= 500) && (samplingRate >= 1)
0028
0029         else
0030             error('Invalid input.');
0031         end
0032 end
0033
0034
0035 %%
0036 % Load command list
0037 [ ~, cmd_strings ] = api_loadCommandList;
0038
0039
0040 %%
0041 % Reconfigure the serial port and set sampling rate
0042 fclose(obj);
0043 set(obj, 'BaudRate', 115200, 'DataBits', 8, 'Parity', 'none', 'StopBits', 1, ...
0044     'FlowControl', 'none', 'Timeout', 1, 'Terminator', 'CR/LF', ...
0045     'InputBufferSize', 2048, 'OutputBufferSize', 2048);
0046 obj.BytesAvailableFcnCount = 48;
0047 obj.BytesAvailableFcn = "";                                              % No callback
function
0048 fopen(obj);
0049
0050 command = dec2hex(samplingRate,4);                                       % Modify the
command sending to the IMU
0051 command = uint8(hex2dec(reshape(command,2,2)'));
0052 cmd_strings{:,8}(8:9) = command;
0053
0054 crc = CRC(cmd_strings{:, 8}(5:9));                                       % Update CRC
0055 cmd_strings{:, 8}(10) = uint8(hex2dec(crc(1:2)));
0056 cmd_strings{:, 8}(11) = uint8(hex2dec(crc(3:4)));
0057
0058 fwrite(obj, cmd_strings{:,8});                                           % Write the
'countingResdAng' command into the IMU
0059
0060 while obj.BytesAvailable <= 0
0061     pause(1);
0062 end
0063
0064 res = fread(obj, obj.BytesAvailable);
0065 if res(8) == 1
0066     output = char(res(13:length(res)))';
0067 else
0068     disp('Sampliling rate setting fails');
0069 end
0070
0071 end
```

api_setOutputRate

PURPOSE

API: set IMU output frequecy division (1-255)(default = 10)

SYNOPSIS

function output = api_setOutputRate(obj, division)

DESCRIPTION

```
 API: set IMU output frequecy division (1-255)(default = 10)
 Frequency division number: if it is needed to set output rate at 50Hz,
 the division should be set as 10 when the sampling rate is 500 Hz.

 obj is the created serial port.

 division is the user defined frequency division (1 - 255)

 output is the updated output rate with char format

 Example:

       output1 = api_setOutputRate(s1)
       output2 = api_setOutputRate(s1, 500);

%
 Validate input arguments
```

CROSS-REFERENCE INFORMATION

This function calls:

- CRC Accurate implementations of the 16-bit CRC-CCITT, used with a look up table.
- api_loadCommandList API: load whole command list

This function is called by:

- main 32A01T commmand test

SOURCE CODE

```
0001 function output = api_setOutputRate(obj, division)
0002 % API: set IMU output frequecy division (1-255)(default = 10)
0003 % Frequency division number: if it is needed to set output rate at 50Hz,
0004 % the division should be set as 10 when the sampling rate is 500 Hz.
0005 %
0006 % obj is the created serial port.
0007 %
0008 % division is the user defined frequency division (1 - 255)
0009 %
0010 % output is the updated output rate with char format
0011 %
0012 % Example:
0013 %
0014 %       output1 = api_setOutputRate(s1)
0015 %       output2 = api_setOutputRate(s1, 500);
0016 %
```

```
0017 %%
0018 % Validate input arguments
0019 switch (nargin)
0020     case 0
0021         error('Invalid input.');
0022     case 1
0023         if isvalid(obj)
0024             division = 10;
0025         end
0026     case 2
0027         if isvalid(obj) && (division == fix(division))...
0028                 && (division <= 255) && (division >= 1)
0029
0030         else
0031             error(message('Invalid input.'));
0032         end
0033 end
0034
0035
0036 %%
0037 % Load command list
0038 [ ~, cmd_strings ] = api_loadCommandList;
0039
0040
0041 %%
0042 % Reconfigure the serial port and set output rate
0043 fclose(obj);
0044 set(obj, 'BaudRate', 115200, 'DataBits', 8, 'Parity', 'none', 'StopBits', 1, ...
0045     'FlowControl', 'none', 'Timeout', 1, 'Terminator', 'CR/LF', ...
0046     'InputBufferSize', 2048, 'OutputBufferSize', 2048);
0047 obj.BytesAvailableFcnCount = 48;
0048 obj.BytesAvailableFcn = "";                                                    % No callback
function
0049 fopen(obj);
0050
0051 command = dec2hex(division, 2);                                                 % Modify the
equipment ID sending to the IMU
0052 command = uint8(hex2dec(command));
0053 cmd_strings{:, 9}(8) = command;
0054
0055 crc = CRC(cmd_strings{:, 9}(5:8));                                             % Update CRC
0056 cmd_strings{:, 9}(9) = uint8(hex2dec(crc(1:2)));
0057 cmd_strings{:, 9}(10) = uint8(hex2dec(crc(3:4)));
0058
0059 fwrite(obj, cmd_strings{:, 9});                                                % Write the
'countingResdAng' command into the IMU
0060
0061 while obj.BytesAvailable <= 0
0062     pause(1);
0063 end
0064
0065 res = fread(obj, obj.BytesAvailable);
0066 output = char(res(13:length(res)))';
0067
0068 end
0069
```

api_setSensitivity

PURPOSE

API: set IMU sensitivity (0.04 - 0.4)(default = 0.12)

SYNOPSIS

function output = api_setSensitivity(obj, sensitivity)

DESCRIPTION

```
API: set IMU sensitivity (0.04 - 0.4)(default = 0.12)
obj is the created serial port.
sensitivity is the user defined IMU sensitivity (0.04 - 0.4)
output is the updated sensitivity with double format
Example:
      output1 = api_setSensitivity(s1)
      output2 = api_setSensitivity(s1, 0.12);
```

CROSS-REFERENCE INFORMATION

This function calls:

- CRC Accurate implementations of the 16-bit CRC-CCITT, used with a look up table.
- api_loadCommandList API: load whole command list
- hexsingle2num Convert single precision IEEE hexadecimal string to number.

This function is called by:

- main 32A01T commmand test

SOURCE CODE

```
0001 function output = api_setSensitivity(obj, sensitivity)
0002 % API: set IMU sensitivity (0.04 - 0.4)(default = 0.12)
0003 %
0004 % obj is the created serial port.
0005 %
0006 % sensitivity is the user defined IMU sensitivity (0.04 - 0.4)
0007 %
0008 % output is the updated sensitivity with double format
0009 %
0010 % Example:
0011 %
0012 %       output1 = api_setSensitivity(s1)
0013 %       output2 = api_setSensitivity(s1, 0.12);
0014 %
0015
0016 %%
0017 % Validate input arguments
0018 switch (nargin)
0019     case 0
```

```
0020         error('Invalid input.');
0021     case 1
0022         if isvalid(obj)
0023             sensitivity = 0.12;
0024         end
0025     case 2
0026         if isvalid(obj)...
0027                 && (sensitivity <= 0.4) && (sensitivity >= 0.04)
0028
0029         else
0030             error('Invalid input.');
0031         end
0032 end
0033
0034
0035 %%
0036 % Load command list
0037 [ ~, cmd_strings ] = api_loadCommandList;
0038
0039
0040 %%
0041 % Reconfigure the serial port and set sensitivity
0042 fclose(obj);
0043 set(obj, 'BaudRate', 115200, 'DataBits', 8, 'Parity', 'none', 'StopBits', 1, ...
0044     'FlowControl', 'none', 'Timeout', 1, 'Terminator', 'CR/LF', ...
0045     'InputBufferSize', 2048, 'OutputBufferSize', 2048);
0046 obj.BytesAvailableFcnCount = 48;
0047 obj.BytesAvailableFcn = "";                                          % No callback
function
0048 fopen(obj);
0049
0050 command = num2hex(single(sensitivity));                              % Modify the
equipment ID sending to the IMU
0051 cmd_strings{:, 10}(8:11) =
uint8(hex2dec([command(1:2);command(3:4);command(5:6);command(7:8)]));
0052
0053 crc = CRC(cmd_strings{:, 10}(5:11));                                 % Update CRC
0054 cmd_strings{:, 10}(12) = uint8(hex2dec(crc(1:2)));
0055 cmd_strings{:, 10}(13) = uint8(hex2dec(crc(3:4)));
0056
0057 fwrite(obj, cmd_strings{:, 10});                                     % Write the
'countingResdAng' command into the IMU
0058
0059 while obj.BytesAvailable <= 0
0060     pause(1);
0061 end
0062
0063 res = fread(obj, obj.BytesAvailable);
0064
0065 display = dec2hex(uint8(res(8:11)));
0066 output = hexsingle2num(reshape(display',1,8));
0067 end
```

api_countingReadSensor

PURPOSE

API: counting read sensor data

SYNOPSIS

function output = api_countingReadSensor(obj, counts, saveData)

DESCRIPTION

```
API: counting read sensor data

obj is the created serial port.

counts is the user defined number (1 - 65535) of data to be collected
(default = 32)

saveData is a logical flag to determine wheter to save the collected data
e.g. save == ture -> save | save == false -> don't save (default)

output is collected data with double format.
Data structure: [gyro_x, gyro_y, gyro_z, acc_x, acc_y, acc_z, ...
                 mag_x, mag_y, mag_z] [rad/s, g, mG]
Data format:
  Gyro: singed 16-bit, 0.0125 deg/s/LSB
  Acc: singed 16-bit, 1/8192 g/LSB
  Mag: singed 16-bit, 1/75 gauss/LSB

Example:

      output1 = api_countingReadSensor(s1)
      output2 = api_countingReadSensor(s1, 100);
      output3 = api_countingReadSensor(s1, 100, true)
```

CROSS-REFERENCE INFORMATION

This function calls:

- CRC Accurate implementations of the 16-bit CRC-CCITT, used with a look up table.
- api_loadCommandList API: load whole command list
- callback_countingReadSensor callback: counting read sensor data

This function is called by:

- main 32A01T commmand test

SOURCE CODE

```
0001 function output = api_countingReadSensor(obj, counts, saveData)
0002 % API: counting read sensor data
0003 %
0004 % obj is the created serial port.
0005 %
0006 % counts is the user defined number (1 - 65535) of data to be collected
0007 % (default = 32)
0008 %
```

