Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

M.M. de Ruiter

C-GKS,
a C implementation of GKS,
the graphical kernel system

Computer Science/Department of Interactive Systems Report CS-R8753 November

The Centre for Mathematics and Computer Science is a research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a nonprofit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O.).

C-GKS, a C Implementation of GKS, the Graphical Kernel System.

M.M. de Ruiter

Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB, Amsterdam, The Netherlands

GKS, the ISO international standard for 2D graphics sofware, is specified in a language independent form. This document contains the GKS language binding to C of the GKS datatypes and functions. Furthermore it contains remarks about the implemented GKS functions. The user of this document is supposed to be familiar with GKS.

1980 Math. Subject Classification : 69K32, 69K34, 69K30

Key Words & Phrases : Computer graphics, graphics systems, standardization

Table of Contents

Report CS-R8753
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

Introduction

Since 1985 GKS[1-3] is the ISO standard for 2D graphics. GKS is specified in a language-independent form. To enable the use of this standard in a UNIX™ environment, a C language binding has been developed.[4] This document contains both the C language binding to GKS, and some notes on how to use the GKS system and where to find the GKS libraries and headerfiles on the machines at our institute.

1. The C Binding for GKS.

This C language binding for GKS specifies how the abstract datatypes used in the GKS document are mapped onto datatypes supported by the C language and how the abstract function names and argument lists of the document are mapped onto C function specifications.

The C-GKS implementation is made available to graphics programmers via a set of libraries, in conjunction with a set of C datatypes.

1.1. GKS Environment

Some C-GKS functions use the path name of the 'gks home directory'. In this directory subdirectories are available in which all C-GKS headerfiles, libraries, files containing the workstation description tables, etc. can be found. Before running a gks application the environment variable **GKS** must be set to this directory, currently being "/usr/local/gks". As an example, for the "csh" this can be done via:

```
setenv GKS /usr/local/gks
```

1.2. Libraries

GKS object code modules can be loaded from the two C-GKS libraries. The library **libgks.a** contains the device independent GKS object code for the routines defined in the GKS 7.4 document, while the library **libgksdrive.a** contains the device dependent object code modules.

The two libraries can be found in the directory /usr/local/gks/lib .

1.3. Header Files

To enable the use of the C-GKS datatypes in a gks application, the file "cgks.h" is to be included. This file must be identical to the cgks.h file used at creating the two C-GKS libraries. The file cgks.h can be found in the directory /usr/local/gks/incl .

1.4. Available Workstations

Several workstation types can be handled by the C-GKS implementation. For each type of workstation a driver is developed to map the device independent GKS code to device dependent code. Furthermore for each of these workstations a GKS workstation description table is available in the C-GKS environment. Description tables of all available workstation are stored in a C-GKS specific format on the file "wsdcap", which can be found in the directory /usr/local/gks/etc .

As it is not known at a call of OPEN_GKS which workstation(s) will be opened, or which workstation description tables will be inquired, all available description tables are installed during OPEN_GKS. This might be time and memory consuming. To reduce the set of available workstation description tables a file called "cgksws" is read during OPEN_GKS to inquire which workstations description tables are to be made available. This file is expected to be in the current working directory. If the file is not available, all the description tables are made available to the application.

The content of the file "cgksws" is a set of strings. For each workstation required a string must be added as defined in table 1. These strings are identical to the 2, 3 or 4 character string used for each workstation on the file "wsdcap". The table also contains the workstation names to be used at a call of open_ws().

cgksws-id	Workstation	name for open_ws()
aed	AED 512	aed512
dmd	Olivetti 5620	dmd5620
tek	tektronix 4014	t4014
sigm	Sigmex 1600	sigm1600
sun	sun 3	sun3
aws	ibm 5080	aws
bb	ibm 7374 plotter	ibm7374
ver	versatec plotter	versatec
pic	ws generating 'pic' file	pic
psc	ws generating postscript file	postsc
mi	metafile input	mi
mo	metafile output	mo
wis	ws independent segment storage	wiss
foo	low level skeleton driver	f4711
bar	high level skeleton driver	bartek

Table 1. Workstations currently supported.

1.5. Data Types

One of the main features of the C-binding is the large set of data types. The C-binding specifies each different GKS data item as an individual type, using the set of GKS abstract types. These are themselves defined using C's basic types. This has the following advantages. The practical one is that this improves the capacity of lint, the C typechecking program, to find bugs in both the implementation and application programs. The aesthetic one is that it encourages application programmers to specify meaningful types for values they supply to GKS.

The user provided data types reside in the file cgkstype.h. Besides these types this file contains some datatypes used only inside the kernel.

What follows is an alphabetic list of all user relevant datatypes.

```
typedef struct {
        Bindex    fa_ix;          /* bundle index */
        Istyle    fa_is;          /* interior style */
        Sindex    fa_si;          /* style index */
        Cindex    fa_ci;          /* fill area colour */
        } AreaRep;                /* fill area representation */

typedef struct {
        Int       as_xsz, as_ysz;
        } ArrSiz;                 /* cell array size */

typedef Int Asflags;              /* integer containing 13 aspect source flags */

typedef Int  Bindex;              /* bundle index for the primitive representation bundles */

typedef enum { FALSE, TRUE } Bool;

typedef Real Charef;              /* character expansion factor */

typedef Real Charht;              /* character height */

typedef Real Charsp;              /* character spacing */

typedef Int  Choice;              /* parameter of request_choice, containing returned choice */

typedef Int  Cindex;              /* colour index */
```

```
typedef struct {
        Real    c_red;          /* red component of colour */
        Real    c_green;        /* green component of colour */
        Real    c_blue;         /* blue component of colour */
        } Colour;

typedef struct {
        Real    d_x, d_y;
        } Dc;                   /* device coordinate */

typedef enum {
        ASAP, BNIL, BNIG, ASTI
        } Defmode;              /* deferral mode */

typedef Int  Devno;             /* input device number */

typedef union {
        struct { /* locator record:  */
            Cindex  loc_col;    /* colour for locator devices */
            Real    loc_rat;    /* rectangle ratio (for pet = -1 only) */
            Seg     *loc_seg;   /* segment for dragging (for pet = -2 only) */
            } loc_rec;
        struct { /* stroke record: */
            Size    stk_bufsz;  /* buffer size */
            Int     stk_posn;   /* initial buffer editing position */
            Cindex  stk_col;    /* colour for stroke input device */
            } stk_rec;
        struct { /* valuator record: */
            Real    val_max;    /* maximum value */
            Real    val_min;    /* minumum value */
            Cindex  val_col;    /* colour for valuator input devices */
            } val_rec;
        struct { /* choice record: */
            Int     cho_nr;     /* number of choices */
            String  *cho_str;   /* set of choices (strings) */
            Int     *cho_nrs;   /* set of choices (numbers) */
            Cindex  cho_col;    /* colour for choice input devices */
            } cho_rec;
        struct { /* pick record: */
            Real    pik_rat;    /* a ratio within which pick succeeds */
            Cindex  pik_col;    /* colour of pick input device */
            Seg     *pik_seg;   /* segment for dragging */
            Wc      pik_pos;    /* cursor coordinate relative to seg. */
            } pik_rec;
        struct { /* string record: */
            Size    str_size;   /* input buffer size */
            Size    str_posn;   /* initial cursor position */
            Cindex  str_col;    /* colour of string input device */
            Cindex  str_bcol;   /* string input device background colour */
            } str_rec;
        } Drecord;              /* input data record */

typedef struct {
        Dc      d_ll, d_ur;     /* lower left and upper right */
        } Drect;                /* device rectangle */

typedef struct {
        Wrect   ea_rect;        /* area for error messages */
        } Earea;                /* Earea can be used by the user defined error function */

typedef Sint Ercode;            /* error code */

typedef Ercode (*Erhandle)();   /* error handling function */
```

```
typedef union {
        struct {                        /* selective bitplane read: */
            Int sbr_bitmask;            /* mask for selective bitplane read */
            } sbr_rec;
        struct {                        /* selective bitplane write: */
            Int sbw_bitmask;            /* mask for selective bitplane write */
            } sbw_rec;
        struct {                        /* transparancy: */
            Bool trp_onoff;             /* transparancy is set on/off */
            } trp_rec;
        } Es_data;                      /* data for GKS escape function */

typedef enum {
        ES_SELRBPL,                     /* selective read from bitplanes */
        ES_SELWBPL,                     /* selective write in bitplanes */
        ES_TRANSPARENT                  /* transparancy */
        } Es_func;                      /* escape function identifiers */

typedef struct {
        Wss     *ev_ws;                 /* workstation identifier */
        Devno   ev_devn;                /* input device number */
        Iclass  ev_class;               /* input class */
        Idata   ev_data;                /* input data */
        } Event;                        /* input event */

#define File FILE                       /* FILE is defined in stdio.h */

typedef struct {
        Tfont   fp_fo;                  /* font */
        Tprec   fp_pr;                  /* precision */
        } FoPr;                         /* text font and precision */

typedef struct {
        Gdpi    gd_id;                  /* gdp identifier */
        Int     gd_satt;                /* set of attributes; integer between 0 and 017 octal; a bit
                                         * each for polyline, polymark, text, fillarea */
        } Gdp;                          /* general drawing primitive information */

typedef enum {
        CIRCLE,
        ARC,
        DISC,                           /* solid circle */
        ELLIPSE,
        ELDISC,                         /* elliptical disc */
        ELARC,                          /* elliptical arc */
        MULPLINE,                       /* multi-polyline */
        HSPLINE,                        /* Hermite-spline */
        BSPLINE,                        /* B-spline */
        BZSPLINE,                       /* Bezier-spline */
        HSPL_CL,                        /* Closed Hermite-spline */
        BSPL_CL,                        /* Closed B-spline */
        BZSPL_CL,                       /* Closed Bezier-spline */
        HSPL_FA,                        /* Fill Area Hermite Spline */
        BSPL_FA,                        /* Fill Area B-Spline */
        BZSPL_FA                        /* Fill Area  Bezier Spline */
        } Gdpi;                         /* general drawing primitive identifier */
```

```
typedef struct {
    Wss         **gk_opws;   /* address of first of list of open workstations */
    Wss         **gk_actv;   /* address of first of list of active workstations */

    Asflags     gk_asf;      /* current 13 aspect source flags; see s_asflags() function */

    Bindex      gk_line;     /* current polyline bundle index */
    Ltype       gk_lrlt;     /* current global linetype */
    Lwidth      gk_lrlw;     /* current global line width */
    Cindex      gk_lrci;     /* current global polyline colour index */

    Bindex      gk_mark;     /* current polymarker bundle index */
    Mtype       gk_mrmt;     /* current global marker type */
    Msize       gk_mrms;     /* current global marker size */
    Cindex      gk_mrci;     /* current polymark colour index */

    Bindex      gk_text;     /* current text index */
    FoPr        gk_txfp;     /* current text font and precision */
    Charef      gk_chef;     /* current char expansion factor */
    Charsp      gk_chsp;     /* current character spacing */
    Cindex      gk_txci;     /* current text colour index */
    Charht      gk_chht;     /* current character height */
    Wc          gk_chup;     /* current character up vector */
    Charht      gk_chwd;     /* current current character width */
    Wc          gk_chbs;     /* current current character base vector */
    Path        gk_path;     /* current text path */
    Talign      gk_txal;     /* current text alignment */

    Bindex      gk_area;     /* current fill area index */
    Istyle      gk_fais;     /* current fill area interior style */
    Sindex      gk_fasi;     /* current fill area style index */
    Cindex      gk_faci;     /* current fill area colour index */

    Wc          gk_ptwd;     /* current pattern width vector */
    Wc          gk_ptht;     /* current pattern height vector */
    Wc          gk_ptref;    /* current pattern reference point */

    Pickid      gk_pikid;    /* current pick identifier */

    Ntran       **gk_ntran;  /* address of first of list of norm transf */
    Ntran       *gk_curnt;   /* address of current norm transf */
    Bool        gk_clip;     /* clipping indicator */
    Nrect       *gk_clprect; /* current cliprectangle */

    Seg         *gk_opsg;    /* open segment (NULL if no segment open) */
    Seginst     *gk_segs;    /* gk_segs->si_this is segment with lowest priority;
                             * gk_segs->si_prev is NULL */

    Quevent     *gk_queue;   /* input queue of event reports */
    Bool        gk_more;     /* more simultaneous events (TRUE = MORE) */
    } Gks;                   /* global statelist */

typedef struct {
    char        op_lev, ip_lev;/* output ('0', '1' or '2') and input level ('a', 'b' or 'c') */
    } GksLevl;               /* GKS level supported */

typedef struct {
    GksLevl     gd_lev;      /* level of GKS */
    Int         gd_nrwt;     /* number of available workstation types */
    Wsd         **gd_avwt;   /* list of available workstation description tables */
    Int         gd_nopws;    /* max number of open workstations */
    Int         gd_nactv;    /* max number of active workstations */
    Int         gd_nwsas;    /* max number of ws associated with a segment */
    Int         gd_nntran;   /* max normalization transformation number */
    } Gksd;                  /* global description table */
```

```
typedef struct {
        Bool        ge_erstate;     /* error state (FALSE = OFF) */
        Erhandle    ge_erh;         /* error handling function */
        Event       *ge_event;      /* overflow event */
        } Gkse;                     /* GKS error statelist */

typedef enum {
        GKCL, GKOP, WSOP, WSAC, SGOP
        } GksState;                 /* global state */

typedef enum {
        LOCATOR, STROKE, VALUATOR, CHOICE, PICK, STRING
        } Iclass;                   /* input class */

typedef union {
        Locate      ev_loc;         /* locator input data */
        Stroke      ev_stk;         /* stroke input data */
        Value       ev_val;         /* valuator input data */
        Choice      ev_cho;         /* choice input data */
        Pick        ev_pik;         /* pick input data */
        String      ev_str;         /* string input data */
        } Idata;                    /* input data, depending on input class */

typedef struct {
        Iclass      id_clas;        /* input class */
        Imode       id_mode;        /* input mode */
        Bool        id_echo;        /* TRUE for 'echo on' */
        Idata       id_ival;        /* initial value */
        Bool        id_stat;        /* initial status for CHOICE device */
        Pet         id_pet;         /* prompt/echo type */
        Drect       id_area;        /* echo area */
        Drecord     id_drec;        /* input data record */
        } Idevice;                  /* Idevice contains status information of input devices, as used by
                                     * the workstation description table and workstation statelist
                                     */

typedef enum {
        REQUEST, EVENT, SAMPLE
        } Imode;                    /* input mode */

typedef char *Grecord;              /* data record for gdp */

typedef struct {
        Int         i_x, i_y;
        } Ic;                       /* integer device coordinate */

typedef unsigned Int;

typedef enum {
        HOLLOW, SOLID, PATTERN, HATCH
        } Istyle;                   /* fill area interior style */

typedef Int  Itmtp;                 /* metafile item type */

typedef char *Itmrecord;            /* metafile item data record */

typedef struct {
        Bindex      lr_ix;          /* bundle index */
        Ltype       lr_lt;          /* line type */
        Lwidth      lr_lw;          /* line width */
        Cindex      lr_ci;          /* colour index */
        } LineRep;                  /* representation of a polyline */
```

```
typedef struct {
        Ntran     *loc_nt;        /* normalization transformation */
        Wc        loc_pt;         /* locator */
        } Locate;                 /* normalisation transformation number and locator
                                   * position as returned by locator input function
                                   */

typedef Sint Ltype;               /* line type */

typedef Real Lwidth;              /* line width */

typedef struct {
        Bindex    mr_ix;          /* bundle index */
        Mtype     mr_mt;          /* marker type */
        Msize     mr_ms;          /* marker size */
        Cindex    mr_ci;          /* colour index */
        } MarkRep;                /* polymark representation */

typedef Real Msize;               /* marker size */

typedef Sint Mtype;               /* marker type */

typedef struct {
        Real      n_x, n_y;
        } Nc;                     /* NDC coordinate */

typedef struct {
        Nc        n_ll, n_ur;     /* lower left and upper right */
        } Nrect;                  /* rectangle in NDC */

typedef struct {
        Wrect     nt_wind;        /* window of normalization transformation */
        Nrect     nt_view;        /* viewport of normalization transformation */
        } Ntran;                  /* ntr defined by window in WC, and viewport in NDC */

typedef enum {
        RIGHT, LEFT, UP, DOWN
        } Path;                   /* text path */

typedef struct {
        Bindex    pa_ix;          /* bundle index */
        Int       pa_n, pa_m;     /* number of columns and number of rows */
        Cindex    *pa_ci;         /* pointer to first of pa_n*pa_m colours, being colourindex of upperleft */
        } PattRep;                /* pattern representation */

typedef Sint Pet;                 /* prompt/echo type */

typedef struct {
        Seg       *pik_seg;       /* name of segment picked */
        Pickid    pik_pid;        /* pick identifier */
        } Pick;                   /* parameter of request pick, containing returned segment name and
                                   * pick identifier. a pik_seg pointer being NULL means NOPICK
                                   */
typedef Int  Pickid;              /* pick identifier */

typedef float Real;

typedef struct {
        .
        .
        .
        } Screen;
        /*
         * Typedef Screen defines the C-GKS di/dd interface. See cgkstype.h and the
         * section "how to install new driver" for the contents of this structure.
         */
```

```
typedef struct {
    Segname  sg_sn;          /* segment name (integer) */
    Wss      **sg_onws;      /* set of associated workstations, ended with NULL pointer */
    Tmat     sg_mat;         /* segment transformation matrix */
    Bool     sg_vis;         /* visibility (TRUE = visible) */
    Bool     sg_hil;         /* highlighting (TRUE = highlighted) */
    Segpri   sg_pri;         /* segment priority */
    Bool     sg_det;         /* segment detectability (TRUE = DETECTABLE) */
    Bhead    *sg_firstbh;    /* pointer to the first information block; internal use only */
    } Seg;                   /* segment statelist */

typedef struct SEGINST {
    struct SEGINST *si_next; /* previous segment (having lower priority) */
    struct SEGINST *si_prev; /* next segment (having higher priority) */
    Seg    *si_this;         /* this segment */
    } Seginst;               /* datatype for the segment administration. a double linked list
                              * is kept in the global statelist and each workstation statelist
                              * to administrate the available segments.
                              */

typedef Int  Segname;        /* segment name */

typedef Real Segpri;         /* segment priority */

typedef Sint Sindex;         /* fill area style index */

typedef int Sint;            /* signed integer */

typedef Int  Size;

typedef char *String;

typedef struct {
    Ntran    *stk_nt;        /* normalization transformation cwof stroke*/
    Int      stk_no;         /* number of points in stroke */
    Wc       *stk_pt;        /* list of points in stroke */
    } Stroke;                /* stroke event */
typedef enum {
    HNORMAL, ALEFT, CENTRE, ARIGHT
    } Talhor;                /* horizontal text alignment */

typedef struct {
    Talhor   ta_hor;         /* horizontal text alignment */
    Talver   ta_ver;         /* vertical text alignment */
    } Talign;

typedef enum {
    VNORMAL, TOP, CAP, HALF, BASE, BOTTOM
    } Talver;                /* vertical text alignment */

typedef struct {
    Bindex   tx_ix;          /* bundle index */
    FoPr     tx_fp;          /* font and precision */
    Charef   tx_chef;        /* character expansion factor */
    Charsp   tx_chsp;        /* character spacing */
    Cindex   tx_ci;          /* text colour index */
    } TextRep;               /* text representation */

typedef Sint Tfont;          /* text font */

typedef Real Tmat[2][3];     /* transformation matrix of 2 by 3 */

typedef enum {
    STR, CHAR, STROKEP
    } Tprec;                 /* text precision */

typedef Real Value;          /* a valuator value */
```

```
typedef struct {
        Real        w_x, w_y;
        } Wc;                       /* world coordinate */

typedef struct {
        Wc          w_ll, w_ur;
        } Wrect;                    /* rectangle in world coordinates */

typedef enum {
        OUTPUT, INPUT, OUTIN, WISS, MO, MI
        } Wscat;                    /* workstation category */

typedef struct {
/* wc */   Wscat       wd_cat;      /* workstation category */
/* wr */   Wsrov       wd_rov;      /* raster or vector display */
           String      wd_type;     /* type field */
           Screen      *wd_screen;  /* di/dd interface */
/* dm */   Int         wd_dmar;     /* dyn. mod. acc. for representations and transformation.
                                     * each entry is represented by one bit */
/* ds */   Int         wd_dmas;     /* dyn mod acc for segment attributes */
/* df */   Defmode     wd_defr;     /* default deferral mode */
/* dw */   Bool        wd_waitr;    /* implicit regeneration (TRUE = regeneration SUPPRESSED */
/* xm */   Dc          wd_metres;   /* device coordinate units (METRES = TRUE) */
/* xr */   Dc          wd_rsize;    /* upper right of disp. surface in device coord. units;
                                     * lowerleft = (0.0, 0.0) */
/* xi */   Ic          wd_isize;    /* disp. surface in integers; number of columns and rows */
/* ln */   Int         wd_nltyp;    /* number of available linetypes */
/* lt */   Ltype       *wd_ltyp;    /* list of available linetypes */
/* lm */   Int         wd_nlwth;    /* number of available linewidths */
/* lw */   Lwidth      *wd_lwth;    /* pointer to 3 linewidths (nom, min, max) */
/* li */   Int         wd_nline;    /* number of predef polyline bundles */
/* lr */   LineRep     *wd_line;    /* table of predefined polyline bundles */
/* mn */   Int         wd_nmktyp;   /* number of available marker types */
/* ma */   Mtype       *wd_mktyp;   /* list of available marker types */
/* ms */   Int         wd_nmksz;    /* number of available marker sizes */
/* mz */   Msize       *wd_mksz;    /* pointer to 3 marker sizes (nom, min, max) */
/* mi */   Int         wd_nmark;    /* number of predefined marker bundles */
/* mr */   MarkRep     *wd_mark;    /* list of predefined marker bundles */
/* tf */   Int         wd_nfopr;    /* number of text font and precision pairs */
/* tp */   FoPr        *wd_fopr;    /* list of text font and precision pairs */
/* te */   Int         wd_nchef;    /* number of available char expansion factors */
/* tx */   Charef      *wd_chef;    /* pointer to min. and max. char expansion factors */
/* tn */   Int         wd_nchht;    /* number of available character heights */
/* th */   Charht      *wd_chht;    /* pointer to min. and max. char heights */
/* ti */   Int         wd_ntext;    /* number of predefined text indices */
/* tr */   TextRep     *wd_text;    /* table of predefined text bundles */
/* fs */   Int         wd_nistyl;   /* number of available interior styles */
/* ft */   Istyle      *wd_istyl;   /* list of available interior styles */
/* fh */   Int         wd_nhatst;   /* number of hatch styles */
/* fc */   Sindex      *wd_hatst;   /* list of hatch styles */
/* fi */   Int         wd_narea;    /* number of predefined fill area indices */
/* fr */   AreaRep     *wd_area;    /* table of predefined fill area bundles */
/* pi */   Int         wd_npatt;    /* number of predefined pattern indices */
/* pr */   PattRep     *wd_patt;    /* table of predefined pattern indices */
/* cx */   Int         wd_ncix;     /* number of available colours or intensities */
/* ca */   Bool        wd_colav;    /* colour available */
/* cn */   Int         wd_ncol;     /* number of predefined colour reps */
/* cc */   Colour      *wd_col;     /* table of predefined colour representations */
/* gn */   Int         wd_ngdp;     /* number of gdp's */
/* gg */   Colour      *wd_gdp;     /* list of available gdp's */
```

```
/* nl  */    Sint        wd_mxpr;      /* max number of polyline bundle table entries; -1 means infinite */
/* nm */     Sint        wd_mxpm;      /* max number of polymark bundle table entries; -1 means infinite */
/* nt  */    Sint        wd_mxtx;      /* max number of text bundle table entries; -1 means infinite */
/* nf  */    Sint        wd_mxfa;      /* max number of fillarea bundle table entries; -1 means infinite */
/* np  */    Sint        wd_mxpa;      /* max number of patterns indicess; -1 means infinte */
/* nc  */    Sint        wd_mxco;      /* max number of colour indices; -1 means infinite */
/* ns  */    Int         *wd_nprio;    /* number of segment priorities supp */
/* ?n  */    Int         wd_ndevs[6];  /* 6 fields to store number of available input devices */
/* ?d  */    Iddescr     *wd_ddesc[6]; /* input device descriptions. ? is one of [aoueyi] */
}Wsd;                                  /* workstation description table */
/*
 * The 2-character codes in the comments in the Wsd structure are the codes as used for each field
 * during the installation of each description table. See "How to add a new driver".
 *
 * If the GKS grouping extension is included then some more fields
 * are included in this type definition. See the Grouping document.[5]
 */
typedef enum {
      VECTOR, RASTER, OTHERDIS
      } Wsrov;                         /* workstation type */
typedef struct {
   Wsd         *ws_wsd;                /* workstation description table;
                                        * can be seen as workstation type field */
   Wsis        *ws_wsis;               /* workstation internal statelist */
   Int         ws_id[3];               /* workstation identifiers (only used by Fortran layer) */
   File        *ws_ifile;              /* connection identifier; input file */
   File        *ws_ofile;              /* connection identifier; output file */
   char        ws_iname[DEVNM_SZ];     /* connection name; input file */
   char        ws_oname[DEVNM_SZ];     /* connection name; output file */
   Bool        ws_actv;                /* TRUE if workstation is active */
   Seginst     *ws_segs;               /* first Seginst has lowest prio; ws_segs -> si_next has higher prio */
   Defmode     ws_defr;                /* deferral mode */
   Bool        ws_waitr;               /* implicit regeneration mode; if TRUE then i.r. suppressed */
   Bool        ws_empty;               /* display surface empty */
   Bool        ws_newfr;               /* new frame action necessary at update */
   Bool        ws_trupd;               /* ws transf update state; notpending = FALSE */
   Nrect       ws_reqw;                /* requested workstation window */
   Nrect       ws_curw;                /* current workstation window */
   Drect       ws_reqv;                /* requested workstation viewport */
   Drect       ws_curv;                /* current workstation viewport */
   Int         ws_nline;               /* number of polyline bundle table entries */
   LineRep     *ws_line;               /* table of defined polyline bundles */
   Int         ws_nmark;               /* number of polymarker bundle table entries */
   MarkRep     *ws_mark;               /* table of defined polymarker bundles */
   Int         ws_ntext;               /* number of text bundle table entries */
   TextRep     *ws_text;               /* table of defined text bundles */
   Int         ws_narea;               /* number of fill area bundle table entries */
   AreaRep     *ws_area;               /* table of defined fill area bundles */
   Int         ws_npatt;               /* number of pattern bundle table entries */
   PattRep     *ws_patt;               /* table of defined pattern bundles */
   Int         ws_ncolr;               /* number of colour table entries */
   Colour      *ws_colr;               /* table of colour representations */
   Idevice     *ws_devs[6];            /* array of lists of all input device states */
} Wss;                                 /* workstation statelist */
 /*
  * if the GKS grouping extension is included then some more fields
```

```
 * are included in this type definition. See the Grouping document.[5]
 */

/*
 * The workstation type parameter is indicated by a variable of type Wstype, being a function pointer:
 */
typedef int (*Wstype)();          /* pointer to function returning int */
```

