



SYMEO 2D Protocol
Protocol description



LPR 2D Protocol

- Binary Protocol & ASCII Protocol -

| | |
|--|-----------|
| 1. INTRODUCTION / BASICS..... | 6 |
| 2. INTERFACES | 7 |
| 3. BINARY FORMAT OF THE PROTOCOL..... | 8 |
| 3.1. DATA TYPES | 8 |
| 3.2. BYTE STUFFING..... | 8 |
| 3.3. GENERAL STRUCTURE..... | 9 |
| 3.4. DATA FIELD | 11 |
| 3.4.1. START..... | 11 |
| 3.4.2. END..... | 11 |
| 3.4.3. TIMESTAMP..... | 12 |
| 3.4.4. POSITION..... | 12 |
| 3.4.5. VELOCITY | 13 |
| 3.4.6. ORIENTATION..... | 13 |
| 3.4.7. POSITION-ERROR | 14 |
| 3.4.8. VELOCITY-ERROR..... | 14 |
| 3.4.9. ORIENTATION-ERROR..... | 15 |

| | |
|---|-----------|
| 3.4.10. USER-DATA..... | 15 |
| 3.4.11. SYSTEM-ERROR..... | 16 |
| 3.4.12. SATELLITE-STATE..... | 16 |
| 3.5. CRC..... | 17 |
| 4. ASCII FORMAT OF THE PROTOCOL..... | 18 |
| 4.1. DATA TYPES..... | 18 |
| 4.2. GENERAL STRUCTURE..... | 18 |
| 4.3. DATA FIELDS..... | 20 |
| 4.3.1. START..... | 20 |
| 4.3.2. END..... | 20 |
| 4.3.3. TIMESTAMP..... | 21 |
| 4.3.4. POSITION..... | 21 |
| 4.3.5. VELOCITY..... | 23 |
| 4.3.6. ORIENTATION..... | 23 |
| 4.3.7. POSITION-ERROR..... | 24 |
| 4.3.8. VELOCITY-ERROR..... | 24 |
| 4.3.9. ORIENTATION-ERROR..... | 25 |
| 4.3.10. USER-DATA..... | 26 |
| 4.3.11. SYSTEM-ERROR..... | 26 |
| 4.3.12. SATELLITE-STATE..... | 28 |

| | |
|--|-----------|
| 4.3.13. CRC..... | 28 |
| 5. BIT MASK SELECTED-FIELDS | 30 |
| 6. CRC CALCULATION..... | 31 |
| 7. ERROR CODES..... | 31 |

History

| Version | Date | Description |
|---------|------------|------------------------|
| 2.2 | 09.12.2008 | Changed Layout |
| 2.3 | 16.04.2009 | Added satellite fields |

1. Introduction / Basics

The Symeo 2D protocol is the interface between the LPR 2D system and the user. To keep the protocol simple (i.e. easy to parse) the following aspects are considered:

- consistent data packet

The protocol has a standard data packet with a fixed length.

- configuration

The structure of the protocol can be configured by the user. It is possible to output only the required data. Therefore you can use also a slower data interface when using less data.

The Protocol is unidirectional. The LPR 2D system sends data to the user, but the LPR 2D system does not receive any data from the user.

You can choose between the binary format and the ASCII format. The formal structure of both protocols is the same.

To configure the protocol you have to set the parameters in a configuration file for the protocol.

Insert the following text in the configuration file to choose the format of the protocol:

```
[Format]  binary
```

or

```
[Format]  ascii
```

2. Interfaces

The 2D protocol can be used with the following interfaces:

- RS-232 (serial interface)
- TCP/IP
- UDP

Not all combinations of interface and protocol format make sense. For example when using the binary protocol with deactivated byte stuffing on a serial interface, the user does not know the beginning of a new data packet.

When choosing the serial interface you have to keep in mind a sufficiently high baud rate.

