

# Radar SDK - Read Me

Radar Sensor Detector (Motion Sensor)

*Devices:*     *iSYS-310x*

*Revision:*    *9*

*Date:*        *15.01.2016*

# Table of content

1.	What is the radar SDK? .....	5
2.	General Information .....	6
3.	Get Radar API running in your own application .....	7
4.	Programming Example with Radar API for iSYS-3106 .....	8
5.	Programming Example with Radar API for iSYS-3104 .....	9
6.	Used types in radar api .....	10
6.1.	Default data types .....	10
6.2.	Used structures and enumerators .....	10
6.2.1.	struct iSYSObjectList_t .....	10
6.2.2.	struct iSYSTrackedObject_t .....	11
6.2.3.	struct iSYSSystemInfo_t .....	12
6.2.4.	struct iSYSDeviceInfo_t .....	13
6.2.5.	struct iSYSTargetListPolar_t .....	13
6.2.6.	struct iSYSTargetPolar_t .....	14
6.2.7.	enum iSYSObjectListError_u .....	15
6.2.8.	enum iSYSObjectType_u .....	15
6.2.9.	enum iSYSModule_t .....	15
6.2.10.	enum iSYSInstallationSide_t .....	16
6.2.11.	enum iSYSTrafficDirection_t .....	16
6.2.12.	enum iSYSTargetListError_u .....	16
7.	Radar API Function Description .....	17
7.1.	Common Functions .....	17
7.1.1.	iSYS_initSystem(...) .....	17
7.1.2.	iSYS_exitSystem(...) .....	17
7.1.3.	iSYS_shutDownSystem(...) .....	17
7.1.4.	iSYS_getStatusSystem(...) Command deprecated .....	18
7.1.5.	iSYS_getSystemError(...) Command not implemented .....	18
7.1.6.	iSYS_restartSensors(...) .....	18
7.1.7.	iSYS_restartSystem(...) .....	19
7.1.8.	iSYS_getSystemInfo(...) .....	19
7.1.9.	iSYS_setStaticIP(...) .....	19
7.1.10.	iSYS_setDhcpIP(...) .....	20
7.1.11.	iSYS_setFrequencyChannel (...) .....	20
7.1.12.	iSYS_getFrequencyChannel (...) .....	21
7.1.13.	iSYS_saveAllParams(...) .....	21
7.1.14.	iSYS_getTargetListPolar(...) .....	21
7.2.	iSYS-3106 functions .....	22
7.2.1.	iSYS_getObjectList(...) .....	22
7.2.2.	iSYS_setInstallationOffsetXDistance (...) .....	22
7.2.3.	iSYS_getInstallationOffsetXDistance (...) .....	23
7.2.4.	iSYS_setInstallationOffsetYDistance (...) .....	23
7.2.5.	iSYS_getInstallationOffsetYDistance (...) .....	23
7.2.6.	iSYS_setInstallationAngle (...) .....	24
7.2.7.	iSYS_getInstallationAngle (...) .....	24

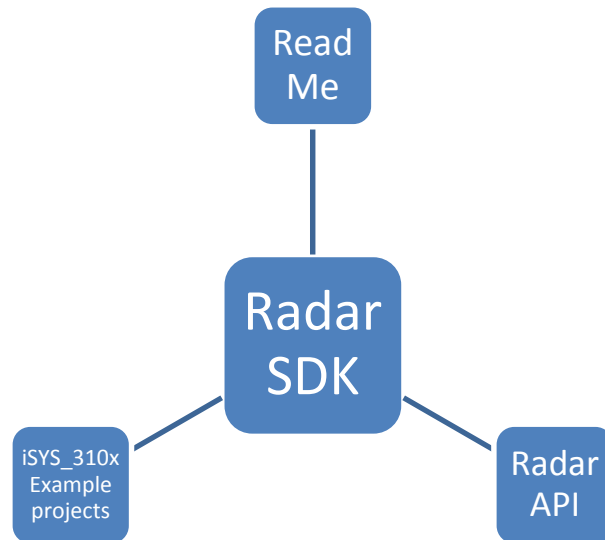
7.2.8.	iSYS_setInstallationLevel (...)	24
7.2.9.	iSYS_getInstallationLevel (...)	25
7.2.10.	iSYS_setTXPower (...)	25
7.2.11.	iSYS_getTXPower (...)	25
7.3.	iSYS-3104 functions	26
7.3.1.	iSYS_getObjectListTrafficMonitoring(...)	26
7.3.2.	iSYS_startYawCalibration(...)	26
7.3.3.	iSYS_getStatusYawCalibration(...)	26
7.3.4.	iSYS_setYawAngle(...)	27
7.3.5.	iSYS_getYawAngle(...)	27
7.3.6.	iSYS_setTrafficMonitoringNumberOfLanes(...)	27
7.3.7.	iSYS_getTrafficMonitoringNumberOfLanes(...)	27
7.3.8.	iSYS_setTrafficMonitoringDistanceToLane1(...)	28
7.3.9.	iSYS_getTrafficMonitoringDistanceToLane1(...)	28
7.3.10.	iSYS_setTrafficMonitoringInstallationHeight(...)	28
7.3.11.	iSYS_getTrafficMonitoringInstallationHeight(...)	28
7.3.12.	iSYS_setTrafficMonitoringLaneParameter(...)	29
7.3.13.	iSYS_getTrafficMonitoringLaneParameter(...)	29
7.3.14.	iSYS_setTrafficMonitoringUdpSocket(...)	30
7.3.15.	iSYS_getTrafficMonitoringUdpSocket(...)	30
8.	How communication between iSYS and dll works	31
8.1.	radarAPI compatibility	31
8.2.	Initialization scheme of radarSDK.dll Version 1-8	32
8.3.	Initialization scheme between radarSDK.dll Version 9-	33
9.	FAQ	34

# History

Document revision	Date	Change log
0.9	2014-10-28	Document created, BG
0.91	2014-10-30	Insert example to use virtual machine, BG
1	2014-11-14	Separate update functionality in an own executable, insert system error codes, BG
2	2014-12-17	Insert new functions an minor changes, BG/TP
3	2015-02-09	<ul style="list-style-type: none"><li>• Bugfix (7.1.3)</li><li>• Deprecated function iSYS_getStatusSystem (7.1.4)</li><li>• Insert new function iSYS_getSystemInfo (7.1.8)</li><li>• Insert new structs iSYSSystemInfo_t and iSYSDeviceInfo_t (6.2.3)</li></ul>
4	2015-03-18	<ul style="list-style-type: none"><li>• Added function to set static IP-address iSYS_setStaticIP (7.1.9)</li><li>• Added function to set static IP-address iSYS_setDhcpIP</li></ul>
5	2015-09-26	<ul style="list-style-type: none"><li>• Bugfix (7.1.8)</li><li>• added a general information (2)</li><li>• removed simtool_3106 (deprecated)</li></ul>
6	2015-10-26	<ul style="list-style-type: none"><li>• removed all Qt*.dll which were used for radarSDK.dll</li><li>• added traffic monitoring functions for iSYS-3104</li><li>• revised documentation</li></ul>
7	2015-11-27	<ul style="list-style-type: none"><li>• slight changes</li><li>• corresponds to radarAPI-Version 7.0</li></ul>
8	2015-12-17	<ul style="list-style-type: none"><li>• info that radarSDK isn't compatible with windows_xp</li><li>• defined new structures and enums for detections from sensor (6.2.5, 6.2.6, 6.2.12)</li><li>• new function to get detections from sensor (7.1.14)</li><li>• corresponds to radarAPI-version 8.0</li><li>• added traffic direction DIRECTION_INACTIVE (6.2.11)</li></ul>
9	2016-01-08	<ul style="list-style-type: none"><li>• added description of timestamp (6.2.2)</li><li>• updated 7.2.11</li><li>• FAQ (9)</li><li>• Description how radarAPI works (8)</li></ul>

## 1. What is the radar SDK?

The radar SDK includes the following components:

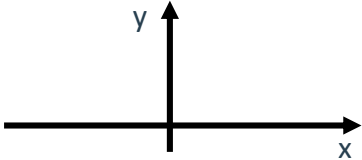
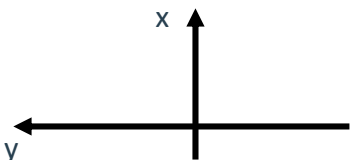


- Read me  
*Description of the following components "How to use the radar SDK?"*
- Radar API  
*DLL and header file for easy access to the iSYS-310x functionality*
- Example project for easy start
  - Visual Studio C++ 2010
  - VB.NET 2010 (only for iSYS-3106)
  - C#.NET 2010 (only for iSYS-3106)

## 2. General Information

- The Radar API was built on MS Windows 7 32-bit system with *Microsoft Visual C++ Compiler 10.0 (x86)*
- The Radar API was tested with MS Windows 7 32-bit
- The Radar API is incompatible with MS Windows XP, Vista and older MS Windows versions
- To guaranty 8 Byte alignment of defined structures the union type is used
- The Radar API is designed for synchronal use only (blocking application)
- Do not modify delivered Radar API files
- Use Radar API specific data types defined in radar\_SDK\_basicTypes.h (see 3)
- Radar API can only be used in one application at the same time

The radarSDK is designed for usage with all iSYS-310x variations. But some commands works only with specific devices.