```
0009 % saveData is a logical flag to determine wheter to save the collected data
0010 % e.g. save == ture -> save | save == false -> don't save (default)
0011 %
0012 % output is collected data with double format.
0013 % Data structure: [gyro_x, gyro_y, gyro_z, acc_x, acc_y, acc_z, ...
0014 %                  mag_x, mag_y, mag_z] [rad/s, g, mG]
0015 % Data format:
0016 %   Gyro: singed 16-bit, 0.0125 deg/s/LSB
0017 %   Acc: singed 16-bit, 1/8192 g/LSB
0018 %   Mag: singed 16-bit, 1/75 gauss/LSB
0019 %
0020 % Example:
0021 %
0022 %       output1 = api_countingReadSensor(s1)
0023 %       output2 = api_countingReadSensor(s1, 100);
0024 %       output3 = api_countingReadSensor(s1, 100, true)
0025 %
0026
0027 %%
0028 % Validate input arguments
0029 switch (nargin)
0030     case 0
0031         error('Invalid input.');
0032     case 1
0033         if isvalid(obj)
0034             counts = 32;
0035             saveData = false;
0036         end
0037     case 2
0038         if isvalid(obj) && (counts == fix(counts))...
0039                 && (counts <= 65535) && (counts >= 1)
0040             saveData = false;
0041         else
0042             error('Invalid input.');
0043         end
0044     case 3
0045         if isvalid(obj) && (counts == fix(counts))...
0046                 && (counts <= 65535) && (counts >= 1) && islogical(saveData)
0047         else
0048             error('Invalid input');
0049         end
0050 end
0051
0052
0053 %%
0054 % Define global variable sensorData to store collected data
0055 global sensorData;
0056 sensorData = [];
0057
0058
0059 %%
0060 % Load command list
0061 [ ~, cmd_strings ] = api_loadCommandList;
0062
0063
0064 %%
0065 % Reconfigure the serial port and counting read sensor data
0066 command = dec2hex(counts,4);                                          % Modify the
command sending to the IMU
0067 command = uint8(hex2dec(reshape(command,2,2)'));
0068 cmd_strings{:,13}(9:10) = command;
0069
0070 crc = CRC(cmd_strings{:, 13}(5:10));                                  % Update CRC
0071 cmd_strings{:, 13}(11) = uint8(hex2dec(crc(1:2)));
0072 cmd_strings{:, 13}(12) = uint8(hex2dec(crc(3:4)));
0073
0074 fclose(obj);
0075 set(obj,'BaudRate',115200,'DataBits',8,'StopBits',1,'Parity','none',...
0076       'FlowControl','none');
0077 obj.BytesAvailableFcnMode = 'byte';
0078 obj.BytesAvailableFcnCount = 32;                                      % Activite the
callback function when receive every 32 byte data
0079 obj.BytesAvailableFcn = @callback_countingReadSensor;                 % Define
callback function
0080 fopen(obj);
0081
0082 fwrite(obj, cmd_strings{:,13});                                       % Write the
'countingResdSensor' command into the IMU
0083 pause(5);
0084 output = sensorData;
0085
```

```
0086 if saveData == true
0087     save('countingReadSensor.txt', 'sensorData', '-ascii');                % Save data to
.txt
0088 end
0089
0090 clear global sensorData;                                                  % Clear global
variable sensorData
0091
0092
0093 end
```

---

api_countingReadQuat

PURPOSE

API: counting read quaternion data

SYNOPSIS

function output = api_countingReadQuat(obj, counts, saveData)

DESCRIPTION

```
API: counting read quaternion data

obj is the created serial port.

counts is the user defined number (1 - 65535) of data to be collected
(default = 32)

saveData is a logical flag to determine wheter to save the collected data
e.g. save == ture -> save | save == false -> don't save (default)

output is collected data with double format.
Data structure:  [q0, q1, q2, q3]
Data format: single precision floating point number

Example:

      output1 = api_countingReadQuat(s1)
      output2 = api_countingReadQuat(s1, 100);
      output3 = api_countingReadQuat(s1, 100, true);
```

CROSS-REFERENCE INFORMATION

This function calls:

- CRC Accurate implementations of the 16-bit CRC-CCITT, used with a look up table.
- api_loadCommandList API: load whole command list
- callback_countingReadQuat callback: counting read quaternion

This function is called by:

- main 32A01T commmand test

SOURCE CODE

```
0001 function output = api_countingReadQuat(obj, counts, saveData)
0002 % API: counting read quaternion data
0003 %
0004 % obj is the created serial port.
0005 %
0006 % counts is the user defined number (1 - 65535) of data to be collected
0007 % (default = 32)
0008 %
0009 % saveData is a logical flag to determine wheter to save the collected data
0010 % e.g. save == ture -> save | save == false -> don't save (default)
0011 %
0012 % output is collected data with double format.
```

```
% Data structure:  [q0, q1, q2, q3]
% Data format: single precision floating point number
%
% Example:
%
%       output1 = api_countingReadQuat(s1)
%       output2 = api_countingReadQuat(s1, 100);
%       output3 = api_countingReadQuat(s1, 100, true);
%

%%
% Validate input arguments
switch (nargin)
    case 0
        error('Invalid input.');
    case 1
        if isvalid(obj)
            counts = 32;
            saveData = false;
        end
    case 2
        if isvalid(obj) && (counts == fix(counts))...
                && (counts <= 65535) && (counts >= 1)
            saveData = false;
        else
            error('Invalid input.');
        end
    case 3
        if isvalid(obj) && (counts == fix(counts))...
                && (counts <= 65535) && (counts >= 1) && islogical(saveData)
        else
            error('Invalid input');
        end
end


%%
% Define global variable sensorData to store collected data
global sensorData;
sensorData = [];


%%
% Load command list
[ ~, cmd_strings ] = api_loadCommandList;


%%
% Reconfigure the serial port and counting read quaternion data
command = dec2hex(counts,4);                                            % Modify the
command sending to the IMU
command = uint8(hex2dec(reshape(command,2,2)'));
cmd_strings{:,15}(9:10) = command;

crc = CRC(cmd_strings{:, 15}(5:10));                                    % Update CRC
cmd_strings{:, 15}(11) = uint8(hex2dec(crc(1:2)));
cmd_strings{:, 15}(12) = uint8(hex2dec(crc(3:4)));

fclose(obj);
set(obj,'BaudRate',115200,'DataBits',8,'StopBits',1,'Parity','none',...
      'FlowControl','none');
obj.BytesAvailableFcnMode = 'byte';
obj.BytesAvailableFcnCount = 30;                                        % Activite the
callback function when receive every 32 byte data
obj.BytesAvailableFcn = @callback_countingReadQuat;                     % Define
callback function
fopen(obj);

fwrite(obj, cmd_strings{:,15});                                         % Write the
'countingResdQuat' command into the IMU
pause(5);
output = sensorData;

if saveData == true
    save('countingReadQuat.txt', 'sensorData', '-ascii');               % Save data to
.txt
end

clear global sensorData;                                                % Clear global
variable sensorData

```

```
0088
0089 end
```

---

api_countingReadAng

PURPOSE

API: counting read Euler angle data

SYNOPSIS

function output = api_countingReadAng(obj, counts, saveData)

DESCRIPTION

```
API: counting read Euler angle data

obj is the created serial port.

counts is the user defined number (1 - 65535) of data to be collected
(default = 32)

saveData is a logical flag to determine wheter to save the collected data
e.g. save == ture -> save | save == false -> don't save (default)

output is collected data with double format.
Data structure:  [roll, pitch, yaw] [rad]
Data format: single precision floating point number

Example:

      output1 = api_countingReadAng(s1)
      output2 = api_countingReadAng(s1, 100);
      output3 = api_countingReadAng(s1, 100, true);
```

CROSS-REFERENCE INFORMATION

This function calls:

- CRC Accurate implementations of the 16-bit CRC-CCITT, used with a look up table.
- api_loadCommandList API: load whole command list
- callback_countingReadAng callback: counting read Euler angle data

This function is called by:

- main 32A01T commmand test

SOURCE CODE

```
0001 function output = api_countingReadAng(obj, counts, saveData)
0002 % API: counting read Euler angle data
0003 %
0004 % obj is the created serial port.
0005 %
0006 % counts is the user defined number (1 - 65535) of data to be collected
0007 % (default = 32)
0008 %
0009 % saveData is a logical flag to determine wheter to save the collected data
0010 % e.g. save == ture -> save | save == false -> don't save (default)
0011 %
0012 % output is collected data with double format.
```

```
0013 % Data structure:  [roll, pitch, yaw] [rad]
0014 % Data format: single precision floating point number
0015 %
0016 % Example:
0017 %
0018 %       output1 = api_countingReadAng(s1)
0019 %       output2 = api_countingReadAng(s1, 100);
0020 %       output3 = api_countingReadAng(s1, 100, true);
0021 %
0022
0023 %%
0024 % Validate input arguments
0025 switch (nargin)
0026     case 0
0027         error('Invalid input.');
0028     case 1
0029         if isvalid(obj)
0030             counts = 32;
0031             saveData = false;
0032         end
0033     case 2
0034         if isvalid(obj) && (counts == fix(counts))...
0035                 && (counts <= 65535) && (counts >= 1)
0036             saveData = false;
0037         else
0038             error('Invalid input.');
0039         end
0040     case 3
0041         if isvalid(obj) && (counts == fix(counts))...
0042                 && (counts <= 65535) && (counts >= 1) && islogical(saveData)
0043         else
0044             error('Invalid input');
0045         end
0046 end
0047
0048
0049 %%
0050 % Define global variable sensorData to store collected data
0051 global sensorData;
0052 sensorData = [];
0053
0054
0055 %%
0056 % Load command list
0057 [ ~, cmd_strings ] = api_loadCommandList;
0058
0059
0060 %%
0061 % Reconfigure the serial port and counting read quaternion data
0062 command = dec2hex(counts,4);                                        % Modify the
command sending to the IMU
0063 command = uint8(hex2dec(reshape(command,2,2)'));
0064 cmd_strings{:,17}(9:10) = command;
0065
0066 crc = CRC(cmd_strings{:, 17}(5:10));                                % Update CRC
0067 cmd_strings{:, 17}(11) = uint8(hex2dec(crc(1:2)));
0068 cmd_strings{:, 17}(12) = uint8(hex2dec(crc(3:4)));
0069
0070 fclose(obj);
0071 set(obj,'BaudRate',115200,'DataBits',8,'StopBits',1,'Parity','none',...
0072       'FlowControl','none');
0073 obj.BytesAvailableFcnMode = 'byte';
0074 obj.BytesAvailableFcnCount = 26;                                    % Activite the
callback function when receive every 32 byte data
0075 obj.BytesAvailableFcn = @callback_countingReadAng;                  % Define
callback function
0076 fopen(obj);
0077
0078 fwrite(obj, cmd_strings{:,17});                                     % Write the
'countingResdAng' command into the IMU
0079 pause(5);
0080 output = sensorData;
0081
0082 if saveData == true
0083     save('countingReadAng.txt', 'sensorData', '-ascii');            % Save data to
.txt
0084 end
0085
0086 clear global sensorData;                                            % Clear global
variable sensorData
0087
```

