IRB-Datei-Zugriff IRBACS-SDK Version 2

Dokumentation

Stand: Dezember 2017

InfraTec GmbH
Infrarotsensorik und Messtechnik
Gostritzer Straße 61 – 63
01217 Dresden/GERMANY

Telefon +49 351 871-8615
Fax +49 351 871-8727
E-Mail service@InfraTec.de
www.InfraTec.de

Inhaltsverzeichnis

1 Einleitung

Die von InfraTec für die Infrarotkameras erstellte Software legt ihre Daten in indizierten Dateien ab, die einen schnellen und komfortablen Zugriff ermöglichen. Um das Handling auch für externe Software zu vereinfachen, ist ein SDK erstellt worden, welches den direkten Zugriff auf IRB-Dateien übernimmt. Diese Bibliothek trägt den Namen IRBACS_xx.DLL bzw. libirbacs_xx.so und ermöglicht insbesondere den Zugriff auf die thermografisch relevanten Eigenschaften der in den IRB-Dateien hinterlegten Infrarotbilder.

Die Intention des SDK besteht darin, die Daten für die Integration in eigene Applikationen zur Verfügung zu stellen. Mit den übernommenen Daten kann die eigene Applikationsumgebung gestaltet werden. Die Rücküberführung der veränderten Daten in die InfraTec-Softwareumgebung ist auch aus Integritätsgründen kein primäres Ziel des SDK.

Der Funktionsumfang umfasst den Zugriff auf die abgelegten Temperaturdaten, auf Informationen der Headerstruktur und temperaturrelevante Korrekturparameter. Auch können veränderte Frames als Einzeldatei abgespeichert werden und in beschränktem Umfang Manipulationen der Headerstrukturen vorgenommen werden.

Folgende Betriebssysteme werden unterstützt:

Windows ab XP -	32 Bit:	irbacs_w32.dll
Windows ab 7 -	64 Bit:	irbacs_w64.dll
Linux – 32 Bit:		libirbacs_l32.so
Linux – 64 Bit:		libirbacs_l64.so

Es gelten folgende Einschränkungen:

- Windows 64 bit : Komprimierte IRB-Dateien der VarioCAM® HD lassen sich mit der 64-Bit Version der irbacs.dll nicht enpacken.
- Resolution-enhanced-IRBs werden nicht berechnet, die Thermogramme werden mit der originalen Kamera-Auflösung behandelt

Für den Echtzeitzugriff auf Daten der Kamerasysteme gibt es ein weiteres SDK (IRBGRAB), welches derzeit ausschließlich als 32-Bit-Dll für Windows zur Verfügung steht.

2 Benutzte Datenstrukturen

Die Datenstrukturen werden gepackt abgelegt, es werden somit keine Leerbytes zur Ausrichtung zwischen den Elementen der Strukturen eingefügt.

2.1 IRB-Header-Struktur

Diese Struktur beschreibt mit ihren Unterstrukturen bild- und temperaturrelevante Informationen des IRB-Formates.

```
TIRBgeomInfo = packed record
    pixelFormat     : Word;
    compression     : Word;
    imgWidth        : Word;
    mgHeight        : Word;
    upperLeftX      : Word;
    upperLeftY      : Word;
    firstValidX     : Word;
    lastValidX      : Word;
    firstValidY     : Word;
    lastValidY      : Word;
    position        : Single;
end;

TIRBobjPars = packed record
    emissivity      : Single;
    objDistance     : Single;
    ambTemp         : Single;
    absoConst       : Single;
    pathTemp        : Single;
    version         : LongInt;
end;

TIRBCalibPars1 = packed record
    cbData          : array[0..1567] of byte;
end;

TIRBImageInfo = packed record
    level           : Single;
    span            : Single;
    imgTime         : TDateTime;
    imgMilliTime    : Single;    // Millisekunden
    imgAccu         : Word;
    imageComment: array[0..79] of Char;
    zoom_hor        : Single;
    zoom_vert       : Single;
    imgMilliTimeEx : SmallInt;  // evtl. Mikrosekunden
end;
```

```
TIRBImageData1 = packed record
      geomInfo        : TIRBgeomInfo;
      objectPars      : TIRBobjPars;
      calibPars       : TIRBCalibPars1;
      imgInfo         : TIRBImageInfo;
 end;

PIRBImageData1 = ^TIRBImageData1;
```