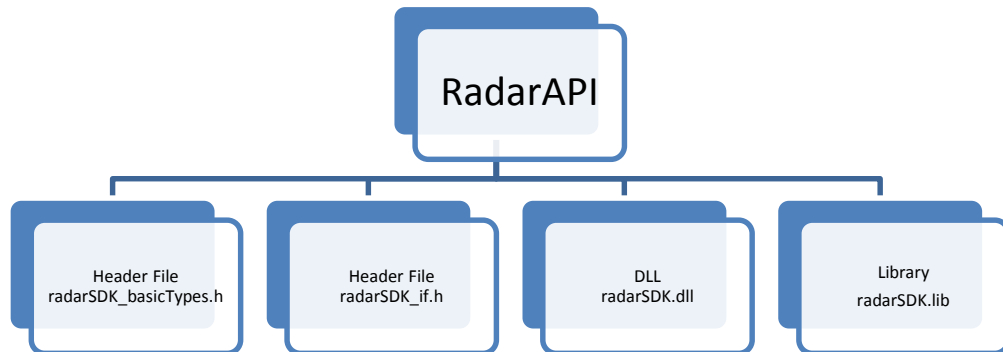
iSYS-3106	iSYS-3104
Area Monitoring	Traffic Monitoring
Master - Slave combination	Master only
no need to define monitored area	lane and other configuration necessary
Area Monitoring coordinate system 	Traffic Monitoring coordinate system 

The documentation of radarSDK function is structured in:

- common functions
- iSYS-3106 functions
- iSYS-3104 functions

### 3. Get Radar API running in your own application

The Radar API consists of the following components:



- radarSDK\_basicTypes.h  
*Contains all basic type definitions used by the Radar SDK*  
*(copy this into your project folder)*
- radarSDK\_if.h  
*Contains all necessary definitions, type definitions, structures and commands featured by the Radar API*  
*(copy this into your project folder)*
- radarSDK.dll  
*Library file which contains all functions referred in the header file*  
*(copy this into the folder where your executable is located)*
- radarSDK.lib  
*Links against \*.dll functions*  
*(copy this into your project folder)*

Each callable function (from Radar API) has a return value that inherits information about status of function. This return value is of type *iSYSResult\_t*. If *iSYSResult\_t* is *ERR\_OK* the command was successfully completed. Otherwise an error occurred during operation.

See also delivered example projects how to use radar API.

## 4. Programming Example with Radar API for iSYS-3106

The following listing shows an example how to use the Radar API for setting installation parameter for an iSYS-3106.

```
#include "radarSDK_if.h"

int main(void){
    iSYSHandle_t handle;
    iSYSResult_t res;
    /* try to connect to system with IP-Address 192.168.178.253 */
    res = iSYS_initSystem(&handle,192,168,178,253);
    if(res != ERR_OK)
        return 0;

    /* set installation parameter */
    res = iSYS_setInstallationOffsetXDistance(handle,ISYS_MASTER,0.2f);
    if(res != ERR_OK)
        return 0;

    res = iSYS_setInstallationOffsetXDistance(handle,ISYS_SLAVE,-0.2f);
    if(res != ERR_OK)
        return 0;

    res = iSYS_setInstallationOffsetYDistance(handle,ISYS_MASTER,0.3f);
    if(res != ERR_OK)
        return 0;

    res = iSYS_setInstallationOffsetYDistance(handle,ISYS_SLAVE,0.3f);
    if(res != ERR_OK)
        return 0;

    res = iSYS_setInstallationAngle(handle,ISYS_MASTER,10.0f);
    if(res != ERR_OK)
        return 0;

    res = iSYS_setInstallationAngle(handle,ISYS_SLAVE,-10.0f);
    if(res != ERR_OK)
        return 0;

    res = iSYS_setInstallationLevel(handle,ISYS_MASTER,6.0f);
    if(res != ERR_OK)
        return 0;

    res = iSYS_setInstallationLevel(handle,ISYS_SLAVE,6.0f);
    if(res != ERR_OK)
        return 0;

    /* after correct initialization get object list */
    iSYSObjectList_t objectList;
    while(1){
        res = iSYS_getObjectList(handle,&objectList,120.0f);
        if((res == ERR_OK) && (objectList.nrofTracks > 0)){
            //Post processing possible
        }
        else{
            //error handling
        }
    }
    /* shutdown before power down system */
    res = iSYS_shutdownSystem(handle);
    if(res != ERR_OK)
        return 0;

    /* de-initialize connected system before finishing own application */
    res = iSYS_exitSystem(handle);
    if(res != ERR_OK)
        return 0;

    return 0;
}
```

Listing 1: Example Code



## 5. Programming Example with Radar API for iSYS-3104

The following listing shows an example how to use the Radar API for setting installation parameter for an iSYS-3104.

**Note:** You have to define the number of lanes before you can configure each lanes.

```
#include "radarSDK_if.h"

int main(void){
    iSYSHandle_t handle;
    iSYSResult_t res;
    /* try to connect to system with IP-Address 192.168.178.253 */
    res = iSYS_initSystem(&handle,192,168,178,253);
    if(res != ERR_OK)
        return 0;

    /* set installation parameter */
    res = iSYS_setYawAngle(handle,-9.0f);
    if(res != ERR_OK)
        return 0;

    res = iSYS_setTrafficMonitoringInstallationHeight(handle,7.0f);
    if(res != ERR_OK)
        return 0;

    res = iSYS_setTrafficMonitoringNumberOfLanes(handle,2);
    if(res != ERR_OK)
        return 0;

    res = iSYS_setTrafficMonitoringDistanceToLane1(handle,1.0f);
    if(res != ERR_OK)
        return 0;

    /* set parameter for lane 1 */
    res = iSYS_setTrafficMonitoringLaneParameter(handle,1,INSTALLATION_SIDE_LEFT,
                                                0.0f,3.5f,30.0f,DIRECTION_APPROACHING);
    if(res != ERR_OK)
        return 0;

    /* set parameter for lane 2 */
    res = iSYS_setTrafficMonitoringLaneParameter(handle,2,INSTALLATION_SIDE_LEFT,
                                                -3.5f,3.5f,30.0f,DIRECTION_APPROACHING);
    if(res != ERR_OK)
        return 0;

    /* start traffic monitoring and application message output */
    res = iSYS_startTrafficMonitoring(handle);
    if(res != ERR_OK)
        return 0;

    /* after correct initialization get object list */
    iSYSObjectList_t objectList;
    while(1){
        res = iSYS_getObjectListTrafficMonitoring(handle,&objectList,60.0f);
        if((res == ERR_OK) && (objectList.nrOfTracks > 0)){
            //Post processing possible
        }
        else{
            //error handling
        }
    }

    /* shutdown before power down system */
    res = iSYS_shutdownSystem(handle);
    if(res != ERR_OK)
        return 0;

    /* de-initialize connected system before finishing own application */
    res = iSYS_exitSystem(handle);
    if(res != ERR_OK)
        return 0;

    return 0;
}
```

Listing 2: Example Code

## 6. Used types in radar api

In the Radar API are specific data types used. You will find these in `radar_SDK_basicTypes.h` and `radarSDK_if.h`.

### 6.1. Default data types

In `radar_SDK_basicTypes.h` are data types defined that are used in the radar api. For compatibility use this data types as well.