```
0088 end
```

---

api_serialReadSensor

PURPOSE

API: serial read sensor data

SYNOPSIS

function output = api_serialReadSensor(obj, saveData)

DESCRIPTION

```
API: serial read sensor data

obj is the created serial port.

saveData is a logical flag to determine wheter to save the collected data
e.g. save == ture -> save | save == false -> don't save (default)

output is collected data with double format.
Data structure: [gyro_x, gyro_y, gyro_z, acc_x, acc_y, acc_z, ...
                 mag_x, mag_y, mag_z] [rad/s, g, mG]
Data format:
  Gyro: singed 16-bit, 0.0125 deg/s/LSB
  Acc: singed 16-bit, 1/8192 g/LSB
  Mag: singed 16-bit, 1/75 gauss/LSB

Example:

      output1 = api_serialReadSensor(s1)
      output2 = api_serialReadSensor(s1, true);
```

CROSS-REFERENCE INFORMATION

This function calls:

- api_loadCommandList API: load whole command list
- callback_serialReadSensor callback: serial read sensor data

This function is called by:

- main 32A01T commmand test

SOURCE CODE

```
0001 function output = api_serialReadSensor(obj, saveData)
0002 % API: serial read sensor data
0003 %
0004 % obj is the created serial port.
0005 %
0006 % saveData is a logical flag to determine wheter to save the collected data
0007 % e.g. save == ture -> save | save == false -> don't save (default)
0008 %
0009 % output is collected data with double format.
0010 % Data structure: [gyro_x, gyro_y, gyro_z, acc_x, acc_y, acc_z, ...
0011 %                  mag_x, mag_y, mag_z] [rad/s, g, mG]
0012 % Data format:
0013 %   Gyro: singed 16-bit, 0.0125 deg/s/LSB
0014 %   Acc: singed 16-bit, 1/8192 g/LSB
```

```
0015 %   Mag: singed 16-bit, 1/75 gauss/LSB
0016 %
0017 % Example:
0018 %
0019 %       output1 = api_serialReadSensor(s1)
0020 %       output2 = api_serialReadSensor(s1, true);
0021 %
0022
0023 %%
0024 % Validate input arguments
0025 switch (nargin)
0026     case 0
0027         error('Invalid input.');
0028     case 1
0029         if isvalid(obj)
0030             saveData = false;
0031         else
0032             error('Invalid input.');
0033         end
0034     case 2
0035         if isvalid(obj) && islogical(saveData)
0036         else
0037             error('Invalid input.');
0038         end
0039 end
0040
0041
0042 %%
0043 % Define global variable sensorData to store collected data
0044 global sensorData;
0045 sensorData = [];
0046
0047
0048 %%
0049 % Load command list
0050 [ ~, cmd_strings ] = api_loadCommandList;
0051
0052
0053 %%
0054 % Reconfigure the serial port and serial read sensor data
0055 fclose(obj);
0056 set(obj,'BaudRate',115200,'DataBits',8,'StopBits',1,'Parity','none',...
0057       'FlowControl','none');
0058 obj.BytesAvailableFcnMode = 'byte';
0059 obj.BytesAvailableFcnCount = 32;                                    % Activite the
callback function when receive every 32 byte data
0060 obj.BytesAvailableFcn = @callback_serialReadSensor;                 % Define
callback function
0061 fopen(obj);
0062
0063 fwrite(obj, cmd_strings{:,14});                                     % Write the
'serialReadQuat' command into the IMU
0064 pause(5);
0065 fwrite(obj, cmd_strings{:,6});                                      % Stop command
0066 output = sensorData;
0067
0068 if saveData == true
0069     save('serialReadSensor.txt', 'sensorData', '-ascii');           % Save data to
.txt
0070 end
0071
0072 clear global sensorData;                                           % Clear global
variable sensorData
0073
0074 end
```

api_serialReadQuat

PURPOSE

API: serial read quaternion

SYNOPSIS

function output = api_serialReadQuat(obj, saveData)

DESCRIPTION

```
API: serial read quaternion

obj is the created serial port.

saveData is a logical flag to determine wheter to save the collected data
e.g. save == ture -> save | save == false -> don't save (default)

output is collected data with double format.
Data structure: [q0, q1, q2, q3]
Data format: single precision floating point number

Example:

      output1 = api_serialReadQuat(s1)
```

CROSS-REFERENCE INFORMATION

This function calls:

- api_loadCommandList API: load whole command list
- callback_serialReadQuat callback: serial read quaternion data

This function is called by:

- main 32A01T commmand test

SOURCE CODE

```
0001 function output = api_serialReadQuat(obj, saveData)
0002 % API: serial read quaternion
0003 %
0004 % obj is the created serial port.
0005 %
0006 % saveData is a logical flag to determine wheter to save the collected data
0007 % e.g. save == ture -> save | save == false -> don't save (default)
0008 %
0009 % output is collected data with double format.
0010 % Data structure: [q0, q1, q2, q3]
0011 % Data format: single precision floating point number
0012 %
0013 % Example:
0014 %
0015 %       output1 = api_serialReadQuat(s1)
0016 %
0017
0018 %%
0019 % Validate input arguments
```

```
0020 switch (nargin)
0021     case 0
0022         error('Invalid input.');
0023     case 1
0024         if isvalid(obj)
0025             saveData = false;
0026         else
0027             error('Invalid input.');
0028         end
0029     case 2
0030         if isvalid(obj) && islogical(saveData)
0031         else
0032             error('Invalid input.');
0033         end
0034 end
0035
0036
0037 %%
0038 % Define global variable sensorData to store collected data
0039 global sensorData;
0040 sensorData = [];
0041
0042
0043 %%
0044 % Load command list
0045 [ ~, cmd_strings ] = api_loadCommandList;
0046
0047
0048 %%
0049 % Reconfigure the serial port and serial read sensor data
0050 fclose(obj);
0051 set(obj,'BaudRate',115200,'DataBits',8,'StopBits',1,'Parity','none',...
0052       'FlowControl','none');
0053 obj.BytesAvailableFcnMode = 'byte';
0054 obj.BytesAvailableFcnCount = 30;                                    % Activite the
callback function when receive every 32 byte data
0055 obj.BytesAvailableFcn = @callback_serialReadQuat;                   % Define
callback function
0056 fopen(obj);
0057
0058 fwrite(obj, cmd_strings{:,16});                                     % Write the
'serialReadQuat' command into the IMU
0059 pause(5);
0060 fwrite(obj, cmd_strings{:,6});                                      % Stop command
0061 output = sensorData;
0062
0063 if saveData == true
0064     save('serialReadQuat.txt', 'sensorData', '-ascii');               % Save data to
.txt
0065 end
0066
0067 clear global sensorData;                                            % Clear global
variable sensorData
0068
0069 end
```

api_serialReadAng

PURPOSE

API: serial read Euler angle data

SYNOPSIS

function output = api_serialReadAng(obj, saveData)

DESCRIPTION

```
API: serial read Euler angle data

obj is the created serial port.

saveData is a logical flag to determine wheter to save the collected data
e.g. save == ture -> save | save == false -> don't save (default)

output is collected data with double format.
Data structure: [roll, pitch, yaw] [rad]
Data format: single precision floating point number

Example:

      output1 = api_serialReadAng(s1);
      output2 = api_serialReadAng(s1, true);
```

CROSS-REFERENCE INFORMATION

This function calls:

- api_loadCommandList API: load whole command list
- callback_serialReadAng callback: serial read Euler angle data

This function is called by:

- main 32A01T commmand test

SOURCE CODE

```
0001 function output = api_serialReadAng(obj, saveData)
0002 % API: serial read Euler angle data
0003 %
0004 % obj is the created serial port.
0005 %
0006 % saveData is a logical flag to determine wheter to save the collected data
0007 % e.g. save == ture -> save | save == false -> don't save (default)
0008 %
0009 % output is collected data with double format.
0010 % Data structure: [roll, pitch, yaw] [rad]
0011 % Data format: single precision floating point number
0012 %
0013 % Example:
0014 %
0015 %       output1 = api_serialReadAng(s1);
0016 %       output2 = api_serialReadAng(s1, true);
0017 %
0018
```

```
0019 %%
0020 % Validate input arguments
0021 switch (nargin)
0022     case 0
0023         error('Invalid input.');
0024     case 1
0025         if isvalid(obj)
0026             saveData = false;
0027         else
0028             error('Invalid input.');
0029         end
0030     case 2
0031         if isvalid(obj) && islogical(saveData)
0032         else
0033             error('Invalid input.');
0034         end
0035 end
0036
0037
0038 %%
0039 % Define global variable sensorData to store collected data
0040 global sensorData;
0041 sensorData = [];
0042
0043
0044 %%
0045 % Load command list
0046 [ ~, cmd_strings ] = api_loadCommandList;
0047
0048
0049 %%
0050 % Reconfigure the serial port and serial read sensor data
0051
0052 fclose(obj);
0053 set(obj,'BaudRate',115200,'DataBits',8,'StopBits',1,'Parity','none',...
0054       'FlowControl','none');
0055 obj.BytesAvailableFcnMode = 'byte';
0056 obj.BytesAvailableFcnCount = 26;                                    % Activite the
callback function when receive every 32 byte data
0057 obj.BytesAvailableFcn = @callback_serialReadAng;                    % Define
callback function
0058 fopen(obj);
0059
0060 fwrite(obj, cmd_strings{:,18});                                     % Write the
'serialReadAng' command into the IMU
0061 pause(5);
0062 fwrite(obj, cmd_strings{:,6});                                      % Stop command
0063 output = sensorData;
0064
0065 if saveData == true
0066     save('serialReadAng.txt', 'sensorData', '-ascii');              % save data to
.txt
0067 end
0068
0069 clear global sensorData;                                            % Clear global
variable sensorData
0070
0071 end
```

api_serialReadQuatMag

PURPOSE

API: serial read quaternion and magnetometer data

SYNOPSIS

function output = api_serialReadQuatMag(obj, saveData)

DESCRIPTION