To identify the **interface of the protocol** write the following line into the configuration file.

```
[Interface]    serial
```

or

```
[Interface]    tcp
```

or

```
[Interface]    udp
```

When choosing the serial interface you have to add also the **COM port**, i.e.:

```
[Port]         1
```

or

```
[Port]         /dev/tty0
```

When choosing **TCP/IP** or **UDP-interface** you have to add a **port** as well, i.e.:

```
[Port]         1234
```

In the TCP/IP mode the program works as a server. This means the receiver (user) has to connect itself to the server port. In the **UDP mode** the program works as a sender. This means you have to enter the **target port of the receiver**. The IP-address has to be entered as well, i.e.:

```
[Destination]  192.168.255.255
```

3. Binary format of the protocol

For the binary format the entire data packet of the 2D protocol is transferred as binary data. This means that the binary format is more compact than the ASCII format, but is not human-readable.

3.1. Data types

The length of all data fields is a multiple of one byte.

The data types are exclusively bit field or integer. Integer can be signed or unsigned. This is specified in the description of each integer data field. Integer data fields with a length of more than one byte are all encoded in network byte order (Big Endian).

3.2. Byte Stuffing

The data packets are transmitted continuously with a constant length. The problem is here to identify the beginning of a data packet. Although an identification character for the beginning of the data packet is sent, exactly this identification can happen to appear in the following data packet. Without an additional technique it is not possible to detect the beginning of the data packet.

If the protocol is used with a TCP/IP interface, the first data packet is first sent when the socket is connected. The first byte of the socket is always the beginning of the data packet. Because all data packets have the same length and the transmission via TCP/IP is error proof, it is possible to read always the same length (bytes) of data packets on the receiver side.

If you use the RS 232 interface there is no proven failure free transmission of data. The receiver might start reading the data at an arbitrary moment. There is no possibility to detect the beginning of the data packet.

To use the protocol for a RS-232 interface the binary data must apply byte stuffing. When byte stuffing is active, reserved symbols are used to identify the beginning and the ending of a data packet. These symbols cannot be used in the regular data stream. Figure 1 shows the principal layout of this binary packet.

| Data field | Symbol |
|------------------------|--------|
| BINARY-START | 0x7e |
| Content of data packet | ... |

| | |
|-------------|------|
| BINARY-STOP | 0x7f |
|-------------|------|

Figure 1: Detection of beginning and ending of data packet

If the reserved symbols are used in the data packet, they have to be substituted by the following symbols:

| Original symbol | substituted in the protocol by |
|-----------------|--------------------------------|
| 0x7d | 0x7d 0x5d |
| 0x7e | 0x7d 0x5e |
| 0x7f | 0x7d 0x5f |

Figure 2: Rules for substitution for byte stuffing

Each time one of the three exclusive symbols occurs in the data packet it is replaced by two other symbols. In the worst case the whole data packet consists of exclusive symbols. In this case the length of the data packet is doubled.

The byte stuffing makes sure that the receiver can identify the BINARY-START field definitively, even if the payload data contains the reserved symbol.

Decoding the byte stuffing at the receiver side can be implemented as following:

When reading symbol 0x7d, discard this symbol and combine the next symbol via XOR-function with 0x20, which will restore the original symbol.

The byte stuffing can only be used for the binary format of the 2D protocol. It is activated per default. If you want to disable byte stuffing you have to enter in the configuration file the following line:

```
[DisableBytestuffing] 1
```

3.3. General Structure

The structure of the data packet of the 2D protocol is identified in the configuration file. For one configuration all data packets have the same length and the same structure. In the configuration file you identify the desired data fields. Figure 3 shows the structure of the data packet for the standard configuration.



| |
|----------|
| START |
| POSITION |
| END |

Figure 3: General structure of data packet with standard configuration

If required additional data fields can be activated. The sequence of the activated data fields is fixed and cannot be changed. Figure 4 shows the structure of the data packet with all activated data fields.

| Data field |
|-------------------|
| START |
| TIMESTAMP |
| POSITION |
| VELOCITY |
| ORIENTATION |
| POSITION-ERROR |
| VELOCITY-ERROR |
| ORIENTATION-ERROR |
| USER-DATA |
| SYSTEM-ERROR |
| SATELLITE-STATE |
| CRC |
| END |

Figure 4: General structure with all possible data fields

The grey data fields (START and END) cannot be deactivated. All other data fields can be enabled and disabled in the configuration file.