Data Type	Definition
<code>bool_t</code>	1 bit boolean
<code>float32_t</code>	32 bit floating point
<code>sint8_t</code> / <code>uint8_t</code>	8 bit signed / unsigned integer
<code>sint16_t</code> / <code>uint16_t</code>	16 bit signed / unsigned integer
<code>sint32_t</code> / <code>uint32_t</code>	32 bit signed / unsigned integer

### 6.2. Used structures and enumerators

There are structures and enumerators used in radar api. These are defined in `radarSDK_if.h` and make function call easier for the user. Please use these types as well.

**Note:** To avoid alignment errors some enumerators are also defined as union and the union type is used.

#### 6.2.1. struct `iSYSObjectList_t`

The `iSYSObjectList_t` is a struct that inherits the information of all tracked objects.

```
typedef struct iSYSObjectList {
    union iSYSObjectListError_u error;
    uint32_t nrOfTracks;
    iSYSTrackedObject_t trackedObjects[MAX_TRACKS];
} iSYSObjectList_t;
```

- `iSYSObjectListError_u`

Inherits information about the object list status (see 0).

- `uint32_t nrOfTracks`

Gives information about the number of active objects (tracks).

### 6.2.2. struct iSYSTrackedObject\_t

The iSYSTrackedObject\_t inherits all information of an active object. There are cartesian as well as polar coordinate information of object. **The reference coordinate system is the pole where the system is mounted.**

```
typedef struct iSYSTrackedObject {
    uint32_t objectID;
    union iSYSObjectType_u objectType;
    uint32_t timeStampUpper; /* time stamp upper */
    uint32_t timeStampLower; /* time stamp lower */
    uint32_t quality; /* track quality */
    float32_t cDistanceX; /* cartesian */
    float32_t cDistanceY; /* cartesian */
    float32_t cVelocityX; /* cartesian */
    float32_t cVelocityY; /* cartesian */
    float32_t pDistance; /* polar */
    float32_t pVelocity; /* polar */
    float32_t pAngle; /* polar */
} iSYSTrackedObject_t;
```

- *uint32\_t objectID*

Identifier for a valid object (doesn't change during lifetime).

- *iSYSObjectType\_u*

Classified object type (see 6.2.8).

**Note:** This information is prepared for future purposes. There is no classification available.

- *uint32\_t timeStampUpper and timeStampLower*

Inherits the number of milliseconds that have passed since 1970-01-01 00:00:00.000 (yyyy-MM-dd hh:mm:ss.zzz). It's **not** the time when the track was build.

Combine it to a 64 bit signed integer (included a conversion in visual studio example project of iSYS-3104).

**Note:** On each system power up time is set to 1970-01-01 00:00:00.000.

```
__int64 timeStamp = (__int64)timeStampUpper << 32 |
                    (__int64)timeStampLower;
```

- *uint32\_t quality*

Indicator for the track quality (equals 10 for best quality).

- *float32\_t cDistanceX; float32\_t cDistanceY*

Distance from reference point to tracked object in x and y direction in meter (cartesian coordinates).

Note: Pay attention for corresponding coordinate system (iSYS-3104 or iSYS-3106).

- *float32\_t cVelocityX ; float32\_t cVelocityY*

Velocity of tracked object in x and y directions in meters per seconds (cartesian coordinates). Pay attention for corresponding coordinate system (iSYS-3104 or iSYS-3106).

Note: Pay attention for corresponding coordinate system (iSYS-3104 or iSYS-3106).

- *float32\_t pDistance*

Distance from reference point to tracked object in meter (polar coordinates).

- *float32\_t pVelocity*

Velocity of tracked object in meters per seconds (polar coordinates).

Note: This information is prepared for future purposes. There is no classification available.

- *float32\_t pAngle*

Angle of tracked object to the reference point in degree (polar coordinates).

### 6.2.3. struct iSYSSystemInfo\_t

The *iSYSSystemInfo\_t* is a structure that inherits information about system and the connected devices.

```
typedef struct iSYSSystemInfo {  
    uint32_t serialNr; /* depends on firmware version of iSYS */  
    uint32_t swVersionMajor; /* software version major of system */  
    uint32_t swVersionMinor; /* software version minor of system */  
    uint32_t hwVersionMajor; /* hardware version major of system */  
    uint32_t hwVersionMinor; /* hardware version minor of system */  
    uint32_t alignmentDummy; /* dummy variable for alignment */  
    iSYSDeviceInfo_t masterInfo /* device Info of master sensor */  
    iSYSDeviceInfo_t slave1Info /* device Info of slave1 sensor */  
    iSYSDeviceInfo_t slave2Info /* device Info of slave2 sensor */  
    iSYSDeviceInfo_t slave3Info /* device Info of slave3 sensor */  
} iSYSSystemInfo_t;
```

Note: *iSYSSystemInfo\_t* is prepared to get information from up to 3 slave sensors (this is for future purpose). Up to now the iSYS-3106 supports only one slave sensor.

#### 6.2.4. struct iSYSDeviceInfo\_t

The *iSYSDeviceInfo\_t* inherits information about connected devices.

```
typedef struct iSYSDeviceInfo {  
    float32_t swVersion; /* software version of connected device */  
    float32_t blVersion; /* bootloader version of connected device */  
    float32_t serialNumber; /* serial number of connected device */  
} iSYSDeviceInfo_t;
```

Note: Devices that are not connected returns -1.

#### 6.2.5. struct iSYSTargetListPolar\_t

The structure *iSYSTargetListPolar\_t* inherits all detections made from the running systems.

```
typedef struct iSYSTargetListPolar {  
    uint16_t nrOfTargetsMaster;  
    uint16_t nrOfTargetsSlave1;  
    uint16_t nrOfTargetsSlave2;  
    uint16_t nrOfTargetsSlave3;  
    union iSYSTargetListError_u errorMaster;  
    union iSYSTargetListError_u errorSlave1;  
    union iSYSTargetListError_u errorSlave2;  
    union iSYSTargetListError_u errorSlave3;  
    iSYSTargetPolar_t targetListMaster[MAX_DETECTIONS];  
    iSYSTargetPolar_t targetListSlave1[MAX_DETECTIONS];  
    iSYSTargetPolar_t targetListSlave2[MAX_DETECTIONS];  
    iSYSTargetPolar_t targetListSlave3[MAX_DETECTIONS];  
} iSYSTargetListPolar_t;
```

- *nrOfTargetsMaster, nrOfTargetsSlave1, nrOfTargetsSlave2, nrOfTargetsSlave3*

Returns the number of valid elements in *iSYSTargetListPolar\_t*

- *iSYSTargetListError\_u*

Inherits information about the list status. (see 6.2.12)

- *iSYSTargetPolar\_t*

Inherits information about the targets. (see 0)

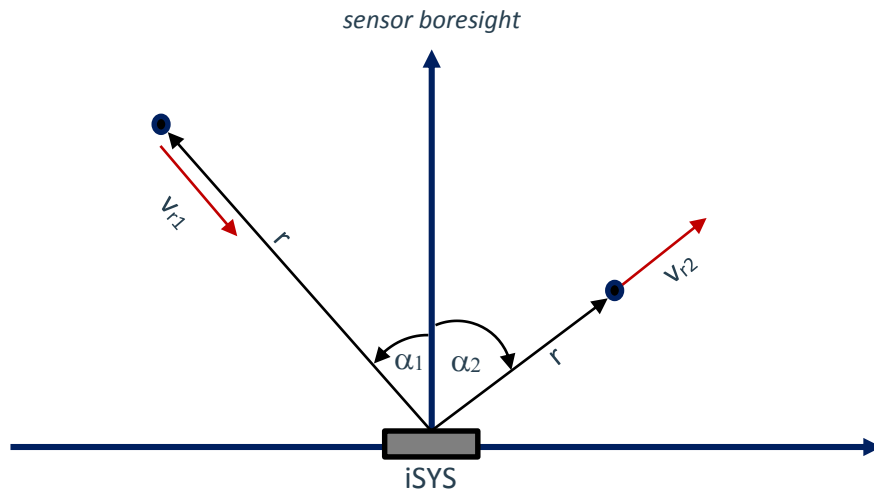
### 6.2.6. struct iSYSTargetPolar\_t

The structure iSYSTargetPolar\_t inherits data of one detection in a polar coordinate system.

```
typedef struct iSYSTargetPolar {  
    bool_t objectAvailable; /* true if target is valid */  
    uint16_t alignmentDummy;  
    float32_t velocity;      /* radial velocity [km/h] */  
    float32_t range;         /* radial distance [m] */  
    float32_t signal;        /* signal indicator */  
    float32_t angle;         /* [°] */  
    uint32_t reserved;  
} iSYSTargetPolar_t;
```

The measured angle is the angle between sensor boresight and target. Detections on the left side of sensor boresight have a positive angle. Detections on the right side of sensor boresight have a negative angle.