1.6. GKS function binding to C.

With each GKS function a list of the relevant typedefinitions is given. Many of the GKS type definitions are structured typedefinitions consisting of one or more of the following three basic types:

typedef unsigned **Int**; (unsigned integer)

typedef int **Sint**; (signed integer)

typedef float **Real**; (floating point number)

Furthermore an enumeration type is introduces for boolean values:

typedef enum { FALSE, TRUE } **Bool**;

1.6.1. Control functions

The GKS control functions and their bindings to C:

	Gks *
OPEN GKS	**open_gks(erh, era, sz, msgfile)**
	Erhandle erh;
	Erarea *era;
IN AMOUNT OF MEM UNITS	Size sz;
IN ERROR FILE	String msgfile;

Diagnostics:
The function name **erh** will be interpreted as the name of user supplied ERROR HANDLING procedure. If no function name is given, i.e. if a NULL pointer is given as parameter, the implementation delivered ERROR HANDLING function is called in case an error occurs.

The parameter **era** can be used by the user supplied error handling function. It is nowhere used in the implementation, and can thus be taken NULL.

The Size parameter **sz** can be used by a local memory manager as a (maximum) size of (a) block(s) of memory. In the current implementation the 'sz' parameter is nowhere used. All dynamic memory management is done via the system functions malloc(), calloc(), realloc() and free().

Errormessages will be sent to a file named **msgfile**. If no messages are written on the msgfile, the file is deleted at a call of close_gks(). If "msgfile" is a NULL string or a NULL pointer, then errors are written on the "stderr" file.

Returned value is of type '**Gks ***' being a pointer to the global statelist. This statelist is allocated by C-GKS and filled with the default values as indicated in the GKS document. The returned pointer can be used by the application program for inquiries of the global statelist.

All parameters can all be taken NULL or NULL pointers.

Relevant C-GKS typedefs:

typedef Ercode (*Erhandle)();/* name of error handling function */

typedef Sint Ercode;

typedef struct { } Erarea; /* empty type; not used in C-GKS */

typedef Int Size;

typedef char *String;

Errors supported: 1, 200, 301, -1, -5, -6

-1 :wsdcap file not found.

-5 :no entry on file called wsdcap for one of the workstation types. Entries on this file are: "aed", "aws", "bar", "bb", "bitm", "cgi", "dmd", "foo", "mi", "mo", "pic", "psc", "sigm", "sun", "tek", "ver", "wis"

-6 :format of information on wsdcap file is wrong.

CLOSE GKS **close_gks()**

Diagnostics:
All dynamically allocated statelists (global statelist, workstation description tables etc.) are freed.

Errors supported: 2

IN WORKSTATION ID	Wss *
OPEN WORKSTATION	**open_ws(ifile, ofile, wtype)**
IN CONNECTION ID	String ifile, ofile;
IN WORKSTATION TYPE	Wstype wtype;

Diagnostics:
The file names **ifile** and **ofile** are the input and output files for the workstation to be opened. A value of NULL, or an empty string, for ifile and/or ofile makes the C-GKS system use stdin and/or stdout for the I/O.

If the workstation is not the terminal the user is working at, the ifile and/or ofile parameter can be filenames to/from another terminal, e.g. "/dev/tty12".

If wstype is metafile_input (mi) 'ifile' must be the file name of the metafile. If wstype is metafile_output (mo) 'ofile' must be the file name of the metafile.

The parameter **wtype** must be a name of one of the available workstation types. Currently the following workstation types are supported:

aed512, aws, bartek, bitmap, cgkscgi, dmd5620, f4711, foo, ibm7374, mi, mo, pic, postsc, psfilter, sigm1600, sun2, sun3, t4014, versatec, wis .

(See the file cgkswstype.h in the "Incl" directory for all currently available workstation_type names).

Returned value of type **'Wss *'** is the pointer to the workstation statelist. This statelist is allocated by C-GKS, and filled partly with default values, partly with values taken from the workstation description table.

Relevant C-GKS types:

```
typedef int (*Wstype)(); /*pointer to function returning int*/
```

Errors supported: 8, 23, 24, 301, -4

-4 :can't find the workstation description table for the device mentioned by **wstype**.

CLOSE WORKSTATION	**close_ws(ws)**
IN WORKSTATION ID	Wss *ws;

Errors supported: 7, 25, 29

ACTIVATE WORKSTATION	**activate(ws)**
IN WORKSTATION ID	Wss *ws;

Errors supported: 6, 25, 29, 33, 35

DEACTIVATE WORKSTATION	**deactivate(ws)**
IN WORKSTATION ID	Wss *ws;

Errors supported: 3, 20, 30, 33, 35

CLEAR WORKSTATION	**clear(ws, always)**
IN WORKSTATION ID	Wss *ws;
IN CONTROL FLAG	Bool always;

Diagnostics:
A control flag **always** being TRUE means ALWAYS, being FALSE means CONDITIONALLY.

Errors supported: 6, 25, 33, 35

REDRAW ALL SEGMENTS ON WORKSTATION **redraw(ws)**
IN WORKSTATION ID Wss *ws;

Errors supported: 7, 25, 33, 35, 36

UPDATE WORKSTATION **update(ws, regen)**
IN WORKSTATION ID Wss *ws;
IN REGENERATION FLAG Bool regen;

Diagnostics:
A regeneration flag **regen** being TRUE means PERFORM, being FALSE means POSTPONE.

Errors supported: 7, 25, 33, 35, 36

SET DEFERRAL STATE **s_defer(ws, def, reg)**
IN WORKSTATION ID Wss *ws;
IN DEFERRAL MODE Defmode def;
IN IMPLICIT REG. MODE Bool reg;

Diagnostics:
Type Defmode is defined as follows:

```
typedef enum {
    ASAP, BNIL, BNIG, ASTI
    } Defmode;
```

A implicit regeneration flag of TRUE means SUPPRESSED, while FALSE means ALLOWED.

Errors supported: 7, 25, 33, 35, 36

MESSAGE **message(ws, s)**
IN WORKSTATION ID Wss *ws;
IN MESSAGE String s;

Diagnostics:
A message s will be sent to the workstation 'ws' in a workstation dependent way. It is only sent if the workstation is capable of handling messages, i.e. if the entry 's_msge' in the DI/DD interface, stored in ws -> wd_wsd -> wd_screen, is unequal to NULL.

Errors supported: 7

ESCAPE **escape(ef, ep)**
IN FUNCTION ID Es_func ef;
IN ESCAPE DATA RECORD Es_data *ep;

Diagnostics:
The escape function to be used is indicated by the variable **ef**, being of type Es_func.

```
typedef enum {
    ES_SELRBPL,         /* selective read from bitplanes */
    ES_SELWBPL,         /* selective write in bitplanes */
    ES_TRANSPARENT      /* transparancy */
    } Es_func;          /* escape function identifiers */
```

The escape function is invoked on all active workstations able to handle escape functions, i.e. workstations with an escape entry in the di/dd interface (Screen).

The following escape facilities are available:

1) ES_SELRBPL Selective bitplane read.

2) ES_SELWBPL Selective bitplane write.

3) ES_TRANSPARENT Set transparancy on/off

Parameter **ep** is a pointer to a user supplied data structure of type Es_data, of which the fields are as follows:

ES_SELRBPL:

For devices with bitplanes, capable of selectively read a set of the available bitplanes, a mask can be set via this escape facility.

For ES_SELRBPL type Es_data is as follows:

```
typedef union {
 struct {
        Int sbr_bitmask; /*mask to be set for selective bitplane read*/
        } sbr_rec;
} Es_data;
```

ES_SELWBPL:

For devices with a bitplane architecture, capable of selectively write a set of the available bitplanes, a mask can be set via this escape function.

For ES_SELRBPL type Es_data is as follows:

```
typedef union {
 struct {
        Int sbw_bitmask; /*mask to be set for selective bitplane write*/
        } sbw_rec;
} Es_data;
```

ES_TRANSPARENT:

For devices capable of setting transparency on/off a flag can be set via this escape function.

For ES_TRANSPARENT type Es_data is as follows:

```
typedef union {
 struct {
        Int trp_onoff; /* transparency flag */
        } trp_rec;
} Es_data;
```

Example:

```
Es_data wmask;
wmask.sbw_rec.sbw_bitmask = 0376;
escape(ES_SELWBPL, &wmask);
```

will cause the first bitplane of all appropriate workstations not longer to be written at following GKS calls sending output to the workstations.

Errors supported:8, 180, 182

1.6.2. Output functions

POLYLINE **polyline(sz, p)**
IN NUMBER OF POINTS Size sz;
IN COORDINATES OF POINTS Wc *p;

Diagnostics:
A polyline is created by connecting **sz** positions stored in a user supplied block, pointed to by Wc pointer **p**.

Relevant C-GKS types:

```
typedef struct {
        Real        w_x, w_y;
        } Wc;
```

Errors supported: 5, 100, 301

POLYMARKER **polymark(sz, p)**
IN NUMBER OF POINTS Size sz;
IN COORDINATES OF POINTS Wc *p;

Diagnostics:
A polymark is created by placing **sz** markers on places indicated by the pointer **p**. p is of type pointer to Wc.

Relevant C-GKS types:

```
typedef struct {
        Real        w_x, w_y;
        } Wc;
```

Errors supported: 5, 100, 301

TEXT **text(st, p)**
IN CHARACTER STRING String st;
IN STARTING POINT Wc *p;

Diagnostics:
A textstring st is placed at position **p**, applying the appropriate text attributes.

Relevant C-GKS types:

```
typedef char * String;

typedef struct {
        Real        w_x, w_y;
        } Wc;
```

Errors supported: 5, 301

FILL AREA **fillarea(sz, p)**
IN NUMBER OF POINTS Size sz;
IN COORDINATES OF POINTS Wc *p;

Diagnostics:

A fillarea is created by connecting **sz** positions indicated by the pointer **p**, where p is of type pointer to Wc. The last position is connected to the first position. The interior of the fillarea is filled according to the currently effective fillarea attributes.

Relevant C-GKS types:

```
typedef struct {
        Real        w_x, w_y;
        } Wc;
```

Errors supported: 5, 100, 301

CELL ARRAY	**cellarray(wr, nr_cols, nr_rows, ci)**
IN CELL RECTANGLE	Wrect *wr;
IN DIMS OF COLOR INDEX ARRAY	Size nr_cols, nr_rows;
IN COLOR INDEX ARRAY	Cindex *ci;

Diagnostics:
A cellarray is drawn, with points P and Q stored in the parameter **wr**. World coordinate P is stored in wr->wr_ll, and coordinate Q in wr->w_ur, both being of type 'Wc'.

The colour indices are stored in an array indicated by the pointer **ci**, where 'ci' is of type pointer to 'Cindex'. ci points to the first of n * m colourindices. These indices are expected to be ordered row wise. The first colour is the colour of the cell at lower left.

Relevant C-GKS types:

```
typedef struct {
        Wc    w_ll, w_ur;
        } Wrect;
typedef struct {
        Real    w_x, w_y;
        } Wc;
typedef Int Cindex;
```

Errors supported: 5, 91, 301

GENERALIZED DRAWING PRIMITIVE	**g_draw(n, p, id, m, gdr)**
IN NUMBER OF POINTS	Size n;
IN COORDINATES OF POINTS	Wc *p;
IN GDP IDENTIFIER	Gdpi id;
	Size m;
IN GDP DATA RECORD	Grecord gdr;

Diagnostics:
A generalized drawing primitive of type **id** is generated on all active workstations capable of generating the specified gdp.

Data relevant for the specified gdp is stored in **sz** worldcoordinates, pointed to by **p**, and in the user supplied gdp data record **gdr**. The byte size of this record is **m**.

The following general drawing primitives are supported:

CIRCLE
A circle is drawn. The number of coordinates to be given is 2, with the first coordinate being the centre of the circle and the second coordinate being a point on the circle.
No gdp data record is interpreted for this gdp type.

ARC
A circular arc is drawn. The number of coordinates to be given is 3, with the first coordinate being its centre, the second coordinate being the starting point of the arc, and the third coordinate being the end point of the arc.
No gdp data record is interpreted for this gdp type.

DISC
A solidly filled circle is drawn. The number of coordinates to be given is 2, with the first coordinate being the centre of the disc and the second coordinate being a point on the disc.
No gdp data record is interpreted for this gdp type.

ELLIPSE
An ellipse is drawn.
No gdp data record is interpreted for this gdp type.

ELDISC
A solid filled ellipse is drawn.
No gdp data record is interpreted for this gdp type.

ELARC
An elliptical arc is drawn.
No gdp data record is interpreted for this gdp type.

MULPLINE

A set of sz/2 polylines is drawn, where each polyline consist of 2 points. The first polyline interprets point p[0] and point p[1], the second polyline interprets p[2] and p[3], and so on. If the number of points 'sz' is odd, the last point in the array 'p' is not used.
No gdp data record is interpreted for this gdp type.

