INSTITUT FÜR
DATENVERARBEITUNGSANLAGEN

TECHNISCHE UNIVERSITÄT
BRAUNSCHWEIG

Imaging with GSEOS

Revision 1.0

IDA-GSEOS-0009

September 2000

Prepared by
Hagen Schmidt

Tables and Figures

1 Scope

1.1 Purpose of this Document

Purpose of this document is to describe the use of GSEOS to process and display image data in the project MSRS.

1.2 Change Record

Table 1 Change Record

Date	Revision	Author	Affected Sections
9/12/2000	1.0	Schmidt	All sections

1.3 Reference Documentation

All GSEOS related documents are listed in the GSEOS Document Index IDA-GSEOS-0000.

1.4 Abbreviations

Table 2 Abbreviations

CU	Camera Unit
EGSE	Electrical Ground Support Equipment
GSEOS	Ground Support Equipment Operating System
MSRS	Multi Spectral High Resolution Sensor
RTL	Runtime Library

2 Introduction

This paper describes a special feature of the Ground Support Operating System 5 (GSEOS 5) – the processing and display of image data, e.g. the handling of bitmaps. First it describes the basics of bitmaps. This is followed by an example of the use of bitmaps in user configuration.

For more detailed information about using the GSEOS configuration language use the document GSEOS Language Description (IDA-GSEOS-0003).

Further more information about the internal structure of GSEOS and its general using is found in the document GSEOS User's Manual (IDA-GSEOS-0001).

The GSEOS Runtime Library Description (IDA-GSEOS-0004) lists predefined functions built in the GSEOS supporting for the development of user configuration.

3 Imaging

3.1 Image Data in GSEOS

The GSEOS software has the capability to display data in various ways. On special kind of data display is the display as image (called as bitmap).

3.1.1 Bitmap Formats

Bitmaps are the equivalent display of raw data as image. There are various color depths of bitmaps - from monochrome black and white (1 bit color depth) to multicolor RGB bitmaps (16, 24 or 32 bit color depth). In contrast to the Windows bitmap format the bitmaps in GSEOS contains the raw data only. The size and color depth is set in the bitmap definition in the appropriate configuration file (see 3.1.2).

Table 3 Bitmap Formats

Color Depth [bit]	Number of Colors	Representation in Memory
1	2 (black and white)	every bit in the data represent 1 pixel in the bitmap: 1: white 0: black
4	16	4 bits in the data represent 1 pixel in the bitmap, i.e. two pixels per byte of the data. The 16 colors are the predefined system colors of Windows.
8	256	8 bits in the data represents 1 pixel in the bitmap, i.e. one pixel per byte of data. The 256 colors are indexes in a predefined standard palette.
16	32768	16 bits in the data represent 1 pixel in the bitmap, i.e. one pixel per short word (two bytes) of data. The colors are stored as RGB value (no palette) in a 5-5-5-bit scheme (The highest bit of the short word is not used). The least 5 bits represent the blue part of the color value, the middle 5 bits represent the green part of the color value and the highest 5 bits represent the red part of the color value: 15 | 14...10 | 9...5 | 4...0 spare | red | green | blue

<table>
<tr>
<td>24</td>
<td>16777216</td>
<td>24 bits in the data represent 1 pixel in the bitmap, i.e. one pixel per triple byte of data.

The colors are stored as RGB values (no palette) in an 8-8-8-bit scheme. The least byte represent the blue part of the color value, the middle byte represent the green part of the color value and the highest byte represent the red part of the color value:
<table>
<tr><td>23...16</td><td>15...8</td><td>7...0</td></tr>
<tr><td>red</td><td>green</td><td>blue</td></tr>
</table>
</td>
</tr>
<tr>
<td>32</td>
<td>16777216</td>
<td>32 bits in the data represent 1 pixel in the bitmap, i.e. one pixel per long word of data.

The colors are stored as RGB values (no palette) in an 8-8-8-bit scheme (The highest byte of the long word is not used). The least byte represent the blue part of the color value, the middle byte represent the green part of the color value and the highest byte represent the red part of the color value:
<table>
<tr><td>31...24</td><td>23...16</td><td>15...8</td><td>7...0</td></tr>
<tr><td>spare</td><td>red</td><td>green</td><td>blue</td></tr>
</table>
</td>
</tr>
</table>

3.1.2 Bitmap Configuration

Bitmaps are special two-dimensional arrays with extended information about resolution and color depth. To define this extended information there are a special bitmap data type in the GSEOS configuration language:

`bitmap` ***depth*** **`[`***width* **`,`** ***height*****`]`** ***varname***

Allowed values for the color depth ***depth*** are 1, 4, 8, 16, 24 and 32 (see Table 3); ***width*** and ***height*** specify the horizontal and vertical dimension of the bitmap in pixels. Please note the different order and syntax of width and height as the C standard array definition.

