REDRAIL

Dokumentation des redRail Übertragungsprotokolls für den Integrator

Version 1.0
Mai 2015

SLR Engineering GmbH
Gartengasse 19
8010 Graz
Austria

G. Loibner
O. Sidla

Date	Type of modification	Author	Version
10.04.2015	Basisversion Dokument	os/gl	0.1
11.06.2015	Text Review, Update Logos Ergänzungen	os/gl	1.0

EINLEITUNG....4
SYSTEMÜBERBLICK....4
DER DATENTRANSFER VON DER KAMERA....5
COMMAND-MESSAGE....7
GET_APPLICATION_PARAM, SET_APPLICATION_PARAM....8
BEISPIEL....9
LIST_APPLICATION_PARAMS....10
GET_APPLICATION_LIST....11
GET_APPLICATION_NAME....12
GET_ACTIVE_APPLICATIONS....13
SEND_FILE....14
RECEIVE_FILE....15
DAS REDRAIL TRANSFER API....16
ABHÄNGIGKEITEN....16
DAS FOLGENDE LISTING ENTHÄLT DIE DOKUMENTATION FÜR DAS TRANSFER API:....16
REDRAILEVENT.H....18
PROGRESSOBSERVER.H....20
LIBREDRAILCLIENTDEF.H....21
DIE DEKODIERUNG VON BILDERN AUF DER CLIENT SEITE....22
REFERENZEN....23

Einleitung

Dieses Dokument beschreibt das Übertragungsprotokoll von SLR Engineering, welches zur Kommunikation zwischen der SLR Engineering Smart-Kamera und einer Anwendung am PC verwendet wird.

Das Protokoll erlaubt die Konfiguration und Wartung der Kamera, als auch das Übertragen von Bild- und Detektionsdaten einer *redRail* Kamera.

Die Kombination aus Hardware (der Smart-Kamera) und Übertragungsprotokoll erlaubt das verschlüsselte, sichere Übertragen von Bilddaten über TCP. Das Protokoll ist so gestaltet, das Verbindungsabbrüche über WLAN oder UMTS erkannt werden und Daten nicht verloren gehen können. Weiters ist das Protokoll an eine potentielle langsame Datenübertragung über GPRS/UMTS angepasst.

Die nötigen Schlüssel zum Dekodieren werden sicher auf einem USB Crypto Key auf der Client Seite gespeichert und bei Bedarf von der SLR Engineering API Software ausgelesen.

Systemüberblick

Das SLR Engineering redRail System ist als Client-Server Architektur aufgebaut. Die *redRail* Kamera übernimmt hierbei die Rolle des Servers, der eine oder mehrere PC Anwendungen bedienen kann.

Daten zwischen SLR Kamera und der PC Anwendung werden über das SLR Protokoll ausgetauscht, wobei üblicherweise eine UMTS Datenverbindung zwischen der SLR Kamera und dem PC besteht.

Die Anwendung am PC spricht über das SLR Protokoll, welches in den folgenden Abschnitten erklärt wird, über die Datenverbindung mit der Kamera.

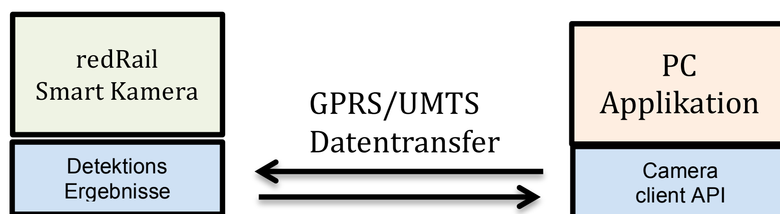


Abbildung 1: Der grundsätzliche Systemaufbau einet redRail Anlage.

Der Datentransfer von der Kamera

Der Datentransfer zwischen Kamera und Endsystem ist so ausgelegt, dass auch unzuverlässige und langsamen Verbindungen für die robuste Übertragung von Detektionsdaten eingesetzt werden können.

Das verwendete Protokoll sichert alle Transfers über Bestätigungen ab, weiters sind alle versendeten Daten von der Kamera sicher verschlüsselt.

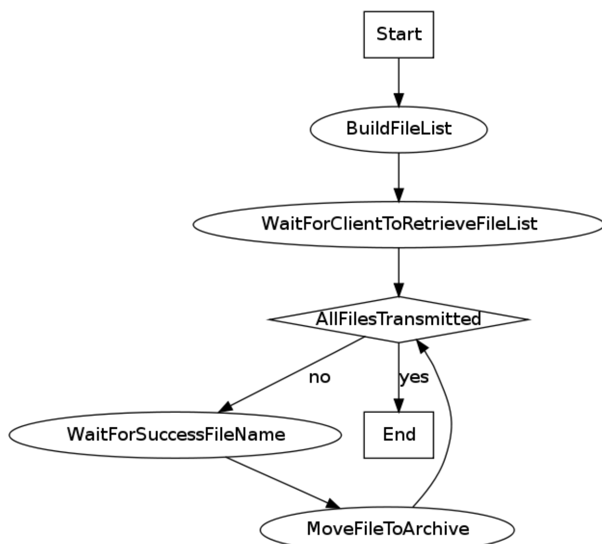


Abbildung 3: a) Ablauf am Server

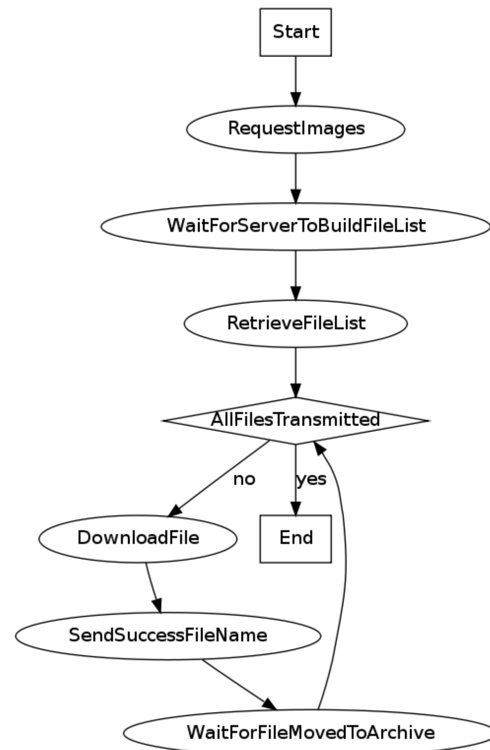
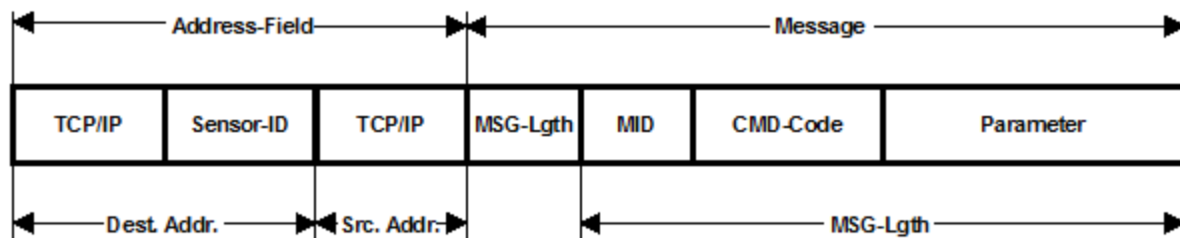