3.4. Data field

In this chapter all data fields are described. Except the two data fields START and END (included in each data packet) each data field can be enabled and disabled by the user.

The unit of data length is one byte.

3.4.1. START

The data field START indicates the beginning of a data packet. It contains furthermore the data length of the whole data packet. One bit mask indicates which data fields are enabled.

| Name | Length | Type | Description |
|-----------------|--------|--------------|---|
| BINARY-START | 1 | unsigned int | Exclusive Symbol 0x7e, which identify the beginning of a data packet |
| LENGTH | 2 | unsigned int | Length of the entire data packet in byte (including the start and end field) |
| SELECTED-FIELDS | 4 | bitmask | The bit mask indicates which data fields are enabled and disabled in the data packet (see chapter 5). |

Entire length of data packet: 7 Byte

3.4.2. END

The data field END indicates the end of each data packet.

| Name | Length | Type | Description |
|-------------|--------|--------------|---|
| BINARY-STOP | 1 | unsigned int | Exclusive symbol 0x7f to identify the end of a data packet. |

Entire length of data packet: 1 Byte

3.4.3. TIMESTAMP

This data field specifies the time when a position was taken. Due to the calculation time for the position this time is always in the past.

Hint: If the hardware does not have a battery-buffered RTC (real time clock) the clock is set to 1.1.1970 00:00:00 at each reboot.

| Name | Length | Type | Description |
|---------|--------|--------------|---|
| TS-SEC | 4 | unsigned int | Number of seconds since 01.01.1970 00:00:00 |
| TS-MSEC | 2 | unsigned int | Additional number of milliseconds (0..999) |

Entire length of data packet: 6 Byte

This data field can be activated by the following line in the configuration file:

```
[SendTimestamp] 1
```

3.4.4. POSITION

This data field displays the 2D position (x,y). Furthermore it shows the reliability of the position.

| Name | Length | Type | Description |
|------------|--------|--------------|--|
| POS-X | 4 | signed int | signed x-position in mm |
| POS-Y | 4 | signed int | signed y-position in mm |
| TRACKSTATE | 1 | unsigned int | reliability of position: 0,1: position is not reliable 2: position is reliable |

Entire length of data packet: 9 Byte

This data field is enabled in each data packet per default. It can be disabled via the following line in the configuration file:

```
[SendPosition] 0
```

The protocol is configured per default, that only a reliable position is output (Track state 2). If also unreliable positions should be output the following line in the configuration file has to be changed:

```
[OutputOnlyLockedTracks] 0
```

3.4.5. VELOCITY

This data field indicates the velocity in x- and y- direction.

Hint: Without knowing the orientation of the vehicle (data field ORIENTATION) it is not possible to identify if the vehicle is moving forward or backward!

| Name | Length | Type | Description |
|-------|--------|------------|--|
| VEL-X | 4 | signed int | Signed velocity in x direction in mm/s |
| VEL-Y | 4 | signed int | Signed velocity in y direction in mm/s |

Entire length of data packet: 8 Byte

This data field can be enabled in the configuration file in the following line:

```
[SendVelocity] 1
```

3.4.6. ORIENTATION

This data field indicates the orientation of the vehicle. The angle is measured counter-clockwise, beginning at the x axis.

| Name | Length | Type | Description |
|-------|--------|--------------|--|
| ANGLE | 2 | unsigned int | Orientation of vehicle in degree (0..359°) |

Entire length of data packet: 2 Byte

This data field can be enabled in the configuration file via the following entry:

[SendOrientation] 1

3.4.7. POSITION-ERROR

This data field indicates the estimated position error (EPE). The EPE is always a positive value.

| Name | Length | Type | Description |
|-----------|--------|--------------|-------------------------------------|
| POS-ERR-X | 4 | unsigned int | estimated error of x-position in mm |
| POS-ERR-Y | 4 | unsigned int | estimated error of y-position in mm |

Entire length of data packet: 8 Byte

This data field can be activated in the configuration file via the following line:

[SendPosError] 1

3.4.8. VELOCITY-ERROR

This data field indicates the estimated velocity error. The value is always positive.

| Name | Length | Type | Description |
|------------|--------|--------------|--|
| VEL-ERR-VX | 4 | unsigned int | estimated error of velocity in x-direction in mm/s |
| VEL-ERR-VY | 4 | unsigned int | estimated error of velocity in y-direction in mm/s |

Entire length of data packet: 8 Byte

This data field can be activated in the configuration file via the following line:

[SendVelError] 1

3.4.9. ORIENTATION-ERROR

This data field indicates the estimated error of orientation. The value is always positive.

| Name | Length | Type | Description |
|-----------|--------|--------------|--|
| ANGLE-ERR | 2 | unsigned int | Estimated error of orientation in degree |

Entire length of data packet: 2 Byte

This data field can be activated in the configuration file via the following line:

```
[SendOrientationError] 1
```

3.4.10. USER-DATA

This data field is used to indicate a user data packet. The meaning of the user data packet depends on the application.

| Name | Length | Type | Description |
|---------------|--------|--------------|-------------|
| USER-DATA-SET | 8 | unsigned int | User data |

Entire length of data packet: 8 Byte

This data field can be activated in the configuration file via the following line:

```
[SendUserData] 1
```

3.4.11. SYSTEM-ERROR

This data field provides information about possible errors of the system. Up to five errors can be displayed simultaneously in one data packet. An error code is sent as long as an error exists.

| Name | Length | Type | Description |
|---------------|--------|--------------|--------------------------------------|
| ERROR-CODE-1 | 1 | unsigned int | Error code of 1 st error |
| ERROR-VALUE-1 | 2 | unsigned int | Error value of 1 st error |
| ERROR-CODE-2 | 1 | unsigned int | Error code of 2 nd error |
| ERROR-VALUE-2 | 2 | unsigned int | Error value of 2 nd error |
| ERROR-CODE-3 | 1 | unsigned int | Error code of 3 rd error |
| ERROR-VALUE-3 | 2 | unsigned int | Error value of 3 rd error |
| ERROR-CODE-4 | 1 | unsigned int | Error code of 4 th error |
| ERROR-VALUE-4 | 2 | unsigned int | Error value of 4 th error |
| ERROR-CODE-5 | 1 | unsigned int | Error code of 5 th error |
| ERROR-VALUE-5 | 2 | unsigned int | Error value of 5 th error |

Entire length of data packet: 15 Byte

A detailed description of all errors is written in chapter 7. If more than 5 errors exist, the special error 0xff is sent as the ERROR-CODE-5.

This data field can be enabled by the following instruction in the configuration file:

```
[SendSystemError] 1
```

3.4.12. SATELLITE-STATE

This data field is used for satellite-based localization and holds information about the positioning quality.

| Name | Length | Type | Description |
|-----------|--------|------------|---|
| SAT-COUNT | 1 | signed int | Number of satellites tracked The single allowed <i>negative</i> value is “-1”, it means “unknown”, e.g. in case of hardware failure. |
| SAT-HDOP | 2 | signed int | 10 * horizontal dilution of precision So the integer 123 would mean a 12.3 HDOP. A HDOP value of “-1” (given as “-10” in this format) means “unknown”, e.g. in case of hardware failure. |

Entire length of data packet: 3 Byte

3.5. CRC

This data field displays the CRC (*cyclic redundancy check*) of each data packet.

| Name | Length | Type | Description |
|--------|--------|--------------|----------------------|
| CRC-16 | 2 | unsigned int | CRC value of message |

Entire length of data field: 2 Byte

A detailed description of CRCs can be found in chapter 6 as well as source code for the CRC.

This data field can be enabled by the following line in the configuration file:

```
[SendCRC] 1
```

4. ASCII Format of the Protocol

For the **ASCII format** of the 2D protocol the entire data packet is transmitted as **ASCII-code** by letters, numbers and some special characters. This means the ASCII code is human-readable. Due to the fewer encoding characters the data is transmitted not as compact as for the binary protocol, so the amount of data increases.