Detections with negative velocity moving towards the sensor. Detections with positive velocity moving away from sensor.



$r$ : Range in meter [m]

$\alpha_1$ : in degree [°] (in this example  $+x^\circ$ )

$\alpha_2$ : in degree [°] (in this example  $-x^\circ$ )

$v_{r1}$ : radial velocity in kilometers per hour [km/h] (in this example  $v_{r1} = -x$  km/h)

$v_{r2}$ : radial velocity in kilometers per hour [km/h] (in this example  $v_{r2} = +x$  km/h)

*Example: Calculation of polar range to x-y-coordinates*

$$x = \sin\left((-1) * \alpha * \frac{\pi}{180}\right) * r; \quad y = \cos\left(\alpha * \frac{\pi}{180}\right) * r;$$

**Note:** The radar sensor can measure only the radial velocity. This is the velocity of the target in the direction towards or away from the sensor. If you want to get the true target velocity you have to take the moving direction into account.

### 6.2.7. enum iSYSObjectListError\_u

The *iSYSObjectListError\_u* (*iSYSObjectListError\_t*) gives information about the object list status. The following statuses are available:

iSYSObjectListError_t	Description
OBJECT_LIST_OK	There are no problems with object list. Post processing is possible.
OBJECT_LIST_FULL	There are more objects available as MAX_TRACKS. Post processing is possible.
OBJECT_LIST_REFRESHED	There are no problems with object list. Post processing is possible.
OBJECT_LIST_ALREADY_REQUESTED	The requested object list was already sent. This happened if time difference to last request is less than 120ms/60ms. Request new list. No post processing necessary. Wait for next object list.

### 6.2.8. enum iSYSObjectType\_u

The *iSYSObjectType\_u* (*iSYSObjectType\_t*) inherits information about the object classification. The following classifications are possible.

iSYSObjectType_t	Description
OBJECT_TYPE_UNDEFINED	No classification possible.
OBJECT_TYPE_HUMAN	The object is classification as human.
OBJECT_TYPE_SMALL_PASSENGER_CAR	The object is classification as passenger car
OBJECT_TYPE_BIG_TRUCK	The object is classified as truck.

Note: This information is prepared for future purposes. There is no classification available.

### 6.2.9. enum iSYSModule\_t

An iSYS-3106 system consists of one master device iSYS-3106 and optional slave device iSYS-3006. To send commands to only one or both of them use the *iSYSModule\_t*.

iSYSModule_t	Description
ISYS_ALL	Send command to all connected devices.
ISYS_MASTER	Send command to iSYS-3106
ISYS_SLAVE	Send command to iSYS-3006

#### 6.2.10. enum iSYSInstallationSide\_t

The iSYS-3104 needs to know traffic monitoring parameters. One of these is the installation side. There are three installation sides available (see User Manual for more information).

iSYSInstallationSide_t	Description
INSTALLATION_SIDE_LEFT	iSYS-3104 is mounted left of the lanes
INSTALLATION_SIDE_RIGHT	iSYS-3104 is mounted right of the lanes
INSTALLATION_SIDE_GANTRY	iSYS-3104 is mounted on a gantry above the lanes

#### 6.2.11. enum iSYSTrafficDirection\_t

For iSYS-3104 user can set different traffic directions for each lane (generate application messages at trigger line). There are three traffic directions available (see User Manual for more information).

iSYSTrafficDirection_t	Description
DIRECTION_INACTIVE	application message inactive
DIRECTION_APPROACHING	application message only for approaching objects
DIRECTION_RECEDING	application message only for receding objects
DIRECTION_BOTH	application message for approaching and receding objects

#### 6.2.12. enum iSYSTargetListError\_u

The *iSYSTargetListError\_u* (*iSYSTargetListError\_t*) gives information about the target list status. The following statuses are available:

iSYSTargetListError_t	Description
TARGET_LIST_OK	There are no problems with target list. Post processing is possible.
TARGET_LIST_FULL	There are more detection available as MAX_DETECTIONS. Post processing is possible.
TARGET_LIST_REFRESHED	There are no problems with target list. Post processing is possible.
TARGET_LIST_ALREADY_REQUESTED	The requested target list was already sent. This happened if time difference to last request is less than 120ms/60ms. Request new list. No post processing necessary. Wait for next target list.



## 7. Radar API Function Description

The radar API function description describes all available functions with calling example. It is structured in 3 sections. There are functions that could be used for all iSYS-310x variations (see 7.1). Functions that are iSYS-3104 or iSYS-3106 specific got their own sections (see 7.2, 7.3).

### 7.1. Common Functions

#### 7.1.1. iSYS\_initSystem(...)

To initialize a connection to a connected iSYS-310x use iSYS\_initSystem (...).

**Input:**

```
iSYSHandle_t *pHandle, uint8_t ipPart1, uint8_t ipPart2, uint8_t ipPart3,
uint8_t ipPart4
```

For each connection to a system user has to create an empty iSYSHandle\_t pHandle that is a unique identifier to communicate with this unique system. This function initiate the connection and initialize pHandle.

**Example: initiate a connection with IP-Address: 192.168.178.253**

```
iSYSHandle_t pHandle;
iSYSResult_t res;
res = iSYS_initSystem(&pHandle,192,168,178,253);
```

#### 7.1.2. iSYS\_exitSystem(...)

To close (disconnect) an existing connection use iSYS\_exitSystem(...).

**Input:**

```
iSYSHandle_t pHandle
```

**Example: Close an existing connection with a valid iSYSHandle\_t pHandle**

```
iSYSResult_t res;
res = iSYS_exitSystem(pHandle);
```

#### 7.1.3. iSYS\_shutDownSystem(...)

Use this command for shut down the iSYS-310x before power off devices.

**Input:**

```
iSYSHandle_t pHandle
```

**Example: Get information about connected system**

```
iSYSResult_t res;
res = iSYS_shutDownSystem(pHandle);
```

#### 7.1.4. **iSYS\_getStatusSystem(...)** **Command deprecated**

To get information about the status of system use `iSYS_getStatusSystem`. The function gives information about serial number of system, software version, hardware version and the number of actual connected sensors.

**Input:**

```
iSYSHandle_t pHandle, float32_t *serialNumber, float32_t
*hardwareVersion, sint8_t *nrOfConnectedSensors
```

**Example: Get information about connected system**

```
iSYSResult_t res;
float32_t serialNr;
float32_t hwVersion;
sint8_t nrOfSensors;
res = iSYS_getStatusSystem(pHandle, &serialNr, &hwVersion, &nrOfSensors);
```

#### 7.1.5. **iSYS\_getSystemError(...)** **Command not implemented**

To get information about errors in system (e. g. `ISYS_SLAVE` doesn't send data) use `iSYS_getSystemError`. There are maximal 32 errors possible. After calling this function the errors will be deleted.

Each bit of the `uint32_t` variable set to 1 represents an error.

**Input:**

```
iSYSHandle_t pHandle, uint32_t *error
```

**Example: Get system errors**

```
iSYSResult_t res;
uint32_t error;
res = iSYS_getSystemError(pHandle, &error);
```

#### 7.1.6. **iSYS\_restartSensors(...)**

To restart the connected devices use `iSYS_restartSensors`.

**Input:**

```
iSYSHandle_t pHandle
```

**Example: Restart connected sensors corresponding to `iSYSHandle_t`**

```
iSYSResult_t res;
res = iSYS_restartSensors(pHandle);
```

### 7.1.7. iSYS\_restartSystem(...)

To restart the complete system (sensor and application) use iSYS\_restartSystem.

**Input:**

```
iSYSHandle_t pHandle
```

**Example: Restart system corresponding to iSYSHandle\_t**

```
iSYSResult_t res;  
res = iSYS_restartSystem(pHandle);
```

### 7.1.8. iSYS\_getSystemInfo(...)

To get information about the status of system use iSYS\_getSystemInfo. The function requests information about serial number of system, software version, hardware version and the number of actual connected sensors (see 6.2.3).