Abbildung 2: b) Ablauf am Client

Das SmartVis Communication Protocol - SVCP

Das **S**mart**V**is **C**ommunication **P**rotocol *SVCP* basiert auf dem Verschicken von Botschaften, sogenannten SVCP-Messages, die wie folgt aufgebaut sind:



Prinzipiell besteht eine *SVCP-Message* aus einem

- *Adress-Feld* und einem
- *Message-Feld*

Das *Adress-Feld* besteht wieder aus folgenden Feldern:

<table>
<tr><th colspan="2">Adressfeld</th><th>Beschreibung</th></tr>
<tr><td rowspan="2">DestAdr</td><td>TCP/IP</td><td rowspan="2">Adresse des Empfängers, sie besteht aus der TCP/IP Adresse und einer ID für die Adressierung eines Server Moduls.</td></tr>
<tr><td>SensorId</td></tr>
<tr><td>SrcAdr</td><td>TCP/IP</td><td>Adresse des Absenders</td></tr>
</table>

Tabelle 1: Adressteil einer SVCP-Message

Das *Message-Feld* einer SVCP-Message besteht aus den folgenden vier Teilfeldern:

MSG-Lgth	Kodiert die Messagelänge. Die MSG-Lgth beschreibt die Länge des Parameter Feldes.
MID	Über die **M**essage**ID** wird eine Message einem bestimmten Request zugeordnet. Der Absender einer Message ist für die Generierung der MID verantwortlich.
CMD-Code	Der CMD-Code beschreibt die auszuführende Operation
Parameter	Im Parameterfeld werden die einem CMD zugeordneten Parameter übertragen.

Tabelle 2: Struktur des Message-Feldes einer SVCP-Message

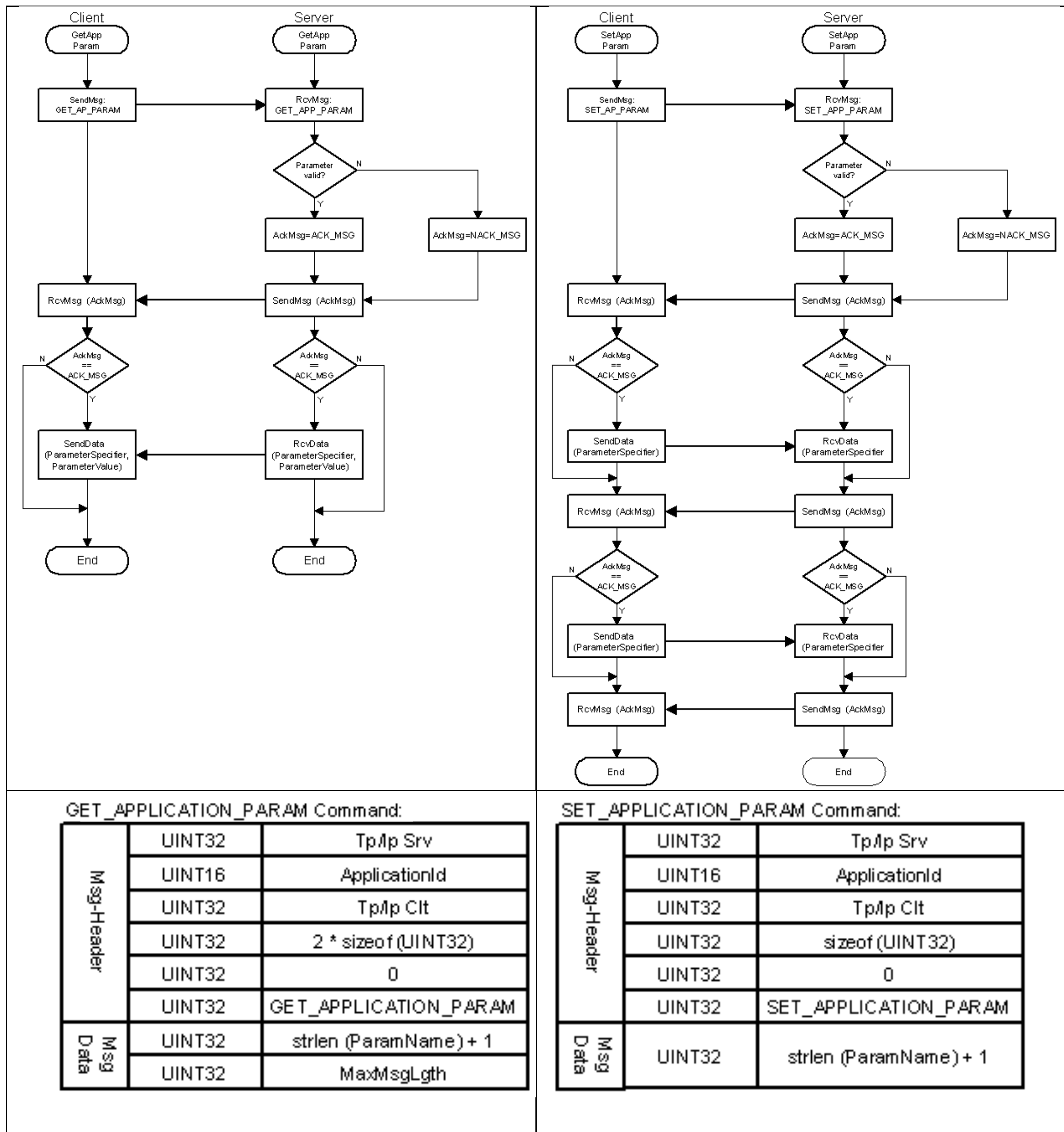
Command-Message

Mit Command-Messages werden die adressierten Empfänger zur Ausführung bestimmter Operationen veranlasst. Die folgende Tabelle gibt eine Übersicht über die verfügbaren CMD.Messages:

Command-Message	**Beschreibung**
Allgemeine Operationen	
GET_APPLICATION_PARAM	Abrufen eines Applikationsparameters
SET_APPLICATION_PARAM	Setzen eines Applikationsparameters
LIST_APPLICATION_PARAMS	Die Namen aller verfügbaren Parameter einer Applikation werden abgerufen
GET_APPLICATION_LIST	Eine Liste der verfügbaren Applikationen wird gelesen.
GET_APPLICATION_NAME	Der Name einer Applikation wird gelesen.
GET_ACTIVE_APPLICATIONS	Eine Liste der auf einem Knoten aktiven Applikationen wird gelesen.
SEND_FILE	Übertragung einer Datei zum Server.
RECEIVE_FILE	Abrufen einer Datei vom Server.