```
API: serial read quaternion and magnetometer data

obj is the created serial port.

saveData is a logical flag to determine wheter to save the collected data
e.g. save == ture -> save | save == false -> don't save (default)

output is collected data with double format.
Data structure: [q0, q1, q2, q3, mag_x, mag_y, mag_z] [-, mG]
Data format: sinlge precision floating point number

Example:

      output1 = api_serialReadQuatMag(s1);
      output2 = api_serialReadQuatMag(s1, true);
```

CROSS-REFERENCE INFORMATION

This function calls:

- api_loadCommandList API: load whole command list
- callback_serialReadQuatMag callback: serial read quaternion and magnetometer data

This function is called by:

- main 32A01T commmand test

SOURCE CODE

```
0001 function output = api_serialReadQuatMag(obj, saveData)
0002 % API: serial read quaternion and magnetometer data
0003 %
0004 % obj is the created serial port.
0005 %
0006 % saveData is a logical flag to determine wheter to save the collected data
0007 % e.g. save == ture -> save | save == false -> don't save (default)
0008 %
0009 % output is collected data with double format.
0010 % Data structure: [q0, q1, q2, q3, mag_x, mag_y, mag_z] [-, mG]
0011 % Data format: sinlge precision floating point number
0012 %
0013 % Example:
0014 %
0015 %       output1 = api_serialReadQuatMag(s1);
0016 %       output2 = api_serialReadQuatMag(s1, true);
0017 %
0018
```

```
0019 %%
0020 % Validate input arguments
0021 switch (nargin)
0022     case 0
0023         error('Invalid input.');
0024     case 1
0025         if isvalid(obj)
0026             saveData = false;
0027         else
0028             error('Invalid input.');
0029         end
0030     case 2
0031         if isvalid(obj) && islogical(saveData)
0032         else
0033             error('Invalid input.');
0034         end
0035 end
0036
0037
0038 %%
0039 % Define global variable sensorData to store collected data
0040 global sensorData;
0041 sensorData = [];
0042
0043
0044 %%
0045 % Load command list
0046 [ ~, cmd_strings ] = api_loadCommandList;
0047
0048
0049 %%
0050 % Reconfigure the serial port and serial read data
0051 fclose(obj);
0052 set(obj,'BaudRate',115200,'DataBits',8,'StopBits',1,'Parity','none',...
0053       'FlowControl','none');
0054 obj.BytesAvailableFcnMode = 'byte';
0055 obj.BytesAvailableFcnCount = 42;                                    % Activite the
callback function when receive every 32 byte data
0056 obj.BytesAvailableFcn = @callback_serialReadQuatMag;                % Define
callback function
0057 fopen(obj);
0058
0059 fwrite(obj, cmd_strings{:,20});                                     % Write the
'serialReadQuatMag' command into the IMU
0060 pause(5);
0061 fwrite(obj, cmd_strings{:,6});                                      % Stop command
0062 output = sensorData;
0063
0064 if saveData == true
0065     save('serialReadQuatACC.txt', 'sensorData', '-ascii');          % Save data to
.txt
0066 end
0067
0068 clear global sensorData;                                            % Clear global
variable sensorData
0069
0070 end
```

api_serialReadQuatAcc

PURPOSE

API: serial read quaternion and accelerometer data

SYNOPSIS

function output = api_serialReadQuatAcc(obj, saveData)

DESCRIPTION

```
API: serial read quaternion and accelerometer data

obj is the created serial port.

saveData is a logical flag to determine wheter to save the collected data
e.g. save == ture -> save | save == false -> don't save (default)

output is collected data with double format.
Data structure: [q0, q1, q2, q3, acc_x, acc_y, acc_z] [-, g]
Data format: single precision floating point number

Example:

      output1 = api_serialReadQuatAcc(s1);
      output2 = api_serialReadQuatAcc(s1, true);
```

CROSS-REFERENCE INFORMATION

This function calls:

- api_loadCommandList API: load whole command list
- callback_serialReadQuatAcc callback: serial read quaternion and accelerometer data

This function is called by:

- main 32A01T commmand test

SOURCE CODE

```
0001 function output = api_serialReadQuatAcc(obj, saveData)
0002 % API: serial read quaternion and accelerometer data
0003 %
0004 % obj is the created serial port.
0005 %
0006 % saveData is a logical flag to determine wheter to save the collected data
0007 % e.g. save == ture -> save | save == false -> don't save (default)
0008 %
0009 % output is collected data with double format.
0010 % Data structure: [q0, q1, q2, q3, acc_x, acc_y, acc_z] [-, g]
0011 % Data format: single precision floating point number
0012 %
0013 % Example:
0014 %
0015 %       output1 = api_serialReadQuatAcc(s1);
0016 %       output2 = api_serialReadQuatAcc(s1, true);
0017 %
0018
```

```
0019 %%
0020 % Validate input arguments
0021 switch (nargin)
0022     case 0
0023         error('Invalid input.');
0024     case 1
0025         if isvalid(obj)
0026             saveData = false;
0027         else
0028             error('Invalid input.');
0029         end
0030     case 2
0031         if isvalid(obj) && islogical(saveData)
0032         else
0033             error('Invalid input.');
0034         end
0035 end
0036
0037
0038 %%
0039 % Define global variable sensorData to store collected data
0040 global sensorData;
0041 sensorData = [];
0042
0043
0044 %%
0045 % Load command list
0046 [ ~, cmd_strings ] = api_loadCommandList;
0047
0048
0049 %%
0050 % Reconfigure the serial port and serial read data
0051 fclose(obj);
0052 set(obj,'BaudRate',115200,'DataBits',8,'StopBits',1,'Parity','none',...
0053       'FlowControl','none');
0054 obj.BytesAvailableFcnMode = 'byte';
0055 obj.BytesAvailableFcnCount = 42;                                    % Activite the
callback function when receive every 32 byte data
0056 obj.BytesAvailableFcn = @callback_serialReadQuatAcc;                % Define
callback function
0057 fopen(obj);
0058
0059 fwrite(obj, cmd_strings{:,21});                                     % Write the
'serialReadQuatAcc' command into the IMU
0060 pause(5);
0061 fwrite(obj, cmd_strings{:,6});                                      % Stop command
0062 output = sensorData;
0063
0064 if saveData == true
0065     save('serialReadQuatACC.txt', 'sensorData', '-ascii');          % Save data to
.txt
0066 end
0067
0068 clear global sensorData;                                            % Clear global
variable sensorData
0069
0070 end
```

api_serialReadQuatAccMag

PURPOSE

API: serial read quaternion accelerometer and magnetometer data

SYNOPSIS

function output = api_serialReadQuatAccMag(obj, saveData)

DESCRIPTION

```
API: serial read quaternion accelerometer and magnetometer data

obj is the created serial port.

saveData is a logical flag to determine wheter to save the collected data
e.g. save == ture -> save | save == false -> don't save (default)

output is collected data with double format.
Data structure: [q0, q1, q2, q3, acc_x, acc_y, acc_z, ...
                 mag_x, mag_y, mag_z] [-, g, mG]
Data format: single precision floating point number

Example:

      output1 = api_serialReadQuatAccMag(s1);
      output2 = api_serialReadQuatAccMag(s1, true);
```

CROSS-REFERENCE INFORMATION

This function calls:

- api_loadCommandList API: load whole command list
- callback_serialReadQuatAccMag callback: serial read quaternion, accelerometer and magnetometer data

This function is called by:

- main 32A01T commmand test

SOURCE CODE

```
0001 function output = api_serialReadQuatAccMag(obj, saveData)
0002 % API: serial read quaternion accelerometer and magnetometer data
0003 %
0004 % obj is the created serial port.
0005 %
0006 % saveData is a logical flag to determine wheter to save the collected data
0007 % e.g. save == ture -> save | save == false -> don't save (default)
0008 %
0009 % output is collected data with double format.
0010 % Data structure: [q0, q1, q2, q3, acc_x, acc_y, acc_z, ...
0011 %                  mag_x, mag_y, mag_z] [-, g, mG]
0012 % Data format: single precision floating point number
0013 %
0014 % Example:
0015 %
```

```
0016 %       output1 = api_serialReadQuatAccMag(s1);
0017 %       output2 = api_serialReadQuatAccMag(s1, true);
0018 %
0019
0020 %%
0021 % Validate input arguments
0022 switch (nargin)
0023     case 0
0024         error('Invalid input.');
0025     case 1
0026         if isvalid(obj)
0027             saveData = false;
0028         else
0029             error('Invalid input.');
0030         end
0031     case 2
0032         if isvalid(obj) && islogical(saveData)
0033         else
0034             error('Invalid input.');
0035         end
0036 end
0037
0038
0039 %%
0040 % Define global variable sensorData to store collected data
0041 global sensorData;
0042 sensorData = [];
0043
0044
0045 %%
0046 % Load command list
0047 [ ~, cmd_strings ] = api_loadCommandList;
0048
0049
0050 %%
0051 % Reconfigure the serial port and serial read sensor data
0052 fclose(obj);
0053 set(obj,'BaudRate',115200,'DataBits',8,'StopBits',1,'Parity','none',...
0054       'FlowControl','none');
0055 obj.BytesAvailableFcnMode = 'byte';
0056 obj.BytesAvailableFcnCount = 54;                                    % Activite the
callback function when receive every 32 byte data
0057 obj.BytesAvailableFcn = @callback_serialReadQuatAccMag;             % Define
callback function
0058 fopen(obj);
0059
0060 fwrite(obj, cmd_strings{:,22});                                     % Write the
'serialReadQuatAcc' command into the IMU
0061 pause(5);
0062 fwrite(obj, cmd_strings{:,6});                                      % Stop command
0063 output = sensorData;
0064
0065 if saveData == true
0066     save('serialReadQuatAccMag.txt', 'sensorData', '-ascii');       % Save data to
.txt
0067 end
0068
0069 clear global sensorData;                                            % Clear global
variable sensorData
0070
0071 end
```

api_serialReadDCM

PURPOSE

API: serial read direct cosine matrix data

SYNOPSIS

function output = api_serialReadDCM(obj, saveData)

DESCRIPTION

```
API: serial read direct cosine matrix data

obj is the created serial port.