HSPLINE

A curved line (cubic Hermite Spline) is drawn through n points. Depending on the gdp data record, this curve is once or twice continuously differentiable.
The size of the gdp data record is sizeof(Int). The gdp data record is interpreted in the following way:

1. If *(Int *)gdr == 2, the curve is twice continuously differentiable;
2. In all other cases, the curve is once continuously differentiable;

BSPLINE

A curved line (cubic Basic Spline or B-Spline) is drawn, controlled by n points. This curve is twice continuously differentiable.
The gdp data record is not interpreted.

BZSPLINE (not yet implemented)

A curved line (cubic Bezier Spline) is controlled by n points, where n is 3k + 1. The line interpolates at any three control points, starting with the first one. This curve is twice continuously differentiable between two subsequent interpolation points and only continuous at the interpolation points. This closed curve is drawn according to the current POLYLINE attributes.
If n is not 3k + 1, the last or the last two control points are ignored.
The gdp data record is not interpreted.

HSPL_CL

A *closed* curved line (cubic Hermite Spline) is drawn through n points. Depending on the gdp data record, this curve is once or twice continuously differentiable. If the first and last point coincide, the last point is ignored.
The size of the gdp data record is sizeof(Int). The gdp data record is interpreted in the following way:

1. If *(Int *)gdr == 2, the curve is twice continuously differentiable;
2. In all other cases, the curve is once continuously differentiable;

BSPL_CL

A *closed* curved line (cubic Basic Spline or B-Spline) is drawn, controlled by n points. This curve is twice continuously differentiable. If the first and the last point coincide, the last point is ignored.
The gdp data record is not interpreted.

BZSPL_CL (not yet implemented)

A *closed* curved line (cubic Bezier Spline) is controlled by n points, where n is 3k. The line interpolates at any three control points, starting with the first one. This curve is twice continuously differentiable between two subsequent interpolation points and only continuous at the interpolation points. This closed curve is drawn according to the current POLYLINE attributes.
If the first and the last point coincide, the last point is ignored.
If n is not 3k, the last or the last two control points are ignored.
The gdp data record is not interpreted.

HSPL_FA

A *closed* curved line (cubic Hermite Spline) is going through n points. Depending on the gdp data record, this curve is once or twice continuously differentiable. The interior of this closed curve is filled according to the current FILL AREA attributes. If the first and the last point coincide, the last point is ignored.
The size of the gdp data record is sizeof(Int). The gdp data record is interpreted in the following way:

1. If *(Int *)gdr == 2, the curve is twice continuously differentiable;
2. In all other cases, the curve is once continuously differentiable;

BSPL_FA

A *closed* curved line (cubic Basic Spline or B-Spline) is controlled by n points. This curve is twice continuously differentiable. The interior of this closed curve is filled according to the current FILL AREA attributes. If the first and the last point coincide, the last point is ignored.
The gdp data record is not interpreted.

BZSPL_FA (not yet implemented)

A *closed* curved line (cubic Bezier Spline) is controlled by n points, where n is 3k. The line interpolates at any three control points, starting with the first one. This curve is twice continuously differentiable between

two subsequent interpolation points and only continuous at the interpolation points. The interior of this closed curve is filled according to the current FILL AREA attributes. If the first and the last point coincide, the last point is ignored. If n is not 3k, the last or the last two control points are ignored.
The gdp data record is not interpreted.

It is workstation dependent which attribute sets are interpreted for each gdp identifier. An inquiry function is available to find out which sets are used for each gdp on each workstation. Currently for each general drawing primitive the attributes sets used are identical on all available workstations. The following sets are globally used.

gdpi:	attributes used:	gdpi:	attributes used:
CIRCLE	polyline	BSPLINE	polyline
ARC	polyline	HSPL_CL	polyline
DISC	fillarea	BZSPLINE	polyline
ELLIPSE	polyline	BSPL_CL	polyline
ELDISC	fillarea	BZSPL_CL	polyline
ELARC	polyline	HSPL_FA	fillarea
MULPLINE	polyline	BSPL_FA	fillarea
HSPLINE	polyline	BZSPL_FA	fillarea

Whether a specific gdp is supported is workstation dependent. An inquiry is available to find out which gdp's are supported on a specific workstation.

The following table indicates which gdp's are supported on each workstation. If a gdp is supported the attributes used are indicated by one or more of the characters *l*, *m*, *t*, and/or *f* (polyline, polymark, text and/or fillarea attributes).

ws supports	CIRCLE	ARC	DISC	ELLIPSE	ELDISC	ELARC	MULPLINE
sigm1600	*l*	*l*	-	-	-	-	
aed512	*l*	-	*f*	-	-	-	-
t4014	-	-	-	-	-	-	-
versatec	*l*	-	*f*	-	-	-	-
aws	*l*	*l*	*f*	*l*	*f*	*l*	*l*
dmd5620	*l*	-	*f*	-	-	-	-
ibm7374	*l*	*l*	-	-	-	-	-
sun2	*l*	-	*f*	-	-	-	-

Relevant C-GKS types:

```
typedef Int Size;

typedef   enum {
          CIRCLE,
          ARC,
          DISC,       /* solid circle */
          ELLIPSE,
          ELDISC,     /* elliptical disc */
          ELARC,      /* elliptical arc */
          MULPLINE,   /* multi-polyline */
          HSPLINE,    /* Hermite-spline */
          BSPLINE,    /* B-spline */
          BZSPLINE,   /* Bezier-spline */
          HSPL_CL,    /* Closed Hermite-spline */
          BSPL_CL,    /* Closed B-spline */
          BZSPL_CL,   /* Closed Bezier-spline */
          HSPL_FA,    /* Fill Area Hermite Spline */
          BSPL_FA,    /* Fill Area B-Spline */
          BZSPL_FA    /* Fill Area  Bezier Spline */
          } Gdpi;     /* general drawing primitive identifier */

typedef char  *Grecord;
```

GKS errors supported: 5, 301

1.6.3. Output Attributes

1.6.3.1. Workstation Independent Primitive Attributes

SET POLYLINE INDEX
IN POLYLINE INDEX

s_pl_i(i)
Bindex i;

Relevant C-GKS types:
typedef Int Bindex;

GKS errors supported: 8, 60

SET LINETYPE
IN LINETYPE

s_lt(lt)
Ltype lt;

Diagnostics:
The linetype value is interpreted by the workstation driver. At this moment the following Linetypes are interpreted/supported:

LTSOLID	solid line
LTDASHED	dashed line
LTDOTTED	dotted line
LTDOTDASH	dotdashed line
LTBLANK	blank line
LTDOUBLE	double line

Relevant C-GKS types:
typedef Sint Ltype;

GKS errors supported: 8, 63

SET LINE WIDTH SCALEFACTOR
IN LINEWIDTH SCALE FACTOR

s_lw(lw)
Lwidth lw;

Diagnostics:
At this moment the linewidth is not interpreted.

Relevant C-GKS types:
typedef Real Lwidth;

GKS errors supported: 8

SET POLYLINE COLOUR INDEX
IN POLYLINE COLOUR INDEX

s_pl_ci(ci)
Cindex ci;

Relevant C-GKS types:
typedef Int Cindex;

GKS errors supported: 8

SET POLYMARKER INDEX
IN POLYMARKER INDEX

s_pm_i(i)
Bindex i;

Relevant C-GKS types:
typedef Int Bindex;

GKS errors supported: 8, 66

SET MARKER TYPE **s_mt(mt)**
IN MARKER TYPE Mtype mt;

Diagnostics:
The markertype value **mt** is interpreted by the workstation driver. At this moment in all drivers the following markertypes are interpreted/supported:

1	MTDOT	dot
2	MTPLUS	plus
3	MTASTERISK	asterisk
4	MTCIRCLE	circle
5	MTCROSS	cross

For markersize MTDOT the scale factor is not interpreted.

Relevant C-GKS types:
typedef Sint Mtype;

GKS errors supported: 8, 69

SET MARKER SIZE SCALE FACTOR **s_msz(msz)**
IN MARKER SIZE SCALE FACTOR Msize msz;

Relevant C-GKS types:
typedef Real Msize;

GKS errors supported: 8

SET POLYMARKER COLOUR INDEX **s_pm_ci(ci)**
IN POLYMARKER COLOUR INDEX Cindex ci;

Relevant C-GKS types:
typedef Int Cindex;

GKS errors supported: 8

SET TEXT INDEX **s_tx_i(i)**
IN TEXT INDEX Bindex i;

Relevant C-GKS types:
typedef Int Bindex;

GKS errors supported: 8, 72

SET TEXT FONT AND PRECISION **s_tx_fp(fp)**
IN TEXT FONT AND PRECISION FoPr *fp;

Diagnostics:
Text font HARDCHAR (i.e. fp->fp_fo = HARDCHAR) is the hard character set of the workstation.
Code is added to use the Berkeley fontpackage (raster oriented). If this package is available (and the implementation is installed with the BERKFONT_define set in cgkslocal.h) then the BF* fonts below are supported:

Tfont	**fontname and description**	**Tfont**	**fontname and description**
HARDCHAR	hard character set of the ws	BFSTAR_I16	stare.i.16
BFCLAR_R18	clarendon.18	BFSTAR_B16	stare.b.16
BFBODO_I10	bodoni.i.10	BFPLAY_R10	playbill.10
BFTIME_B10	times.b.10	BFDELE_B12	delegate.b.12
BFMETE_R12	meteor.r.12	BFFIX_R14	fix.14
BFMETE_I10	meteor.i.10		

Code is added to use the Hershey fontpackage (line oriented). If these fonts are available (and the implementation is installed with the HERSHEY_define set in cgkslocal.h) then the HF* fonts below are supported:

Tfont	Hershey font	Tfont	Hershey font
HFCOMP_R13	compl.r.13	HFDUPL_R21	dupl.roman
HFCOMP_I13	compl.i.13	HFTRIP_R21	tripl.roman
HFSIMP_R21	simpl.roman	HFTRIP_I21	tripl.italic
HFSIMP_S21	simpl.script	HFGOTH_G21	gothic.germ
HFCOMP_R21	compl.roman	HFGOTH_E21	gothic.engl
HFCOMP_I21	compl.italic	HFGOTH_I21	gothic.ital
HFCOMP_S21	compl.script		

Relevant C-GKS types:

```
typedef Sint Tfont;
typedef enum {
        STR, CHAR, STROKEP
        } Tprec;
typedef struct {
        Tfontfp_fo;
        Tprec       fp_pr;
        } FoPr;
```

GKS errors supported: 8, 75

SET CHAR. EXP. FACTOR — **s_ch_ef(f)**
IN CHARACTER EXP.FACTOR — Charef f;

Relevant C-GKS types:

```
typedef Real Charef;
```

GKS errors supported: 8

SET CHARACTER SPACING — **s_ch_sp(sp)**
IN CHARACTER SPACING — Charsp sp;

Relevant C-GKS types:

```
typedef Real Charsp;
```

GKS errors supported: 8

SET TEXT COLOUR INDEX — **s_tx_ci(ci)**
IN TEXT COLOUR INDEX — Cindex ci;

Relevant C-GKS types:

```
typedef Int Cindex;
```

GKS errors supported: 8

SET CHARACTER HEIGHT — **s_ch_ht(ht)**
IN CHARACTER HEIGHT — Charht ht;

Relevant C-GKS types:

```
typedef Real Charht;
```

GKS errors supported: 8

SET CHARACTER UP VECTOR
IN CHAR UP VECTOR

s_ch_up(v)
Wc *v;

Relevant C-GKS types:

```
typedef struct {
        Real        w_x, w_y;
        } Wc;
```

GKS errors supported: 8

SET TEXT PATH
IN TEXT PATH

s_tx_pt(pt)
Path pt;

Relevant C-GKS types:

```
typedef enum {
        RIGHT, LEFT, UP, DOWN,
        } Path;
```

GKS errors supported: 8

SET TEXT ALIGNEMENT
IN TEXT ALIGNMENT

s_tx_al(ta)
Talign *ta;

Relevant C-GKS types:

text alignment:

```
typedef     struct {
        Talhor      ta_hor;
        Talver      ta_ver;
        } Talign;
```

horizontal text alignment:

```
typedef     enum {
        HNORMAL, ALEFT, CENTRE, ARIGHT
        } Talhor;
```

vertical text alignment:

```
typedef     enum {
        VNORMAL, TOP, CAP, HALF, BASE, BOTTOM
        } Talver;
```

GKS errors supported: 8

SET FILL AREA INDEX
IN FILLAREA INDEX

s_fa_i(i)
Bindex i;

Relevant C-GKS types:

typedef Int Bindex;

GKS errors supported: 8, 80

SET FILL AREA INTERIOR STYLE **s_fa_is(is)**
IN FILLAREA INTERIOR STYLE Istyle is;

For simple devices all filling is generated in software. This can be time consuming, especially for PATTERN. On intelligent devices, having a high level DI/DD, filling is supposed to be done on the device.

For a HOLLOW fillarea the boundary is drawn in the current fillarea colour.

PATTERNs are calculated and generated in software, and are treated in a way similar to cellarrays. This includes that patterns can be rotated, scaled etc. For the available patterns see SET PATTERN REPRESENTATION (pa_rep()).

For interior style HATCH fillareas are filled according to the current style index. See SET FILL AREA STYLE INDEX for the possible values.

Relevant C-GKS types:

```
typedef enum {
        HOLLOW, SOLID, PATTERN, HATCH
        } Istyle;
```

GKS errors supported: 8

SET FILLAREA STYLE INDEX **s_fa_si(si)**
IN FILAREA STYLE INDEX Sindex si;

Diagnostics:
For interior style HATCH the following values for the interior style are supported on at least one device.

HSHOR	-1	horizontal lines
HSVER	-2	vertical lines
HSGRID	-3	grid of horizontal and vertical lines
HSTRELL	-4	trellis
HSRDIAG	-5	diagonal lines, going up from left to right
HSLDIAG	-6	diagonal lines, going down

Relevant C-GKS types: typedef Sint Sindex;

GKS errors supported: 8, 84

SET FILLAREA COLOURINDEX **s_fa_ci(ci)**
IN FILLAREA COLOUR INDEX Cindex ci;

Relevant C-GKS types:

```
typedef Int Cindex;
```

GKS errors supported: 8

SET PATTERN SIZE **s_patsiz(ps)**
IN PATTERN SIZE Wc *ps;

Relevant C-GKS types:

```
typedef struct {
        Real        w_x, w_y;
        } Wc;
```

GKS errors supported: 8

SET PATTERN REF. POINT **s_patpt(rp)**
IN PATTERN REF. POINT Wc *rp;

Relevant C-GKS types:

```
typedef struct {
        Real        w_x, w_y;
        } Wc;
```

GKS errors supported: 8

SET ASPECT SOURCE FLAGS **s_asflags(mask)**
IN LIST OF ASPECT SOURCE FLAGS Asflags mask;

Diagnostics:
The parameter 'mask' is an (octal) integer of which the 13 lowest bits indicate the 13 aspect source flags. A 0 bit means BUNDLED, a 1 bit means INDIVIDUAL. Initially all flags are set to BUNDLED.
The order of the flag bits is as follows:

01	linetype ASF	0100	text font and precision ASF
02	linewidth scalefactor ASF	0200	character expansion factor ASF
04	polyline col index ASF	0400	character spacing ASF
010	marker type ASF	01000	text colour index ASF
020	marker size scale factor ASF	02000	fill area interior style ASF
040	polymarker colour index ASF	04000	fillarea style index ASF
		010000	fill area colour index ASF

As an example s_asflags(017777) sets all flags to INDIVIDUAL, s_asflags(02) sets the linewidth flag to INDIVIDUAL and all other flags to BUNDLED.

Relevant C-GKS types:

```
typedef Int Asflags;
```

GKS errors supported: 8

SET PICK IDENTIFIER **s_pikid(id)**
IN PICK IDENTIFIER Pickid id;

Relevant C-GKS types:

```
typedef Int Pickid;
```

GKS errors supported: 8

1.6.3.2. Workstation Attributes (Representations)

SET POLYLINE REPRESENTATION	**pl_rep(ws, pl)**
IN WORKST IDENTIFIER	Wss *ws;
IN POLYLINE INDEX IN LINETYPE IN LINEWIDTH SCALE FACTOR IN COLOUR INDEX	LineRep *pl;

Diagnostics:
A user supplied data structure, pointed to by **pl**, and containing the representation values, is copied to the workstation statelist. The index of the representation is part of the structure, and therefore no index parameter is needed in the function call.
For remarks about the attributes see also the individual attribute setting functions.

Relevant C-GKS types:

```
typedef struct {
    Bindex    lr_ix;
    Ltype     lr_lt;
    Lwidth    lr_lw;
    Cindex    lr_ci;
    } LineRep;
typedef Int Bindex;
typedef Sint Ltype; /*see SET LINETYPE */
typedef Real Lwidth;
typedef Int Cindex;
```

GKS errors supported: 7, 25, 33, 35, 36, 63, 93

SET POLYMARK REPRESENTATION	**pm_rep(ws, mk)**
IN WORKST IDENTIFIER	Wss *ws;
IN POLYMARKER INDEX IN MARKER TYPE IN MARKER SIZE SCALE FACTOR IN COLOUR INDEX	MarkRep *mk;

Diagnostics:
A user supplied data structure, pointed to by **mk**, and containing the representation values, is copied to the workstation statelist. The index of the representation is part of the structure, and therefore no index parameter is needed in the function call.
For remarks about the attributes see also the individual attribute setting functions.

Relevant C-GKS types:

```
typedef struct {
    Bindex    mr_ix;
    Mtype     mr_mt;
    Msize     mr_ms;
    Cindex    mr_ci;
    } MarkRep;
```

GKS errors supported: 7, 25, 33, 35, 36, 69, 93

SET TEXT REPRESENTATION **tx_rep(ws, tx)**
IN WORKST IDENTIFIER Wss *ws;
IN TEXT INDEX
IN TEXT FONT
IN TEXT PRECISION
IN CHAR EXP FACTOR TextRep *tx;
IN CHAR SPACING
IN COLOUR INDEX

Diagnostics:
A user supplied data structure, pointed to by **tx**, and containing the representation values, is copied to the workstation statelist. The index of the representation is part of the structure, and therefore no index parameter is needed in the function call.
For remarks about the attributes see also the individual attribute setting functions.

Relevant C-GKS types:

```
typedef struct {
    Bindex    tx_ix;
    FoPr      tx_fp;
    Charef    tx_chef;
    Charsp    tx_chsp;
    Cindex    tx_ci;
    } TextRep;
typedef Int Bindex;
typedef struct {
    Tfont     fp_fo;
    Tprec     fp_pr;
    } FoPr;
typedef Sint Tfont;
typedef enum { STR, CHAR, STROKEP } Tprec;
typedef Real Charef;
typedef Real Charsp;
typedef Int Cindex;
```

GKS errors supported: 7, 25, 33, 35, 36, 76, 77, 93

SET FILL AREA REPRESENTATION **fa_rep(ws, fa)**
IN WORKST IDENTIFIER Wss *ws;
IN FILL AREA INDEX
IN INTERIOR STYLE
IN STYLE INDEX AreaRep *fa;
IN COLOUR INDEX
IN COLOUR INDEX

Diagnostics:
A user supplied data structure, pointed to by **fa**, and containing the representation values, is copied to the workstation statelist. The index of the representation is part of the structure, and therefore no index parameter is needed in the function call.
For remarks about the attributes see also the individual attribute setting functions.

Relevant C-GKS types:

```
typedef struct {
    Bindex    fa_ix;
    Istyle    fa_is;
    Sindex    fa_si;
    Cindex    fa_ci;
    } AreaRep;
typedef enum { HOLLOW, SOLID, PATTERN, HATCH } Istyle; typedef Sint Sindex;
typedef Int Cindex;
```

GKS errors supported: 7, 25, 33, 35, 36, 85, 86, 93

SET PATTERN REPRESENTATION	**pa_rep(ws, pa)**
IN WORKST IDENTIFIER	Wss *ws;
IN PATTERN ARRAY IN PATTERN INDEX IN DIM OF PATTERN ARRAY	PattRep *pa;

Diagnostics:
A user supplied data structure, pointed to by **pa**, and containing the representation values, is copied to the workstation statelist. The index of the representation is part of the structure, and therefore no index parameter is needed in the function call.
For remarks about the attributes see also the individual attribute setting functions.