GET_APPLICATION_PARAM, SET_APPLICATION_PARAM

Abrufen bzw. Setzen von Applikations-Parametern wird mit diesen Parametern gesteuert. Der Kommunikationsablauf für das *GET_APPLICATION* bzw. *SET_APPLICATION* Kommando sieht wie folgt aus:



GET_APPLICATION_PARAM Command:

Msg-Header	UINT32	Tp/Ip Srv
	UINT16	ApplicationId
	UINT32	Tp/Ip Clt
	UINT32	2 * sizeof (UINT32)
	UINT32	0
	UINT32	GET_APPLICATION_PARAM
Msg Data	UINT32	strlen (ParamName) + 1
	UINT32	MaxMsgLgth

SET_APPLICATION_PARAM Command:

Msg-Header	UINT32	Tp/Ip Srv
	UINT16	ApplicationId
	UINT32	Tp/Ip Clt
	UINT32	sizeof (UINT32)
	UINT32	0
	UINT32	SET_APPLICATION_PARAM
Msg Data	UINT32	strlen (ParamName) + 1

Abbildung 4: Die Überragung von Steuerparametern

Beispiel

Nachfolgend findet sich ein Beispiel für den Befehl GET_APPLICATION_PARAM:

Send MSG:

Send:

Destination	ApplicationID	Source	Msg Length	MessageID	CMD Code
c0a80de9	00000000	aacfe0b0	0000000c	00000000	0000000d

Receive:

Ack
0000fffc

Send:

Data
0000001f 00002000 000003e8

Receive MSG:

Receive:

Destination	ApplicationID	Source	Msg Length	MessageID	CMD Code
aacfe0b0	00000000	c0a80de9	00000004	00000000	0000fffc

Send**:**

Ack
0000fffc

Receive:

Value
00000000

Send Data:

Parameter Specifier
72657175 65737449 6e695061 72616d65 74657252 6573756c 742e4348 415200

Receive MSG:

Receive:

Destination	ApplicationID	Source	Msg Length	MessageID	CMD Code
aacfe0b0	00000000	c0a80de9	00000004	00000000	0000fffc

Send**:**

Ack
0000fffc

Receive:

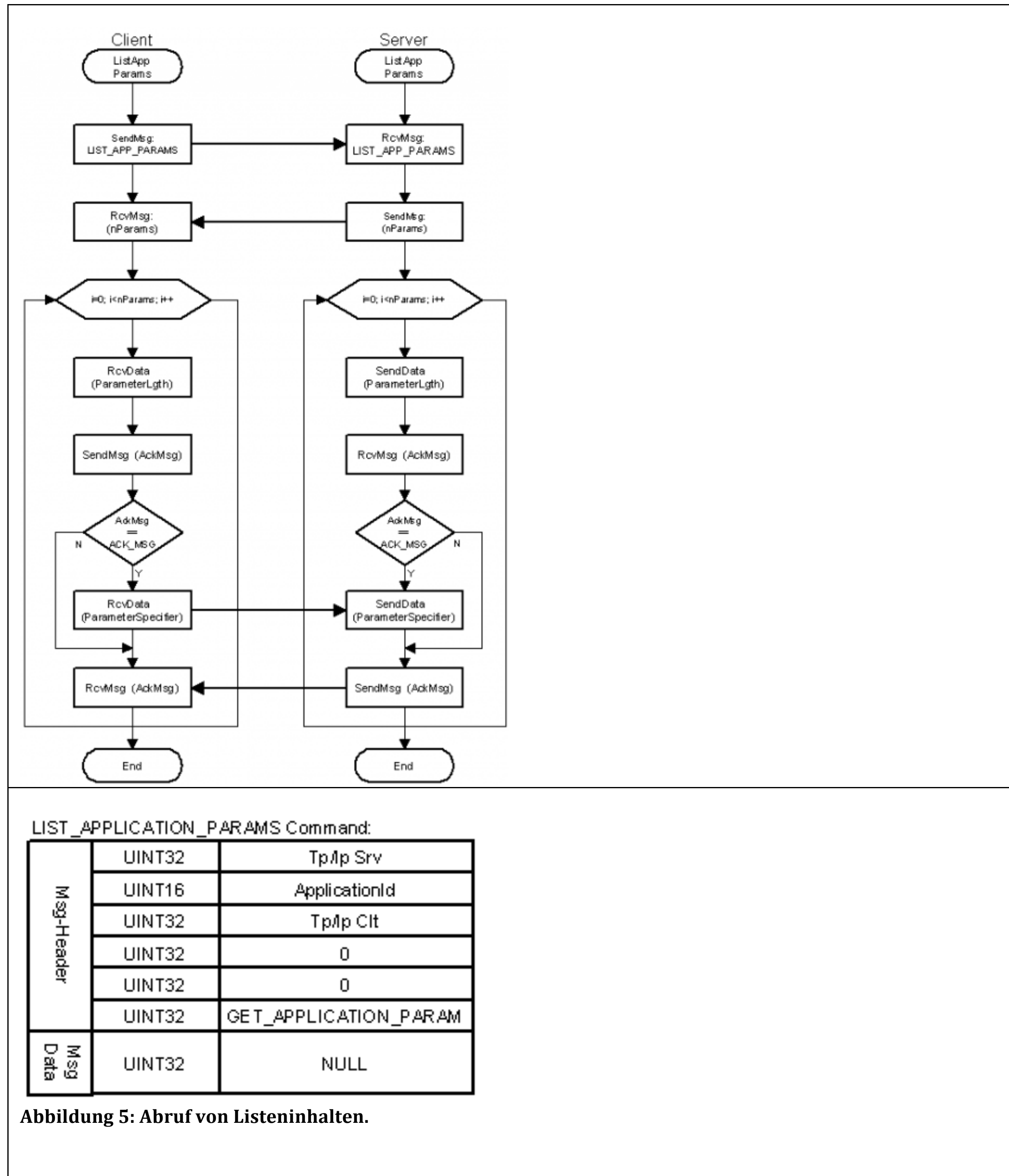
Value
0000fffc

Receive Data:

Value
1

LIST_APPLICATION_PARAMS

Dieser parameter erlaubt das Abrufen der Liste aller verfügbaren Parameter einer Applikation.



LIST_APPLICATION_PARAMS Command:

Msg-Header	UINT32	Tp/Ip Srv
	UINT16	ApplicationId
	UINT32	Tp/Ip Clt
	UINT32	0
	UINT32	0
	UINT32	GET_APPLICATION_PARAM
Msg Data	UINT32	NULL

Abbildung 5: Abruf von Listeninhalten.

GET_APPLICATION_LIST

Eine Liste der auf dem Server verfügbaren Applikationen wird ausgelesen.