saveData is a logical flag to determine wheter to save the collected data
e.g. save == ture -> save | save == false -> don't save (default)

output is collected data with double format.
Data structure: [ data0, data1, data2;
                  data3, data4, data5;
                  data6, data7, data8;]
Data format: single precision floating point number

Example:

      output1 = api_serialReadDCM(s1);
      output2 = api_serialReadDCM(s1, true);
```

CROSS-REFERENCE INFORMATION

This function calls:

- api_loadCommandList API: load whole command list
- callback_serialReadDCM callback: serial read direct cosine matrix

This function is called by:

- main 32A01T commmand test

SOURCE CODE

```
0001 function output = api_serialReadDCM(obj, saveData)
0002 % API: serial read direct cosine matrix data
0003 %
0004 % obj is the created serial port.
0005 %
0006 % saveData is a logical flag to determine wheter to save the collected data
0007 % e.g. save == ture -> save | save == false -> don't save (default)
0008 %
0009 % output is collected data with double format.
0010 % Data structure: [ data0, data1, data2;
0011 %                   data3, data4, data5;
0012 %                   data6, data7, data8;]
0013 % Data format: single precision floating point number
0014 %
0015 % Example:
0016 %
```

```
0017 %       output1 = api_serialReadDCM(s1);
0018 %       output2 = api_serialReadDCM(s1, true);
0019 %
0020
0021 %%
0022 % Validate input arguments
0023 switch (nargin)
0024     case 0
0025         error('Invalid input.');
0026     case 1
0027         if isvalid(obj)
0028             saveData = false;
0029         else
0030             error('Invalid input.');
0031         end
0032     case 2
0033         if isvalid(obj) && islogical(saveData)
0034         else
0035             error('Invalid input.');
0036         end
0037 end
0038
0039
0040 %%
0041 % Define global variable sensorData to store collected data
0042 global sensorData;
0043 sensorData = [];
0044
0045
0046 %%
0047 % Load command list
0048 [ ~, cmd_strings ] = api_loadCommandList;
0049
0050
0051 %%
0052 % Reconfigure the serial port and serial read sensor data
0053 fclose(obj);
0054 set(obj,'BaudRate',115200,'DataBits',8,'StopBits',1,'Parity','none',...
0055       'FlowControl','none');
0056 obj.BytesAvailableFcnMode = 'byte';
0057 obj.BytesAvailableFcnCount = 50;                                    % Activite the
callback function when receive every 32 byte data
0058 obj.BytesAvailableFcn = @callback_serialReadDCM;                    % Define
callback function
0059 fopen(obj);
0060
0061 fwrite(obj, cmd_strings{:,23});                                     % Write the
'serialReadDCM' command into the IMU
0062 pause(5);
0063 fwrite(obj, cmd_strings{:,6});                                      % Stop command
0064 output = sensorData;
0065
0066 if saveData == true
0067     save('serialReadDCM.txt', 'sensorData', '-ascii');              % Save data to
.txt
0068 end
0069
0070 clear global sensorData;                                            % Clear global
variable sensorData
0071
0072 end
```

api_serialReadAngGyroAccMag

PURPOSE

API: serial read Euler angle, gyroscope, accelerometer, and magnetometer

SYNOPSIS

function output = api_serialReadAngGyroAccMag(obj, saveData)

DESCRIPTION

```
API: serial read Euler angle, gyroscope, accelerometer, and magnetometer

obj is the created serial port.

saveData is a logical flag to determine wheter to save the collected data
e.g. save == ture -> save | save == false -> don't save (default)

output is collected data with double format.
Data structure: [roll, pitch, yaw, gyro_x, gyro_y, gyro_z, acc_x, ...
                 acc_y, acc_z, mag_x, mag_y, mag_z] [rad, rad/s, g, mG]
Data format: single precision floating point number

Example:

      output1 = api_serialReadAngGyroAccMag(s1)
```

CROSS-REFERENCE INFORMATION

This function calls:

- api_loadCommandList API: load whole command list
- callback_serialReadAngGyroAccMag callback: serial read angle, gyroscope, accelerometer and magnetometer

This function is called by:

- main 32A01T commmand test

SOURCE CODE

```
0001 function output = api_serialReadAngGyroAccMag(obj, saveData)
0002 % API: serial read Euler angle, gyroscope, accelerometer, and magnetometer
0003 %
0004 % obj is the created serial port.
0005 %
0006 % saveData is a logical flag to determine wheter to save the collected data
0007 % e.g. save == ture -> save | save == false -> don't save (default)
0008 %
0009 % output is collected data with double format.
0010 % Data structure: [roll, pitch, yaw, gyro_x, gyro_y, gyro_z, acc_x, ...
0011 %                  acc_y, acc_z, mag_x, mag_y, mag_z] [rad, rad/s, g, mG]
0012 % Data format: single precision floating point number
0013 %
0014 % Example:
0015 %
0016 %       output1 = api_serialReadAngGyroAccMag(s1)
```

```
0017 %
0018
0019 %%
0020 % Validate input arguments
0021 switch (nargin)
0022     case 0
0023         error('Invalid input.');
0024     case 1
0025         if isvalid(obj)
0026             saveData = false;
0027         else
0028             error('Invalid input.');
0029         end
0030     case 2
0031         if isvalid(obj) && islogical(saveData)
0032         else
0033             error('Invalid input.');
0034         end
0035 end
0036
0037
0038 %%
0039 % Define global variable sensorData to store collected data
0040 global sensorData;
0041 sensorData = [];
0042
0043
0044 %%
0045 % Load command list
0046 [ ~, cmd_strings ] = api_loadCommandList;
0047
0048
0049 %%
0050 % Reconfigure the serial port and serial read sensor data
0051
0052 fclose(obj);
0053 set(obj,'BaudRate',115200,'DataBits',8,'StopBits',1,'Parity','none',...
0054       'FlowControl','none');
0055 obj.BytesAvailableFcnMode = 'byte';
0056 obj.BytesAvailableFcnCount = 62;                                        % Activite the
callback function when receive every 32 byte data
0057 obj.BytesAvailableFcn = @callback_serialReadAngGyroAccMag;                      % Define
callback function
0058 fopen(obj);
0059
0060 fwrite(obj, cmd_strings{:,24});                                         % Write the
'countingResdSensor' command into the IMU
0061 pause(5);
0062 fwrite(obj, cmd_strings{:,6});                                          % Stop command
0063 output = sensorData;
0064
0065 if saveData == true
0066     save('serialReadAngGyroAccMag.txt', 'sensorData', '-ascii');        % save data to
.txt
0067 end
0068
0069 clear global sensorData;                                                % Clear global
variable sensorData
0070
0071 end
```

callback_countingReadSensor

PURPOSE

callback: counting read sensor data

SYNOPSIS

function callback_countingReadSensor(obj, ~)

DESCRIPTION

```
callback: counting read sensor data
automatically called after opening the serial port

obj is the created serial port.
```

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

- api_countingReadSensor API: counting read sensor data

SOURCE CODE

```
0001 function callback_countingReadSensor(obj, ~)
0002 % callback: counting read sensor data
0003 % automatically called after opening the serial port
0004 %
0005 % obj is the created serial port.
0006 %
0007 global sensorData;
0008
0009 received = fscanf(obj);                                          % read 32 bytes
each time
0010
0011 if length(received) == 32
0012     gyro_x = dec2bin(uint8(received(11:12)),8);
0013     gyro_x = reshape(gyro_x',1,16);
0014     gyro_x = double(typecast(uint16(bin2dec(gyro_x)),'int16'))*0.0125;
0015
0016     gyro_y = dec2bin(uint8(received(13:14)),8);
0017     gyro_y = reshape(gyro_y',1,16);
0018     gyro_y = double(typecast(uint16(bin2dec(gyro_y)),'int16'))*0.0125;
0019
0020     gyro_z = dec2bin(uint8(received(15:16)),8);
0021     gyro_z = reshape(gyro_z',1,16);
0022     gyro_z = double(typecast(uint16(bin2dec(gyro_z)),'int16'))*0.0125;
0023
0024     acc_x = dec2bin(uint8(received(17:18)),8);
0025     acc_x = reshape(acc_x',1,16);
0026     acc_x = double(typecast(uint16(bin2dec(acc_x)),'int16'))/8192;
0027
0028     acc_y = dec2bin(uint8(received(19:20)),8);
0029     acc_y = reshape(acc_y',1,16);
0030     acc_y = double(typecast(uint16(bin2dec(acc_y)),'int16'))/8192;
0031
0032     acc_z = dec2bin(uint8(received(21:22)),8);
```

```
0033     acc_z = reshape(acc_z',1,16);
0034     acc_z = double(typecast(uint16(bin2dec(acc_z)),'int16'))/8192;
0035
0036     mag_x = dec2bin(uint8(received(23:24)),8);
0037     mag_x = reshape(mag_x',1,16);
0038     mag_x = double(typecast(uint16(bin2dec(mag_x)),'int16'))/75;
0039
0040     mag_y = dec2bin(uint8(received(25:26)),8);
0041     mag_y = reshape(mag_y',1,16);
0042     mag_y = double(typecast(uint16(bin2dec(mag_y)),'int16'))/75;
0043
0044     mag_z = dec2bin(uint8(received(27:28)),8);
0045     mag_z = reshape(mag_z',1,16);
0046     mag_z = double(typecast(uint16(bin2dec(mag_z)),'int16'))/75;
0047
0048     sensorData = [sensorData; gyro_x, gyro_y, gyro_z, acc_x, acc_y, ...     % Record and
save data
0049         acc_z, mag_x, mag_y, mag_z];
0050
0051 end
0052
0053 end
```

callback_countingReadQuat

PURPOSE

callback: counting read quaternion

SYNOPSIS

function callback_countingReadQuat(obj, ~)

DESCRIPTION

```
callback: counting read quaternion
automatically called after opening the serial port

obj is the created serial port.
```

CROSS-REFERENCE INFORMATION

This function calls:

- hexsingle2num Convert single precision IEEE hexadecimal string to number.

This function is called by:

- api_countingReadQuat API: counting read quaternion data

SOURCE CODE