4.1. Data Types

Each data packet consists of characters/ character chains, numbers (optional with decimal point und algebraic sign) and underline character. Each data packet consist of one row, terminated by the special character LF (*line feed*\n, ASCII-Code 0x0a).

All numbers have a prescribed **fixed quantity of characters**. If less numbers are required for the value, the value has to be filled by zeros (prefixed / postfixed).

Integer values exist as well as floating point numbers. The numbers in each data field are described as follows:

- + Sign, always (+ or -)
- Sign, only if value is negative (then -)
- # Single decimal number / character
- . Decimal point (only for floating point numbers)

Example:

The floating-point number to encode is: 12.34

description: +###.####

→ coded number: +012.3400

4.2. General Structure

A **configuration file** is the basis of the 2D protocol. Each data packet consists of one text row – the end of each data packet is terminated by the ASCII-STOP sign \n.

Once configured the configuration file each data packet has the same length and the same structure. The configuration file specifies the data fields. Figure 5 shows the structure of the data field for the standard configuration:

| Data field |
|------------|
| START |
| POSITION |
| END |

Figure 5: General structure of data field for the standard configuration

If desired additional data fields can be activated. The sequence of data field is hereby fix and cannot be changed. Figure 6 the structure of a data packet with all possible data fields in one data packet.

| Data field |
|-------------------|
| START |
| TIMESTAMP |
| POSITION |
| VELOCITY |
| ORIENTATION |
| POSITION-ERROR |
| VELOCITY-ERROR |
| ORIENTATION-ERROR |
| USER-DATA |
| SYSTEM-ERROR |
| SATELLITE-STATE |
| CRC |
| END |

Figure 6: General structure of data packet with all possible data fields

The grey data fields (START and END) cannot be deactivated. All other data fields can be enabled and disabled in the configuration file.

4.3. Data fields

In this chapter all data fields are described. Except the two data fields START and END (included in each data packet) all data field can be enabled and disabled by the user.

The unit of data length is one byte.

4.3.1. START

The data field START indicates the beginning of a data packet. It contains furthermore the data length of the whole data packet. A bit field indicates which data fields are enabled.

| Name | Length | Description |
|-----------------|--------|--|
| ASCII-START | 1 | ASCII sign A (0x41) |
| LENGTH | 3 | Length of the entire data packet in byte (including the start and end field) Character Coding: ### |
| SELECTED-FIELDS | 8 | The bit mask indicates which data fields are enabled and disabled in the data packet. (see chapter 5). Character coding: ##### The bit field is coded hexadecimal! |
| UNDERLINE | 1 | Underline character (ASCII Code 0x5F) |

Entire length of data field: 13 Byte

4.3.2. END

The data field END indicates the end of each data packet. It consists of the single symbol \n.

| Name | Length | Description |
|------------|--------|----------------------|
| ASCII-STOP | 1 | ASCII sign \n (0x0a) |

Entire length of data packet: 1 Byte

4.3.3. TIMESTAMP

This data field specifies the time when a position was taken. Due to the calculation time for the position this time is always in the past.

Hint: On hardware platforms without a battery-buffered RTC (real time clock) the clock is set to 1.1.1970 00:00:00 at each reboot.

| Name | Length | Description |
|-----------|--------|--|
| TIME | 4 | ASCII character chain time |
| TS-SEC | 10 | Number of seconds since 01.01.1970 00:00:00 Character coding: ##### |
| UNDERLINE | 1 | Underline character (ASCII Code 0x5F) |
| TS-MSEC | 3 | Additional number of milliseconds (0..999) Character coding: ### |
| UNDERLINE | 1 | Underline character (ASCII Code 0x5F) |

Entire length of data packet: 19 Byte

This data field can be activated by the following line in the configuration file:

```
[SendTimestamp] 1
```

4.3.4. POSITION

This data field is enabled in each data packet per default. It can be disabled via the following line in the configuration file:

This data field displays the 2D position (x,y). Furthermore it displays the reliability of the position.