Relevant C-GKS types:

```
typedef struct {
    Bindex    pa_ix;
    Int       pa_n, pa_m;
    Cindex    *pa_ci;
    } PattRep;
typedef Int Bindex;
tpyedef Int Cindex;
```

GKS errors supported: 7, 25, 33, 35, 36, 79, 90

SET COLOUR REPRESENTATION	**col_rep(ws, ci, col)**
IN WORKSTATION IDENTIFIER	Wss *ws;
IN COLOUR INDEX	Cindex ci;
IN COLOUR	Colour *col;

Diagnostics:
A call with the appropriate parameters will cause a structured field in a workstation dependent array to be filled with a red, blue and green value.

Relevant C-GKS types:

```
typedef struct {
        Real    c_red;
        Real    c_green;
        Real    c_blue;
        } Colour;
```

GKS errors supported: 7, 25, 33, 35, 36, 93, 96

1.6.4. Transformation Functions

Relevant C-GKS typedefs:

```
typedef struct {
    Wc   w_ll, w_ur;
    } Wrect; /* rectangle in world coordinates (window)*/
typedef struct {
    Nc   n_ll, n_ur;
    } Nrect; /* rectangle in normalized devices coordinates (viewport)*/
typedef struct {
    Wrect   nt_wind;
    Nrect   nt_view;
    } Ntran;
    /* normalization transformation, characterized by its window and viewport
     */
```

1.6.4.1. Normalization Transformations

SET WINDOW	**s_window(nt, wr)**
IN TRANSFORMATION NUMBER	Ntran *nt;
IN WINDOW LIMITS	Wrect *wr;

Diagnostics:

nt must be a pointer to one of the normalization transformations available via the global statelist. For example, if the global statelist pointer, returned by open_gks(), is 'gkss' then a call of s_window(gkss -> gk_ntran[i], wr) sets the Wrect of the ith normalization transformation to the contents of the Wrect pointed to by 'wr'.
It is a GKS error to try to change the first normalization tranformation having index 0.

GKS errors supported: 8, 50, 51

SET VIEWPORT	**s_viewport(nt, nr)**
IN TRANSFORMATION NUMBER	Ntran *nt;
IN VIEWPORT LIMITS	Nrect *nr;

Diagnostics:

nt must be a pointer to one of the normalization transformations available via the global statelist. For example, if the global statelist pointer, returned by open_gks(), is 'gkss' then a call of s_viewport(gkss -> gk_ntran[i], nr) sets the Nrect of the ith normalization transformation to the contents of the Nrect pointed to by 'nr'.
It is a GKS error to try to change the first normalization tranformation having index 0.

GKS errors supported: 8, 50, 51, 52

SET VIEWPORT INPUT PRIORITY	**s_vip(nt, rnt, hi)**
IN TRANSFORMATION NUMBER	Ntran *nt;
IN REFERENCE TRANSFORMATION NUMBER	Ntran *rnt;
IN RELATIVE PRIORITY	Bool hi;

Diagnostics:
Bool hi is TRUE for HIGHER, FALSE else. For 'nt' and 'rnt' see s_window().

GKS errors supported: 8, 50

SELECT NORMALIZATION TRANSFORMATION	**sel_cntran(nt)**
IN TRANSFORMATION NUMBER	Ntran *nt;

GKS errors supported: 8, 50

SET CLIPPING INDICATOR **s_clip(flg)**
IN CLIPPING INDICATOR Bool flg;

Diagnostics:
Bool flg is TRUE for CLIP, FALSE else.

GKS errors supported: 8

1.6.4.2. Workstation Transformations

SET WORKSTATION WINDOW **s_w_wind(ws, nr)**
IN WORKST ID Wss *ws;
IN WORKST WINDOW LIMITS Nrect *nr;

Diagnostics:
Nrect pointer 'nr' is a pointer to a user defined Nrect.

Relevant C-GKS typedefs:

```
typedef struct {
    Nc   n_ll, n_ur;
    } Nrect;
```

GKS errors supported: 7, 25, 33, 36, 51, 53

SET WORKSTATION VIEWPORT **s_w_view(ws, dr)**
IN WORKST ID Wss *ws;
IN WORKST VIEWPORT LIMITS Drect *dr;

Diagnostics:
Drect pointer 'dr' is a pointer to a user defined Drect.

Relevant C-GKS typedefs:

```
typedef struct {
    Real   d_x, d_y;
    } Dc; /* device coordinate*/
typedef struct {
    Dc   d_ll, d_ur;
    } Drect; /* rectangle in device coordinates */
```

GKS errors supported: 7, 25, 33, 36, 51, 54

1.6.5. Segment Functions.

1.6.5.1. Segment Manipulation Functions.

The following C-GKS typedefenitions are relevant for the GKS segments.

```
typedef Int Segname;
typedef Real Tmat[2][3];
typedef Real Segpri;

typedef   struct {
          Segname   sg_sn;       /*segment name */
          Wss       **sg_onws;   /*pointer to list of associated ws, followed by NULL ptr*/
          Tmat      sg_mat;      /* segment transformation matrix */
          Bool      sg_vis;      /* segment visibility */
          Bool      sg_hil;      /* segment highlighting */
          Segpri    sg_pri;      /* segment priority */
          Bool      sg_det;      /* segment detectability */
          Bhead     *sg_firstbh; /* pointer to first of linked list of segment blocks */
          } Seg;
```

CREATE SEGMENT — **Seg * newseg(n)**
SEGMENT NAME — Segname n;

Relevant C-GKS typedefinitions:
typedef Int Segname;

GKS errors supported: 3, 121

CLOSE SEGMENT — **closeg()**

Relevant C-GKS typedefs:
typedef Int Segname;

GKS errors supported: 4

RENAME SEGMENT — **renameseg(old, new)**
IN OLD SEGMENT NAME — Segname old;
IN NEW SEGMENT NAME — Segname new;

Relevant C-GKS typedefs:
typedef Int Segname;

GKS errors supported: 121, 122

DELETE SEGMENT — **zapseg(seg)**
IN SEGMENT NAME — Seg *seg;

Relevant C-GKS typedefinitions:
For typedef Seg see at beginning of this section.

GKS errors supported: 7, 122, 125

DELETE SEG. FROM W.S. — **delseg(ws, seg)**
IN WORKSTATION IDENTIFIER — Wss *ws;
IN SEGMENT NAME — Seg *seg;

Relevant C-GKS typedefinitions:
For typedef Seg see at beginning of this section.

GKS errors supported:7, 25, 33, 35, 122, 123, 125

ASSOCIATE SEG. WITH W.S. **sendseg(ws, seg)**
IN WORKSTATION IDENTIFIER Wss *ws;
IN SEGMENT NAME Seg *seg;

Relevant C-GKS typedefinitions:

GKS errors supported: 6, 25, 27, 33, 35, 122, 124

COPY SEG. TO W.S. **copyseg(ws, seg)**
IN WORKSTATION ID. Wss *ws;
IN SEGMENT NAME Seg *seg;

Relevant C-GKS typedefinitions:

For typedef **Seg** see at beginning of this section.

GKS errors supported: 6, 25, 27, 33, 35, 36, 122, 124

INSERT SEGMENT **insseg(seg, mat)**
IN SEGMENT NAME Seg *seg;
IN TRANSFORMATION MATRIX Tmat *mat;

Relevant C-GKS typedefinitions:

typedef Int Segname;
typedef Real Tmat[2][3];
typedef Real Segpri;
For typedef **Seg** see at beginning of this section.

GKS errors supported 122, 124, 125

1.6.5.2. Segment Attributes

TRANSFORM SEGMENT **transeg(seg, mat)**
IN SEGMENT NAME Seg *seg;
IN TRANSFORMATION MATRIX Tmat *mat;

Relevant C-GKS typedefinitions:

typedef Int Segname;
typedef Real Tmat[2][3];
typedef Real Segpri;
For typedef **Seg** see at beginning of this section.

GKS errors supported: 7, 122

SET VISIBILITY **s_segvis(seg, vis)**
IN SEGMENT NAME Seg *seg;
IN VISIBILITY Bool vis;

Relevant C-GKS typedefinitions:

For typedef **Seg** see at beginning of this section.

GKS errors supported: 7, 122

SET HIGHLIGHTING	**s_hilite(seg, hlt)**
IN SEGMENT NAME	Seg *seg;
IN HIGHLIGHTING	Bool hlt;

Diagnostics:
This function has no any effect on workstations with a low level di/dd interface. The function sends the appropriate parameters to the device via the screen entry for segment highlighting.

GKS errors supported: 7, 122

SET SEGMENT PRIORITY	**s_segpri(seg, pri)**
IN SEGMENT NAME	Seg *seg;
IN SEGMENT PRIORITY	Segpri pri;

Diagnostics:
The segment priority 'pri' can be taken between 0.0 and 1.0. 1.0 stands for the highest priority. At opening a segment a default priority of 0.0 is taken. At redrawing segments with equal priority the older segment will be shown first.

Relevant C-GKS typedefinitions:

typedef Int Segname;

typedef Real Tmat[2][3];

typedef Real Segpri;

For typedef **Seg** see at beginning of this section.

GKS errors supported: 7, 122, 126

SET DETECTABILITY	**s_detect(seg, det)**
IN SEGMENT NAME	Seg *seg;
IN DETECTABILITY	Bool det;

Relevant C-GKS typedefinitions:

typedef Int Segname;

typedef Real Tmat[2][3];

typedef Real Segpri;

For typedef **Seg** see at beginning of this section.

GKS errors supported: 7, 122

1.6.5.2.1. GKS extension: activate and deactivate segment [†]

ACTIVATE SEGMENT	**act_seg()**

If the segment state of the open segment is already active this function has no effect.

GKS errors supported: 4

DEACTIVATE SEGMENT	**deact_seg()**

If the segment state of the open segment is already inactive this function has no effect.

GKS errors supported: 4

[†] see section on "Local extension to GKS"

1.6.6. Input Functions

1.6.6.1. Initialize Input Functions

INITIALIZE LOCATOR

```
init_loc(ws, dn, val, pet, ea, dr)
Wss *ws;
Devno dn;
Locate *val;
Pet pet;
Drect *ea;
Drecord *dr;
```

Diagnostics:
The following GKS echotypes are supported for at least one device:

PE_LOC_1	1	device dependent firmware cursor (a tracking cross on most devices).
PE_LOC_CH	2	crosshair
PE_LOC_TC	3	tracking cross
PE_LOC_RB	4	rubber band
PE_LOC_RR	5	rubber rectangle

For locator initialization the Drecord **dr** contains a field indicating the colour to be used for cursor drawing rubber band, etc. A colour equal 0 may cause the cursor to be drawn in the background colour!

```
Cindex   dr->loc_rec.loc_col ;
```

GKS errors supported: 7, 25, 38, 51, 140, 141, 144, 145

INITIALIZE STROKE

```
init_ske(ws, dn, val, pet, ea, dr)
Wss *ws;
Devno dn;
Stroke *val;
Pet pet;
Drect *ea;
Drecord *dr;
```

Diagnostics:
The following GKS echotypes are supported for at least one device:

PE_SKE_1	1	a device cursor indicates a position. there is no echo of the whole stroke or part of the stroke.
PE_SKE_RB	4	connected rubber lines.

The Drecord **dr** contains for the init_ske() function the following fields:

```
Size dr->stk_rec.stk_bufsz     /* size of buffer available during stroke measure process*/
Real dr->stk_rec.stk_posn      /* initial position of cursor in stroke buffer*/
Cindex dr->stk_rec.stk_col     /* colour for prompt/echo*/
```

GKS errors supported: 7, 25, 38, 51, 140, 141, 144, 145

INITIALIZE VALUATOR

```
init_val(ws, dn, val, pet, ea, dr)
Wss *ws;
Devno dn;
Value *val;
Pet pet;
Drect *ea;
Drecord *dr;
```

Diagnostics:
The following GKS echotypes are supported for at least one device:

PE_VAL_1	1	Device dependent. On most devices same as type 2.
PE_VAL_TH	2	A 'thermometer' is shown having the size of the given echo area. A valuator request can be made by placing the cursor somewhere in the thermometer and firing the trigger. Depending on the echoarea the thermometer is laying or standing.
PE_VAL_DR	3	a digital representation can be typed in the echo area rectangle

The Drecord **dr** contains for the init_val() function the following fields:

```
Real   dr->val_rec.val_max /*maximum value */
Real   dr->val_rec.val_min /*minimum value */
Cindex dr->val_rec.val_col /*colour*/
```

GKS errors supported: 7, 25, 38, 51, 140, 141, 144, 145

INITIALIZE CHOICE

```
init_cho(ws, dn, val, pet, ea, dr)
Wss *ws;
Devno dn;
Choice *val;
Pet pet;
Drect *ea;
Drecord *dr;
```

Diagnostics:
The initial status parameter of this function, being OK or NOCHOICE, can be set by giving the intial choice value (val) a value either > 0 (OK) or a value of 0 (NOCHOICE).

The following prompt/echo types are supported:

PE_CHO_1	1	device dependent pet
PE_CHO_TC	3	a vertical menu with dr->cho_rec.cho_nr of choices, each rectangle in the menu containing a string from the string array from dr. Each item in menu is contained in small box.
PE_CHO_TC	4	a rectangle is shown in which a choice number can be typed.
PE_CHO_SG	5	menu consists of a segment
PE_CHO_HM	-1	a horizontal menu with dr->cho_rec.cho_nr of choices, each rectangle in the menu containing a string from the string array from dr.
PE_CHO_CM	-2	a vertical menu is of dr->cho_rec.cho_nr choices is shown, with each rectangle in the menu coloured according to the colourindices in the dr->cho_rec.cho_nrs (being a pointer to an array of ch_nr integers).
PE_CHO_VM2	-3	a vertical menu with dr->cho_rec.cho_nr of choices, each rectangle in the menu containing a string from the string array from dr. The selected item is drawn diapositive.

The Drecord **dr** contains for the init_cho() function the following fields:

```
Int     dr->cho_rec.cho_nr     /*number of choices */
String  *dr->cho_rec.cho_str   /*list of strings containing the choices */
Int     *dr->cho_rec.cho_nrs   /* list of choices, being numbers or colourindices */
Cindex  dr->cho_rec.cho_col    /*colour*/
```

GKS errors supported: 7, 25, 38, 51, 140, 141, 144, 145

INITIALIZE PICK

```
init_pik(ws, dn, val, pet, ea, dr)
Wss *ws;
Devno dn;
Pick *val;
Pet pet;
Drect *ea;
Drecord *dr;
```

Diagnostics:
The initial status parameter of this function, being OK or NOPICK, can be set by giving the intial pick value (pik) a value either > 0 (OK) or a value of 0 (NOPICK).

The following GKS echotypes are supported for at least one device:

PE_PIK_1	1	a device dependent cursor is prompted to allow the user to indicate the pick position
PE_PIK_NF	-1	no feedback is given after picking a segment.

The Drecord **dr** contains for the init_pik() function the following fields:

```
Real    dr->pik_rec.pik_rat  /* sizes of the small rectangle in which pick is recognized
                             * will be determined by this ratio of the viewport
                             */
Cindex  dr->pik_rec.pik_col  /* colour */
```

GKS errors supported: 7, 25, 38, 51, 140, 141, 144, 145

INITIALIZE STRING

```
init_str(ws, dn, val, pet, ea, dr)
Wss *ws;
Devno dn;
String val; /* initial string */
Pet pet;
Drect *ea;
Drecord *dr;
```

Diagnostics:
The following GKS echotypes are supported for at least one device:

PE_STR_1	1	The boundary of the echo area is drawn with the initial string. The cursor is placed under the initial buffer position. As soon as a character is typed the remaining part of the initial string is deleted, both on the screen and in the current value of the string measure process.
PE_STR_2	2	same as PE_STR_1, however whithout drawing the boundary of the echo rectangle.

The Drecord **dr** contains for the init_str() function the following fields:

```
Size    dr->str_rec.str_size   /* size of buffer containing initial value */
Size    dr->str_rec.str_posn   /* initial position in buffer */
Cindex  dr->str_rec.str_col    /* colour */
```

GKS errors supported: 7, 25, 38, 51, 140, 141, 144, 145

1.6.6.2. Set Input Mode functions

SET LOCATOR MODE

```
loc_mode(ws, dn, mo, echo)
Wss *ws;
Devno dn;
Imode mo;
Bool echo;
```

GKS errors supported: 7, 25, 38, 140

SET STROKE MODE

```
ske_mode(ws, dn, mo, echo)
Wss *ws;
Devno dn;
Imode mo;
Bool echo;
```

GKS errors supported: 7, 25, 38, 140

SET VALUATOR MODE

```
val_mode(ws, dn, mo, echo)
Wss *ws;
Devno dn;
Imode mo;
Bool echo;
```

GKS errors supported: 7, 25, 38, 140

SET CHOICE MODE

```
cho_mode(ws, dn, mo, echo)
Wss *ws;
Devno dn;
Imode mo;
Bool echo;
```

GKS errors supported: 7, 25, 38, 140

SET PICK MODE

```
pik_mode(ws, dn, mo, echo)
Wss *ws;
Devno dn;
Imode mo;
Bool echo;
```

GKS errors supported: 7, 25, 37, 140

SET STRING MODE

```
str_mode(ws, dn, mo, echo)
Wss *ws;
Devno dn;
Imode mo;
Bool echo;
```

GKS errors supported: 7, 25, 38, 140

REQUEST LOCATOR

```
Bool req_loc(ws, dn, val)
Wss *ws;
Devno dn;
Locate *val;
```

GKS errors supported: 7, 25, 38, 140, 141

REQUEST STROKE

```
Bool req_ske(ws, dn, val)
Wss *ws;
Devno dn;
Stroke *val;
```

Diagnostics:
Request stroke is implemented with 2 prompt echo types (see INIT STROKE). During the building of a stroke the 'x'-key is interpreted as a 'break', while the 'q'-key is interpreted as 'stop adding new points or replacing old points'.

GKS errors supported: 7, 25, 38, 140, 141

REQUEST VALUATOR

Bool req_val(ws, dn, val)
Wss *ws;
Devno dn;
Value *val;

GKS errors supported: 7, 25, 38, 140, 141

REQUEST CHOICE

Bool req_cho(ws, dn, val)
Wss *ws;
Devno dn;
Choice *val;

Diagnostics:
A returned Bool value of FALSE means NONE.
A returned Bool value of TRUE and *val == 0 means NOCHOICE.
A returned Bool value of TRUE and *val != 0 means OK.

GKS errors supported: 7, 25, 38, 140, 141

REQUEST PICK

Bool req_pik(ws, dn, val)
Wss *ws;
Devno dn;
Pick *val;

Diagnostics:
A returned Bool value of FALSE means NONE.
A returned Bool value of TRUE and val->pik_seg == 0 means NOPICK.
A returned Bool value of TRUE and val->pik_seg != 0 means OK.

GKS errors supported: 7, 25, 37, 140, 141

REQUEST STRING

Bool req_str(ws, dn, val)
Wss *ws;
Devno dn;
String val;

Diagnostics:
The returned string will be stored in memory delivered by the application program, pointed to by val. The size of this block of memory must be equal to the buffer size specified in the device data record, being filled with default values or set by init_str().

GKS errors supported: 7, 25, 38, 140, 141

SAMPLE LOCATOR

smp_loc(ws, dn, val)
Wss *ws;
Devno dn;
Locate *val;

Diagnostics:
This function is only available for workstations with a high level di/dd interface for the input functions. Currently there is no software support for asynchronous I/O.

GKS errors supported: 7, 20, 38, 140, 142

SAMPLE STROKE

smp_ske(ws, dn, val)
Wss *ws;
Devno dn;
Stroke *val;

Diagnostics:
This function is only available for workstations with a high level di/dd interface for the input functions. Currently there is no software support for asynchronous I/O.