```
0001 function callback_countingReadQuat(obj, ~)
0002 % callback: counting read quaternion
0003 % automatically called after opening the serial port
0004 %
0005 % obj is the created serial port.
0006 %
0007 global sensorData;
0008
0009 received = fscanf(obj);                                          % Read 30 bytes
each time
0010
0011 if length(received) == 30
0012     q0 = flip(dec2hex(uint8(received(11:14))));
0013     q0 = hexsingle2num(reshape(q0',1,8));
0014
0015     q1 = flip(dec2hex(uint8(received(15:18))));
0016     q1 = hexsingle2num(reshape(q1',1,8));
0017
0018     q2 = flip(dec2hex(uint8(received(19:22))));
0019     q2 = hexsingle2num(reshape(q2',1,8));
0020
0021     q3 = flip(dec2hex(uint8(received(23:26))));
0022     q3 = hexsingle2num(reshape(q3',1,8));
0023
0024     sensorData = [sensorData; q0, q1, q2, q3 ];                    % Record and
save data
0025
0026 end
0027
0028 end
```

callback_countingReadAng

PURPOSE

callback: counting read Euler angle data

SYNOPSIS

function callback_countingReadAng(obj, ~)

DESCRIPTION

```
callback: counting read Euler angle data
automatically called after opening the serial port

obj is the created serial port.
```

CROSS-REFERENCE INFORMATION

This function calls:

- hexsingle2num Convert single precision IEEE hexadecimal string to number.

This function is called by:

- api_countingReadAng API: counting read Euler angle data

SOURCE CODE

```
0001 function callback_countingReadAng(obj, ~)
0002 % callback: counting read Euler angle data
0003 % automatically called after opening the serial port
0004 %
0005 % obj is the created serial port.
0006 %
0007 global sensorData;
0008
0009 received = fscanf(obj);                                          % Read 30 bytes
each time
0010
0011 if length(received) == 26
0012     roll = flip(dec2hex(uint8(received(11:14))));
0013     roll = hexsingle2num(reshape(roll',1,8));
0014
0015     pitch = flip(dec2hex(uint8(received(15:18))));
0016     pitch = hexsingle2num(reshape(pitch',1,8));
0017
0018     yaw = flip(dec2hex(uint8(received(19:22))));
0019     yaw = hexsingle2num(reshape(yaw',1,8));
0020
0021     sensorData = [sensorData; roll, pitch, yaw];                   % Record and
save data
0022
0023 end
0024
0025 end
```

callback_serialReadSensor

PURPOSE

callback: serial read sensor data

SYNOPSIS

function callback_serialReadSensor(obj, ~)

DESCRIPTION

```
callback: serial read sensor data
automatically called after opening the serial port

obj is the created serial port.
```

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

- api_serialReadSensor API: serial read sensor data

SOURCE CODE

```
0001 function callback_serialReadSensor(obj, ~)
0002 % callback: serial read sensor data
0003 % automatically called after opening the serial port
0004 %
0005 % obj is the created serial port.
0006 %
0007 global sensorData;
0008
0009 received = fscanf(obj);                                          % Read 32 bytes
each time
0010
0011 if length(received) == 32
0012     gyro_x = dec2bin(uint8(received(11:12)),8);
0013     gyro_x = reshape(gyro_x',1,16);
0014     gyro_x = double(typecast(uint16(bin2dec(gyro_x)),'int16'))*0.0125;
0015
0016     gyro_y = dec2bin(uint8(received(13:14)),8);
0017     gyro_y = reshape(gyro_y',1,16);
0018     gyro_y = double(typecast(uint16(bin2dec(gyro_y)),'int16'))*0.0125;
0019
0020     gyro_z = dec2bin(uint8(received(15:16)),8);
0021     gyro_z = reshape(gyro_z',1,16);
0022     gyro_z = double(typecast(uint16(bin2dec(gyro_z)),'int16'))*0.0125;
0023
0024     acc_x = dec2bin(uint8(received(17:18)),8);
0025     acc_x = reshape(acc_x',1,16);
0026     acc_x = double(typecast(uint16(bin2dec(acc_x)),'int16'))/8192;
0027
0028     acc_y = dec2bin(uint8(received(19:20)),8);
0029     acc_y = reshape(acc_y',1,16);
0030     acc_y = double(typecast(uint16(bin2dec(acc_y)),'int16'))/8192;
0031
0032     acc_z = dec2bin(uint8(received(21:22)),8);
```

```
0033      acc_z = reshape(acc_z',1,16);
0034      acc_z = double(typecast(uint16(bin2dec(acc_z)),'int16'))/8192;
0035
0036      mag_x = dec2bin(uint8(received(23:24)),8);
0037      mag_x = reshape(mag_x',1,16);
0038      mag_x = double(typecast(uint16(bin2dec(mag_x)),'int16'))/75;
0039
0040      mag_y = dec2bin(uint8(received(25:26)),8);
0041      mag_y = reshape(mag_y',1,16);
0042      mag_y = double(typecast(uint16(bin2dec(mag_y)),'int16'))/75;
0043
0044      mag_z = dec2bin(uint8(received(27:28)),8);
0045      mag_z = reshape(mag_z',1,16);
0046      mag_z = double(typecast(uint16(bin2dec(mag_z)),'int16'))/75;
0047
0048      sensorData = [sensorData; gyro_x, gyro_y, gyro_z, acc_x, acc_y, ...     % Record and
save data
0049          acc_z, mag_x, mag_y, mag_z];
0050  end
0051
0052  end
```

callback_serialReadQuat

PURPOSE

callback: serial read quaternion data

SYNOPSIS

function callback_serialReadQuat(obj, ~)

DESCRIPTION

```
callback: serial read quaternion data
automatically called after opening the serial port

obj is the created serial port.
```

CROSS-REFERENCE INFORMATION

This function calls:

- hexsingle2num Convert single precision IEEE hexadecimal string to number.

This function is called by:

- api_serialReadQuat API: serial read quaternion

SOURCE CODE

```
0001 function callback_serialReadQuat(obj, ~)
0002 % callback: serial read quaternion data
0003 % automatically called after opening the serial port
0004 %
0005 % obj is the created serial port.
0006 %
0007 global sensorData;
0008
0009 received = fscanf(obj);                                          % Read 30 bytes
each time
0010
0011 if length(received) == 30
0012     q0 = flip(dec2hex(uint8(received(11:14))));
0013     q0 = hexsingle2num(reshape(q0',1,8));
0014
0015     q1 = flip(dec2hex(uint8(received(15:18))));
0016     q1 = hexsingle2num(reshape(q1',1,8));
0017
0018     q2 = flip(dec2hex(uint8(received(19:22))));
0019     q2 = hexsingle2num(reshape(q2',1,8));
0020
0021     q3 = flip(dec2hex(uint8(received(23:26))));
0022     q3 = hexsingle2num(reshape(q3',1,8));
0023
0024     sensorData = [sensorData; q0, q1, q2, q3];
0025 end
0026 end
```

callback_serialReadAng

PURPOSE

callback: serial read Euler angle data

SYNOPSIS

function callback_serialReadAng(obj, ~)

DESCRIPTION

```
callback: serial read Euler angle data
automatically called after opening the serial port

obj is the created serial port.
```

CROSS-REFERENCE INFORMATION

This function calls:

- hexsingle2num Convert single precision IEEE hexadecimal string to number.

This function is called by:

- api_serialReadAng API: serial read Euler angle data

SOURCE CODE

```
0001 function callback_serialReadAng(obj, ~)
0002 % callback: serial read Euler angle data
0003 % automatically called after opening the serial port
0004 %
0005 % obj is the created serial port.
0006 %
0007 global sensorData;
0008
0009 received = fscanf(obj);                                          % Read 26 bytes
each time
0010
0011 if length(received) == 26
0012     roll = flip(dec2hex(uint8(received(11:14))));
0013     roll = hexsingle2num(reshape(roll',1,8));
0014
0015     pitch = flip(dec2hex(uint8(received(15:18))));
0016     pitch = hexsingle2num(reshape(pitch',1,8));
0017
0018     yaw = flip(dec2hex(uint8(received(19:22))));
0019     yaw = hexsingle2num(reshape(yaw',1,8));
0020
0021     sensorData = [sensorData; roll, pitch, yaw];                   % Record and
save data
0022 end
0023 end
```

callback_serialReadQuatMag

PURPOSE

callback: serial read quaternion and magnetometer data

SYNOPSIS

function callback_serialReadQuatMag(obj, ~)

DESCRIPTION

```
callback: serial read quaternion and magnetometer data
automatically called after opening the serial port

obj is the created serial port.
```

CROSS-REFERENCE INFORMATION

This function calls:

- hexsingle2num Convert single precision IEEE hexadecimal string to number.

This function is called by:

- api_serialReadQuatMag API: serial read quaternion and magnetometer data

SOURCE CODE

```
0001 function callback_serialReadQuatMag(obj, ~)
0002 % callback: serial read quaternion and magnetometer data
0003 % automatically called after opening the serial port
0004 %
0005 % obj is the created serial port.
0006 %
0007 global sensorData;
0008
0009 received = fscanf(obj);                                          % Read 42 bytes
each time
0010 if length(received) == 42
0011     q0 = flip(dec2hex(uint8(received(11:14))));
0012     q0 = hexsingle2num(reshape(q0',1,8));
0013
0014     q1 = flip(dec2hex(uint8(received(15:18))));
0015     q1 = hexsingle2num(reshape(q1',1,8));
0016
0017     q2 = flip(dec2hex(uint8(received(19:22))));
0018     q2 = hexsingle2num(reshape(q2',1,8));
0019
0020     q3 = flip(dec2hex(uint8(received(23:26))));
0021     q3 = hexsingle2num(reshape(q3',1,8));
0022
0023     mag_x = flip(dec2hex(uint8(received(27:30))));
0024     mag_x = hexsingle2num(reshape(mag_x',1,8));
0025
0026     mag_y = flip(dec2hex(uint8(received(31:34))));
0027     mag_y = hexsingle2num(reshape(mag_y',1,8));
0028
0029     mag_z = flip(dec2hex(uint8(received(35:38))));
```

```
0030     mag_z = hexsingle2num(reshape(mag_z',1,8));
0031
0032     sensorData = [sensorData; q0, q1, q2, q3, mag_x, mag_y, mag_z];          % Record and
save data
0033
0034 end
0035 end
```

---

callback_serialReadQuatAcc

PURPOSE

callback: serial read quaternion and accelerometer data

SYNOPSIS

function callback_serialReadQuatAcc(obj, ~)

DESCRIPTION

```
callback: serial read quaternion and accelerometer data
automatically called after opening the serial port

obj is the created serial port.
```

CROSS-REFERENCE INFORMATION

This function calls:

- hexsingle2num Convert single precision IEEE hexadecimal string to number.

This function is called by:

- api_serialReadQuatAcc API: serial read quaternion and accelerometer data

SOURCE CODE