| Name | Length | Description |
|------------|--------|--|
| X | 1 | ASCII character x |
| POS-X | 10 | Signed x-Position in meters Character coding: +#####.### |
| UNDERLINE | 1 | Underline character (ASCII Code 0x5F) |
| Y | 1 | ASCII sign y |
| POS-Y | 10 | Signed y-Position in meters Character coding: +#####.### |
| UNDERLINE | 1 | Underline character (ASCII Code 0x5F) |
| TRACKSTATE | 1 | Number for the reliability of the position: 0,1: position is not reliable 2: Position is reliable Character coding: # |
| UNDERLINE | 1 | Underline character (ASCII Code 0x5F) |

Entire length of data packet: 26 Byte

This data field is enabled in the standard configuration. It can be deactivated by the following entry in the configuration file:

```
[SendPosition] 0
```

The protocol is configured per default, that only a reliable position is output (Track state 2). If unreliable positions should be output as well, the following line in the configuration file has to be changed:

```
[OutputOnlyLockedTracks] 0
```

4.3.5. VELOCITY

This data field indicates the velocity in x- and y- direction.

Hint: Without knowing the orientation of the vehicle (data field ORIENTATION) it is not possible to identify if the vehicle is moving forward or backward!

| Name | Length | Description |
|-----------|--------|--|
| VX | 2 | ASCII character chain vx |
| VEL-X | 6 | Signed velocity in x-direction in m/s Character coding: ###.## |
| UNDERLINE | 1 | Underline character (ASCII Code 0x5F) |
| VY | 2 | ASCII character chain vy |
| VEL-Y | 6 | Signed velocity in y-direction in m/s Character coding: ###.## |
| UNDERLINE | 1 | Underline character (ASCII Code 0x5F) |

Entire length of data packet: 18 Byte

This data field can be enabled in the configuration file in the following line:

```
[SendVelocity] 1
```

4.3.6. ORIENTATION

This data field indicates the orientation of the vehicle. The angle is measured counter-clockwise, beginning at the x axis.

| Name | Length | Description |
|-------|--------|--|
| O | 1 | ASCII character o |
| ANGLE | 3 | Orientation of vehicle in degree (0..359°) Character coding: ### |

| | | |
|-----------|---|---------------------------------------|
| UNDERLINE | 1 | Underline character (ASCII Code 0x5F) |
|-----------|---|---------------------------------------|

Entire length of data packet: 5 Byte

This data field can be enabled in the configuration file via the following entry:

```
[SendOrientation] 1
```

4.3.7. POSITION-ERROR

This data field indicates the estimated position error (EPE). The EPE is always a positive value.

| Name | Length | Description |
|-----------|--------|---|
| EX | 2 | ASCII character chain ex |
| POS-ERR-X | 5 | Estimated error of x-position in m Number coding: ##.## |
| UNDERLINE | 1 | Underline character (ASCII Code 0x5F) |
| EY | 2 | ASCII character chain ey |
| POS-ERR-Y | 5 | Estimated error of y-position in m Number coding: ##.## |
| UNDERLINE | 1 | Underline character (ASCII Code 0x5F) |

Entire length of data packet: 16 Byte

This data field can be activated in the configuration file via the following line:

```
[SendPosError] 1
```

4.3.8. VELOCITY-ERROR

This data field indicates the estimated velocity error. The value is always positive.

| Name | Length | Description |
|-----------|--------|--|
| EVX | 3 | ASCII character chain evx |
| POS-ERR-X | 5 | Estimated error of velocity in x-direction in m/s Character coding: ### |
| UNDERLINE | 1 | Underline character (ASCII Code 0x5F) |
| EVY | 3 | ASCII character chain evy |
| POS-ERR-Y | 5 | Estimated error of velocity in y-direction in mm/s Character coding: ### |
| UNDERLINE | 1 | Underline character (ASCII Code 0x5F) |

Entire length of data packet: 18 Byte

This data field can be activated in the configuration file via the following line:

```
[SendVelError] 1
```

4.3.9. ORIENTATION-ERROR

This data field indicates the estimated error of orientation. The value is always positive.