GKS errors supported: 7, 20, 38, 140, 142

SAMPLE VALUATOR

smp_val(ws, dn, val)
Wss *ws;
Devno dn;
Value *val;

Diagnostics:
This function is only available for workstations with a high level di/dd interface for the input functions. Currently there is no software support for asynchronous I/O.

GKS errors supported: 7, 20, 38, 140, 142

SAMPLE CHOICE

smp_cho(ws, dn, val)
Wss *ws;
Devno dn;
Choice *val;

Diagnostics:
This function is only available for workstations with a high level di/dd interface for the input functions. Currently there is no software support for asynchronous I/O.

GKS errors supported: 7, 20, 38, 140, 142

SAMPLE PICK

smp_pik(ws, dn, val)
Wss *ws;
Devno dn;
Pick *val;

Diagnostics:
This function is only available for workstations with a high level di/dd interface for the input functions. Currently there is no software support for asynchronous I/O.

GKS errors supported: 7, 20, 37, 140, 142

SAMPLE STRING

smp_str(ws, dn, val)
Wss *ws;
Devno dn;
String val;

Diagnostics:
This function is only available for workstations with a high level di/dd interface for the input functions. Currently there is no software support for asynchronous I/O.

GKS errors supported: 7, 20, 38, 140, 142

AWAIT EVENT

Event *
await(sec)
Real sec;

Diagnostics:
This function is only available for workstations with a high level di/dd interface for the input functions. Currently there is no software support for asynchronous I/O.

If the returned Event pointer is NULL, there was no entry in the queue; NULL stands for the possible value NONE of the returned input class.

AWAIT EVENT is specified as a function returning a pointer to the current event. This is a structure containing the identification of the device generating the event, and a union of the types returned by each device class. GET <device class> functions are thus redundant, the application can refer to the appropriate fields of the current event structure directly.

Relevant C-GKS types:

```
typedef struct {
    Wss       *ev_ws;
    Devno     ev_devn;
    Iclass    ev_class;
    Idata     ev_data;
    } Event;  /* input event */
typedef struct QUEVENT {
    struct QUEVENT    *qu_next; /* next event */
    Bool      qu_last;          /* last of simultaneous events  */
    Event     *qu_event;        /* this event */
    } Quevent;                  /* event in queue */
```

GKS errors supported: 7, 151

FLUSH DEVICE EVENTS

flsh_ev(ws, class, dn)
Wss *ws;
Iclass class;
Devno dn;

Diagnostics:
This function is only available for workstations with
a high level di/dd interface for the input functions.
Currently there is no software support for asynchronous I/O.

GKS errors supported: 7, 20, 38, 140

There are no GET CLASS functions in the C-GKS library. In fact these functions are superfluous as the current event report contains already all the information returned with a GET CLASS function. This is possible due to the use of a 'union' type for 'Idata', containing the input data. A call of AWAIT EVENT returns an Event pointer, containing the current class and input data, structured in a way depending on the input class.

After a call of 'await()' a pointer to the current event is returned to the application program. Thus, after

```
Event *event;
event = await(sec);
```

The current event is determined by the event pointer. The current event class is determined by the field event->ev_class. If no current event is available then a NULL pointer is returned by await(). The effect of a GET CLASS function is equivalent to inspection of the information of the event->ev_data field.

/* LOCATOR */

If the event->ev_class = LOCATOR, then event->ev_data.ev_loc.loc_nt contains a pointer to a normalization transformation, and event->ev_data.ev_loc.loc_pt contains a locator position. It is a (not reported) error to try to get this information if the input class of the current event is unequal to LOCATOR.

/* STROKE */

If the event -> ev_class = STROKE, then event -> ev_data . ev_stk . stk_nt contains a pointer to a normalization transformation, event -> ev_data . ev_stk . stk_no contains the number of points, and event -> ev_data . ev_stk . stk_pt contains a pointer to the positions. It is a (not reported) error to try to get this information if the input class of the current event is unequal to STROKE.

/* VALUATOR */

If the event->ev_class = VALUATOR, then event->ev_data.ev_val contains a value. It is a (not reported) error to try to get this information if the input class of the current event is unequal to VALUATOR.

/* CHOICE */

If the event->ev_class = CHOICE, then event->ev_data.ev_cho contains a choice. It is a (not reported) error to try to get this information if the input class of the current event is unequal to CHOICE.

/* PICK */

If the event->ev_class = PICK, then event->ev_data.ev_pik contains a Pick field. It is a (not reported) error to try to get this information if the input class of the current event is unequal to PICK.

/* STRING */

If the event->ev_class = STRING, then event->ev_data.ev_str contains a String. It is a (not reported) error to try to get this information if the input class of the current event is unequal to STRING.

/* TYPEDEFS */

```
typedef   struct {
          Wss         *ev_ws;
          Devno       ev_devn;
          Iclass      ev_class;
          Idata       ev_data;
          } Event;

typedef   union {
          Locate      ev_loc;
          Stroke      ev_stk;
          Value       ev_val;
          Choice      ev_cho;
          Pick        ev_pik;
          String      ev_str;
          } Idata;
```

1.6.7. Metafile Functions

For the C-GKS metafiles the following type definitions are relevant:

```
typedef Int Itmtp;          /* metafile item type */
typedef char *Itmrecord;    /* metafile item data record */
typedef Int Size;           /*data record length */
```

WRITE ITEM TO GKSM	**wr_itm(ws, type, length, datrec)**
IN WORKSTATION IDENTIFIER	Wss *ws;
IN ITEM TYPE	Itmtp type;
IN ITEM DATA RECORD LENGTH	Size length;
IN ITEM DATA RECORD	Itmrecord datrec;

GKS errors supported: 7, 25, 30, 32, 160, 161.

GET ITEM TYPE FROM GKSM	**get_itmtp(ws, type, length)**
IN WORKSTATION IDENTIFIER	Wss *ws;
OUT ITEM TYPE	Itmtp *type;
OUT ITEM DATA RECORD LENGTH	Size *length;

Diagnostics:
The user supplied parameters 'type' and 'length' will contain on return the type of the current item, and length (in bytes) of the current item data record in the metafile.

GKS errors supported: 7, 25, 34, 162, 163.

READ ITEM FROM GKSM	**rd_itm(ws,maxrec,datrec)**
IN WORKSTATION IDENTIFIER	Wss *ws;
IN MAXIMUM DATA RECORD LENGTH	Size maxrec;
OUT ITEM DATA RECORD	Itmrecord datrec;

Diagnostics:
The contents of the current item in the metafile are copied to the user supplied record 'datrec'. The length of this data record is given by the parameter 'maxrec'. By inspecting first the current data record via a call of **get_itmtp()** a temporary record with the correct size can be supplied to a call of **rd_itm()**.

GKS errors supported: 7, 25, 34, 162, 165, 166.

INTERPRET ITEM	**intrp_itm(type, length, datrec)**
IN ITEM TYPE	Itmtp type;
IN ITEM DATA RECORD LENGTH	Size length;
IN ITEM DATA RECORD	Itmrecord datrec;

Diagnostics:
The function **intrp_itm()** takes the contents of the item data record supplied by the application programs (as delivered by **rd_itm()**, and causes appropriate changes in the set of GKS state variables and generates appropriate graphical output as determined by the metafiles specification.

The interpretation of an item can cause the same C-GKS errors as a call of the corresponding GKS function.

GKS errors supported: 7, 161, 164, 165, 167, 168.

The following items are currently not interpreted:

The ESCAPE item

1.6.8. Inquiry Macros

To inquire values of the various statelists GKS specifies 75 INQUIRE functions. Nearly all these inquire functions can be implemented as macros, producing in-line code. They all inquire a field of one of the available statelists, and return the value of that particular field. The argument 'fn' is a field name of the structure.

INQ GKS DESCRIPTION TABLE
inq_gkd(fn)

INQ GKS STATE
inq_gkss(fn)

INQ WORKST. DESCR. TABLE
inq_wsd(wsd, fn)
Wsd *wsd;

The wsd pointer can be obtained via wss->ws_wsd or the global list of Wsd pointers, stored in the global description table 'gksd', being a pointer to a structure of type Gksd. A utility function i_wsd_from_wst() is available to get a specific Wsd pointer.

INQ WORKSTATION STATE
inq_wss(ws, fn)
Wss *ws;

INQ SEGMENT STATE
inq_seg(sg, fn)
Seg *sg;

The C binding almost completely eliminates inquiry functions as such. The application is provided with the data type definitions of the structures implementing the various GKS statelists. Whenever a statelist is created, for example by OPEN_SEGMENT, the corresponding function returns the address of the newly created statelist structure.

The application could thus refer to the fields of the structure directly. It would seem attractive to replace the multitude of inquiry functions with a function for each statelist whose arguments were the statelist pointer and the field name within it, but:

- fieldnames alone are not valid expressions in C
- specifying the type of the value returned by such a function is problematic.

Instead the application invokes inquiry macros, looking like functions with these arguments. They are converted into in-line C by the preprocessor and thereby escape the restrictions on the use of field names. So that they may safely be called from error handling routines, GKS inquiry functions do not themselves generate errors. Instead, they set an output parameter indicating whether the value inquired for is available. In C, this precaution is not required. If the application has a valid statelist address, then the value is available. Otherwise, the value is both unavailable and inaccessible.
As an example the code

```
Bindex ix;
ix = inq_gkss(gk_line);
```

will get you the current polyline index (under the condition that the global state is not GKCL).

On systems not able to handle structure assignments, calls concerning structured type definitions can be done with the ASS macro:

```
Wc  upvec;
ASS(upvec , inq_gkss(gk_chup));
```

where ASS is a macro available in the C-GKS system to perform structure assignments.

To use the inquiry macros in an application the file cgksinq.h must be included in the application program. This will be done automatically if the file cgks.h is included in the application program.

See the previous chapters in this document or the file cgkstype.h for the definitions of all the types.

Some of the inquiry functions couldn't be implemented as C macro's, being:

INQ PIXEL ARRAY DIMENSIONS
INQ PIXEL ARRAY
INQ PIXEL
INQ TEXT EXTENT

They are implemented as functions.

In order to enable the use of the GKS inquiry functions as real functions, including all the proposed error checking, a C function binding is developed too.

1.6.8.1. Inquire operating state (via macros)

INQ OPERATING STATE	call:	return val of type:
OUT operating state value	**inq_state()**	GksState

1.6.8.2. Inquire GKS description table (macros)

INQ LEVEL OF GKS	call:	return val of type:
Out level of GKS	**inq_gkd(gd_lev)**	GksLevl
INQ LIST OF AV WS TYPES	call:	return val of type:
Out nr of available ws types	**inq_gkd(gd_nrwt)**	Int
Out list of available ws types	**inq_gkd(gd_avwt)**	Wsd **
INQ WS MAX NUMBERS	call:	return val of type:
Out max nr of simult open ws	**inq_gkd(gd_nopws)**	Int
Out max nr of simult active ws	**inq_gkd(gd_nactv)**	Int
Out max nr of ws assoc. with seg.	**inq_gkd(gd_nwsas)**	Int
INQ MAX NORM TRANSF NR	call:	return val of type:
Out max norm transf number	**inq_gkd(gd_nntran)**	Int(is array size)

1.6.8.3. Inquire GKS state list (macros)

INQ SET OF OPEN WORKSTATIONS	call:	return val of type:
Out number of open workstations		
Out list of open workstations	**inq_gkss(gk_opws)**	Wss **

The list contains pointers to workstation statelists, followed by a NULL pointer.

INQ SET OF ACTIVE WORKSTATIONS	call:	return val of type:
Out number of active workstations		
Out list of active workstations	**inq_gkss(gk_actv)**	Wss **

The return val points to a contiguous list of pointers to workstation statelists, followed by a NULL pointer.

INQ CURR PRIMITIVE ATTR VALUES	call:	return val of type:
Out curr polyline index	**inq_gkss(gk_line)**	Bindex
Out curr polymarker index	**inq_gkss(gk_mark)**	Bindex
Out curr text index	**inq_gkss(gk_text)**	Bindex
Out curr character height	**inq_gkss(gk_chht)**	Charht
Out curr character up vector	**inq_gkss(gk_chup)**	Wc
Out curr character width	**inq_gkss(gk_chwd)**	Charht
Out curr character base vector	**inq_gkss(gk_chbs)**	Wc
Out curr text path	**inq_gkss(gk_path)**	Path
Out curr text alignment	**inq_gkss(gk_txal)**	Talign
Out curr fill area index	**inq_gkss(gk_area)**	Bindex
Out curr pattern width vector	**inq_gkss(gk_ptwd)**	Wc

Out curr pattern height vector	**inq_gkss(gk_ptht)**	Wc
Out curr pattern reference pt	**inq_gkss(gk_ptref)**	Wc
INQ CURR PICK IDENT VALUE	call:	return val of type:
Out curr pick identifier	**inq_gkss(gk_pikid)**	Pickid
INQ CURR INDIV ATTRIB VALUES	call:	return val of type:
Out curr linetype	**inq_gkss(gk_lrlt)**	Ltype
Out curr linewidth scalefactor	**inq_gkss(gk_lrlw)**	Lwidth
Out curr polyline colour index	**inq_gkss(gk_lrci)**	Cindex
Out curr marker type	**inq_gkss(gk_mrmt)**	Mtype
Out curr marker size scale factor	**inq_gkss(gk_mrms)**	Msize
Out curr polymarker colour index	**inq_gkss(gk_mrci)**	Cindex
Out curr text font and precision	**inq_gkss(gk_txfp)**	FoPr
Out curr char expansion factor	**inq_gkss(gk_chef)**	Charef
Out curr character spacing	**inq_gkss(gk_chsp)**	Charsp
Out curr text colour index	**inq_gkss(gk_txci)**	Cindex
Out curr fill area interior style	**inq_gkss(gk_fais)**	Istyle
Out curr fill area style index	**inq_gkss(gk_fasi)**	Sindex
Out curr fill area colour index	**inq_gkss(gk_faci)**	Cindex
Out curr list of asp. source flags	**inq_gkss(gk_asf)**	Int
INQ CURR NORM TRANSF NR	call:	return val of type:
Out current normal transf nr	**inq_gkss(gk_curnt)**	Ntran *
INQ LIST OF NORM TRANSF NRS	call:	return val of type:
Out list of norm transf numbers	**inq_gkss(gk_ntran)**	Ntran **
INQ NORM TRANSFORMATION	call:	return val of type:
Out window limits		
Out viewport limits		
In i = normalization transform	**inq_gkss(gk_ntran[i])**	Ntran *
INQ CLIPPING	call:	return val of type:
Out clipping indicator	**inq_gkss(gk_clip)**	Bool
Out clipping rectangle	**inq_gkss(gk_curnt->nt_view)**	Nrect
INQ NAME OF OPEN SEGMENT	call:	return val of type:
Out name of open segment	**inq_gkss(gk_opsg)**	Seg *
INQ SET OF SEGM NAMES IN USE	call:	return val of type:
Out nr of segment names in use		
Out set of segment names in use	**inq_gkss(gk_segs)**	Seginst *

If the Grouping Extension [5]
is enabled then this inquiry only returns segments outside groups.

INQ MORE SIMULT EVENTS	call:	return val of type:
Out more simultaneous events	**inq_gkss(gk_more)**	Bool

relevant typedef:

```
typedef struct QUEVENT {
        struct QUEVENT      *qu_next;
        Bool                qu_last;
        Event               *qu_event;
        } Quevent;
```

A value of 'qu_last' being FALSE means there are more simultaneous events.

1.6.8.4. Inquire workstation state list (macros)

Some inquiry functions can have a parameter SET/REALIZED. In the following macro's all the values are taken from the workstation statelist and therefore the option SET is supposed.

INQ WS CONNECTION & TYPE	call:	return val of type:
Out connection identifier	**inq_wss(ws, ws_ifile)**	File * (inp taken from this file)
Out connection identifier	**inq_wss(ws, ws_iname)**	String (inp device name)
Out connection identifier	**inq_wss(ws, ws_ofile)**	File * (outp sent to this file)
Out connection identifier	**inq_wss(ws, ws_oname)**	String (outp device name)
Out workstation type	**inq_wss(ws, ws_wsd)**	Wsd *

INQ WORKST STATE	call:	return val of type:
Out workstation state	**inq_wss(ws, ws_actv)**	Bool

INQ WS DEF & UPD STATES	call:	return val of type:
Out deferral mode	**inq_wss(ws, ws_defr)**	Defmode
Out implicit regeneration mode	**inq_wss(ws, ws_waitr)**	Bool
Out display surface empty	**inq_wss(ws, ws_empty)**	Bool
Out new frame necess at update	**inq_wss(ws, ws_newfr)**	Bool

INQ LIST OF POLYLINE INDICES	call:	return val of type:
Out nr of pol. bundle table entries	**inq_wss(ws, ws_nline)**	Int
Out list of defined polyline indices	**inq_wss(ws, ws_line)**	LineRep *

INQ POLYLINE REPR	call:	return val of type:
Out polyline representation	**inq_wss(ws, ws_line[i-1])**	LineRep

INQ LIST OF POLYM INDICES	call:	return val of type:
Out nr of pom bundle table entr.	**inq_wss(ws, ws_nmark)**	Int
Out list of defined polym. indices	**inq_wss(ws, ws_mark)**	MarkRep *

INQ POLYMARKER REPR	call:	return val of type:
Out polymark representation	**inq_wss(ws, ws_mark[i-1])**	MarkRep

INQ LIST OF TEXT INDICES	call:	return val of type:
Out nr of txt bundle table entr.	**inq_wss(ws, ws_ntext)**	Int
Out list of defined text indices	**inq_wss(ws, ws_text)**	TextRep *

INQ TEXT REPRESENTATION	call:	return val of type:
Out text representation	**inq_wss(ws, ws_text[i-1])**	TextRep

INQ TEXT EXTENT	**Ercode textext(ws, pt, st, cat, tr)**
IN WORKST ID	Wss *ws;
IN TEXT POSITION	Wc *pt;
IN STRING	String st;
OUT CONCATENATION POINT	Wc *cat;
OUT TEXT EXTENT RECTANGLE	Wc *tr;

Diagnostics:
The function "inquire text extent" couldn't be implemented as a macro.

INQ LIST OF FILL AREA INDICES	call:	return val of type:
Out nr of fill bundle table entries	**inq_wss(ws, ws_narea)**	Int
Out list of defined fillarea indices	**inq_wss(ws, ws_area)**	AreaRep *

INQ FILLAREA REPRESENTATION	call:	return val of type:
Out fill area representation	**inq_wss(ws, ws_area[i-1])**	AreaRep

INQ LIST OF PATT INDICES	call:	return val of type:
Out nr of patt bundle table entr.	**inq_wss(ws, ws_npatt)**	Int
Out list of defined patt. indices	**inq_wss(ws, ws_patt)**	PattRep *
INQ PATTERN REPR	call:	return val of type:
Out pattern representation	**inq_wss(ws, ws_patt[i-1])**	PattRep
INQ LIST OF COL INDICES	call:	return val of type:
Out nr of colour table entries	**inq_wss(ws, ws_colr)**	Colour *
Out list of colour indices	**inq_wss(ws, ws_ncolr)**	Int
INQ COLOUR REPR	call:	return val of type:
In i = colour index		
Out colour	**inq_wss(wss, ws_colr[i])**	Colour
INQ WORKST TRANSF	call:	return val of type:
Out ws transform update state	**inq_wss(ws, ws_trupd)**	Bool
Out req workstation window	**inq_wss(ws, ws_reqw)**	Nrect
Out current workstation window	**inq_wss(ws, ws_curw)**	Nrect
Out req workstation viewp	**inq_wss(ws, ws_reqv)**	Drect
Out current workstation viewport	**inq_wss(ws, ws_curv)**	Drect
INQ SET OF SEG NAMES ON WS	call:	return val of type:
Out nr of segment names		
Out set of stored segms on this ws	**inq_wss(ws, ws_segs)**	Seginst *

INQ LOCATOR DEV STATE	call:	return value of type:
	inq_wss(ws, ws_devs[(Int)LOCATOR] [devno - 1])	Idevice
INQ STROKE DEV STATE	call:	return value of type:
	inq_wss(ws, ws_devs[(Int)STROKE] [devno - 1])	Idevice
INQ VALUATOR DEV STATE	call:	return value of type:
	inq_wss(ws, ws_devs[(Int)VALUATOR] [devno - 1])	Idevice
INQ CHOICE DEV STATE	call:	return value of type:
	inq_wss(ws, ws_devs[(Int)CHOICE] [devno - 1])	Idevice
INQ PICK DEVICE STATE	call:	return value of type:
	inq_wss(ws, ws_devs[(Int)PICK] [devno - 1])	Idevice
INQ STRING DEV STATE	call:	return value of type:
	inq_wss(ws, ws_devs[(Int)STRING] [devno - 1])	Idevice

1.6.8.5. Inquire workstation description table (macros)

Utility Function.