**Input:**

```
iSYSHandle_t pHandle, iSYSSystemInfo *systemInfo
```

**Example: Get information about connected system**

```
iSYSResult_t res;  
iSYSSystemInfo_t systemInfo;  
res = iSYS_getSystemInfo(pHandle, &systemInfo);
```

### 7.1.9. iSYS\_setStaticIP(...)

The default static IP-address of iSYS-3106 is 192.168.178.253. But the static IP-address could also be changed with function iSYS\_setStaticIP.

Please ensure that the used iSYSHandle\_t pHandle is already correct initialised. After successfully changing the IP-address pHandle will be set to NULL and has to be renewed.

**Attention:** Please ensure that each device in network has an unique IP-address.  
There is no check for valid IP settings!!!

**Input:**

```
iSYSHandle_t pHandle, uint8_t ipPart1, uint8_t ipPart2, uint8_t ipPart3,  
uint8_t ipPart4, uint8_t subnetPart1, uint8_t subnetPart2, uint8_t subnetPart3,  
uint8_t subnetPart4, uint8_t stdGatewayPart1, uint8_t stdGatewayPart2, uint8_t  
stdGatewayPart3, uint8_t stdGatewayPart4
```

**Example:** Change the IP-address of *pHandle* to:

. IP-address: 192.168.178.2  
. subnet mask: 255.255.255.0  
. standard gateway: 192.168.178.5

```
iSYSResult_t res;  
res = iSYS_setStaticIP(pHandle,192,168,10,2,255,255,255,0,192,168,178,5);
```

#### 7.1.10. iSYS\_setDhcpIP(...)

For changing the iSYS-3106 network configuration to DHCP use the function `iSYS_setDhcpIP(...)`

Please ensure that the used `iSYSHandle_t pHandle` is already correct initialised. After successfully changing the network configuration to DHCP mode `pHandle` will be set to NULL and has to be renewed.

**Input:**

```
iSYSHandle_t pHandle
```

**Example:** Change network configuration to DHCP mode of *pHandle*:

```
iSYSResult_t res;  
res = iSYS_setDhcpIP(pHandle);
```

#### 7.1.11. iSYS\_setFrequencyChannel (...)

The frequency channel of the devices could be changed with the function `iSYS_setFrequencyChannel`. If there is more than one iSYS-310x running at the same time it is recommended to change frequency channel (as explained in User Manual). To change the frequency channel iSYS-3104 you must use `ISYS_MASTER` as module parameter.

**Input:**

```
iSYSHandle_t pHandle, iSYSModule_t module, uint8_t channel
```

The frequency channel should set for each module separate to a different channel number.

**Example:** Change frequency channel to avoid interference with other iSYS-3106 systems:

```
iSYSResult_t res;  
res = iSYS_setFrequencyChannel(pHandle, ISYS_MASTER, 1);  
res = iSYS_setFrequencyChannel(pHandle, ISYS_SLAVE, 8);
```

**Example:** Change frequency channel to avoid interference with other iSYS-3104 systems:

```
iSYSResult_t res;  
res = iSYS_setFrequencyChannel(pHandle, ISYS_MASTER, 3);
```

### 7.1.12. iSYS\_getFrequencyChannel (...)

To read out frequency channel use the function iSYS\_getFrequencyChannel.

**Input:**

```
iSYSHandle_t pHandle, iSYSModule_t module, uint8_t *channel
```

**Example: Get the frequency channel of iSYS-3106 and iSYS-3006**

```
iSYSResult_t res;
uint8_t valueMaster;
uint8_t valueSlave;
res = iSYS_getFrequencyChannel(pHandle, ISYS_MASTER, &valueMaster);
res = iSYS_getFrequencyChannel(pHandle, ISYS_SLAVE, &valueSlave);
printf("Installation level of iSYS-3106:%i", &valueMaster);
printf("Installation level of iSYS-3006:%i", &valueSlave);
```

**Example: Get the frequency channel of iSYS-3104**

```
iSYSResult_t res;
uint8_t valueMaster;
res = iSYS_getFrequencyChannel(pHandle, ISYS_MASTER, &valueMaster);
printf("Installation level of iSYS-3104:%i", &valueMaster);
```

### 7.1.13. iSYS\_saveAllParams(...)

To save your settings for the next restart use iSYS\_saveAllParams.

**Input:**

```
iSYSHandle_t pHandle
```

**Example:**

```
iSYSResult_t res;
res = iSYS_saveAllParams(pHandle);
```

### 7.1.14. iSYS\_getTargetListPolar(...)

To get the list of detections in polar coordinates use iSYS\_getTargetListPolar.

**Input:**

```
iSYSHandle_t pHandle, iSYSTargetListPolar_t *targetList, float32_t
timeoutMs
```

The information of detected targets are in the structure iSYSTargetListPolar\_t. The function iSYS\_getTargetListPolar fetches the data from the connected system and save it in targetList. With timeoutMs user can set a timeout in milliseconds when the function shall exit.

**Note:** timeoutMs should be at least 60ms.

**Example: Requests an polar target list of a valid iSYSHandle\_t with a timeout of 100ms**

```
iSYSTargetListPolar_t pTargetList;  
iSYSResult_t res;  
res = iSYS_getTargetListPolar(handle, &pTargetList, 100);
```

## 7.2. iSYS-3106 functions

If you're using iSYS-3106 please use the following functions to request object lists and set/get the installation parameters (the reference point for installation parameters is the mounted pole).

**NOTE:** It is essential to set the installation parameter carefully that the tracker works well.

### 7.2.1. iSYS\_getObjectList(...)

User can request an object list of a connected system with iSYS\_getObjectList. **Pay attention to coordinate system!**

**Input:**

```
iSYSHandle_t pHandle, iSYSObjectList_t *objectList, float32_t timeoutMs
```

The information of valid objects are in the structure iSYSObjectList\_t. Therefore user has to create an empty iSYSObjectList\_t \*pObjectList. The function iSYS\_getObjectList fetches the data from the connected system and save it in pObjectList. With timeoutMs user can set a timeout in milliseconds when the function shall exit.

**Note:** timeoutMs should be at least 120ms.

**Example: Requests an object list of a valid iSYSHandle\_t pHandle with a timeout of 100ms**

```
iSYSObjectList_t pObjectList;  
iSYSResult_t res;  
res = iSYS_getObjectList(handle, &pObjectList, 100);
```

### 7.2.2. iSYS\_setInstallationOffsetXDistance (...)

The x-distance in meter between the devices and the pole (reference point) is set with the function iSYS\_setInstallationOffsetXDistance.

**Input:**

```
iSYSHandle_t pHandle, iSYSModule_t module, float32_t xDistance
```

The installation offset in x distance should set for each module.

**Example: Set the installation offset in distance x to 0.2m for Master and -0.2m for Slave.**

```
iSYSResult_t res;  
res = iSYS_setInstallationOffsetXDistance(pHandle, ISYS_MASTER, 0.2f);  
res = iSYS_setInstallationOffsetXDistance(pHandle, ISYS_SLAVE, -0.2f);
```

### 7.2.3. iSYS\_getInstallationOffsetXDistance (...)

To read out the x-distance in meter use the function iSYS\_getInstallationOffsetXDistance.

**Input:**

```
iSYSHandle_t pHandle, iSYSModule_t module, float32_t *xDistance
```

**Example: Get the installation offset in distance x**

```
iSYSResult_t res;
float32_t valueMaster;
float32_t valueSlave;
res = iSYS_getInstallationOffsetXDistance(pHandle, ISYS_MASTER, &valueMaster);
res = iSYS_getInstallationOffsetXDistance(pHandle, ISYS_SLAVE, &valueSlave);
printf("Offset distance X of iSYS-3106:%f", &valueMaster);
printf("Offset distance X of iSYS-3006: %f", &valueSlave);
```

### 7.2.4. iSYS\_setInstallationOffsetYDistance (...)

The y-distance in meter between the devices and the pole (reference point) is set with the function iSYS\_setInstallationOffsetYDistance.

**Input:**

```
iSYSHandle_t pHandle, iSYSModule_t module, float32_t yDistance
```

The installation offset in y distance should set for each module.