```
0001 function callback_serialReadQuatAcc(obj, ~)
0002 % callback: serial read quaternion and accelerometer data
0003 % automatically called after opening the serial port
0004 %
0005 % obj is the created serial port.
0006 %
0007 global sensorData;
0008
0009 received = fscanf(obj);                                          % Read 42 bytes
each time
0010 if length(received) == 42
0011     q0 = flip(dec2hex(uint8(received(11:14))));
0012     q0 = hexsingle2num(reshape(q0',1,8));
0013
0014     q1 = flip(dec2hex(uint8(received(15:18))));
0015     q1 = hexsingle2num(reshape(q1',1,8));
0016
0017     q2 = flip(dec2hex(uint8(received(19:22))));
0018     q2 = hexsingle2num(reshape(q2',1,8));
0019
0020     q3 = flip(dec2hex(uint8(received(23:26))));
0021     q3 = hexsingle2num(reshape(q3',1,8));
0022
0023     acc_x = flip(dec2hex(uint8(received(27:30))));
0024     acc_x = hexsingle2num(reshape(acc_x',1,8));
0025
0026     acc_y = flip(dec2hex(uint8(received(31:34))));
0027     acc_y = hexsingle2num(reshape(acc_y',1,8));
0028
0029     acc_z = flip(dec2hex(uint8(received(35:38))));
```

```
0030      acc_z = hexsingle2num(reshape(acc_z',1,8));
0031
0032      sensorData = [sensorData; q0, q1, q2, q3, acc_x, acc_y, acc_z];          % Record and
save data
0033
0034  end
0035  end
```

---

callback_serialReadQuatAccMag

PURPOSE

callback: serial read quaternion, accelerometer and magnetometer data

SYNOPSIS

function callback_serialReadQuatAccMag(obj, ~)

DESCRIPTION

```
callback: serial read quaternion, accelerometer and magnetometer data
automatically called after opening the serial port

obj is the created serial port.
```

CROSS-REFERENCE INFORMATION

This function calls:

- hexsingle2num Convert single precision IEEE hexadecimal string to number.

This function is called by:

- api_serialReadQuatAccMag API: serial read quaternion accelerometer and magnetometer data

SOURCE CODE

```
0001 function callback_serialReadQuatAccMag(obj, ~)
0002 % callback: serial read quaternion, accelerometer and magnetometer data
0003 % automatically called after opening the serial port
0004 %
0005 % obj is the created serial port.
0006 %
0007 global sensorData;
0008
0009 received = fscanf(obj);                                          % Read 54 bytes
each time
0010 if length(received) == 54
0011     q0 = flip(dec2hex(uint8(received(11:14))));
0012     q0 = hexsingle2num(reshape(q0',1,8));
0013
0014     q1 = flip(dec2hex(uint8(received(15:18))));
0015     q1 = hexsingle2num(reshape(q1',1,8));
0016
0017     q2 = flip(dec2hex(uint8(received(19:22))));
0018     q2 = hexsingle2num(reshape(q2',1,8));
0019
0020     q3 = flip(dec2hex(uint8(received(23:26))));
0021     q3 = hexsingle2num(reshape(q3',1,8));
0022
0023     acc_x = flip(dec2hex(uint8(received(27:30))));
0024     acc_x = hexsingle2num(reshape(acc_x',1,8));
0025
0026     acc_y = flip(dec2hex(uint8(received(31:34))));
0027     acc_y = hexsingle2num(reshape(acc_y',1,8));
0028
0029     acc_z = flip(dec2hex(uint8(received(35:38))));
```

```
0030      acc_z = hexsingle2num(reshape(acc_z',1,8));
0031
0032      mag_x = flip(dec2hex(uint8(received(39:42))));
0033      mag_x = hexsingle2num(reshape(mag_x',1,8));
0034
0035      mag_y = flip(dec2hex(uint8(received(43:46))));
0036      mag_y = hexsingle2num(reshape(mag_y',1,8));
0037
0038      mag_z = flip(dec2hex(uint8(received(47:50))));
0039      mag_z = hexsingle2num(reshape(mag_z',1,8));
0040
0041      sensorData = [sensorData; q0, q1, q2, q3, acc_x, acc_y, acc_z, ...
0042                                                mag_x, mag_y, mag_z];         % Record and
save data
0043 end
0044 end
```

callback_serialReadDCM

PURPOSE

callback: serial read direct cosine matrix

SYNOPSIS

function callback_serialReadDCM(obj, ~)

DESCRIPTION

```
callback: serial read direct cosine matrix
automatically called after opening the serial port

obj is the created serial port.
```

CROSS-REFERENCE INFORMATION

This function calls:

- hexsingle2num Convert single precision IEEE hexadecimal string to number.

This function is called by:

- api_serialReadDCM API: serial read direct cosine matrix data

SOURCE CODE

```
0001 function callback_serialReadDCM(obj, ~)
0002 % callback: serial read direct cosine matrix
0003 % automatically called after opening the serial port
0004 %
0005 % obj is the created serial port.
0006 %
0007 global sensorData;
0008
0009 received = fscanf(obj);                                          % Read 50 bytes
each time
0010 if length(received) == 50
0011     e00 = flip(dec2hex(uint8(received(11:14))));
0012     e00 = hexsingle2num(reshape(e00',1,8));
0013
0014     e01 = flip(dec2hex(uint8(received(15:18))));
0015     e01 = hexsingle2num(reshape(e01',1,8));
0016
0017     e02 = flip(dec2hex(uint8(received(19:22))));
0018     e02 = hexsingle2num(reshape(e02',1,8));
0019
0020     e10 = flip(dec2hex(uint8(received(23:26))));
0021     e10 = hexsingle2num(reshape(e10',1,8));
0022
0023     e11 = flip(dec2hex(uint8(received(27:30))));
0024     e11 = hexsingle2num(reshape(e11',1,8));
0025
0026     e12 = flip(dec2hex(uint8(received(31:34))));
0027     e12 = hexsingle2num(reshape(e12',1,8));
0028
0029     e20 = flip(dec2hex(uint8(received(35:38))));
```

```
0030     e20 = hexsingle2num(reshape(e20',1,8));
0031
0032     e21 = flip(dec2hex(uint8(received(39:42))));
0033     e21 = hexsingle2num(reshape(e21',1,8));
0034
0035     e22 = flip(dec2hex(uint8(received(43:46))));
0036     e22 = hexsingle2num(reshape(e22',1,8));
0037
0038
0039     DCM = [e00, e01, e02;
0040            e10, e11, e12;
0041            e20, e21, e22;];
0042
0043     sensorData = [sensorData; DCM];                                    % Record and
save data
0044
0045 end
0046 end
```

---

callback_serialReadAngGyroAccMag

PURPOSE

callback: serial read angle, gyroscope, accelerometer and magnetometer

SYNOPSIS

function callback_serialReadAngGyroAccMag(obj, ~)

DESCRIPTION

```
callback: serial read angle, gyroscope, accelerometer and magnetometer
automatically called after opening the serial port

obj is the created serial port.
```

CROSS-REFERENCE INFORMATION

This function calls:

- hexsingle2num Convert single precision IEEE hexadecimal string to number.

This function is called by:

- api_serialReadAngGyroAccMag API: serial read Euler angle, gyroscope, accelerometer, and magnetometer

SOURCE CODE

```
0001 function callback_serialReadAngGyroAccMag(obj, ~)
0002 % callback: serial read angle, gyroscope, accelerometer and magnetometer
0003 % automatically called after opening the serial port
0004 %
0005 % obj is the created serial port.
0006 %
0007 global sensorData;
0008
0009 received = fscanf(obj);                                                  % Read 62 bytes
each time
0010 if length(received) == 62
0011     roll = flip(dec2hex(uint8(received(11:14))));
0012     roll = hexsingle2num(reshape(roll',1,8));
0013
0014     pitch = flip(dec2hex(uint8(received(15:18))));
0015     pitch = hexsingle2num(reshape(pitch',1,8));
0016
0017     yaw = flip(dec2hex(uint8(received(19:22))));
0018     yaw = hexsingle2num(reshape(yaw',1,8));
0019
0020     gyro_x = flip(dec2hex(uint8(received(23:26))));
0021     gyro_x = hexsingle2num(reshape(gyro_x',1,8));
0022
0023     gyro_y = flip(dec2hex(uint8(received(27:30))));
0024     gyro_y = hexsingle2num(reshape(gyro_y',1,8));
0025
0026     gyro_z = flip(dec2hex(uint8(received(31:34))));
0027     gyro_z = hexsingle2num(reshape(gyro_z',1,8));
0028
```

```
0029      acc_x = flip(dec2hex(uint8(received(35:38))));
0030      acc_x = hexsingle2num(reshape(acc_x',1,8));
0031
0032      acc_y = flip(dec2hex(uint8(received(39:42))));
0033      acc_y = hexsingle2num(reshape(acc_y',1,8));
0034
0035      acc_z = flip(dec2hex(uint8(received(43:46))));
0036      acc_z = hexsingle2num(reshape(acc_z',1,8));
0037
0038      mag_x = flip(dec2hex(uint8(received(47:50))));
0039      mag_x = hexsingle2num(reshape(mag_x',1,8));
0040
0041      mag_y = flip(dec2hex(uint8(received(51:54))));
0042      mag_y = hexsingle2num(reshape(mag_y',1,8));
0043
0044      mag_z = flip(dec2hex(uint8(received(55:58))));
0045      mag_z = hexsingle2num(reshape(mag_z',1,8));
0046
0047      sensorData = [sensorData; roll, pitch, yaw, gyro_x, gyro_y, gyro_z,...
0048          acc_x, acc_y, acc_z, mag_x, mag_y, mag_z];
0049  end
0050  end
```

CRC

PURPOSE

Accurate implementations of the 16-bit CRC-CCITT, used with a look up table.

SYNOPSIS

function crc = CRC(data)

DESCRIPTION

```
Accurate implementations of the 16-bit CRC-CCITT, used with a look up table.