The C-GKS inquiry macros, having a workstation type as input parameter, expect a workstation description table pointer to indicate the workstation type. To obtain the workstation description table pointer belonging to a workstation type, indicated by a string, a utility function i_wsd_from_wst() is available.

INQ WSD FROM WORKS TYPE

```
Ercode i_wsd_from_wst(ws_type, wsd)
Wsd **wsd;
String ws_type;
```

Diagnostics:
On return the user supplied Wsd pointer **wsd** contains the workstation description table pointer belonging to workstation type **ws_type**. Currently the parameter **ws_type** can be one of the following strings:
"aed", "aws", "bar", "bb", "bitm", "cgi", "dmd", "foo", "mi", "mo", "pic", "psc", "sigm", "sun", "tek", "ver", "wis"

GKS errorindicators checked: 8

INQ WORKST. CATEGORY	call:	return val of type:
Out workstation category	**inq_wsd(wsd, wd_cat)**	Wscat
INQ WORKST. CLASSIF	call:	return val of type:
	inq_wsd(wsd, wd_rov)	Wsrov
INQ DISPLAY SPACE SIZE	call:	return val of type:
Out device coord units	**inq_wsd(wsd, wd_metres)**	Bool
Out max disp surf in dev crd units	**inq_wsd(wsd, wd_rsize)**	Dc
Out max disp surf in raster units	**inq_wsd(wsd, wd_isize)**	Ic
INQ DYNAMIC MOD OF WS ATTRIB	call:	return val of type:
	inq_wsd(wsd,wd_dmar)	Int

The field **wd_dmar** is an integer. If the rightmost, lowest, bit is the first bit then the bits in wd_dmar stand for (a bit being 0 means IRG, a bit being 1 means IMM.):

polyline bundle repr. changeable:	1th bit
polym bundle repr. changeable:	2nd bit
text bundle repr. changeable:	3rd bit
fill area bundle repr. changeable:	4th bit
pattern repr. changeable:	5th bit
colour repr. changeable:	6th bit
workst. transf. changeable:	7th bit

INQ DEFLT DEFER STATE VALUES	call:	return val of type:
def. value for deferral mode	**inq_wsd(wsd,wd_defr)**	Defmode
def. value for impl. regen. mode	**inq_wsd(wsd,wd_waitr)**	Bool (TRUE=suppressed)
INQ POLYLINE FACILITIES	call:	return val of type:
	inq_wsd(wsd, wd_nltyp)	Int
	inq_wsd(wsd, wd_ltyp)	Ltype *
	inq_wsd(wsd, wd_nlwth)	Int
	inq_wsd(wsd, wd_lwth)	Lwidth * (3 values)
	inq_wsd(wsd, wd_nline)	Int
INQ PREDEF. POLYLINE REPR	call:	return val of type:
	inq_wsd(wsd, wd_line[i - 1])	LineRep
INQ POLYMARKER FACILITIES	call:	return val of type:
	inq_wsd(wsd, wd_nmktyp)	Int
	inq_wsd(wsd, wd_mktyp)	Mtype *
	inq_wsd(wsd, wd_nmksz)	Int
	inq_wsd(wsd, wd_mksz)	Msize * (3 values)
	inq_wsd(wsd, wd_nmark)	Int
INQ PREDEF. POLYMARK. REP	call:	return val of type:
	inq_wsd(wsd, wd_mark[i - 1])	MarkRep

	call:	return val of type:
INQ TEXT FACILITIES	**inq_wsd(wsd, wd_nfopr)**	Int
	inq_wsd(wsd, wd_fopr)	FoPr *
	inq_wsd(wsd, wd_nchht)	Int
	inq_wsd(wsd, wd_chht)	Charht * (2 values)
	inq_wsd(wsd, wd_nchef)	Int
	inq_wsd(wsd, wd_chef)	Charef * (2 values)
	inq_wsd(wsd, wd_ntext)	Int
INQ PREDEF. TEXT REPR	call:	return val of type:
	inq_wsd(wsd, wd_text[i - 1])	TextRep
INQ FILLAREA FACILITIES	call:	return val of type:
	inq_wsd(wsd, wd_nistyl)	Int
	inq_wsd(wsd, wd_istyl)	Istyle *
	inq_wsd(wsd, wd_nhatst)	Int
	inq_wsd(wsd, wd_hatst)	Sindex *
	inq_wsd(wsd, wd_narea)	Int
INQ PREDEF. FILL AREA REPR	call:	return val of type:
	inq_wsd(wsd, wd_area[i - 1])	AreaRep
INQ PATTERN FACILITIES	call:	return val of type:
	inq_wsd(wsd, wd_npatt)	Int
INQ PREDEF. PATTERN REPR	call:	return val of type:
	inq_wsd(wsd, wd_patt[i - 1])	PattRep
INQ COLOUR FACILITIES	call:	return val of type:
	inq_wsd(wsd, wd_ncix)	Int
	inq_wsd(wsd, wd_colav)	Bool
	inq_wsd(wsd, wd_ncol)	Int
INQ PREDEF. COLOUR REPR	call:	return val of type:
	inq_wsd(wsd, wd_col[i])	Colour
INQ LIST OF AVAILABLE GDP'S	call:	return val of type:
	inq_wsd(wsd, wd_ngdp)	Int
	inq_wsd(wsd, wd_gdp)	Gdp *
INQ GEN. DRAWING PRIMITIVE	call:	return val of type:
	use previous inquiry	
INQ MAX LNGTH OF WS ST. TABLES	call:	return val of type:
	inq_wsd(wsd, wd_mxpr)	Sint
	inq_wsd(wsd, wd_mxpm)	Sint
	inq_wsd(wsd, wd_mxtx)	Sint
	inq_wsd(wsd, wd_mxfa)	Sint
	inq_wsd(wsd, wd_mxpa)	Sint
	inq_wsd(wsd, wd_mxco)	Sint
for these 6 entries a value -1 means infinite		
INQ NR OF SEG PRIO'S SUPPORTED	call:	return val of type:
	inq_wsd(wsd, wd_nprio)	Int

INQ DYN MOD OF SEGM ATTR	call:	return val of type:
	inq_wsd(wsd, wd_dmas)	Int

The field wd_dmas is an integer. If the bits of this integer are numbered from RIGHT to LEFT then the bits stand for (A bit being 0 means IRG, a bit being 1 means IMM.) :

segment transf. changeable:	
visib changeable from vis to invis:	2nd bit
visib changeable from invis to vis:	3nd bit
highlighting changeable:	4th bit
segment priority changeable:	5th bit
adding primitives to the open seg:	6th bit
segm deletion immediately visible:	7th bit

IQ NR AV LOGIC INP DEVS	call:	return val of type:
	inq_wsd(wsd, wd_ndevs[(Int)LOCATOR])	Int
	inq_wsd(wsd, wd_ndevs[(Int)STROKE])	Int
	inq_wsd(wsd, wd_ndevs[(Int)VALUATOR])	Int
	inq_wsd(wsd, wd_ndevs[(Int)CHOICE])	Int
	inq_wsd(wsd, wd_ndevs[(Int)PICK])	Int
	inq_wsd(wsd, wd_ndevs[(Int)STRING])	Int

IQ DFLT LOCAT DEV ST	call:	return val:
	inq_wsd(wsd, wd_ddescr[(Int)LOCATOR] [devno - 1])	Iddescr
IQ DFLT STROKE DEV ST	call:	return val:
	inq_wsd(wsd, wd_ddescr[(Int)STROKE] [devno - 1])	Iddescr
IQ DFLT VALUAT DEV ST	call:	return val:
	inq_wsd(wsd, wd_ddescr[(Int)VALUATOR] [devno - 1])	Iddescr
IQ DFLT CHOICE DEV ST	call:	return val:
	inq_wsd(wsd, wd_ddescr[(Int)CHOICE] [devno - 1])	Iddescr
IQ DFLT PICK DEV ST	call:	return val:
	inq_wsd(wsd, wd_ddescr[(Int)PICK] [devno - 1])	Iddescr
IQ DFLT STRING DEV ST	call:	return val:
	inq_wsd(wsd, wd_ddescr[(Int)STRING] [devno - 1])	Iddescr

1.6.8.6. Inquire segment statelist (macros)

IQ SET OF ASSOC WS	call:	return val of type:
	inq_seg(seg, sg_onws)	array of fields of type 'Wss *', followed by a NULL pointer. Maximum number of ptrs unequal NULL is gksd->gd_nwsas

IQ SEGMENT ATTRIBUTES	call:	return val of type:
segm transf mat	**inq_seg(seg, sg_mat)**	Tmat
visibilty	**inq_seg(seg, sg_vis)**	Bool
highlighting	**inq_seg(seg, sg_hili)**	Bool(FALSE=NORMAL)
segment priority	**inq_seg(seg, sg_pri)**	Segpri
detectability	**inq_seg(seg, sg_det)**	Bool

For the segment extension concerning activation or deactivation of segments, an extra inquiry function is added. This inquiry function is not available as an inline macro, but only as a C function.

1.6.8.7. Pixel Inquiries

These inquiries couldn't be implemented as macros.

IQ PIXEL ARRAY DIMENSIONS	**Ercode pxl_siz(ws, wr, sz)**
IN WORKST ID	Wss *ws;
IN 2 POINTS	Wrect *wr;
OUT DIM. OF PIXEL ARRAY	ArrSiz *sz;

IQ PIXEL ARRAY	**Ercode pxl_arr(ws, p, sz, inv, arr)**
IN WORKST ID	Wss *ws;
IN POINT	Wc *p;
IN DIM. OF COLOUR ARRAY	ArrSiz *sz;
OUT PRESENCE OF INV. VALUES	Bool *inv;
OUT COLOUR INDEX ARRAY	Cindex *arr;

IQ PIXEL	**Ercode pixel(ws, p, ci)**
IN WORKST ID	Wss *ws;
IN POINT	Wc *p;
OUT COLOUR INDEX	Cindex *ci;

1.6.8.8. Inquiry for the GKS error statelist.

IQ INPUT QUEUE OVERFLOW	call:	return val of type:
	inq_gke(ge_event)	Event *

1.6.9. Inquiry Functions

In order to enable the use of the GKS inquiry functions as real functions, including all the proposed error checking, all inquiries are also available as functions.

1.6.9.1. Inquire operating state (functions)

INQ OPERATING STATE	**Ercode i_state(gk)**
	GksState kz*gk;

Diagnostics:
On return the user supplied GksState parameter **gk** is filled with the current global state.

GKS errors supported: none

1.6.9.2. Inquire GKS description table (functions)

INQ LEVEL OF GKS	**Ercode i_level(level)**
	GksLevl *level;

Diagnostics:
On return the user supplied 'level' parameter is filled with the GKS level.

GKS errors supported: 8

INQ LIST OF AV WS TYPES

```
Ercode i_wstypes(nrwt, avwt)
Int      *nrwt;
Wsd      ***avwt;
```

Diagnostics:
The user supplied variable **nrwt** is filled with the number of available workstation types. The parameter **avwt** will point to a GKS system list of 'nrwt' pointers to all available workstation description tables. The Wsd field **wd_type** contains a string, indicating the workstation type.

GKS errorindicators checked: 8

INQ WS MAX NUMBERS

```
Ercode i_ws_max(nopws, nactv, nwsas)
Int      *nopws, *nactv, *nwsas;
```

Diagnostics:
On return the user supplied integers **nopws, nactv** and **nwsas** are filled with the maximum number of simultaneously open/ active workstations, and the maximum number of workstations associated with a segment.

GKS errorindicators checked: 8

INQ MAX NORM TR NUMBER

```
Ercode i_max_ntr(maxntrix)
Int      *maxntrix;
```

Diagnostics:
On return the user supplied integer **maxntrix** is filled with the maximum normalization transformation number, being 1 less than the total number of normalization transformations.

GKS errorindicators checked: 8

1.6.9.3. Inquire GKS state list (functions)

INQ SET OF OPEN WS

```
Ercode i_opws(nopws, opws)
Int      *nopws;
Wss      ***opws;
```

Diagnostics:
On return the user supplied integer **nopws** is filled with the number of open workstations. The pointer **opws** will point to a system owned array of nopws WSS pointers.

GKS errorindicators checked: 8

INQ SET OF ACTIVE WS

```
Ercode i_actv(nactv, actv)
Int      *nactv;
Wss      ***actv;
```

Diagnostics:
On return the user supplied integer **nactv** is filled with the number of active workstations. The pointer **actv** will point to a system owned array of nactv Wss pointers, representing the active workstations.

GKS errorindicators checked: 8

INQ CURR PRIM ATT VALUES

Ercode i_pr_atts(line, mark, text, chht, chup, chwd, chbs, path, txal, area, ptwd, ptht, ptref)
Bindex *line, *mark, *text;
Charht *chht, *chwd;
Wc *chup, *chbs;
Path *path;
Talign *txal;
Bindex *area;
Wc *ptwd, *ptht, *ptref;

Diagnostics:
On return the user supplied parameters are filled according to the current values in the global statelist.

GKS errorindicators checked: 8

INQ CURR PICK ID VALUE

Ercode i_pickid(pikid)
Pickid *pikid;

Diagnostics:
On return the user supplied pikid is filled with the current pick identifier value.

GKS errorindicators checked: 8

INQ CURR INDIV ATT VALUES

Ercode i_indv_atts(lrlt, lrlw, lrci, mrmt, mrms, mrci, txfp, chef, chsp, txci, fais, fasi, faci, asf)
Ltype *lrlt;
Lwidth *lrlw;
Cindex *lrci;
Mtype *mrmt;
Msize *mrms;
Cindex *mrci;
FoPr *txfp;
Charef *chef;
Charsp *chsp;
Cindex *txci;
Istyle *fais;
Sindex *fasi;
Cindex *faci;
Asflags *asf;

Diagnostics:
On return the user supplied parameters are filled according to the current values in the global statelist.

GKS errorindicators checked: 8

INQ CUR NTR

Ercode i_c_ntran(ntr)
Ntran **ntr;

Diagnostics:
On return the user supplied **ntr** pointer points to the systems current normalization transformation.

GKS errorindicators checked: 8

INQ LIST OF NTR

Ercode i_l_ntran(ntrix, sz)
Ntrix ntrix[];
Int sz;

Diagnostics:
On return the user supplied array of Ntrix pointers, having size **sz**, is filled with the normalization transformation indices, ordered by input priority. If the array cannot contain all the indices an errorcode -18 is returned, and the array is filled with only the first 'sz' indices.

GKS errorindicators checked: 8, -18

INQ A NORM TRANS

Ercode i_ntran(ix, ntr)
Ntrix ix;
Ntran *ntr;

Diagnostics:
On return the user supplied Ntran variable, pointed to by **ntr** is filled with the normalization transformation with index **ix.**

GKS errorindicators checked: 8, 50

INQ CLIPPING

Ercode i_clip(clip, nrect)
Bool *clip;
Nrect *nrect;

Diagnostics:
On return the user supplied parameter **clip** is filled with the clipping indicator, being TRUE for CLIP and FALSE for NOCLIP. The **nrect** pointer will point to the current clipping rectangle.

GKS errorindicators checked: 8

INQ NAME OF OPEN SEGMENT

Ercode i_open_seg(seg)
Seg **seg;

Diagnostics:
On return the user supplied segment pointer **seg** will contain the current segment pointer.

GKS errorindicators checked: 4

INQ SET OF SEGM NAMES IN USE

Ercode i_set_seg(nrnames, segnames, sn_sz)
Int *nrnames;
Segname *segnames;
Int sn_sz;

Diagnostics:
On return the user supplied integer nrnames will contain the number of segment names in use, while the user supplied array **segnames**, being of size **sn_sz**, will contain the segment names. If *nrnames is larger than sn_sz, only the first sn_sz entries are filled, and error -18 is returned.

GKS errorindicators checked: 7, -18

INQ MORE SIMULT. EVENTS

Ercode i_sim_events(more)
Bool *more;

Diagnostics:
On return the user supplied boolean **more** is filled, where TRUE means MORE, and FALSE means NOMORE.

GKS errorindicators checked: 7

1.6.9.4. Inquire workstation state list (functions)

INQ WS CONN AND TYPE

Ercode i_ws_con(ws, ifile, ofile, iname, oname, wstyp)
Wss *ws;
File **ifile, **ofile;
String *iname, *oname, *wstyp;

Diagnostics:
On return the user supplied file pointers **ifile** and **ofile** will contain the filepointers for input and output. The user supplied character pointers **iname** and **oname** will point to strings with the filenames as given at the open_ws() call. The parameter wstyp will point to a string indicating the workstation type Currently the following 'wstyp' strings are supported:
"aed", "aws", "bar", "bb", "bitm", "cgi", "dmd", "foo", "mi", "mo", "pic", "psc", "sigm", "sun", "tek", "ver", "wis"

GKS errorindicators checked: 7, 25, 33, 35

INQ WS STATE

Ercode i_ws_state(ws, active)
Wss *ws;
Bool *active;

Diagnostics:
On return the user supplied parameter **active** will be TRUE if the workstation is active, else FALSE.

GKS errorindicators checked: 7, 25, 33, 35

INQ WS DEFER AND UPD STATE

Ercode i_ws_dfup(ws, defr, wait, empty, newframe)
Wss *ws;
Defmode *defr;
Bool *wait, *empty, *newframe;

Diagnostics:
On return the user supplied parameter **defr** will be filled with the deferral mode. The parameter **waitr** will be TRUE if the implicit regeneration mode is suppressed, else FALSE. If the display surface is empty **empty** will be TRUE, else FALSE. If a newframe action is necessary the boolean **newframe** will be TRUE, else FALSE.

GKS errorindicators checked: 7, 25, 33, 35, 36

INQ LIST OF POLYLINE INDICES

Ercode i_l_plix(ws, nreps, indices, nr_indices)
Wss *ws;
Int *nreps;
Bindex *indices;
Int nr_indices;

Diagnostics:
On return the user supplied parameter **nreps** will be filled with the number of bundle table entries. A user supplied list of indices, pointed to by **indices**, being of size **nr_indices**, is filled with the defined primitive indices. If this array cannot hold all the indices only the first nr_indices fields are filled, and an error is returned.

GKS errorindicators checked: 7, 25, 33, 35, 36, -18

INQ POLYLINE REP

Ercode i_pl_rep(ws, ix, realized, rep)
Wss *ws;
Bindex ix;
Bool realized;
LineRep *rep;

Diagnostics:
On return the user supplied representation is filled with the inquired information.

GKS errorindicators checked: 7, 25, 33, 35, 36, 60, 61

INQ LIST OF POLYMARK INDICES

Ercode i_l_pmix(ws, nreps, indices, nr_indices)
Wss *ws;
Int *nreps;
Bindex *indices;
Int nr_indices;

Diagnostics:
On return the user supplied parameter **nreps** will be filled with the number of bundle table entries. A user supplied list of indices, pointed to by **indices**, being of size **nr_indices**, is filled with the defined primitive indices. If this array cannot hold all the indices only the first nr_indices fields are filled, and error -18 is returned.

GKS errorindicators checked: 7, 25, 33, 35, 36, -18

INQ POLYMARK REP

Ercode i_pm_rep(ws, ix, realized, rep)
Wss *ws;
Bindex ix;
Bool realized;
MarkRep *rep;

Diagnostics:
On return the user supplied representation is filled with the inquired information.