| Name | Length | Description |
|-----------|--------|--|
| EO | 2 | ASCII character chain eo |
| ANGLE-ERR | 3 | Estimated error of orientation in degree Character coding: ### |
| UNDERLINE | 1 | Underline character (ASCII Code 0x5F) |

Entire length of data packet: 6 Byte

This data field can be activated in the configuration file via the following line:

[SendOrientationError] 1

4.3.10. USER-DATA

This data field is used to indicate a user data packet. The meaning of the user data packet depends on the application.

| Name | Length | Description |
|---------------|--------|---|
| USER | 4 | ASCII character chain user |
| USER-DATA-SET | 16 | User data Character coding: ##### User data is coded hexadecimal! |
| UNDERLINE | 1 | Underline character (ASCII Code 0x5F) |

Entire length of data packet: 21 Byte

This data field can be activated in the configuration file via the following line:

[SendUserData] 1

4.3.11. SYSTEM-ERROR

This data field provides information about possible errors of the system. Up to five errors can be displayed simultaneously in one data packet. An error code is sent as long as an error exists.

| Name | Length | Description |
|--------------|--------|-------------------------------------|
| ERR | 3 | ASCII character chain err |
| ERROR-CODE-1 | 2 | Error code of 1 st error |

| | | |
|---------------|---|---|
| | | Character coding: ## Value is coded hexadecimal! |
| ERROR-VALUE-1 | 4 | Error value of 1 st error Character coding: #### Value is coded hexadecimal! |
| UNDERLINE | 1 | Underline character (ASCII Code 0x5F) |
| ERROR-CODE-2 | 2 | Error code of 2 nd error |
| ERROR-VALUE-2 | 4 | Error value of 2 nd error |
| UNDERLINE | 1 | Underline character (ASCII Code 0x5F) |
| ERROR-CODE-3 | 2 | Error code of 3 rd error |
| ERROR-VALUE-3 | 4 | Error value of 3 rd error |
| UNDERLINE | 1 | Underline character (ASCII Code 0x5F) |
| ERROR-CODE-4 | 2 | Error code of 4 th error |
| ERROR-VALUE-4 | 4 | Error value of 4 th error |
| UNDERLINE | 1 | Underline character (ASCII Code 0x5F) |
| ERROR-CODE-5 | 2 | Error code of 5 th error |
| ERROR-VALUE-5 | 4 | Error value of 5 th error |
| UNDERLINE | 1 | Underline character (ASCII Code 0x5F) |

Entire length of data packet: 38 Byte

A detailed description of all errors is written in chapter 7. If there are more than five errors the special error 0xff is sent as the ERROR-CODE-5.

This data field can be enabled by the following instruction in the configuration file:

```
[SendSystemError] 1
```

4.3.12. SATELLITE-STATE

This data field is used for satellite-based localization and holds information about the positioning quality.

| Name | Length | Description |
|-----------|--------|--|
| SAT | 3 | ASCII character chain sat |
| SAT-COUNT | 2 | Number of satellites tracked Character coding: ## (or "-1" for "unknown") |
| UNDERLINE | 1 | Underline character (ASCII Code 0x5F) |
| SAT-HDOP | 4 | Horizontal dilution of precision Character coding: ##.# (or "-1.0" for "unknown") |
| UNDERLINE | 1 | Underline character (ASCII Code 0x5F) |

Entire length of data packet: 11 Byte

4.3.13. CRC

This data field displays the CRC (*cyclic redundancy check*) of each data packet.

| Name | Length | Description |
|--------|--------|---|
| CRC | 3 | ASCII character chain crc |
| CRC-16 | 4 | CRC-value of message Character coding: #### Value is coded hexadecimal! |

Entire length of data field: 7 Byte

A detailed description of CRCs as well as source code for the CRC can be found in chapter 6.

This data field can be enabled by the following line in the configuration file:

[SendCRC] 1

5. Bit Mask SELECTED-FIELDS

The bit mask SELECTED-FIELDS is part of the data field START. It indicates which bit fields are activated. Once the bit mask is configured, each data packet has the same length and the same structure. Therefore the value of the bit mask SELECTED-FIELDS is constant. By evaluation of the bit mask at the receiver side it is possible to check for plausibility of the configuration file.