2.2 IRB-Calib-Struktur

Diese Struktur gibt Zugriff auf die für die Kalibrierung benutzten Daten:

```
TIRBCalibData = packed record
      Version         : Integer;
      Count           : Integer;
      Values          : array [0..269] of Single;
 end;
```

2.3 Korrekturdaten-Struktur

Diese Struktur gibt Zugriff auf die für die Korrektur benutzten Daten:

```
TIRBCorrPars = packed record
          epsilon   : Double;
          envTemp   : Double;
          tau       : Double;
          pathTemp  : Double;
          lambda    : Double;
          deltaLambda : Double;
end;
```

2.4 Analogdatenstruktur

Diese Struktur wird von readIRBLoggerdata bereitgestellt.

```
TIRBDataLoggerDataItem = packed record
        TimeStamp      : Double;
        Value          : Double;
end;

TIRBDataLoggerEntry = packed record
        Size           : integer; // Größe der Daten (Header + Daten) eines Kanals in Byte
        HeaderSize     :  integer; // sizeof(TIRBDataLoggerEntry)
        ChannelName    : array[0..31] of Char;
        DataType       : integer; // 0 = ANALOG IN
        DataCount      : integer;
end; // Danach kommen die Daten als TIRBDataLoggerDataItem. Man liest solange ein, bis die Größe des eingelesenen Bereichs der Größe des Gesamteintrages entspricht.

 TIRBDataLoggerDataItem = packed record
   TimeStamp          : Double;
   Value              : Double;
 end;

 PIRBDataLoggerEntry = ^TIRBDataLoggerEntry;
 TIRBDataLoggerEntry = packed record
   Size             : integer; // Größe der Daten (Header + Daten) eines Kanals in Byte
   HeaderSize       : integer; // sizeof(TIRBDataLoggerEntry)
   ChannelName      : array[0..31] of Char;
   DataType         : integer; // 0 = ANALOG IN
   DataCount        : integer;
 end;                         // Danach kommen die Daten mit je ein Double TimeStamp und ein Double Wert (TIRBDataLoggerDataItem)
```

3 Auflistung der DLL-Funktionen

Die Bibliotheken sind in der Programmiersprache Pascal erstellt, die Aufrufkonvention ist stdcall.

3.1 version

function version(var mainver, subver : Integer): Boolean;

Zweck:	Dient zur Ermittlung der Dll-Version der Bibliothek und kann vom Programm zum Integritätscheck verwendet werden.	
Parameter:	mainver:	Hauptversion
	subver:	Unterversion
Rückgabe:	True bei Erfolg	
	False bei Misserfolg	

3.2 loadIRB

function loadIRB(const fn : PAnsiChar) : PtrUInt;

Zweck:	Öffnet eine IRB-Datei zur Bearbeitung und gibt einen Handle zurück und aktiviert das erste Frame in der IRB-Datei.	
Parameter:	fn:	Pfadname der IRB-Datei
Rückgabe:	Referenz auf Verwaltungsstruktur	
	im Fehlerfall wird 0 zurückgegeben	

3.3 unloadIRB

procedure unloadIRB(const aHandle : PtrUInt);

Zweck:	Gibt die zu einer geöffneten IRB-Datei zugehörige Verwaltungsstruktur in DLL wieder frei.	
Parameter:	aHandle:	mit loadIRB erzeugte Referenz
Rückgabe:	keine	

3.4 getFrameCount

function getFrameCount(const aHandle : PtrUInt): Integer;

Zweck:	Ermittelt die Anzahl der in einer IRB-Sequenzdatei enthaltenen Einzelbilder.	
Parameter:	aHandle:	mit loadIRB erzeugte Referenz
Rückgabe:	Anzahl der IR-Bilder	
	im Fehlerfall wird 0 zurückgegeben	

3.5 getIRBIndices

function getIRBIndices(const aHandle : PtrUInt; const irbIdxList : PInteger) : Integer ;