GKS errorindicators checked: 7, 25, 33, 35, 36, 66, 67

INQ LIST OF TEXT INDICES

Ercode i_l_txix(ws, nreps, indices, nr_indices)
Wss *ws;
Int *nreps;
Bindex *indices;
Int nr_indices;

Diagnostics:
On return the user supplied parameter **nreps** will be filled with the number of bundle table entries. A user supplied list of indices, pointed to by **indices**, being of size **nr_indices**, is filled with the defined primitive indices. If this array cannot hold all the indices only the first nr_indices fields are filled, and error -18 is returned.

GKS errorindicators checked: 7, 25, 33, 35, 36, -18

INQ TEXT REP

Ercode i_tx_rep(ws, ix, realized, rep)
Wss *ws;
Bindex ix;
Bool realized;
TextRep *rep;

Diagnostics:
On return the user supplied representation is filled with the inquired information.

GKS errorindicators checked: 7, 25, 33, 35, 36, 72, 73

INQ TEXT EXTENT
IN WORKST ID
IN TEXT POSITION
IN STRING
OUT CONCAT PT
OUT TEXT EXTENT RECTANGLE

Ercode textext(ws, pt, st, cat, tr)
Wss *ws;
Wc *pt;
String st;
Wc *cat;
Wc *tr;

Diagnostics:
On return concatenation point **cat**, and text rectangle **tr** are filled for the input String **st**.

GKS errorindicators checked: 7, 25, 39

INQ LIST OF FILL AREA INDICES

Ercode i_l_faix(ws, nreps, indices, nr_indices)
Wss *ws;
Int *nreps;
Bindex *indices;
Int nr_indices;

Diagnostics:
On return the user supplied parameter **nreps** will be filled with the number of bundle table entries. A user supplied list of indices, pointed to by **indices**, being of size **nr_indices**, is filled with the defined primitive indices. If this array cannot hold all the indices only the first nr_indices fields are filled, and error -18 is returned.

GKS errorindicators checked: 7, 25, 33, 35, 36, -18

INQ FILL AREA REP

Ercode i_fa_rep(ws, ix, realized, rep)
Wss *ws;
Bindex ix;
Bool realized;
AreaRep *rep;

Diagnostics:
On return the user supplied representation is filled with the inquired information.

GKS errorindicators checked: 7, 25, 33, 35, 36, 80 , 81

INQ LIST OF PATTERN INDICES

Ercode i_l_paix(ws, nreps, indices, nr_indices)
Wss *ws;
Int *nreps;
Bindex *indices;
Int nr_indices;

Diagnostics:
On return the user supplied parameter **nreps** will be filled with the number of bundle table entries. A user supplied list of indices, pointed to by **indices**, being of size **nr_indices**, is filled with the defined primitive indices. If this array cannot hold all the indices only the first nr_indices fields are filled, and error -18 is returned.

GKS errorindicators checked: 7, 25, 33, 35, 36, -18

INQ PATTERN REP

Ercode i_pa_rep(ws, ix, realized, rep)
Wss *ws;
Bindex ix;
Bool realized;
PattRep *rep;

Diagnostics:
On return the user supplied representation is filled with the inquired information.

GKS errorindicators checked: 7, 25, 33, 35, 36, 85, 88, 90

INQ LIST OF COLOUR INDICES

Ercode i_l_cix(ws, ncols, cols, cols_sz)
Wss *ws;
Int *ncols;
Cindex *cols;
Int cols_sz;

Diagnostics:
On return the user supplied parameter **ncols** contains the number of colour table entries. Th Cindex array, pointed to by **cols**, and being of size **cols_sz**, will be filled with the colour indices. If the array is not sufficient, only the first cols_sz entries will be filled, and error -18 is returned.

GKS errorindicators checked: 7, 25, 33, 35, 36, -18

INQ COLOUR REP

Ercode i_col_rep(ws, cix, realized, rep)
Wss *ws;
Cindex cix; /* colours are indexed up from 0! */
Bool realized;
Colour *rep;

Diagnostics:
On return the user supplied Colour variable **rep** is filled with the red , green, and blue value of colour with index **cix**.

GKS errorindicators checked: 7, 25, 33, 35, 36, 93, 94

INQ WS TRANSFORMATION

Ercode i_ws_tran(ws, pending, reqw, curw, reqv, curv)
Wss *ws;
Bool *pending;
Nrect *reqw;
Nrect *curw;
Drect *reqv;
Drect *curv;

Diagnostics:
On return the user supplied parameters **reqw, curw, reqv and curv** are filled with the rectangles asked for. If the transformation update state is PENDING , pending will be TRUE, else FALSE.

GKS errorindicators checked: 7, 25, 33, 35, 36

INQ SET OF SEGMENT pAMES ON WS

Ercode i_segs_ws(ws, nrnames, sname, sn_sz)
Wss *ws;
Int *nrnames;
Segnam*sname;
Int sn_sz;

Diagnostics:
On return the user supplied parameter **nrnames** is filled with the number of segment names on workstation. The user supplied array **sname** is filled with the segment names. If the array is not large enough to hold all the names, only the first sn_sz names are stored and an error -18 is returned.

GKS errorindicators checked: 7, 25, 33, 35, -18

INQ LOC DEV STATE

Ercode i_loc_state(ws, dn, realized, idev)
Wss *ws;
Devno dn;
Bool realized;
Idevice *idev;

Diagnostics:
On return the user supplied pointer **idev** will point to a system owned structure of type Idevice, containing all the input device information of device **dn.**

GKS errorindicators checked: 7, 25, 38, 140

INQ STROKE DEV STATE

Ercode i_ske_state(ws, dn, realized, idev)
Wss *ws;
Devno dn;
Bool realized;
Idevice *idev;

Diagnostics:
On return the user supplied pointer **idev** will point to a system owned structure of type Idevice, containing all the input device information of device **dn.**

GKS errorindicators checked: 7, 25, 38,140

INQ VALUATOR DEV STATE

Ercode i_val_state(ws, dn, realized, idev)
Wss *ws;
Devno dn;
Bool realized;
Idevice *idev;

Diagnostics:
On return the user supplied pointer **idev** will point to a system owned structure of type Idevice, containing all the input device information of device **dn.**

GKS errorindicators checked: 7, 25, 38, 140

INQ CHOICE DEV STATE

Ercode i_cho_state(ws, dn, realized, idev)
Wss *ws;
Devno dn;
Bool realized;
Idevice *idev;

Diagnostics:
On return the user supplied pointer **idev** will point to a system owned structure of type Idevice, containing all the input device information of device **dn**.

GKS errorindicators checked: 7, 25, 38, 140

INQ PICK DEV STATE

Ercode i_pik_state(ws, dn, realized, idev)
Wss *ws;
Devno dn;
Bool realized;
Idevice *idev;

Diagnostics:
On return the user supplied pointer **idev** will point to a system owned structure of type Idevice, containing all the input device information of device **dn**.

GKS errorindicators checked: 7, 25, 37, 140

INQ STRING DEV STATE

Ercode i_str_state(ws, dn, realized, idev)
Wss *ws;
Devno dn;
Bool realized;
Idevice *idev;

Diagnostics:
On return the user supplied pointer **idev** will point to a system owned structure of type Idevice, containing all the input device information of device **dn**.

GKS errorindicators checked: 7, 25, 38, 140

1.6.9.5. Inquire workstation description table (functions)

Utility Function.

The C inquiry functions having a workstation type as input parameter expect a workstation description table pointer to indicate the workstation type. To obtain the workstation description table pointer belonging to a workstation type indicated by a string, the utility function i_wsd_from_wst() is available.
Currently the parameter **ws_type** can be one of the following strings:
"aed", "aws", "bar", "bb", "bitm", "cgi", "dmd", "foo", "mi", "mo", "pic", "psc", "sigm", "sun", "tek", "ver", "wis"

INQ WSD FROM WORKS TYPE

Ercode i_wsd_from_wst(ws_type, wsd)
Wsd **wsd;
String ws_type;

Diagnostics:
On return the user supplied Wsd pointer field **wsd** contains the workstation description table pointer belonging to workstation type **ws_type**.

GKS errorindicators checked: 8

INQ WS CATEGORY

Ercode i_ws_cat(wsd, wscat)
Wsd *wsd;
Wscat *wscat;

Diagnostics:
On return the user supplied parameter **wscat** contains the workstation category.

GKS errorindicators checked: 8, 22, 23

INQ WS_CLASS

Ercode i_ws_class(wsd, wsclass)
Wsd *wsd;
Wsrov *wsclass;

Diagnostics:
On return the user supplied parameter **wsclass** contains the workstation class.

GKS errorindicators checked: 8, 22, 23, 39

INQ MAX DISP SURFACE

Ercode i_disp_size(wsd, metres, maxdc, isize)
Wsd *wsd;
Bool *metres;
Dc *maxdc;
Ic *isize;

Diagnostics:
On return the user supplied parameter **metres** will be filled with TRUE if device coordinates are in metres, else FALSE. The parameter **maxdc** will contain the maximum display surface size in device coordinates. The parameter **isize** will contain the maximum display surface size in raster units.

GKS errorindicators checked: 8, 22, 23, 31, 33, 36

INQ DYN MOD OF WS ATTS

Ercode i_dm_ws_atts(wsd, attflags)
Wsd *wsd;
Int *attflags;

Diagnostics:
On return the user supplied parameter **attflags** contains the workstation idea of the dynamic modification settings. Each bit contains an IMM flag, as follows:

polyline representation	if (attflags & 01) == 1	then IMM else IRG
polymark representation	if (attflags & 02) == 1	then IMM else IRG
text representation	if (attflags & 04) == 1	then IMM else IRG
fill area representation	if (attflags & 010) == 1	then IMM else IRG
pattern representation	if (attflags & 020) == 1	then IMM else IRG
colour representation	if (attflags & 040) == 1	then IMM else IRG
ws transformattion	if (attflags & 0100) == 1	then IMM else IRG

GKS errorindicators checked: 8, 22, 23, 39

INQ DEF DEFERRAL ST

Ercode i_dd_state(wsd, defmode, reg_suppr)
Wsd *wsd;
Defmode *defmode;
Bool *reg_suppr;

Diagnostics:
On return the user supplied parameter **defmode** will contain the default deferral mode, while the parameter **reg_suppr** will be TRUE if the default value for implicit regeneration mode will be SUPPRESSED, else FALSE.

GKS errorindicators checked: 8, 22, 23, 39

INQ POLYLINE FAC

Ercode i_pl_fac(wsd, nltyp, ltyp, nlwth, lwidths, nline)
Wsd *wsd;
Int *nltyp;
Ltype **ltyp;
Int *nlwth;
Lwidth **lwidths;
Int *nline;

Diagnostics:
On return the user supplied parameter **nltyp** will contain the number of available linetypes. The pointer **ltyp** will point to a system owned list of *ltyp available linetypes. The number of linewidths will be stored in **nlwth**. The Lwidth pointer **lwidths** will point to a system owned list of 3 Linewidth's representing the nominal, minimal and maximal linewidth. The parameter **nline** will contain the number of predefined representation.

GKS errorindicators checked: 8, 22, 23, 39

INQ PREDEF POLYLINE REP

Ercode i_pr_plrep(wsd, ix, rep)
Wsd *wsd;
Bindex ix;
LineRep *rep;

Diagnostics:
On return the user supplied structure **rep** is filled with the predefined representation for index **ix.**

GKS errorindicators checked: 8, 22, 23, 39, 60, 62

INQ POLYMARKER FAC

Ercode i_pm_fac(wsd, nmtyp, mtyp, nmsz, msizes, nmark)
Wsd *wsd;
Int *nmtyp;
Mtype **mtyp;
Int *nmsz;
Msize **msizes;
Int *nmark;

Diagnostics:
On return the user supplied parameter **nmtyp** will contain the number of available marker types. The pointer **mtyp** will point to a system owned list of *mtyp available marker types. The number of available marker sizes will be stored in **nmsz**. The Mszie pointer **msizes** will point to a system owned list of 3 Linewidth's representing the nominal, minimal and maximal marker size. The parameter **nmark** will contain the number of predefined representation.

GKS errorindicators checked: 8, 22,23, 39

INQ PREDEF POLYMARK REP

Ercode i_pr_pmrep(wsd, ix, rep)
Wsd *wsd;
Bindex ix;
MarkRep *rep;

Diagnostics:
On return the user supplied structure **rep** is filled with the predefined representation for index **ix.**

GKS errorindicators checked: 8, 22, 23, 39, 66, 68

INQ TEXT FACILITS

```
Ercode i_tx_fac(wsd, nfopr, foprs, nchht, chhts,
                nchef, chefs, ntext)
Wsd     *wsd;
Int     *nfopr, *nchht, *nchef, *ntext;
FoPr    **foprs;
Charef  **chefs;
Charht  **chhts;
```

Diagnostics:
On return the user supplied parameter **nfopr** is filled with the number of font and precision pairs. **foprs** points to the systems array of nfopr font and precisions. **nccht** contains the number of available character heights. **chhts** points to a array of 2 elements containing the minimimal and maximal character height. The parameter **nchef** contains the number of available character expansion factors, while the parameter **chefs** points to an array of 2 character expansion factors (minimal and maximal exp. factor). The number of predefined representations is stored in **ntext.**

GKS errorindicators checked: 8, 22, 23, 39

INQ PREDEF TEXT REP

```
Ercode i_pr_txrep(wsd, ix, rep)
Wsd     *wsd;
Bindex  ix;
TextRep *rep;
```

Diagnostics:
On return the user supplied structure **rep** is filled with the predefined representation for index **ix.**

GKS errorindicators checked: 8, 22, 23, 39, 72, 74

INQ FILL AREA FACILIT

```
Ercode i_fa_fac(wsd, nistyl, istyles, nhatst,
                hatstyles, narea)
Wsd     *wsd;
Int     *nistyl, *nhatst, *narea;
Istyle  **istyles;
Sindex  **hatstyles;
```

Diagnostics:
On return the user supplied parameters **nistyl, nhatst** and **narea** are filled with the number of available fill area interior styles, available hatch styles and predefined fill area representations. **istyles** points to the system list of available fill area styles, while **hatstyles** points to the systems list of available hatch styles.

GKS errorindicators checked: 8, 22, 23, 39

INQ PREDEF FILL AREA REP

```
Ercode i_pr_farep(wsd, ix, rep)
Wsd     *wsd;
Bindex  ix;
AreaRep *rep;
```

Diagnostics:
On return the user supplied structure **rep** is filled with the predefined representation for index **ix.**

GKS errorindicators checked: 8, 22, 23, 39, 80, 82

INQ PATTERN FACILITES

```
Ercode i_pat_fac(wsd, npatt)
Wsd     *wsd;
Int     *npatt;
```

Diagnostics:
On return the user supplied parameter **npatt** is filled with the number of predefined pattern representations.

GKS errorindicators checked: 8, 22, 23, 39

INQ PREDEF PATT REP

Ercode i_pr_parep(wsd, ix, rep)
Wsd *wsd;
Bindex ix;
PattRep *rep;

Diagnostics:
On return the user supplied structure **rep** contains the predefined representation for index **ix**.

GKS errorindicators checked: 8, 22, 23, 39, 85, 89, 90

INQ COL FACILITIES

Ercode i_col_fac(wsd, ncix, colav, ncol)
Wsd *wsd;
Int *ncix;
Bool *colav;
Int *ncol;

Diagnostics:
On return the user supplied parameter **ncix** contains the number of available colour indices. **colav** is TRUE is colour is available, else FALSE. The number of predefined colour representations is stored in the parameter **ncol**.

GKS errorindicators checked: 8, 22, 23, 39

INQ PREDEF COLOUR REP

Ercode i_pr_colrep(wsd, cix, col)
Wsd *wsd;
Cindex cix;
Colour *col;

Diagnostics:
The user supplied structure **col** is filled with the colour repr. for index **cix**.

GKS errorindicators checked: 8

INQ LIST OF AV GDP's

Ercode i_l_gdps(wsd, ngdp, gdp)
Wsd *wsd;
Int *ngdp;
Gdp **gdp;

Diagnostics:
On return the user supplied parameter **ngdp** contains the number of available gdp's. **gdp** points to the systems array of size 'ngdp', containing the available gdp's.

GKS errorindicators checked: 8, 22, 23, 39

INQ GEN DRAW PRIM

Ercode i_gdp(wsd, gdpi, nr_sets, atts)
Wsd *wsd;
Gdpi gdpi;
Int *nr_sets;
Int *atts;

Diagnostics:
On return the user supplied parameter **nr_sets** contains the number of sets of attributes used. **atts** will contain a mask for the sets of attributes used, where:
if ((*atts & 01) != 0) then polyline attributes set used.
if ((*atts & 02) != 0) then polymarker attributes set used.
if ((*atts & 04) != 0) then text attributes set used.
if ((*atts & 010) != 0) then fill area attributes set used.

GKS errorindicators checked: 8, 22, 23, 39, 41

INQ MAX WS TABLES

Ercode i_mx_tables(wsd, mxpl, mxpm, mxtx, mxfa, mxpat, mxcol)

```
Wsd     *wsd;
Sint    *mxpl;
Sint    *mxpm;
Sint    *mxtx;
Sint    *mxfa;
Sint    *mxpat;
Sint    *mxcol;
```

Diagnostics:
On return the user supplied parameters **mxpl, mxpm, mxtx, mxfa, mxpat** and **mxcol** will contain the maximum number of bundle table entries for polyline, polymark, text and fillarea, and the maximum number of indices for patterns and colours. A value of -1 means 'infinite'.

GKS errorindicators checked: 8, 22, 23, 39

INQ NR OF SEG PRIO SUPP

Ercode i_nr_sgprio(wsd, nr_segprio)

```
Wsd     *wsd;
Int     *nr_segprio;
```

Diagnostics:
The user supplied parameter **nr_segprio** is filled with the number of segment priorities supported.

GKS errorindicators checked: 8, 22, 23, 39

INQ DYN MOD OF SEG ATTS

Ercode i_dm_seg_atts(wsd, attflags)

```
Wsd     *wsd;
Int     *attflags;
```

Diagnostics:
On return the user supplied parameter **attflags** contains a mask for the dynamic modification settings for segment attributes, where:

for segm. transf.	if (*attflags & 01) == 1	then IMM else IRG
for vis>invis	if (*attflags & 02) == 1	then IMM else IRG
for invis>vis	if (*attflags & 04) == 1	then IMM else IRG
for hilighting	if (*attflags & 010) == 1	then IMM else IRG
for segment priority	if (*attflags & 020) == 1	then IMM else IRG
for adding prims	if (*attflags & 040) == 1	then IMM else IRG

GKS errorindicators checked: 8, 22, 23

INQ NR OF AV LOGICAL INP DEVS

Ercode i_nr_idev(wsd, nr_loc, nr_ske, nr_val, nr_cho, nr_pik, nr_str)

```
Wsd     *wsd;
Int     *nr_loc, *nr_ske, *nr_val;
Int     *nr_cho, *nr_pik, *nr_str;
```

Diagnostics:
On return the user supplied parameters are filled with the number of locator, stroke, valuator, choice, pick and string devices.