Like all other data structures the configuration of the bitmaps can performed in the GSEOS configuration files (with the extension *.g) in batch configuration files (with the extension *.gb) or direct in the definition of an Acticve QLook Item.

You may use unions to get an easily access to the bitmap or you can also cast any array with the appropriate size (the size MUST fit) to the needed bitmap:

```
//-----------------------------------------------------------------------------
// sample of using unions to get an easily acces to the bitmap data
//-----------------------------------------------------------------------------
union
{
  bitmap 32 [1024, 768] bmp;
  ULONG                 aul[1024 * 768];
} uPicture;

//-----------------------------------------------------------------------------
// sample of a bitmap cast
//-----------------------------------------------------------------------------
ULONG aulPicture[768][1024];

uPicture.bmp = (bitmap 32 [1024, 768])aulPicture;
```

3.2 Configuration Example

This section describes the handling of bitmaps on the concrete example of the experiment MSRS.

3.2.1 Data Format of the CU data

The sensor data of the camera sensors have a bit width of 10 bits. Because of the architecture of the mass memory, the data are stored in a special format:

Table 4 Data Format of CU data

Word 0	Word 1	Word 2	Word 3	Word 4
bit 02	bit 22	bit 42	bit 62	bit 00
bit 03	bit 23	bit 43	bit 63	bit 01
bit 04	bit 24	bit 44	bit 64	bit 10
bit 05	bit 25	bit 45	bit 65	bit 11
bit 06	bit 26	bit 46	bit 66	bit 20
bit 07	bit 27	bit 47	bit 67	bit 21
bit 08	bit 28	bit 48	bit 68	bit 30
bit 09	bit 29	bit 49	bit 69	bit 31
bit 12	bit 32	bit 52	bit 72	bit 40
bit 13	bit 33	bit 53	bit 73	bit 41
bit 14	bit 34	bit 54	bit 74	bit 50
bit 15	bit 35	bit 55	bit 75	bit 51
bit 16	bit 36	bit 56	bit 76	bit 60
bit 17	bit 37	bit 57	bit 77	bit 61
bit 18	bit 38	bit 58	bit 78	bit 70
bit 19	bit 39	bit 59	bit 79	bit 71

There is a function in the GSEOS Runtime Library to decode this format:

```
ULONG DeformatCUData (USHORT &ausDest[], const USHORT &ausCU[], ULONG ulLen)
```

The 10 bit sensor data are decoded to a 16 bit data stream to avoid huge performance losses on the post processing. This function is called in the decoder of the CCU raw data block _DppSim (same as DPPSIM I/F).

Because of the architecture of Windows, there are no way to display bitmaps with more color depth than 3 x 8 bit (Windows supports 8 bit DACs only). So 8 bit only of the camera data can be displayed. In the example the lowest two bits of the data are cut.

The example decodes respectively three sensor channels to a RGB picture. Therefore there a two blocks defined **`CUPic0`** and **`CUPic1`** to retrieve the picture data. Every of these two blocks contains a bitmap with the dimension of 530x435 pixels and a color depth of 32 bit (RGB 8-8-8). To submit the appending of sensor data in the picture there are three length-counter. If the decoding of the picture is finished, e.g. all three counter reach the maximum of (530x435) no more picture decoding is done until the picture is cleared (all counters reset to zero).

The picture can be displayed in the QLook easily by creating an Image Item with the expression:

`CUPic0.bmp` or `CUPic1.bmp`

To separate one color channel in this block you may mask out the appropriate color channel:

- to separate the red color channel:
  `(CUPic0.aul & 0x00ff0000)` or
  `(CUPic1.aul & 0x00ff0000)`
- to separate the green color channel:
  `(CUPic0.aul & 0x0000ff00)` or
  `(CUPic1.aul & 0x0000ff00)`
- to separate the blue color channel:
  `(CUPic0.aul & 0x000000ff)` or
  `(CUPic1.aul & 0x000000ff)`