**Example: Set the installation offset in distance y 0.3m for Master and -0.3m for Slave**

```
iSYSResult_t res;
res = iSYS_setInstallationOffsetYDistance(pHandle, ISYS_MASTER, 0.3f);
res = iSYS_setInstallationOffsetYDistance(pHandle, ISYS_SLAVE, 0.3f);
```

Alternative use if offsets of both devices are equal:

```
iSYSResult_t res;
res = iSYS_setInstallationOffsetYDistance (pHandle, ISYS_ALL, 0.3f);
```

### 7.2.5. iSYS\_getInstallationOffsetYDistance (...)

To read out the y-distance in meter use the function iSYS\_getInstallationOffsetYDistance.

**Input:**

```
iSYSHandle_t pHandle, iSYSModule_t module, float32_t *yDistance
```

**Example: Get the installation offset in distance y**

```
iSYSResult_t res;
float32_t valueMaster;
float32_t valueSlave;
res = iSYS_getInstallationOffsetYDistance(pHandle, ISYS_MASTER, &valueMaster);
res = iSYS_getInstallationOffsetYDistance(pHandle, ISYS_SLAVE, &valueSlave);
printf("Offset distance Y of iSYS-3106:%f", &valueMaster);
printf("Offset distance Y of iSYS-3006:%f", &valueSlave);
```

### 7.2.6. iSYS\_setInstallationAngle (...)

The installation angle in degree of the devices is set with the function iSYS\_setInstallationAngle.

**Input:**

```
iSYSHandle_t pHandle, iSYSModule_t module, float32_t angle
```

The installation angle should be set for each module.

**Example: Set the installation angle to 10° for Master and -10° for Slave.**

```
iSYSResult_t res;  
res = iSYS_setInstallationAngle(pHandle, ISYS_MASTER, 10.0f);  
res = iSYS_setInstallationAngle(pHandle, ISYS_SLAVE, -10.0f);
```

### 7.2.7. iSYS\_getInstallationAngle (...)

To read out installation angles in degree use the function iSYS\_getInstallationAngle.

**Input:**

```
iSYSHandle_t pHandle, iSYSModule_t module, float32_t *angle
```

**Example: Get the installation angle.**

```
iSYSResult_t res;  
float32_t valueMaster;  
float32_t valueSlave;  
res = iSYS_getInstallationAngle(pHandle, ISYS_MASTER, &valueMaster);  
res = iSYS_getInstallationAngle(pHandle, ISYS_SLAVE, &valueSlave);  
printf("Installation angle of iSYS-3106:%f", &valueMaster);  
printf("Installation angle of iSYS-3006:%f", &valueSlave);
```

### 7.2.8. iSYS\_setInstallationLevel (...)

The installation height in meter from ground to the devices is set with the function iSYS\_setInstallationLevel.

**Input:**

```
iSYSHandle_t pHandle, iSYSModule_t module, float32_t level
```

The installation level should be set for each module.

**Example:**

```
iSYSResult_t res;  
res = iSYS_setInstallationLevel(pHandle, ISYS_MASTER, 6.0f);  
res = iSYS_setInstallationLevel(pHandle, ISYS_SLAVE, 6.0f);
```

Alternative use if heights of both devices are equal:

```
iSYSResult_t res;  
res = iSYS_setInstallationLevel(pHandle, ISYS_ALL, 6.0f);
```



### 7.2.9. iSYS\_getInstallationLevel (...)

To read out installation height (level) in meter use the function iSYS\_getInstallationLevel.

**Input:**

```
iSYSHandle_t pHandle, iSYSModule_t module, float32_t *level
```

**Example: Get the installation level**

```
iSYSResult_t res;
float32_t valueMaster;
float32_t valueSlave;
res = iSYS_getInstallationLevel(pHandle, ISYS_MASTER, &valueMaster);
res = iSYS_getInstallationLevel(pHandle, ISYS_SLAVE, &valueSlave);
printf("Installation level of iSYS-3106:%f", &valueMaster);
printf("Installation level of iSYS-3006:%f", &valueSlave);
```

### 7.2.10. iSYS\_setTXPower (...)

The TX power of the devices is set with the function iSYS\_setTXPower. Regard the regulations due power limitations while adjusting the TX power (see User Manual). A value of 0 -> 18dBm and a value of 255 -> 34dBm. The scaling between is linear. A change in the value of 10 means that the output power changes about 0.625dB (e.g. value of 128 -> 26dBm).

**Input:**

```
iSYSHandle_t pHandle, iSYSModule_t module, uint8_t power
```

**Example: Set the TX power for maximum range performance**

```
iSYSResult_t res;
res = iSYS_setTXPower(pHandle, ISYS_MASTER, 255);
res = iSYS_setTXPower(pHandle, ISYS_SLAVE, 255);
```

Alternative use if both devices are set to the same TX power:

```
iSYSResult_t res;
res = iSYS_setTXPower(pHandle, ISYS_ALL, 255);
```

### 7.2.11. iSYS\_getTXPower (...)

To read out TX power use the function iSYS\_getTXPower. A value of 0 means 18dBm and a value of 255 means 34dBm. The scale between is linear (e.g. value of 128 -> 26dBm).

**Input:**

```
iSYSHandle_t pHandle, iSYSModule_t module, uint8_t *power
```

```
iSYSResult_t res;
uint8_t valueMaster;
uint8_t valueSlave;
res = iSYS_getTXPower(pHandle, ISYS_MASTER, &valueMaster);
res = iSYS_getTXPower(pHandle, ISYS_SLAVE, &valueSlave);
printf("TX power of iSYS-3106:%i", &valueMaster);
printf("TX power of iSYS-3006:%i", &valueSlave);
```

## 7.3. iSYS-3104 functions

### 7.3.1. iSYS\_getObjectListTrafficMonitoring(...)

User can request an object list of a connected system with iSYS\_getObjectListTrafficMonitoring.

**Pay attention to coordinate system!**

**Input:**

```
iSYSHandle_t pHandle, iSYSObjectList_t *objectList, float32_t timeoutMs
```

The information of a valid object is in the structure iSYSObjectList\_t. Therefore user has to create an empty iSYSObjectList\_t \*pObjectList. With timeoutMs user can set a timeout in milliseconds when the function shall exit.

**Note:** Recommended is a timeoutMs of 60 because the iSYS-3104 has a cycle time of 60ms.

**Example: Request an object list of a valid iSYSHandle\_t pHandle with a timeout of 60ms**

```
iSYSObjectList_t pObjectList;  
res = iSYS_getObjectListTrafficMonitoring(handle, &pObjectList, 60.0f);
```

### 7.3.2. iSYS\_startYawCalibration(...)

To set the yaw angle of iSYS-3104 user can start an automatically yaw calibration with the function iSYS\_startYawCalibration. The function iSYS\_getStatusYawCalibration returns the status information of yaw calibration.

**Input:**

```
iSYSHandle_t pHandle, uint32_t nrOfTracks
```

**Example: Start yaw calibration with 20 tracked objects**

```
res = iSYS_startYawCalibration(handle, 20);
```

### 7.3.3. iSYS\_getStatusYawCalibration(...)

To get status information about the running yaw calibration use iSYS\_getStatusYawCalibration. This function returns the progress and the calculated yaw angle. As long as yaw calibration is running the return value of progress is between 0 (0%) and 1 (100%), the value of yaw angle is -100.

**Input:**

```
iSYSHandle_t pHandle, float32_t *progress, float32_t *yawAngle
```

**Example: Get status of actual yaw calibration.**

```
float32_t progress;  
float32_t yawAngle;  
res = iSYS_getStatusYawCalibration(handle, &progress, &yawAngle);
```

#### 7.3.4. iSYS\_setYawAngle(...)

User can set the yaw angle of iSYS-3104 manually with iSYS\_setYawAngle.

**Input:**

```
iSYSHandle_t pHandle, float32_t yawAngle
```

**Example: Set yaw angle to 10°.**

```
res = iSYS_setYawAngle(handle, 10.0f);
```

#### 7.3.5. iSYS\_getYawAngle(...)

User can get the yaw angle of iSYS-3104 with iSYS\_getYawAngle.

**Input:**

```
iSYSHandle_t pHandle, float32_t *yawAngle
```

**Example: Get yaw angle.**

```
float32_t yawAngle;  
res = iSYS_getYawAngle(handle, &yawAngle);
```

#### 7.3.6. iSYS\_setTrafficMonitoringNumberOfLanes(...)

For traffic monitoring it is necessary to set the number of observed lanes. Use the function iSYS\_setTrafficMonitoringNumberOfLanes to set the number of lanes.