Zweck: Ermittelt die in der IRB-Datei enthaltenen Indexe der Thermobilder, die sog. IRB-Indices
Parameter: aHandle: mit loadIRB erzeugte Referenz
irbIdxLst: Durch Aufruf mit nil erhält man die Anzahl der IRB-Indexe in der IRB-Datei zurück. Der zu allokierende Speicherbereich muss mindestens die Größe Anzahl IRB-Indexe * SizeOf(Integer) haben. Übergibt man den Zeiger auf diesen Speicherbereich, werden von der Funktion getIRBIndices alle IRB-Indexe der IRB-Datei darin abgelegt.
Rückgabe: Anzahl der in der IRB-Datei enthaltenen IRB-Indexe
im Fehlerfall wird 0 zurückgegeben

3.6 getFrameNumber

function getFrameNumber(const aHandle : PtrUInt) : Integer;

Zweck: Ermitteln des IRB-Indices des aktivierten Thermobildes
Parameter: aHandle: mit loadIRB erzeugte Referenz
Rückgabe: IRB-Index des aktiven Frames
im Fehlerfall wird 0 zurückgegeben

3.7 setFrameNumber

procedure setFrameNumber(const aHandle : PtrUInt; const frno : Integer);

Zweck: Aktivieren des Thermobildes mit dem IRB-Index frno
Parameter: aHandle: mit loadIRB erzeugte Referenz
frno: Nummer des zu aktivierenden Thermobildes, ausgehend von der internen Nummerierung per indexID
Rückgabe: keine, Erfolg der Aktion kann mit getFrameNumber überprüft werden

3.8 getFrameNbByArrayIdx

function getFrameNbByArrayIdx(const aHandle : PtrUInt) : Integer;

Zweck: Gibt den Array-Index des aktivierten Thermobildes (Aufsteigende Nummerierung der Frames beginnend mit 0)
Parameter: aHandle: mit loadIRB erzeugte Referenz
Rückgabe: (-1) – bei Fehler
Array-Index des aktiven Frames beginnend mit 0

3.9 setFrameNbByArrayIdx

procedure setFrameNbByArrayIdx(const aHandle : PtrUInt; const frno : Integer);

Zweck:	Aktivieren des Thermobildes mit dem Array-Index frno	
Parameter:	aHandle:	mit loadIRB erzeugte Referenz
	frno:	Array-Index des zu aktivierenden Thermobildes in der Sequenz (frno beginnend mit 0 bis Thermobildanzahl-1)
Rückgabe:	keine, der Erfolg der Aktion kann mit getFrameNbByArrayIdx überprüft werden	

3.10 getTempBBXY

function getTempBBXY(const aHandle : PtrUInt; const xx, yy : Integer) : Double;

Zweck:	Ermittelt Blackbody-Temperatur an der mit (xx,yy) adressierten Stelle im Thermobild	
Parameter:	aHandle:	mit loadIRB erzeugte Referenz
	xx:	X-Koordinate des Messpunktes (0-indiziert)
	yy:	Y-Koordinate des Messpunktes (0-indiziert)
Rückgabe:	Temperaturwert in Kelvin, im Fehlerfall wird 0 zurückgegeben	

3.11 getTempXY

function getTempXY(const aHandle : PtrUInt; const xx, yy : Integer) : Double;

Zweck:	Ermittelt korrigierte Temperatur an der mit (xx,yy) adressierten Stelle im Thermobild	
Parameter:	aHandle:	mit loadIRB erzeugte Referenz
	xx:	X-Koordinate des Messpunktes
	yy:	Y-Koordinate des Messpunktes
Rückgabe:	Temperaturwert in Kelvin im Fehlerfall wird 0 zurückgegeben	

3.12 getDigValXY

function getDigValXY(const aHandle : PtrUInt; const xx, yy : Integer) : Double;

Zweck:	Ermittelt Digitalwert an der mit (xx,yy) adressierten Stelle im Thermobild	
Parameter:	aHandle:	mit loadIRB erzeugte Referenz
	xx:	X-Koordinate des Messpunktes
	yy:	Y-Koordinate des Messpunktes
Rückgabe:	Digitalwert im Fehlerfall wird 0 zurückgegeben	

3.13 readPixelData

function readPixelData(const aHandle : PtrUInt; const pData: Pointer; const what : Integer) : Integer;