Each data field is presented as a single bit in the bit mask. If the data field is activated the related bit is set.

Figure 7 shows the allocation of the single bits and data fields. Bit 31 is the most significant bit and is found in the ASCII format of the protocol on the left most side.

Important: The order of bits in this bit mask does not correspond to the order of data fields in a data packet. The order of the data fields is described in chapter 3.3 (binary format) or chapter 4.2 (ASCII format).

| Bit | Data field | Bit | Data field |
|-----|------------|-----|-------------------|
| 31 | - | 15 | - |
| 30 | - | 14 | - |
| 29 | - | 13 | - |
| 28 | - | 12 | - |
| 27 | - | 11 | - |
| 26 | - | 10 | SATELLITE-STATE |
| 25 | - | 9 | CRC |
| 24 | - | 8 | SYSTEM-ERROR |
| 23 | - | 7 | USER-DATA |
| 22 | - | 6 | ORIENTATION-ERROR |
| 21 | - | 5 | VELOCITY-ERROR |
| 20 | - | 4 | POSITION-ERROR |
| 19 | - | 3 | ORIENTATION |
| 18 | - | 2 | VELOCITY |
| 17 | - | 1 | POSITION |
| 16 | - | 0 | TIMESTAMP |

Figure 7: Allocation Bitmask – Data fields

6. CRC Calculation

To detect errors during data transmission, the data field CRC can be activated. For CRC the CRC-16-IBM is used with the polynomial $x^{16}+x^{15}+x^2+1$. The CRC is applied to all previous data fields of the data packet except the data field START.

Example source code for CRC calculation in C:

```
// Holds a table to calculate crc16 values
static Uint16 crc_table[256];

// Initializes the CRC table
// MUST BE RUN before first crc calculation
Void InitCRCTable( void )
{
    int i, j;

    Uint16 k;

    for (i = 0; i < 256; i++)
        crc_table[i] = i;

    for (i = 0; i < 256; i++)
    {
        k = 0xC0C0;
        for (j = 1; j < 256; j <= 1)
        {
            if (i & j)
                crc_table[i] ^= k;
            k = ((k & 0x7FFF) << 1) ^ 0x4003;
        }
    }
}

// Adds calculation for an 8 bit value to crc.
// Initially crc should be zero.
Uint16 CalcCRC8(Uint16 crc, Uint8 value)
{
    crc = (crc >> 8) ^ crc_table[(crc & 0xFF) ^ value];
    return crc;
}
```

7. Error Codes

If the system identifies self-contained an error (self-diagnosis), a system error is sent. The error is sent as long as the error is not repaired. I.e. if a defect transponder is recognized as defect then the appropriate error code is sent until the transponder is able to make a correct measurement.

A special case is the error code 0xff. This error is sent if more than 5 errors appear at the same time. Therefore not all error codes can be transmitted. This special error code is only sent for the last error code of the system failure (ERROR-CODE-5).

An error consists always of the error code. Optional it can feature an error value which consists of additional information to the error code.

Overview of error codes:

| Code | Meaning |
|------|----------------------------|
| 0x01 | No connection to LPR-B DSP |
| 0x02 | Transponder defect |
| | |
| 0xff | Further errors active |

Detailed description of error codes and error values:

| Code | Description |
|------|--|
| 0x01 | In the last 3 seconds there was no receiving from the LPR-B DSP possible. Possible reasons: - physical connection to the DSP is interrupted - DSP is configured with failures - DSP is defect |
| 0x02 | A transponder was not detected in a LPR 2D system over a longer period. The failure value indicates the LPR address of the defect transponder. A transponder is first detected as a defect transponder, if its measurement is not received at different positions. Sometimes it is possible that the transponder is only hidden or his radius is out of range. |
| | |
| 0xff | At the moment more than 5 errors are active. The error value indicates the |

| | |
|--|-----------------------------------|
| | number of activated errors (> 5). |
|--|-----------------------------------|