**Input:**

```
iSYSHandle_t pHandle, uint8_t nrOfLanes
```

**Example: Set the number of lanes to 4.**

```
res = iSYS_setTrafficMonitoringNumberOfLanes(handle, 4);
```

#### 7.3.7. iSYS\_getTrafficMonitoringNumberOfLanes(...)

To read back the number of lanes use iSYS\_setTrafficMonitoringNumberOfLanes.

**Input:**

```
iSYSHandle_t pHandle, uint8_t *nrOfLanes
```

**Example: Get number of lanes.**

```
uint8_t nrOfLanes;  
res = iSYS_getTrafficMonitoringNumberOfLanes(handle, &nrOfLanes);
```

### 7.3.8. iSYS\_setTrafficMonitoringDistanceToLane1(...)

To set the position of iSYS-3104 due to y-axis use iSYS\_setTrafficMonitoringDistanceToLane1. With this function you can set an offset between the origin of coordinate system and the iSYS-3104.

**Note:** If you set a value for distance to lane 1 different to zero it is recommended to set starting position of lane 1 to zero.

**Input:**

```
iSYSHandle_t pHandle, float32_t distToLane1
```

**Example: Set the distance to lane 1 to 0.5m.**

```
res = iSYS_setTrafficMonitoringDistanceToLane1(handle,0.5f);
```

### 7.3.9. iSYS\_getTrafficMonitoringDistanceToLane1(...)

To get the distance of iSYS-3104 to lane 1 use iSYS\_getTrafficMonitoringDistanceToLane1.

**Input:**

```
iSYSHandle_t pHandle, float32_t *distToLane1
```

**Example: Get distance to lane 1.**

```
float32_t distToLane1;  
res = iSYS_getTrafficMonitoringDistanceToLane1(handle,&distToLane1);
```

### 7.3.10. iSYS\_setTrafficMonitoringInstallationHeight(...)

To set the installation height of iSYS-3104 use iSYS\_setTrafficMonitoringInstallationHeight.

**Input:**

```
iSYSHandle_t pHandle, float32_t instHeight
```

**Example: Set installation height to 4m.**

```
res = iSYS_setTrafficMonitoringInstallationHeight(handle,4.0f);
```

### 7.3.11. iSYS\_getTrafficMonitoringInstallationHeight(...)

To get the installation height of iSYS-3104 use iSYS\_getTrafficMonitoringInstallationHeight.

**Input:**

```
iSYSHandle_t pHandle, float32_t *instHeight
```

**Example: Get distance to lane 1.**

```
float32_t instHeight;  
res = iSYS_getTrafficMonitoringInstallationHeight(handle,&instHeight);
```

### 7.3.12. iSYS\_setTrafficMonitoringLaneParameter(...)

User must set specific parameters for each lane. This could be done with iSYS\_setTrafficMonitoringLaneParameter. Parameters are equal to iGUI-3104 lane configuration.

**Note:** Before you can set parameters for each lane set the correct number of lanes (see 7.3.6).

Parameters	Description	Example
uint8_t laneNumber	Number of lane which should be set	1
iSYSInstallationSide_t side	installation side of iSYS-3104	left
float32_t y	start position of lane (depends on installation side) [m]	0.0
float32_t laneWidth	lane width [m]	3.75
float32_t triggerLine	distance where the application message should be triggered [m]	40.0
iSYSTrafficDirection_t direction	moving direction of objects that should trigger the application message	both

**Note:** If you set a starting position of lane 1 different to zero it is recommended to set distance to lane 1 to zero.

**Input:**

```
iSYSHandle_t pHandle, uint8_t laneNumber, iSYSInstallationSide_t side,
float32_t y, float32_t laneWidth, float32_t triggerLine,
iSYSTrafficDirection_t direction
```

**Example:** Set the example parameters from table above.

```
res = iSYS_setTrafficMonitoringLaneParameter(handle, 1,
INSTALLATION_SIDE_LEFT, 0.0f, 3.75f, 40.0f, DIRECTION_BOTH);
```

### 7.3.13. iSYS\_getTrafficMonitoringLaneParameter(...)

Read back lane parameters for each lane using iSYS\_getTrafficMonitoringLaneParameter.

**Input:**

```
iSYSHandle_t pHandle, uint8_t laneNumber, iSYSInstallationSide_t *side,
float32_t *y, float32_t *laneWidth, float32_t *triggerLine,
iSYSTrafficDirection_t *direction
```

**Example:** Get parameters for lane 1.

```
iSYSInstallationSide_t side;
float32_t y;
float32_t laneWidth;
float32_t triggerLine;
iSYSTrafficDirection_t direction;
res = iSYS_getTrafficMonitoringLaneParameter(handle, 1, &side, &y,
&laneWidth, &triggerLine, &direction);
```

#### 7.3.14. iSYS\_setTrafficMonitoringUdpSocket(...)

Application messages are output on RS-485 interface. In addition it is possible to activate application message output on UDP port. The messages will be sent over both interfaces. **User has to set the UDP port number.** Be sure that chosen port is open and available (e.g. 62200).

**Input:**

```
iSYSHandle_t pHandle, uint8_t enable, uint32_t portNumber
```

**Example: Set UDP active with port 62200**

```
res = iSYS_setTrafficMonitoringUdpSocket(handle, 1, 62200);
```

#### 7.3.15. iSYS\_getTrafficMonitoringUdpSocket(...)

It is possible to check if application message output on UDP port is already active and read back the port number by using iSYS\_getTrafficMonitoringUdpSocket.

**Input:**

```
iSYSHandle_t pHandle, uint8_t enable, uint32_t portNumber
```

**Example:**

```
res = iSYS_getTrafficMonitoringUdpSocket(handle, &enable, &portNumber);
```

## 8. How communication between iSYS and dll works

Network communication between iSYS-310x and radarSDK.dll works with TCP/IP. This protocol ensures that no data getting lost. If there is a corrupted data package detected by receiver (due to bad network connection), there will be a data package retransmission (also known as TCP-Retransmission) requested by receiver.

The communication works in a synchronous way. That means if you send a command, function blocks the calling thread until it receives acknowledge from data or a function timeout expired e.g. timeout in function to get object list (see chapter 7.2.1 or 7.3.1).

If timeout expired, e.g. because of TCP-Retransmission, data will be discarded and return value of function equals not ERR\_OK.