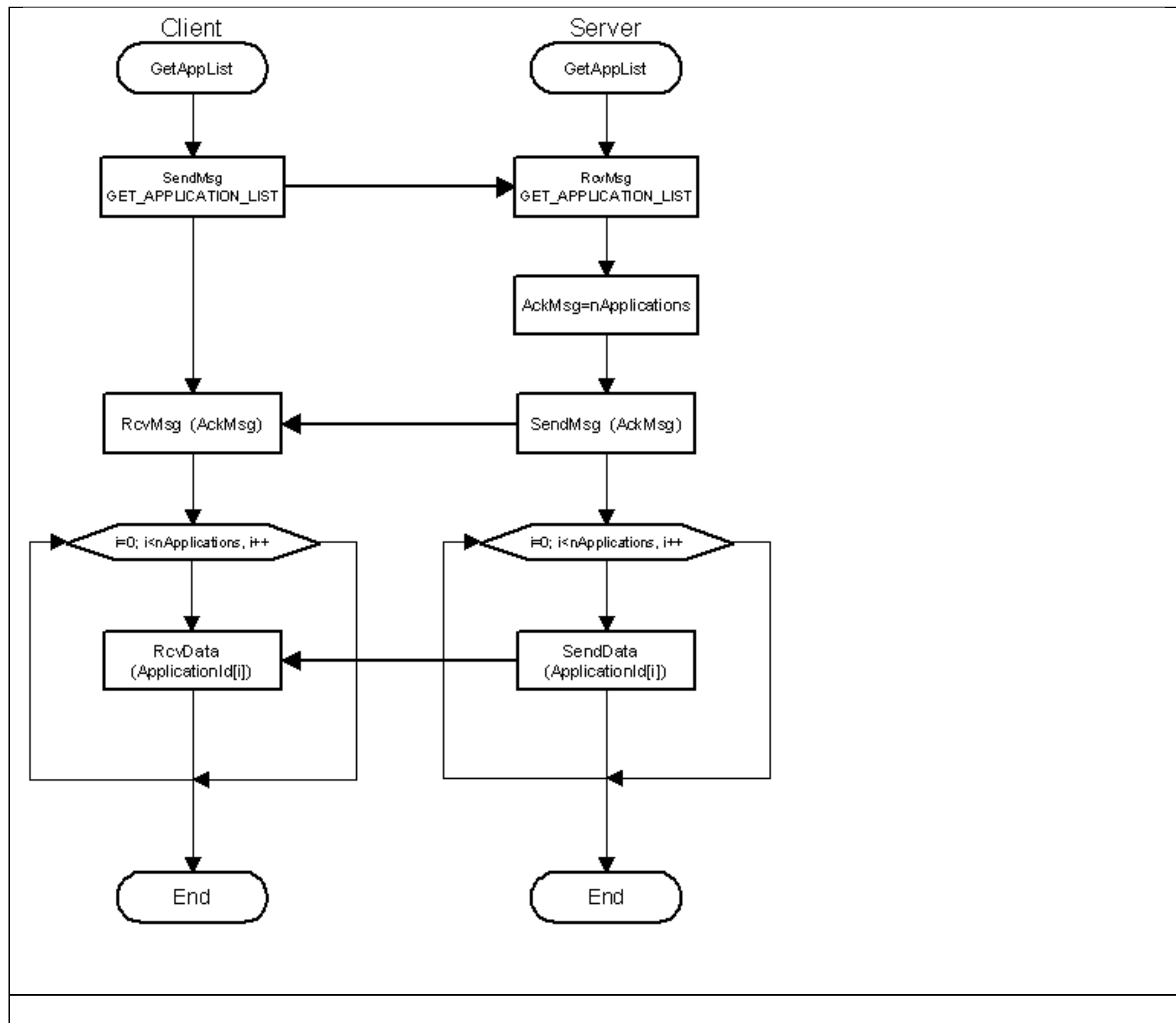


Abbildung 6: Einlesen von Listen am Server.

GET_APPLICATION_NAME

Der Name einer Applikation wird gelesen.

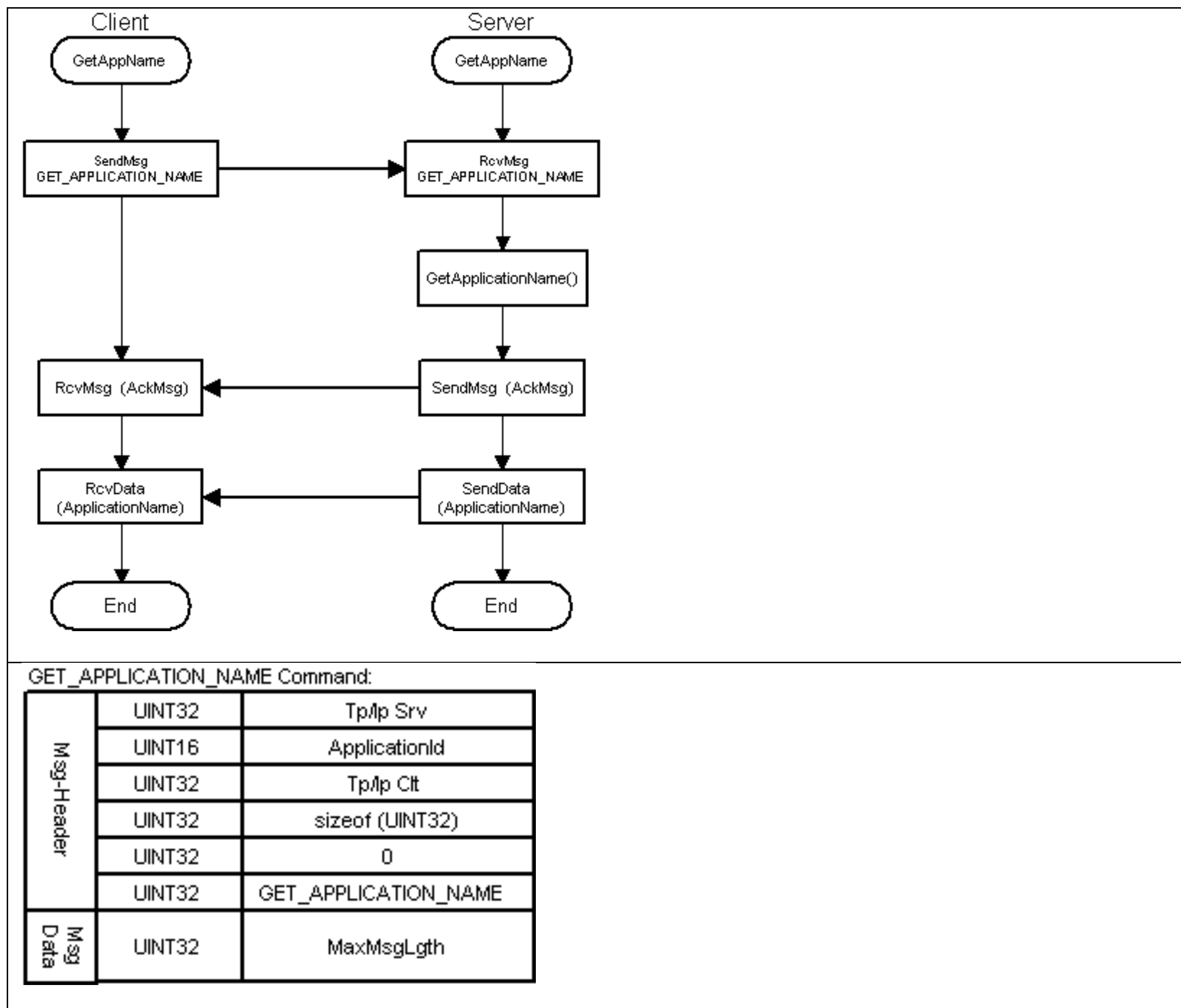


Abbildung 7: Das Lesen eines Applikation-Namens.

GET_ACTIVE_APPLICATIONS

Eine Liste der auf dem Server aktiven Applikationen wird ausgelesen.