Zweck:	Füllt den übergebenen Speicherberech mit den im Thermogramm Daten, je nachdem,, was über what angefordert wurde (Digitalwerte, Blackbody-Temperaturen, korrigierte Temperaturen).	
Parameter:	aHandle:	mit loadIRB erzeugte Referenz
	pData:	Zieladresse des Datenfeldes (Double) (vom Aufrufer zu allokieren) bei nil wird die Größe des zu allokierenden Speichers in Byte zurückgegeben
	what:	zu ermittelnde Werte 0 - Blackbody-Temperaturen in K (Datentyp Double) 1 – Korrigierte Temperaturen in K (Datentyp Double) 2 – Digitalwerte (Datentyp Double) 3 - Blackbody Temperaturen in K (Datentyp Single)
Rückgabe:	Länge des Feldes in Byte im Fehlerfall wird 0 zurückgegeben	

3.14 readIRBData

function readIRBData(const aHandle : PtrUInt; const pIRBFrame: PIRBImageData1) : Integer;

Zweck:	Liest die Rohdaten (ggf. auch komprimiert) incl. Header für das ausgewählte Bild	
Parameter:	aHandle:	mit loadIRB erzeugte Referenz
	pIRBFrame:	Zieladresse des Datenfeldes (vom Aufrufer zu allokieren) bei nil wird die Größe des zu allozierenden Speichers in Byte zurückgegeben
Rückgabe:	Länge des Feldes in Byte im Fehlerfall wird 0 zurückgegeben	

3.15 readIRBDataUncompressed

function readIRBDataUncompressed(const aHandle : PtrUInt; const pIRBFrame: PIRBImageData1) : Integer;

Zweck:	Liest unkomprimierte Rohdaten incl. Header für das ausgewählte Bild	
Parameter:	aHandle:	mit loadIRB erzeugte Referenz
	where:	Zieladresse des Datenfeldes (vom Aufrufer zu allokieren) bei nil wird die Größe des zu allokzierenden Speichers in Byte zurückgegeben
Rückgabe:	Länge des Feldes in Byte im Fehlerfall wird 0 zurückgegeben	

3.16 convertPixelToKelvin

function(const aHandle : PtrUInt; const pData: Pointer; const cnt: Integer; const corrpars : Pointer): Integer;

Zweck:	Wandelt Rohwerte (Digitalwerte) unter Benutzung der internen Kalibrierung und der übergebenen Korrekturparameter in Temperaturwerte	
Parameter:	aHandle:	mit loadIRB erzeugte Referenz
	pData:	Quell/Zieladresse des Datenfeldes vom Typ Double (vom Aufrufer zu allokieren und zu füllen)
	cnt:	Größe des zu konvertierenden Bereiches in Byte
	Corrpars:	Struktur mit Korrekturparametern (s.o. TIRBCorrPars) oder nil (die internen Korrekturparameter werden verwendet)
Rückgabe:	Anzahl der berechneten Werte im Fehlerfall wird 0 zurückgegeben	

Beispiel:

```
FIrbacsHnd := _irbacs64_loadIRB( filenam ); //  Laden der IRB-Datei
//  Ermitteln des Speicherbedarfs
iMemSize := _irbacs64_readPixelData( FIrbacsHnd, nil, 2 );
   Getmem( kelvdat, iMemSize );
   iMemSize := _irbacs64_readPixelData( FIrbacsHnd, kelvdat, 2 ); // Laden
der Pixel als Rohwerte

… Manipulieren der Rohwerte ( im Double-Format )
pDoub := PDouble( kelvdat );
For iPixel := 0 to ( iMemSize div SizeOf( Double ) )-1 do
Begin
  pDoub^ := pDoub^ - 10;
  Inc( pDoub );
end;
…
Corrpars.epsilon := 0.95;
Corrpars.envtemp := 293.15; // in K
Corrpars.tau        := 0.8;
Corrpars.pathtemp := 283.15; // in K
Corrpars.lambda             := 0;  // Nutzung des Standardwertes
Corrpars.deltaLambda    := 0;  // Nutzung des Standardwertes

Res  :  =  _irbacs64_convertPixelToKelvin(FIrbacsHnd,  kelvdat,  iPixelSize,
@corrpars );
```