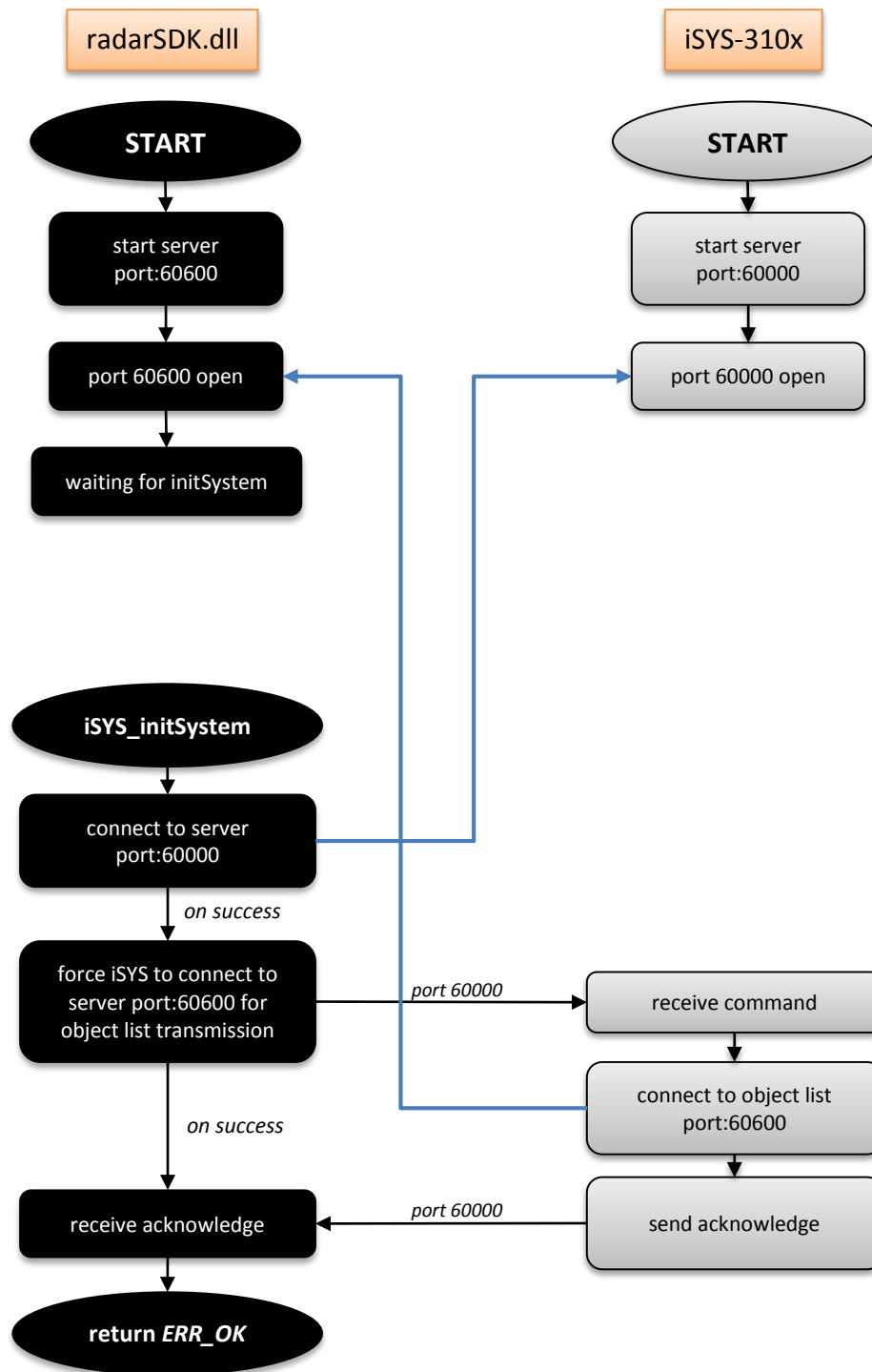
How often a TCP-Retransmission is necessary depends on the quality of your network connection.

### 8.1. radarAPI compatibility

*.dll-Version	iSYS-3106	iSYS-3104
1 - 8	TNK_0001_0000 – TNK_0001_4000	TNK_0001_0000 - TNK_0001_3001
9 - x	---	TNK_0001_4000 -

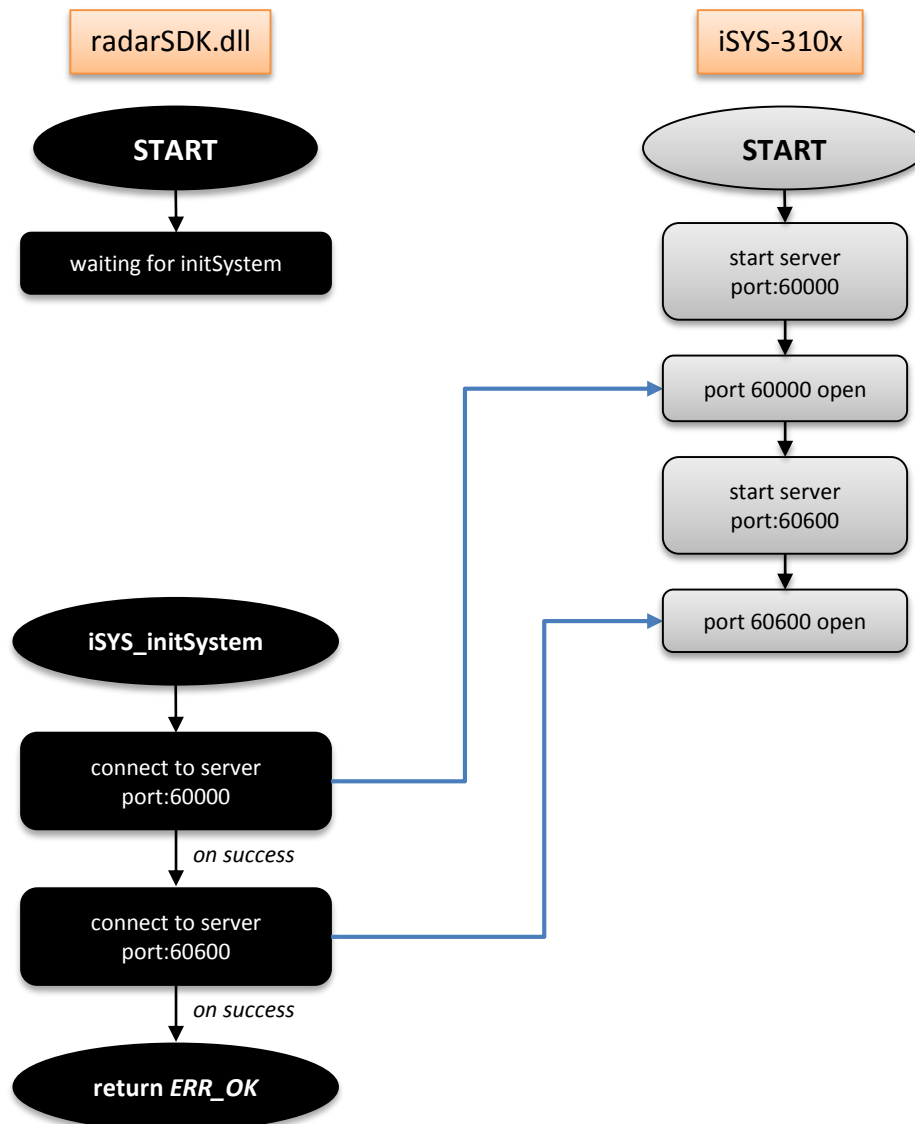
\*.dll Version 9 is backward compatible with old firmware versions of iSYS-310x

## 8.2. Initialization scheme of radarSDK.dll Version 1-8





### 8.3. Initialization scheme between radarSDK.dll Version 9-



If there is an “old” firmware on iSYS running (see chapter 8.1) \*.dll starts a server on port 60600 during *iSYS\_initSystem* function and performs the initialization described in chapter 8.2.

## 9. FAQ

- **How quickly can I request an iSYSObjectList?**

This depends on the quality of your network connection. With a good network connection a request time of 60ms should be no problem. When you have a slow network and the transmission takes longer than the defined timeout you will lose the datapacket. Moreover a rational request time depends on the system you use. An **iSYS-3106** has a cycle time of 120ms so a new iSYSObjectList might be requested every 120ms.

An **iSYS-3104** has a cycle time of 60ms so a new iSYSObjectList might be requested every 60ms.

There is no problem to request iSYSObjectList faster than the processing cycle time. If you receive an old list you already received an information in the iSYSObjectListError\_u (OBJECT\_LIST\_ALREADY\_REQUESTED) is included.

- **Is it possible for targets to stall in the buffer and be returned at a later time?**

No, this should not happen. After every processing cycle - every 60ms(iSYS3104) or 120ms(iSYS3106) - the internal iSYSObjectList buffer will be replaced with actual data.

- **Is a buffer overrun of iSYSObjectList possible if I request data in a slow cycle?**

No, there is no buffer overrun possible because iSYS only sends iSYSObjectList if an iSYSObjectList is requested.

- **What is the maximum number of possible tracked objects?**

The iSYS-310x has a limit of maximum 60 tracked object.

- **It seems that I lost many iSYSObjectList's. What can I do to optimize my performance?**

This could happen because of TCP-Retransmissions of data packages (see chapter 8). Check your network connection in this case.

- **I can't connect to an iSYS-310x that is "pingable" in the command prompt.**

This can have several reasons:

- Check (e. g. in task manager) if there are still applications running which uses radarSDK.dll (e.g. iGUI-3104).
- Check if all necessary ports (60000, 60500, 60600) are unblocked in your network. The firewall can block the communication.
- Restart iSYS-310x by power up and try it again.
- Ask your local IT-department for help. They should have experience with blocked ports or general TCP/IP connections

### InnoSenT GmbH

Am Roedertor 30  
97499 Donnersdorf  
GERMANY

Tel.: +49 95289518-0  
E-Mail: [info@innosent.de](mailto:info@innosent.de)  
[www.innosent.de](http://www.innosent.de)