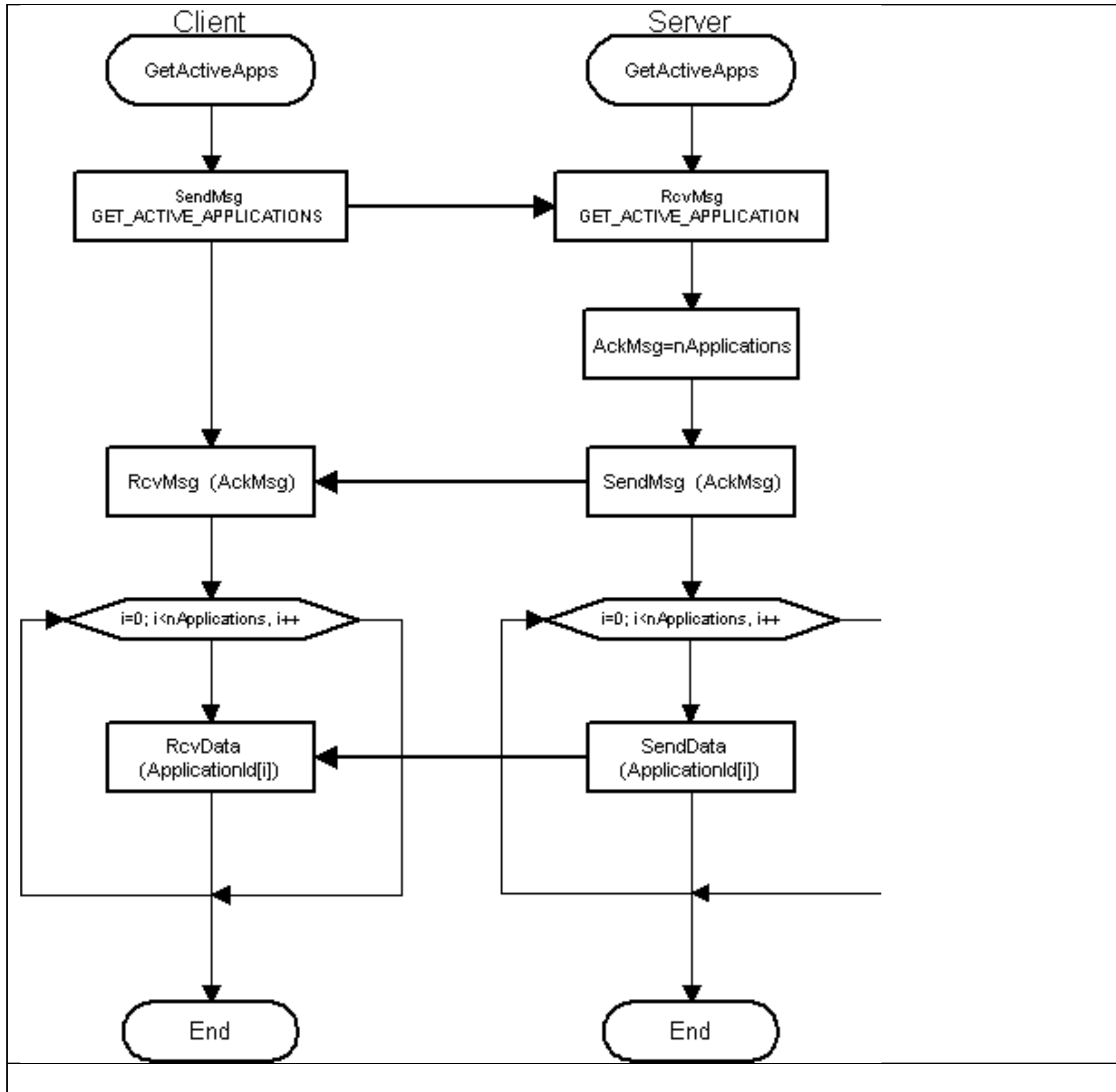


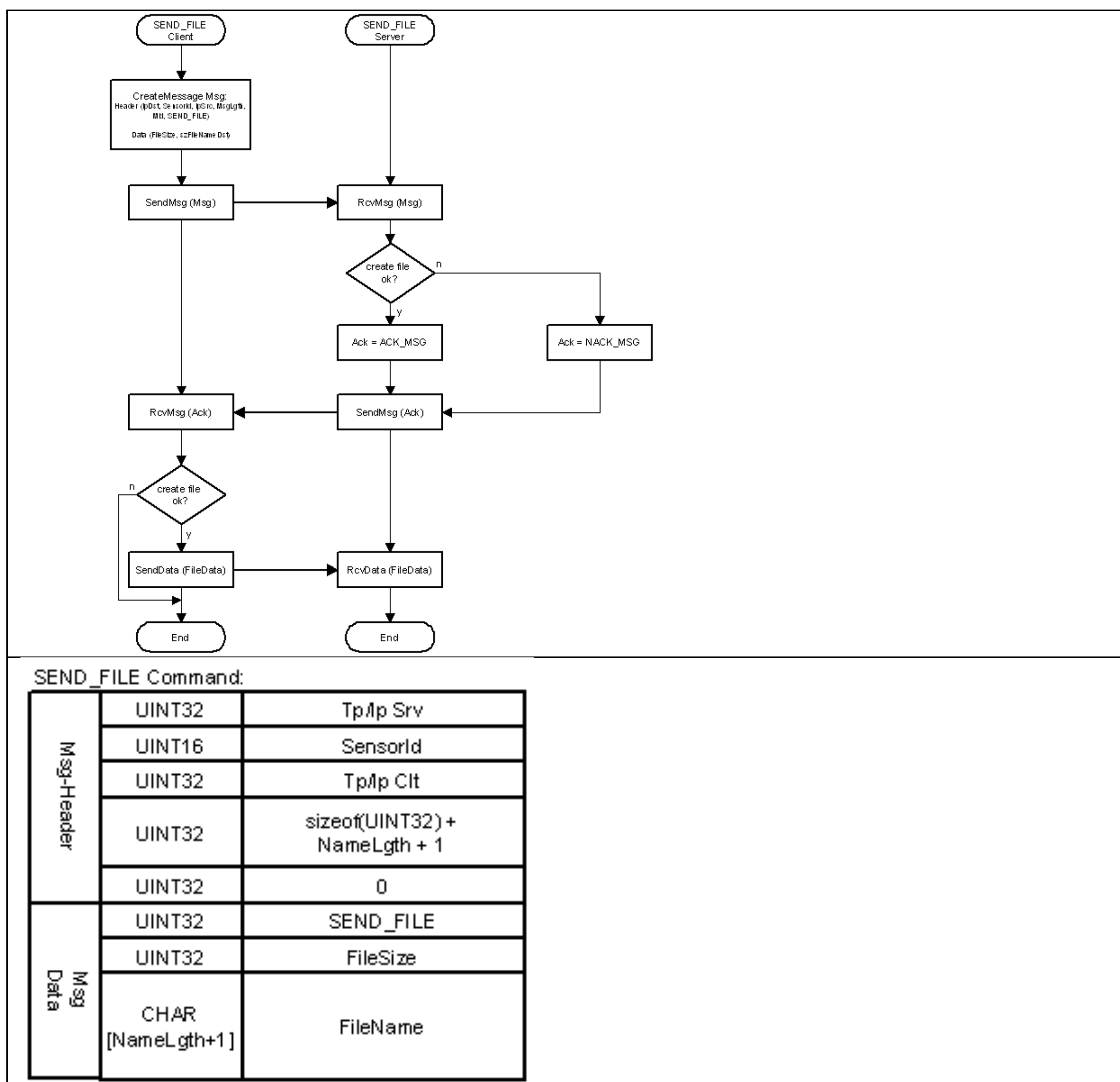
Abbildung 8: Auslesen aller aktiven Applikationen am Server.