3.17 getParam

function getParam(const aHandle :PtrUInt; const what : Integer; var aValue : Double): Boolean;

Zweck: Liest einen Parameterwert vom Typ Single (float) aus einer IRB-Datei

Parameter:
- aHandle: mit loadIRB erzeugte Referenz
- what: Welcher Parameterwert soll in Value zurückgegeben werden:
  - 0 - Bildbreite
  - 1 - Bildhöhe
  - 2 – Mitteltemperatur (Level) in K
  - 3 – Spreizung (Span
  - 4 - Emissionsgrad
  - 5 - Entfernung
  - 6 - Pfadtemperatur
  - 7 - Umgebungstemperatur
  - 8 - Absorption
  - 9 - IRB-Dataversion (100 o. 101)
  - 10 - Zoom (1 = 100%)
  - 14 - PixelFormat :
    - 1 = BYTE (8 Bit, ohne Vorzeichen)
    - 2 = WORD / (16 Bit, ohne Vorzeichen)

0xF084 = 61572 = SINGLE / FLOAT (32 Bit, mit Vorzeichen)
15 - Transmission (tau)
16 - Schwerpunktwellenlänge Lambda
17 - deltaLambda
18 - Millisekunden-Zeitstempel
19 - Absolutzeitstempel (seit 30.12.1899 12.00 Uhr, 1 = 1 Tag)
20 – Kalibrierbereichsuntergrenze in K
21 – Kalibrierbereichsobergrenze in K
22 – Triggerstatus
23 – Integrationszeit in µsec
24 – HFOV in °
25 – VFOV in °
26 – Smooth
27 – Kameratemperatur in K
28 – Sensortemperatur in K
29 – Prozessinterface Digitalwerte (optional)
30 – Prozessinterface Analogwert 1 (optional)
31 – Prozessinterface Analogwert 2 (reserviert)
32 – Prozessinterface Analogwert 3 (reserviert)
33 – Prozessinterface Analogwert 4 (reserviert)
34 – Prozessinterface Analogwert 5 (reserviert)

aValue: ausgelesener Rückgabewert

Rückgabe: True – Erfolg mit gültigem Wert in aValue
False – Fehler

3.18 setParam

function setParam(const aHandle : PtrUInt; const what : Integer; const aValue : Double) : Boolean;

Zweck: Ändert Parameter in einem IRB-Bild
mit saveSingleFrame() werden diese Änderungen dann in einer Einzeldatei gespeichert

Parameter: aHandle: mit loadIRB erzeugte Referenz
what: siehe getParam
aValue: Übergabewert

Rückgabe: True – Erfolg
False – Fehler

3.19 getParamS

function getParamS(const aHandle: PtrUInt; const what: Integer; const aValue: PAnsiChar) : Boolean;

Zweck: Liest einen Parameterwert vom Typ PAnsiChar aus einer IRB-Datei

Parameter: aHandle: mit loadIRB erzeugte Referenz
what: Spezifikation des abgefragten Parameters
11 - Kameraname
12 - Kamera Serien-Nr.
13 - Objektivname

	Value:	Übergabe eines Zeigers auf mindestens 256 Byte (bzw. cMaxParamString) großen Speicherbereich. Es werden maximal 256 Byte mit einem nullterminiertem String beschrieben.
Rückgabe:	True – Erfolg mit gültiger Zeichenkette in aValue False – Fehler	

3.20 getMilliTime

function getMilliTime(const aHandle : PtrUInt) : Double;

Zweck:	Liest den ms-Zeitstempel für das ausgewählte Bild	
Parameter:	aHandle:	mit loadIRB erzeugte Referenz
Rückgabe:	Hochaufgelöster ms-Zeitstempel im Fehlerfall 0	

3.21 getFrameTimeStamp

function getFrameTimeStamp(const aHandle : PtrUInt; timestamp : TSystemTime) :Boolean;

Zweck:	Liest das Datum und die Uhrzeit für das ausgewählte Bild	
Parameter:	aHandle:	mit loadIRB erzeugte Referenz
Rückgabe:	True – Erfolg (gültiger Zeitstempel in timestamp) False – Fehler	