If you want to create grayscale pictures you have to set R = G = B, e.g. all three color-channels with the same data.

```
//---------------------------------------------------------------------------
//
// Project: MSRS
// File:    CCUSensPic.g
//
// Description:
//
//   Sample Application to show the "How To" of decoding of the CCU sensors
//   as pictures.
//
// Copyright (C) 2000, IDA
// Author: Hagen Schmidt
//
//---------------------------------------------------------------------------

// Sample Picture with a size of 530x435 pixels
#define PicMaxLen               (530 * 435)



//***************************************************************************
// block type definitions
//***************************************************************************

// the sample picture structure
typedef struct
{
  ULONG ulBLength;             // length of blue channel data
  ULONG ulGLength;             // length of green channel data
  ULONG ulRLength;             // length of red channel data
  union
  {
    BITMAP 32 [530, 435] bmp;  // the picture bitmap
                               //(530 * 435 pixels RGB 32bit)
    ULONG  aul[];              // data structure on same address
  };
}
tsCUPic;

blockdef tsCUPic tblkCUPic0 CUPic0; // picture of channel 0
blockdef tsCUPic tblkCUPic1 CUPic1; // picture of channel 1



//***************************************************************************
// functions
//***************************************************************************
//---------------------------------------------------------------------------
//
// VOID DecodeSensorPic (ULONG ulId, UCHAR &aucData[], ULONG ulLength)
//
// Description:
//
//   Decode sensor data as picture channels
//
// Parameters:  ULONG   ulId     - Sensor ID
//              USHORT &aus[]    - sensor data as USHORTs (16bit)
//              ULONG   ulLength - data length (USHORTs)
//
// Return:      -
//
//---------------------------------------------------------------------------
```

```
VOID DecodeSensorPic (ULONG ulId, USHORT &aus[], ULONG ulLength)
{
  ULONG ul;

  switch (ulId)
  {
    // sensor 0 as blue color channel
    case ID_CUS_S0:
         if (CUPic0.ulBLength < PicMaxLen)
         {
           tblkCUPic0 bCUPic = CUPic0;
           for (ul = 0;
               (ul < ulLength) && ((bCUPic.ulBLength+ul) < PicMaxLen); ++ul)
           {
             bCUPic.aul[bCUPic.ulBLength+ul] |= ((ULONG)(aus[ul]>>2));
           }
           bCUPic.ulBLength += ul;
           sendcopy (bCUPic);
         }
         break;

    // sensor 3  as blue color channel
    case ID_CUS_S3:
         if (CUPic1.ulBLength < PicMaxLen)
         {
           tblkCUPic1 bCUPic = CUPic1;
           for (ul = 0;
               (ul < ulLength) && ((bCUPic.ulBLength+ul) < PicMaxLen); ++ul)
           {
             bCUPic.aul[bCUPic.ulBLength+ul] |= ((ULONG)(aus[ul]>>2));
           }
           bCUPic.ulBLength += ul;
           sendcopy (bCUPic);
         }
         break;

    // sensor 1 as green color channel
    case ID_CUS_S1:
         if (CUPic0.ulGLength < PicMaxLen)
         {
           tblkCUPic0 bCUPic = CUPic0;
           for (ul = 0;
               (ul < ulLength) && ((bCUPic.ulGLength+ul) < PicMaxLen); ++ul)
           {
             bCUPic.aul[bCUPic.ulGLength+ul] |= (((ULONG)(aus[ul]>>2)) << 8);
           }
           bCUPic.ulGLength += ul;
           sendcopy (bCUPic);
         }
         break;

    // sensor 4 as green color channel
    case ID_CUS_S4:
         if (CUPic1.ulGLength < PicMaxLen)
         {
           tblkCUPic1 bCUPic = CUPic1;
           for (ul = 0;
               (ul < ulLength) && ((bCUPic.ulGLength+ul) < PicMaxLen); ++ul)
           {
             bCUPic.aul[bCUPic.ulGLength+ul] |= (((ULONG)(aus[ul]>>2)) << 8);
           }
           bCUPic.ulGLength += ul;
```

```
            sendcopy (bCUPic);
          }
          break;

    // sensor 2 as red color channel
    case ID_CUS_S2:
          if (CUPic0.ulRLength < PicMaxLen)
          {
            tblkCUPic0 bCUPic = CUPic0;
            for (ul = 0;
                (ul < ulLength) && ((bCUPic.ulRLength+ul) < PicMaxLen); ++ul)
            {
              bCUPic.aul[bCUPic.ulRLength+ul] |= (((ULONG)(aus[ul]>>2)) << 16);
            }
            bCUPic.ulRLength += ul;
            sendcopy (bCUPic);
          }
          break;

    // sensor 5 as red color channel
    case ID_CUS_S5:
          if (CUPic1.ulRLength < PicMaxLen)
          {
            tblkCUPic1 bCUPic = CUPic1;
            for (ul = 0;
                (ul < ulLength) && ((bCUPic.ulRLength+ul) < PicMaxLen); ++ul)
            {
              bCUPic.aul[bCUPic.ulRLength+ul] |= (((ULONG)(aus[ul]>>2)) << 16);
            }
            bCUPic.ulRLength += ul;
            sendcopy (bCUPic);
          }
          break;
  }
} // DecodeSensorPic
```