SEND_FILE

Übertragung einer Datei zum Server wird mit diesem Parameter gesteuert.

Der Client schickt zunächst eine Message mit der Dateigröße (in Byte) und dem Filenamen an den Server. Der Server versucht daraufhin, die spezifizierte Datei anzulegen.

Falls die Datei erfolgreich angelegt werden konnte, antwortet der Server mit einer ACK_MSG, andernfalls mit einer NACK_MSG.
Im Falle einer ACK_MSG liest der Client die Inputdatei in einen Puffer und überträgt diesen Puffer an den Server. Dieser liest die empfangenen Daten ebenfalls in einen Puffer und speichert sie anschließend auf der angegebenen Output Datei ab.



Msg-Header	UINT32	Tp/Ip Srv
	UINT16	SensorId
	UINT32	Tp/Ip Clt
	UINT32	sizeof(UINT32) + NameLgth + 1
	UINT32	0
Msg Data	UINT32	SEND_FILE
	UINT32	FileSize
	CHAR [NameLgth+1]	FileName

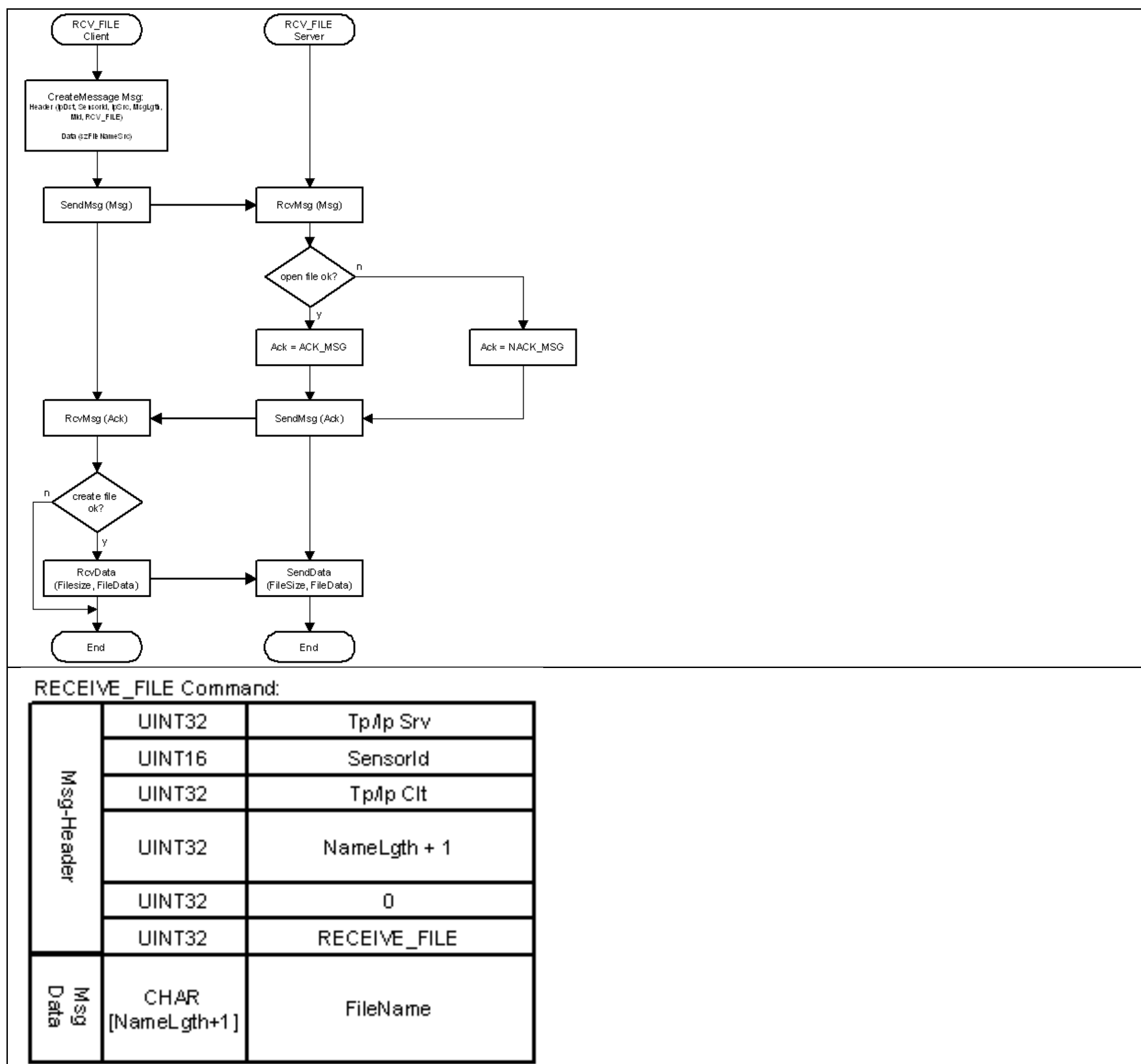
Abbildung 9: Das Überragen einer Datei zum Server.

RECEIVE_FILE

Empfanden einer Datei von einem Modul.

Der Client versucht zunächst die Output-Datei zu öffnen. Falls die Output-Datei angelegt werden kann, sendet der Client den Namen der Input-Datei ein der Server. Der Server versucht daraufhin, die Inputdatei zu öffnen. Falls diese Operation erfolgreich ausgeführt werden konnte, wird an den Client eine ACK_MSG übertragen andernfalls antwortet der Server mit einer NACK_MSG.

Im Falle, dass die Inputdatei erfolgreich geöffnet werden konnte, sendet der Server zunächst die Filegröße (in Byte) und anschließend die Filedaten. Der Client lies die Filedaten in einen Puffer und schreibt diese Daten anschließend auf die spezifizierte Output-Datei.



RECEIVE_FILE Command:

	Typ	Inhalt
Msg-Header	UINT32	Tp/Ip Srv
	UINT16	SensorId
	UINT32	Tp/Ip Clt
	UINT32	NameLgth + 1
	UINT32	0
	UINT32	RECEIVE_FILE
Msg Data	CHAR [NameLgth+1]	FileName

Abbildung 10: Das Empfangen einer Datei von der Kamera.

Das redRail Transfer API

Grundsätzlich kann die Umsetzung des Protokolls in jeder Entwicklungsumgebung mit dem in den vorigen Abschnitten beschriebenem Protokoll durchgeführt werden.

SLR Engineering bietet jedoch auch eine Implementierung des Protokolls als Windows als Bibliothek für Windows und Linux an, die in eigenen Anwendungen eingesetzt werden kann.