TSystemTime : unter Delphi und C++ übliche SystemTime-Struktur folgenden Inhalts:

```
type TSystemTime = record
  Year: Word;              Year part
  Month: Word;             Month part
  DayOfWeek: Word;         Day of week part
  Day: Word;               Day of month part
  Hour: Word;              Hour of the day
  Minute: Word;            Minute of the hour
  Second: Word;            Second of the minute
  MilliSecond: Word;       Milliseconds in the second
end;
```

3.22 getIRBCalibData*

function getIRBCalibData(const aHandle : PtrUInt; var IRBCalibData : TIRBCalibData) : Boolean;

Zweck:	Liest die Kalibrierwerte einer IRB-Datei aus	
Parameter:	aHandle:	mit loadIRB erzeugte Referenz
	IRBCalibData:	Struktur mit Kalibrierwerten (wenn IRBCalibData.Version=8 oder 9, beziehen sich die Einträge im Array IRBCalibData.Values auf einen Temperaturwert.
Rückgabe:	True – Erfolg False – Fehler	

**Die Funktion ist nur noch aus Kompatibilitätsgründen mit der äteren Version 1 im Interface enthalten. Ihre Funktionalität erübrigt sich durch die Funktion convertPixelToKelvin.*

3.23 getIRBHeader*

function getIRBHeader(const aHandle : PtrUInt; var aIRBHeader : TIRBImageData1) : Boolean;

Zweck:	Auslesen des IRB-Headers eines IR-Bildes	
Parameter:	aHandle:	mit loadIRB erzeugte Referenz
	aIRBHeader:	IRB-Header ohne Bilddaten
Rückgabe:	True – Erfolg	
	False – Fehler	

**Die Funktion ist nur noch aus Kompatibilitätsgründen mit der äteren Version 1 im Interface enthalten. Ihre Funktionalität erübrigt sich durch die Funktion convertPixelToKelvin.*

3.24 setIRBHeader*

function setIRBHeader(const aHandle : PtrUInt; const aIRBHeader : TIRBImageData1) : Boolean;

Zweck:	Rückschreiben des IRB-Headers in ein IR-Bild	
Parameter:	aHandle:	mit loadIRB erzeugte Referenz
	aIRBHeader:	IRB-Header ohne Bilddaten
Rückgabe:	True – Erfolg	
	False – Fehler	

**Die Funktion ist nur noch aus Kompatibilitätsgründen mit der äteren Version 1 im Interface enthalten. Ihre Funktionalität erübrigt sich durch die Funktion convertPixelToKelvin.*

3.25 saveSingleFrame

function saveSingleFrame(const aHandle : PtrUInt; fn : PChar; pIRBFrame : Pointer) : Boolean;

Zweck:	Speichert eine Irb-Datei unter angegebenen Namen	
Parameter:	aHandle:	mit loadIRB erzeugte Referenz
	fn:	Pfadname der IRB-Datei
	pIRBFrame:	Zeiger auf IRB-Frame (IRBHeader und Bilddaten). Diese Daten sollten z.B. mit der Funktion readIRBDataUncompressed gelesen worden sein. Ist der Wert nil, so wird das aktivierte Frame gespeichert.
Rückgabe:	True – Erfolg	
	False – Fehler	

3.26 exportVisBitmap

function exportVisBitmap(const aHandle : PtrUInt; const fn : PAnsiChar) : Boolean;

Zweck:	Speichert das visuelle Bild einer IRB-Datei unter dem angegebenen Namen im entsprechenden Format ab	
Parameter:	aHandle:	mit loadIRB erzeugte Referenz
	fn:	Pfadname der zu speichernden Bilddatei

Rückgabe: True – Erfolg
False – Fehler

3.27 audioComment

function audioComment(const aHandle : PtrUInt; const fn : PAnsiChar) : Boolean;

Zweck: Speichert den Audiokommentar einer IRB-Datei unter dem angegeben Dateinamen ab
Parameter: aHandle: mit loadIRB erzeugte Referenz
fn: Vollständiger Dateiname (inkl. Pfad) der zu speichernden Audiodatei
Rückgabe: True – Erfolg
False – Fehler