Ref: https://www.mathworks.com/matlabcentral/fileexchange/47682-crc-16-ccitt-m

The CRC calculation is based on following generator polynomial:
G(x) = x16 + x12 + x5 + 1

The register initial value of the implementation is: 0xFFFF

used data = string -> 1 2 3 4 5 6 7 8 9

Online calculator to check the script:
http://www.lammertbies.nl/comm/info/crc-calculation.html
```

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

- api_countingReadAng API: counting read Euler angle data
- api_countingReadQuat API: counting read quaternion data
- api_countingReadSensor API: counting read sensor data
- api_setEquipID API: set equipment ID (1-80)(default = 1)
- api_setOutputRate API: set IMU output frequecy division (1-255)(default = 10)
- api_setSamplingRate API: set sampling rate (1-500) [Hz](default = 500)
- api_setSensitivity API: set IMU sensitivity (0.04 - 0.4)(default = 0.12)

SOURCE CODE

```
0001 function crc = CRC(data)
0002 % Accurate implementations of the 16-bit CRC-CCITT, used with a look up table.
0003 %
0004 % Ref: https://www.mathworks.com/matlabcentral/fileexchange/47682-crc-16-ccitt-m
0005 %
0006 % The CRC calculation is based on following generator polynomial:
0007 % G(x) = x16 + x12 + x5 + 1
0008 %
0009 % The register initial value of the implementation is: 0xFFFF
0010 %
0011 % used data = string -> 1 2 3 4 5 6 7 8 9
0012 %
0013 % Online calculator to check the script:
0014 % http://www.lammertbies.nl/comm/info/crc-calculation.html
0015 %
0016 %
0017
0018 % crc look up table
0019 Crc_ui16LookupTable=[0,4129,8258,12387,16516,20645,24774,28903,33032,37161,41290,45419,49548,...
```

```
0020
53677,57806,61935,4657,528,12915,8786,21173,17044,29431,25302,37689,33560,45947,41818,54205,...
0021
50076,62463,58334,9314,13379,1056,5121,25830,29895,17572,21637,42346,46411,34088,38153,58862,...
0022
62927,50604,54669,13907,9842,5649,1584,30423,26358,22165,18100,46939,42874,38681,34616,63455,...
0023
59390,55197,51132,18628,22757,26758,30887,2112,6241,10242,14371,51660,55789,59790,63919,35144,...
0024
39273,43274,47403,23285,19156,31415,27286,6769,2640,14899,10770,56317,52188,64447,60318,39801,...
0025
35672,47931,43802,27814,31879,19684,23749,11298,15363,3168,7233,60846,64911,52716,56781,44330,...
0026
48395,36200,40265,32407,28342,24277,20212,15891,11826,7761,3696,65439,61374,57309,53244,48923,...
0027
44858,40793,36728,37256,33193,45514,41451,53516,49453,61774,57711,4224,161,12482,8419,20484,...
0028
16421,28742,24679,33721,37784,41979,46042,49981,54044,58239,62302,689,4752,8947,13010,16949,...
0029
21012,25207,29270,46570,42443,38312,34185,62830,58703,54572,50445,13538,9411,5280,1153,29798,...
0030
25671,21540,17413,42971,47098,34713,38840,59231,63358,50973,55100,9939,14066,1681,5808,26199,...
0031
30326,17941,22068,55628,51565,63758,59695,39368,35305,47498,43435,22596,18533,30726,26663,6336,...
0032
2273,14466,10403,52093,56156,60223,64286,35833,39896,43963,48026,19061,23124,27191,31254,2801,6864,...
0033
10931,14994,64814,60687,56684,52557,48554,44427,40424,36297,31782,27655,23652,19525,15522,11395,...
0034
7392,3265,61215,65342,53085,57212,44955,49082,36825,40952,28183,32310,20053,24180,11923,16050,3793,7920];

0035
0036 % data=[1 2 4 1]; % ~ string '1 2 3 4 5 6 7 8 9'
0037
0038 ui16RetCRC16 = hex2dec('FFFF');
0039 for I=1:length(data)
0040     ui8LookupTableIndex = bitxor(data(I),uint8(bitshift(ui16RetCRC16,-8)));
0041     ui16RetCRC16 =
bitxor(Crc_ui16LookupTable(double(ui8LookupTableIndex)+1),mod(bitshift(ui16RetCRC16,8),65536));
0042 end
0043
0044 crc = dec2hex(ui16RetCRC16,4);
0045
0046
0047
0048
0049
0050
0051
0052
0053
0054
```

hexsingle2num

PURPOSE

Convert single precision IEEE hexadecimal string to number.

SYNOPSIS

```
function x = hexsingle2num(s)
```

DESCRIPTION

```
Convert single precision IEEE hexadecimal string to number.

Ref: https://www.mathworks.com/matlabcentral/fileexchange/6927-hexsingle2num

HEXSINGLE2NUM(S), where S is a 8 character string containing a
hexadecimal number, returns a double type number equal to the IEEE single
precision floating point number it represents. Fewer than 8 characters
are padded on the right with zeros.

If S is a character array, each row is interpreted as a single precision
number (and returned as a double).

NaNs, infinities and denorms are handled correctly.

Example:
  hexsingle2num('40490fdb') returns Pi.
  hexsingle2num('bf8') returns -1.

See also HEX2NUM.
```

CROSS-REFERENCE INFORMATION

This function calls:

This function is called by:

- api_setSensitivity API: set IMU sensitivity (0.04 - 0.4)(default = 0.12)
- callback_countingReadAng callback: counting read Euler angle data
- callback_countingReadQuat callback: counting read quaternion
- callback_serialReadAng callback: serial read Euler angle data
- callback_serialReadAngGyroAccMag callback: serial read angle, gyroscope, accelerometer and magnetometer
- callback_serialReadDCM callback: serial read direct cosine matrix
- callback_serialReadQuat callback: serial read quaternion data
- callback_serialReadQuatAcc callback: serial read quaternion and accelerometer data
- callback_serialReadQuatAccMag callback: serial read quaternion, accelerometer and magnetometer data
- callback_serialReadQuatMag callback: serial read quaternion and magnetometer data

SOURCE CODE

```
0001 function x = hexsingle2num(s)
0002 % Convert single precision IEEE hexadecimal string to number.
0003 %
0004 % Ref: https://www.mathworks.com/matlabcentral/fileexchange/6927-hexsingle2num
0005 %
0006 % HEXSINGLE2NUM(S), where S is a 8 character string containing a
0007 % hexadecimal number, returns a double type number equal to the IEEE single
0008 % precision floating point number it represents. Fewer than 8 characters
0009 % are padded on the right with zeros.
0010 %
0011 % If S is a character array, each row is interpreted as a single precision
0012 % number (and returned as a double).
0013 %
0014 % NaNs, infinities and denorms are handled correctly.
0015 %
0016 % Example:
0017 %   hexsingle2num('40490fdb') returns Pi.
0018 %   hexsingle2num('bf8') returns -1.
0019 %
0020 % See also HEX2NUM.
0021
0022 % Based on Matlab's hex2num.
0023 % Note: IEEE Standard 754 for floating point numbers
0024 %
0025 %  Floating point numbers are represented as:
0026 %  x = +/- (1+f)*2^e
0027 %
0028 %  doubles: 64 bits
0029 %           Bit 63       (1 bit)  = sign (0=positive, 1=negative)
0030 %           Bit 62 to 52 (11 bits)= exponent biased by 1023
0031 %           Bit 51 to 0  (52 bits)= fraction f of the number 1.f
0032 %  singles: 32 bits
0033 %           Bit 31       (1 bit)  = sign (0=positive, 1=negative)
0034 %           Bit 30 to 23 (8 bits) = exponent biased by 127
0035 %           Bit 22 to 0  (23 bits)= fraction f of the number 1.f
0036
0037 % 21 June 2005 Fixed bug with underflow.
0038 %    Bug found by Matthias Noell (matthias.noell@heidelberg.com)
0039
0040 if iscellstr(s), s = char(s); end
0041 if ~ischar(s)
0042     error('Input to hexsingle2num must be a string.')
0043 end
0044 if isempty(s), x = []; return, end
0045
0046 [row,col] = size(s);
0047 blanks = find(s==' '); % Find the blanks at the end
0048 if ~isempty(blanks), s(blanks) = '0'; end % Zero pad the shorter hex numbers.
0049
0050 % Convert characters to numeric digits.
0051 % More than 8 characters are ignored
0052 % For double: d = zeros(row,16);
0053 d = zeros(row,8);
0054 d(:,1:col) = abs(lower(s)) - '0';
0055 d = d + ('0'+10-'a').*(d>9);
0056 neg = d(:,1) > 7;
0057 d(:,1) = d(:,1)-8*neg;
0058
0059 if any(d > 15) | any(d < 0)
0060     error('Input string to hexsingle2num should have just 0-9, a-f, or A-F.')
0061 end
0062
0063 % Floating point exponent.
0064 % For double: e = 16*(16*(d(:,1)-4) + d(:,2)) + d(:,3) + 1;
0065 % For double: e = 256*d(:,1) + 16*d(:,2) + d(:,3) - 1023;
0066 expBit = (d(:,3) > 7);
0067 e = 32*d(:,1) + 2*d(:,2) + expBit - 127;
0068 d(:,3) = d(:,3)-8*expBit;  % Remove most sig. bit of d(:,3) which belongs to exponent
0069
0070 % Floating point fraction.
0071 % For double: sixteens = [16;256;4096;65536;1048576;16777216;268435456];
0072 % For double: sixteens2 = 268435456*sixteens(1:6);
0073 % For double: multiplier = 1./[sixteens;sixteens2];
0074 % For double: f = d(:,4:16)*multiplier;
0075 sixteens = [16;256;4096;65536;1048576;16777216];
0076 multiplier = 2./[sixteens];
0077 f = d(:,3:8)*multiplier;
0078
0079 x = zeros(row,1);
0080 % Scale the fraction by 2 to the exponent.
0081 % For double: overinf = find((e>1023) & (f==0));
```

```
0082 overinf = find((e>127) & (f==0));
0083 if ~isempty(overinf), x(overinf) = inf; end
0084
0085 % For double: overNaN = find((e>1023) & (f~=0));
0086 overNaN = find((e>127) & (f~=0));
0087 if ~isempty(overNaN), x(overNaN) = NaN; end
0088
0089 % For double: underflow = find(e<-1022);
0090 underflow = find(e<-126);
0091 if ~isempty(underflow), x(underflow) = pow2(f(underflow),-126); end
0092
0093 % For double: allothers = find((e<=1023) & (e>=-1022));
0094 allothers = find((e<=127) & (e>=-126));
0095 if ~isempty(allothers), x(allothers) = pow2(1+f(allothers),e(allothers)); end
0096
0097 negatives = find(neg);
0098 if ~isempty(negatives), x(negatives) = -x(negatives); end
```