Abhängigkeiten

Folgende Bibliotheken werden benötigt, um die Transfer API zu nutzen:

- svlib (svcore) 2.0.0 (SLR Engineering)
- opencv (opencv_core) 2.4.9
- boost (boost_system, boost_date_time) 1.55.0

Das folgende Listing enthält die Dokumentation für das Transfer API:

(redrailclient.h)

```
#include "libredrailclientdef.h"
#include <string>
#include <vector>
#include <boost/date_time/posix_time/posix_time.hpp>
#include "progressobserver.h"
#include "redrailevent.h"
#include <svcore/include/svdef.h>
#include <opencv2/core/core.hpp>

using namespace boost::posix_time;

/*!
* Provides file transfer and event building operations for redrail images.
*
*/
class LIBREDRAILCLIENT_EXPORT RedrailClient
{
public:

    /*! Constructor
    * Constructs a new RedrailClient object
    */
    RedrailClient();

    /*! Connect to remote host.
    * Connects to the server given the ip and the port.
    *
    * \param [in] ip    The remote ip or host name
    * \parma [in] port  The remote port
    */
    void connect( std::string ip,
                  std::string port );

    /*! Disconnect from the remote host.
    * Disconnects the client from the server.
    *
    */
    void disconnect();

    /*! Destructor
    * Destroys the RedrailClient object
    *
    */
    ~RedrailClient();

    /*! Transfer result images from server.
    * Retrieves all event images from the redrail server and stores them
    * into the given local destination directory (default = /events).
    *
    * \param [in] destinationDir    The local directory (default = /events)
    * \param [in] timeoutMS         The timeout when communication with the server.
```

```
*                                Default = 1000.
*
* \return number of transfered files
*/
void transferFromServer(int32& numTransfered,
                        std::string destinationDir = "/events",
                        int32 timeoutMS = 1000,
                        bool transferUnverified = false);

/*! Cancel a current active transfer.
* The currently active transfer will be canceled as soon as the
* current image is transfered.
*
*/
void cancelTransfer();

/*! Build events from redrail images.
* Builds a list of events by analyzing all files in the given directory.
*
* \param [in] eventDir    The event directory (default = /events)
*/
std::vector<RedrailEvent> buildEvents(std::string eventDir = "/events");

/*! Build events from redrail images.
* Builds a list of events by analyzing all files in the given directory.
*
* \param [in] eventDir     The event directory (default = /events)
* \param [in] timeDuration boost::posix_time::time_duration object
*                          representing the maximum duration between
*                          two events to be merged (default = 2 seconds)
* \param [in] maxNumEventFiles After successfully merging of events these
*                              events are pruned for display (default = 10).
*/
std::vector<RedrailEvent> buildAndMergeEvents(
                                std::string eventDir = "/events",
                                time_duration timeDuration = time_duration(0,0,2,0),
                                std::size_t maxNumEventFiles = 10);

/*! Registers an observer for the transfer progress
* The given object is informed over the transfer progress
*
* \param [in] observer      The observing object which will be informed of the progress.
*
*/
inline void setProgressObserver(ProgressObserver* observer)
{
    mProgressObserver = observer;
}

/*! Retrieves the time from a given fileName
* Parses the fileName and returns the boost::posix_time::ptime object
* correspondending to the fileName's time string.
*
* \param [in]  fileName          The fileName to parse.
*
* \return The ptime object holding the fileNames time.
*/
boost::posix_time::ptime parseTimeFromFileName(std::string fileName) const;

/*! Retrieves the state from a given fileName
* Parses the fileName and returns the RedrailEvent::State
* correspondending to the fileName's state string.
*
* \param [in]  fileName          The fileName to parse.
*
* \return The RedrailEvent::State.
*/
RedrailEvent::State      parseStateFromFileName(std::string fileName) const;

/*! Get the prefix of a filename.
* Parses the fileName and returns its prefix which
* is the ID of the redrail system.
*
* \param [in] fileName      The fileName to parse.
*/
std::string       parsePrefixFromFileName(std::string fileName) const;

/*!
```

```
    * Used for sorting the RedrailEvents in descending order.
    *
    */
    struct
    {
        bool operator() (RedrailEvent a,RedrailEvent b) { return (a>b);}
    }mDescendingSorter;


private:
    void*               mClient;
    std::string         mEventDir;
    ProgressObserver*   mProgressObserver;
    bool                        mCancel;
};
```

redrailevent.h

```
#include <string>
#include <vector>
#include <boost/date_time/posix_time/posix_time.hpp>

class LIBREDRAILCLIENT_EXPORT RedrailEvent
{
public:

    /*! Annotation state of the Event
    * Using the client software a redrail event can
    * have one out of four possible annotation states which
    * are coded into the filename.
    */
    enum State
    {
        Detection,            // filename: _ff_
        NoViolation,          // filename: _tf_
        Violation,            // filename: _ft_
        Punishment            // filename: _tt_
    };

    /*! Create a new redrail event.
    * Creates a new Redrail Event object.
    */
    RedrailEvent(void);

    /*! Destructor
    *
    */
    ~RedrailEvent(void);

    /*! Comparison of two redrail events
    * RedrailEvents are equal, when their time is equal.
    */
    bool operator==(const RedrailEvent& re) const;

    /*! Comparison of two redrail events
    * RedrailEvents are compared using their time.
    */
    bool operator<(const RedrailEvent& re) const;

    /*! Comparison of two redrail events
    * RedrailEvents are compared using their time.
    */
    bool operator>(const RedrailEvent& re) const;

    /*! Adds a file to the redrail event
    * The given file is added at the back of the internal file list.
    *
    * \param  [in]    The file path to be added.
    */
    void addFile(std::string file);

    /*! Removes a file from the redrail event
    * The file at the given index is removed.
    *
    * \param [in] fileThe file index to be removed.
    */
```

```
    void removeFile(std::size_t index);

    /*! Update the file at the given index
    * The file at the given index is replaced with the one passed.
    *
    * \param [in] index The file index to be replaced.
    * \param [in] file  The replacement file.
    */
    void updateFile(std::size_t index, std::string file);

    /*! Retrieves the file list.
    * Returns the whole list of file paths.
    *
    * \return All file paths belonging to the redrail event.
    */
    std::vector<std::string>& getFiles();

    /*! Removes all files from the redrail event.
    * All file paths are removed from the redrail event.
    */
    void clearFiles();

    /*! Sets the time of the event.
    * The given time is set to the event time. The time is used
    * for comparison and sorting of events.
    *
    * \param [in] timeThe new event time.
    */
    void setTime(boost::posix_time::ptime time);

    /*! Retrieves the current redrail event time.
    * Returns the current redrail event time. The time is used
    * for comparison and sorting of redrail events.
    *
    * \return The new redrail event time.
    */
    boost::posix_time::ptime getTime();

    /*! Sets the state of the redrail event.
    * There are four possible redrail event states:
    *
    *          Detection:     Indicates, that this is a redrail event not viewed at all.
    *          Delete:        Indicates, that this redrail event is deleted.
    *          Violation:     Indicates, that this redrail event holds a violation but that
    *                         this violation was not punished.
    *          Punishment:    Indicatest, that this redrail event is completly finished.
    *
    * \param [in] state       The new redrail event state.
    */
    void setState(State state);

    /*! Retrieves the state of the redrail event.
    * There are four possible redrail event states:
    *
    *          Detection:     Indicates, that this is a redrail event not viewed at all.
    *          Delete:        Indicates, that this redrail event is deleted.
    *          Violation:     Indicates, that this redrail event holds a violation but that
    *                         this violation was not punished.
    *          Punishment:    Indicatest, that this redrail event is completly finished.
    *
    * \return The current redrail event state.
    */
    State getState();

    /*! Transforms a state enum object to a string representing the state.
    *
    * \param [in] state       The redrail event state.
    * \return The string representation of the event state.
    */
    std::string stateToString(RedrailEvent::State state) const;

    /*! Returns the prefix.
    *
    * The prefix represents the camera from where the file was downloaded.
    *
    * \return The prefix representing the image's origin.
    */
    std::string getPrefix() const;
```

```
    /*! Set the prefix.
     *
     * The prefix represents the camera from where the file was downloaded.
     *
     * \param [in] prefix      The events origin = the file name prefix.
     */
    void setPrefix(const std::string& prefix);

    /*! Get the filename of the ini file for this event.
     *
     * The ini file has as extionsion .txt.
     *
     * \return The ini filename of the event.
     */
    std::string getIniFilename() const;

    /*! Set the ini filename for this event.
     *
     * Set the ini filename upon event generation for this event.
     *
     * \param [in] iniFilename The ini filename of the event.
     */
    void setIniFilename( const std::string& iniFilename );

private:

    boost::posix_time::ptime  mTime;
    std::vector<std::string>  mFiles;
    std::string                                     mIniFilename;
    State                                           mState;
    std::string                                     mPrefix;
};
```

progressobserver.h

```
#include "libredrailclientdef.h"
#include <svcore/include/svdef.h>

/*! Base class for observing the file transfer progress.
* This class is an implementation of the observer pattern used in
* the RedrailClient to provide a way to inform calling objects of
* the current file transfer progress.
*
* One has to create a derived class from ProgressObserver and register
* an object of this class at the RedrailClient by calling the method
* RedrailClient::setProgressObserver()
*
* Then the RedrailClient will call the methods setNumFiles(),
* setProgress() and setProgressText() of the passed object when a
* transfer is in progress.
*/
class LIBREDRAILCLIENT_EXPORT ProgressObserver
{

public:

    /*! Informs about the number of files to be downloaded.
    * This method is called after the file list has been downloaded from the
    * redrail camera.
    *
    * \param [in]    numFiles    The number of files to be downloaded.
    */
    virtual void setNumFiles(int32 numFiles) = 0;

    /*! Sets the index of the current file download.
    * For each new file to be downloaded this method is called with
    * increasing progress value.
    *
    * \param [in]    progress    The current file index which is downloaded.
    */
    virtual void setProgress(int32 progress) = 0;

    /*! Sets an informative text about the current progress.
    * This method is called whenever a new action is done in the transfer progress.
    * It is called for connection creation, file list download and for each single file.
```

```
    *
    * \param [in]   text        Textual description of the current transfer progress.
    */
    virtual void setProgressText(std::string text) = 0;
};
```

libredrailclientdef.h

```
#ifdef _DLL
	#ifdef LIBREDRAILCLIENT_EXPORTS
		#define LIBREDRAILCLIENT_EXPORT __declspec(dllexport)
	#else
		#define LIBREDRAILCLIENT_EXPORT __declspec(dllimport)
	#endif
#else
	#define LIBREDRAILCLIENT_EXPORT
#endif
```

Die Dekodierung von Bildern auf der Client Seite

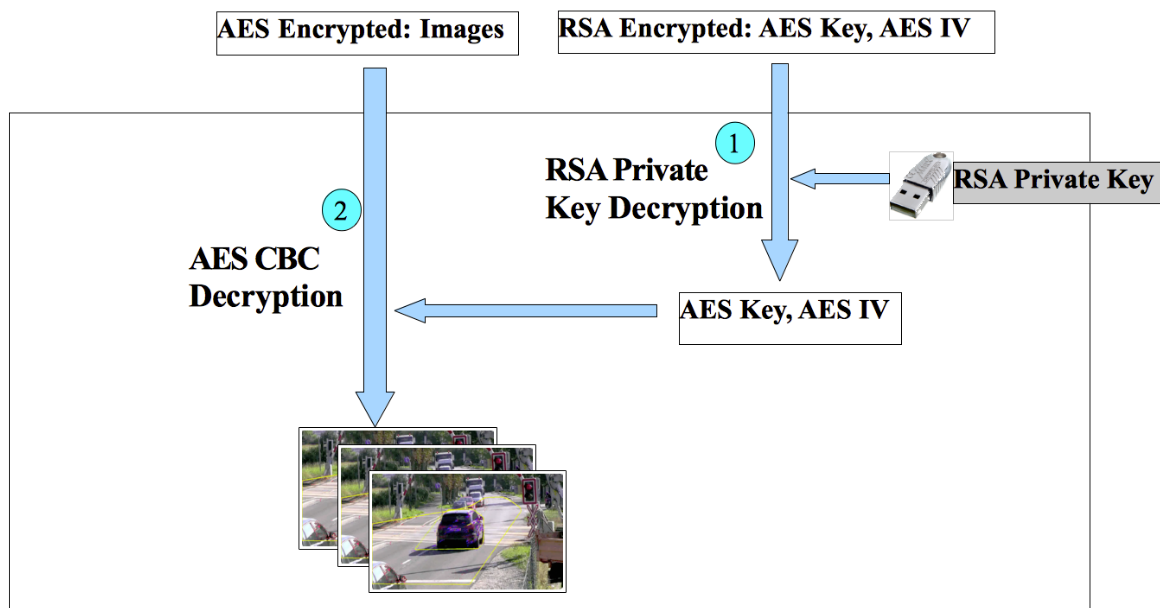
Die Bilddaten auf der Kamera sind durch AES-Verschlüsselung bzw. RSA Verschlüsselung und AES- IV auf der SLR Smart Camera geschützt.

Die Dekodierung der Bilddaten erfolgt nach folgendem Schema:

I) Zunächst muss der Client die AES -Parameter entschlüsseln. Dies wird durch die Entschlüsselung mit der Kunden RSA Private Key (1) durchgeführt . Der private Schlüssel muss auf einem Hardware-Dongle gespeichert werden , um den Schlüssel aus , die verteilt verhindern.

II) Danach werden die verschlüsselten Bilder mit den restaurierten AES -Parameter und die AES -CBC- Entschlüsselungs-Algorithmus (2) wieder entschlüsselt.

Die folgende Abbildung zeigt , wie die Entschlüsselung funktioniert :



Bemerkung: Zu Details zum Entschlüsseln der Daten von der Kamera kontaktieren Sie bitte SLR Engineering. Aus Sicherheitsgründen werden die Details zum Entschlüsseln nicht publiziert.

Referenzen

http://en.wikipedia.org/wiki/Block_cipher_modes_of_operation#Cipher-block_chaining_.28CBC.29
http://en.wikipedia.org/wiki/RSA_%28algorithm%29
http://www.cryptotech.com/products_cryptoboxusb.php
http://www.cryptopp.com/