GKS errorindicators checked: 8, 22, 23

INQ DFLT LOC DEV DATA

Ercode i_locdev(wsd, dn, idev_descr)
Wsd *wsd;
Devno dn;
Iddescr **idev_descr;

Diagnostics:
On return the user supplied pointer **idev_descr** points to the systems input device description, stored in a structure of type Iddescr. For the type definitions of interest see the end of this section

GKS errorindicators checked: 8, 22, 23, 38, 140

INQ DFLT STROKE DEV DESCR

Ercode i_skedev(wsd, dn, idev_descr)
Wsd *wsd;
Devno dn;
Iddescr **idev_descr;

Diagnostics:
On return the user supplied pointer **idev_descr** points to the systems input device description, stored in a structure of type Iddescr. For the type definitions of interest see the end of this section

GKS errorindicators checked: 8, 22, 23, 38, 140

INQ DFLT VAL DEV DATA

Ercode i_valdev(wsd, dn, idev_descr)
Wsd *wsd;
Devno dn;
Iddescr **idev_descr;

Diagnostics:
On return the user supplied pointer **idev_descr** points to the systems input device description, stored in a structure of type Iddescr. For the type definitions of interest see the end of this section

GKS errorindicators checked: 8, 22, 23, 38, 140

INQ DFLT CHOICE DEV DATA

Ercode i_chodev(wsd, dn, idev_descr)
Wsd *wsd;
Devno dn;
Iddescr **idev_descr;

Diagnostics:
On return the user supplied pointer **idev_descr** points to the systems input device description, stored in a structure of type Iddescr. For the type definitions of interest see the end of this section

GKS errorindicators checked: 8, 22, 23, 38, 140

INQ DFLT PICK DEV DATA

Ercode i_pikdev(wsd, dn, idev_descr)
Wsd *wsd;
Devno dn;
Iddescr **idev_descr;

Diagnostics:
On return the user supplied pointer **idev_descr** points to the systems input device description, stored in a structure of type Iddescr. For the type definitions of interest see the end of this section

GKS errorindicators checked: 8, 22, 23, 38, 140

INQ DFLT STRING DEV DATA **Ercode i_strdev(wsd, dn, idev_descr)**

```
Wsd       *wsd;
Devno     dn;
Iddescr   **idev_descr;
```

Diagnostics:
On return the user supplied pointer **idev_descr** points to the systems input device description, stored in a structure of type Iddescr. For the type definitions of interest see the end of this section

GKS errorindicators checked: 8, 22, 23, 38, 140

For the default input device data the following typedefinitions are relevant.

```
typedef struct {
    Idevice   idd_idev;      /* default input device values */
    Int       idd_max;       /* maximum value field */
    Int       idd_npet;      /* number of prompt/echo types */
    Pet       *idd_pets;     /* list of prompt/echo types */
    } Iddescr;

typedef struct {
    Iclass    id_clas;       /* LOCATOR, STROKE etc */
    Imode     id_mode;       /* not used for default dev. data */
    Bool      id_echo;       /* not used for default dev. data*/
    Idata     id_ival;       /* default initial value */
    Pet       id_pet;        /* not used for default dev. data */
    Drect     id_area;       /* default echo area */
    Drecord   id_drec;       /* default data record */
    } Idevice;
```

See chapter 1 for the data record 'Drecord' type definition.

```
typedef union {
    Locate    ev_loc;
    Stroke    ev_stk;
    Value     ev_val;
    Choice    ev_cho;
    Pick      ev_pik;
    String    ev_str;
    } Idata;

typedef int Pet;        /* prompt/echo type */
```

1.6.9.6. Inquire segment state list (functions)

INQ SET OF ASSOCIATED WS **Ercode i_ass_ws(segname, nr_ws, ws)**

```
Segname   segname;
Int       *nr_ws;
Wss       ***ws;
```

Diagnostics:
On return the user supplied parameter **nr_ws** contains the number of associated workstations, while **ws** will point to the systems array of 'nr_ws' workstation pointers.

GKS errorindicators checked: 7, 120, 122

INQ SEG ATT **Ercode i_segatt(segname, seg)**

```
Segname  segname;
Seg      **seg;
```

Diagnostics:
On return the segment pointer **seg** will point to the segment structure belonging to the segment with name **segname**. The Seg structure will contain among others the information asked for. The Seg type contains the following fields:

```
typedef  struct {
         Segname   sg_sn;
         Wss       *sg_onws[NWSAS+1];
         Tmat      sg_mat;
         Bool      sg_vis;
         Bool      sg_hil;
         Segpri    sg_pri;
         Bool      sg_det;
         Bhead     *sg_firstbh;
         Int       *sg_local;
         } Seg;
```

where

```
#define NWSAS 8     /* maximum number of ws associated with a segment */
```

GKS errorindicators checked: 7, 120, 122

IQ PIXEL ARRAY DIMENSIONS **Ercode pxl_siz(ws, wr, sz)**

```
IN WORKST ID                Wss     *ws;
IN 2 POINTS                 Wrect   *wr;
OUT DIM. OF PIXEL ARRAY     ArrSiz  *sz;
```

IQ PIXEL ARRAY **Ercode pxl_arr(ws, p, sz, inv, arr)**

```
IN WORKST ID                Wss     *ws;
IN POINT                    Wc      *p;
IN DIM. OF COLOUR ARRAY     ArrSiz  *sz;
OUT PRESENCE OF INV. VALUES Bool    *inv;
OUT COLOUR INDEX ARRAY      Cindex  *arr;
```

IQ PIXEL **Ercode pixel(ws, p, ci)**

```
IN WORKST ID                Wss     *ws;
IN POINT                    Wc      *p;
OUT COLOUR INDEX            Cindex  *ci;
```

1.6.9.7. Inquire GKS error state list (functions)

INQUIRE INPUT QUEUE OVERFLOW	**Ercode i_inp_oflw(event)**
	Event *event;

Diagnostics:
If the input queue has overflowed since the last invocation of INQUIRE INPUT QUEUE OVERFLOW the user supplied structure **event** contains on return the workstation pointer, device number, and input class, of the logical input device that caused the overflow.

Typedef 'Event' looks as follows:

```
typedef struct {
    Wss     *ev_ws;
    Devno   ev_devn;
    Iclass  ev_class;
    Idata   ev_data;
    } Event;
```

GKS errorindicators checked: 8, 22, 23

1.6.10. Utility functions

The utility functions of GKS, and their C bindings are:

	Tmat *
EVALUATE TRANSFORM MATRIX	**ev_trmat(p, shift, rot, scale, ndcoord)**
IN FIXED POINT	Real *p;
IN SHIFT VECTOR	Real *shift;
IN ANGLE	Real rot;
IN SCALE FACTORS	Real *scale;
IN COORDINATE SW	Bool ndcoord; /*TRUE if NDC, FALSE if WC */
OUT SEGM TRANSF MATRIX	(returned value)

Remarks:

As it is not known of which type the fixed point and the shift vector are, these parameters are defined being of type "Real *" .

	Tmat *
ACCUMULATE TRANSF MATRIX	**acc_trmat(segmat, p, shift, rot, scale, ndcoord)**
IN SEGM TR MATRIX	Tmat *segmat;
IN FIXED POINT	Real *p;
IN SHIFT VECTOR	Real *shift;
IN ANGLE	Real rot;
IN SCALE FACTORS	Real *scale;
IN COORDINATE SW	Bool ndcoord; /*TRUE if NDC, FALSE if WC */
OUT SEGM TRANSF MATRIX	(returned value)

Remarks:

As it is not known of which type the fixed point and the shift vector are, these parameters are defined of type "Real *" .

1.6.11. Error Handling

The C binding of the error handling functions are:

EMERGENCY CLOSE GKS **emergency()**

Remarks:

See one of next chapters for the possible values for the **rout** parameter.

ERROR HANDLING **er_hand(erc, rout)**
IN ERROR NUMBER Ercode erc;
IN GKS PROCEDURE ID Int rout;

Remarks:

The user can replace the ERROR HANDLING function by defining an er_hand() function in the application or in one of the user libraries. During linking this library must be called before the gks library. The parameters of this er_hand() function must be identical to those of the C-GKS er_hand() function. The error file is already defined in open_gks().

See one of next chapters, or the headerfile cgksfun.h, for the possible values for the **rout** parameter.

ERROR LOGGING **er_logg(erc, rout)**
IN ERROR NUMBER Ercode erc;
IN GKS PROCEDURE ID Int rout;

Remarks:

See one of next chapters, or the headerfile cgksfun.h, for the possible values for the **rout** parameter.

2. Local extensions to GKS

Two extensions of GKS are defined and implemented. The Segment Grouping extension provides a facility for efficient clipping and transformation of segments for high function workstations. See the document "Segment Grouping, an Extension to The Graphical Kernel System".[5] The document contains an appendix with the C binding of the extra grouping functions.

The second extension creates the possibility to make segments temporary inactive. This extension is described here.

2.1. Activate and deactivate segment.

Good interactive programming includes a lot of feedback. To allow more complex prompts for input devices GKS provides a facility to use segments as a prompt for a choice device. However not always all the feedback can be generated in this way by the GKS input function, especially when the format is special. Therefore sometimes the application itself must be able to generate the appropriate feedback, and the only way the program can achieve this is by calling one or more GKS outputfunctions. This may cause problems, for example when a segment is open. Feedback output primitives not directly generated by the user are then put in the segment. The most appropriate solution is closing the segment temporarily. Calling close_seg() causes the stopping of the insertion. According to the GKS standard this unfortunately includes that no more information can be put in the segment afterwards; the segment can not be reopened again.

By using more than one workstation the actual creation of a picture and the feedback for the interaction can be arranged on different workstations. So one workstation for example can be used as a "scratch workstation", showing the currently available menus and the current state of the attributes, while another workstation shows the intermediate result of the picture creation. It is not strictly necessary to have two screens/devices for such a configuration. If the driver is smart enough one real device can be split up in two virtual workstations, where one workstation uses the left part of the screen while the other uses the right part.

The "more than one workstation approach" however does not solve the previous mentioned problem. Using two open workstations at the same time makes the application more complex, while still feedback, generated by the application by calling GKS outputprimitive functions, will be sent to all open workstations. A call of DEACTIVATE WORKSTATION would stop the sending of the output to one specific workstation, but this still gives the problem again with the unwanted data in the segment.

To solve the prompt problem an extra feature is added to the implementation. By allowing the segment to be put in an INACTIVE state the segment can remain into existence, although temporarily no information will be stored in the open segment. This can be done by calling the function "deactivate segment": deact_seg(), a function without any parameters. A call of "activate segment": act_seg() enables further insertion of information in the open segment.

Addition of the functions act_seg() and deact_seg() to the GKS system can be done in two different ways. The first approach is making it an extension of GKS. This includes the addition of a new global state being the state SEGMENT ACTIVE, and two segment control functions. Now to add info to an open segment the segment must be made active by calling the function ACTIVATE SEGMENT. Before closing a segment it must be made inactive by calling DEACTIVATE SEGMENT. Information is only put in a segment when the segment is active. An open segment can be switched more than one time between the states active and inactive. The two added functions are treated and implemented as GKS functions including error checking for the statevariables.

A weaker approach will be to let a global boolean variable, set to TRUE at the opening of GKS, tell whether or not info is stored in an open segment. The only task of the functions act_seg() and deact_seg() is the control of this boolean. The advantage of this approach is that a programmer, not aware of this extra feature, doesn't notice any difference with the standard.

At this moment the last approach is implemented. Furthermore an extra inquiry function is defined to get the current state of the open segment.

3. Error detection, error reaction, error recovery.

What follows is a list of GKS errors, preceded by their implementation name. Error numbers are according to GKS version 7.4. Error numbers are followed by a 1 character code ('a', 'b' or 'c'), indicating the C-GKS error reaction as it is currently implemented.

Error reaction after detection of a GKS error is dependent of the error type. Errors of type 'a' and 'c' causes an error message report to be sent to the messagefile and to all active/open workstations capable of accepting messages.

Errors of type 'a' are followed by an immediate return from the calling GKS routine to the application program.

Errors of type 'c' causes an 'emergency close GKS' to be called, followed by stopping the execution of the program, without returning to the calling GKS procedure.

Some errors, like a negative number of points (Size is defined as an unsigned integer) can not occur in C-GKS. Therefor no check on these errors is implemented.

Most functions return either GE_NO_ERR, or, if a GKS error occured, the error number. Some C-GKS functions are implemented such that they return a pointer to a structure, like open_ws() and newseg(), or a value, like req_loc(). In case an error occurs these functions return with a NULL pointer, or a NULL value.

error	7.4 nr	type	description
GE_NO_ERR	0	-	No GKS error occured
B.2 States			
GE_NOGKCL	1	c	GKS shall be in state GKCL
GE_NOGKOP	2	c	GKS shall be in state GKOP
GE_NOWSAC	3	c	GKS shall be in state WSAC
GE_NOSGOP	4	c	GKS shall be in state SGOP
GE_NOWASO	5	c	GKS shall be in state WSAC or SGOP
GE_NOWOWA	6	c	GKS shall be in state WSOP or WSAC
GE_NOWOAS	7	c	GKS shall be in state WSOP, WSAC or SGOP
GE_INGKCL	8	c	GKS shall be in state GKOP, WSOP, WSAC or SGOP
B.3 Workstations			
WE_IDINVD	20	c	Spec workstation identifier is inv.
WE_CIDINV	21	c	spec connection identifier is inv.
WE_TYPINV	22	c	Spec workstation type is inv.
WE_TYPNEX	23	c	Spec workstation type does not exist
WE_TYPNOP	24	c	Spec workstation type is open
WE_NOTOPN	25	c	Spec workstation is not open
WE_CNNTOP	26	c	Spec workstation cannot be opened
WE_WISNOP	27	c	Ws independent segment storage is not open
WE_WISOPN	28	c	Ws independent segment storage is already open
WE_ACTIVE	29	c	Spec workstation is active
WE_NACTIV	30	c	Spec workstation is not active
WE_CTMFOU	31	c	Spec workstation is of category MO
WE_NMFOUT	32	c	Spec workstation is not of category MO
WE_CTMFIN	33	c	Spec workstation is of category MI
WE_NMFINP	34	c	Spec workstation is not of category MI
WE_CATINP	35	c	Spec workstation is of category INPUT
WE_ISWISS	36	c	Spec workstation is WISS
WE_NOUTIN	37	c	Spec workstation is not of category OUTIN
WE_NINOIN	38	c	Spec workstation is neither of category INPUT nor of OUTIN
WE_NOUOIN	39	c	Spec workstation is neither of category OUTPUT nor of OUTIN
WE_PXLSRB	40	c	Spec workstation has no pixel store

error	7.4 nr	type	description
			readback capability
WE_NOTGDP	41	c	Spec workstation type is not able to do spec gdp
WE_MXOPEX	42	c	Max number of simultaneous open workstation would be exceeded
WE_MXACEX	43	c	Max number of simultaneous active workstation would be exceeded
B.4 Transformations			
TE_TNRINV	50	c	Transformation nr is inv.
TE_RCTINV	51	c	Rectangle definition is inv.
TE_VPOUTS	52	c	Viewport is not within the NDC unit square
TE_WSWOUT	53	c	Ws window is not within the NDC unit square
TE_WSVOUT	54	c	Ws viewport is not within the display space
B.5 Output attributes			
PE_LIXINV	60	c	pol. ix is inv.
PE_LRNDWS	61	c	A repr for spec pol.ix is not def. on this ws
PE_LRNPWS	62	c	A repr for spec pol.ix is not predf. on this ws
PE_LTZERO	63	c	linetype is equal to zero
PE_LTNSUP	64	c	spec linetype not supported on this ws
PE_LWSLZE	65	c	linewidth scalefactor is less than zero
PE_MIXINV	66	c	polymarker ix is inv.
PE_MRNDWS	67	c	a repr for spec pom ix is not def on this ws
PE_MRNPWS	68	c	a repr for spec pom ix is not predef on this ws
PE_MTZERO	69	c	marker type is equal zero
PE_MTNSUP	70	c	specif. markertype is not supported on this ws
PE_MSSLZE	71	c	marker size scalefactor is less than zero
PE_TIXINV	72	c	text ix is inv.
PE_TRNDWS	73	c	a rep for the spec txt ix is not def on this ws
PE_TRNPWS	74	c	a rep for spec txt ix is not predef on this ws
PE_TFZERO	75	c	text font is equal zero
PE_FNSPWS	76	c	req text font is not supp for spec prec on ws
PE_CEFLEZ	77	c	char expansion fac is less than or equal zero
PE_CHTLEZ	78	c	character height is less than or equal zero
PE_LCUZRO	79	c	length of character up vector is equal zero
PE_FIXINV	80	c	fillarea ix is inv.
PE_FRNDWS	81	c	repr for specfd farea ix is not def. on this ws
PE_FRNPWS	82	c	repr for spec farea ix is not predef on this ws
PE_ISNSWS	83	c	spec farea inter style is not supp on this ws
PE_SIXZRO	84	c	style (pattern or hatch) ix is equal 0
PE_PIXINV	85	c	spec pattern ix is inv.
PE_HSNSWS	86	c	spec hatch style is not supported on this ws
PE_PSZNPO	87	c	pattern size value is not positive
PE_PRNDWS	88	c	a repr for spec pattern index has not been defined on this workstation
PE_PRNPWS	89	c	a repr for spec pattern index has not been predefined on this workstation
PE_NOPATT	90	c	int style PATTERN is not supported on this ws
PE_DMCINV	91	c	Dimensions of colour array inv.
PE_CIXLZO	92	c	colour ix is less than 0
PE_CIXINV	93	c	colour ix is inv.
PE_CRNDWS	94	c	repr for spec col ix is not def on this ws
PE_CRNPWS	95	c	repr for spec col ix is not predef on this ws
PE_COLOUT	96	c	colour is outside range [0, 1]
PE_PCKINV	97	c	pick identifier is inv.
B.6 Output primitives			
AE_INVNRC	100	c	inv. nrs of coordinates

error	7.4 nr	type	description
AE_INVSTR	101	c	inv. code in string
AE_INVGDI	102	c	gdp identifier is inv.
AE_INVCGD	103	c	content of gdp record is inv.
AE_AWSNGD	104	c	at least one active workstation is not able to do gdp
AE_WSNGDP	105	c	a workstation is not able to generate spec gdp under the current transf. and clipping rectangle
B.7 Segments			
SE_SGNINV	120	c	spec segment name is inv.
SE_SGNUSE	121	a	spec segment name is already in use
SE_SGNXST	122	c	spec segment does not exist
SE_SGNSWS	123	c	spec segment does not exist on spec ws
SE_SGNXWI	124	c	spec segment does not exist on WISS
SE_SGOPEN	125	c	spec segment is open
SE_SGPOUT	126	c	segment priority is outside range [0, 1]
B.8 Input			
IE_DVNPWS	140	c	spec input device is not present on ws
IE_DVNRQM	141	c	input device is not in REQUEST mode
IE_DVNSMM	142	c	input device is not in SAMPLE mode
IE_NOSEMD	143	c	EVENT, SAMPLE mode not avail at this GKS level
IE_PETNSU	144	c	spec prompt echo type not supported on this ws
IE_AREAOU	145	c	echo area is outside display space
IE_DRCINV	146	c	content of input data record is inv.
IE_QOVERF	147	c	input queue has overflowed
IE_QNOVSO	148	c	input queue not overflowed since it was opened or the last invocation of INQ INP QUEUE OVERFLOW
IE_QOVWSC	149	c	input queue has overflowed, but associated workstation closed
IE_NOVCER	150	c	no input value of correct class in curr ev rep
IE_TOUTIN	151	c	timeout is inv.
IE_IVAINV	152	c	initial value is inv.
IE_NPSKGB	153	c	nr of points in in. stroke greater than bu.sze
IE_LISGDM	154	c	length of in. str. greater than a def. max
B.9 Metafiles			
ME_ITYNAL	160	c	item type is not allowed for user items
ME_ITLINV	161	c	item length is inv.
ME_NITMMF	162	c	no item is left in metafile
ME_ITMINV	163	c	metafile item is inv.
ME_ITTINV	164	c	item type is not a valid item
ME_IDRINV	165	c	cont of item data rec is inv for spec item typ
ME_MDRINV	166	c	maximum item data record length is inv.
ME_UINITP	167	c	user item can not be interpreted
ME_FUNSUP	168	c	spec func is not supp in this level of GKS
B.10 Escape			
EE_FNTSUP	180	c	spec escape function is not supported
EE_FIDINV	181	c	spec escape function identification is inv.
EE_EDRINV	182	c	contents of escape data record are inv.

error	7.4 nr	type	description
B.11 Miscellaneous			
GE_ERFINV	200	c	spec err file is inv.
B.12 System			
XE_STOVGK	300	c	storage overflow has occured in GKS
XE_STOVSS	301	c	storage overflow has occured in segement storage
XE_IOREAD	302	c	I/O err has occured while reading
XE_IOWRIT	303	c	I/O err has occured while writing
XE_IOSEND	304	c	I/O err has occured while sending data to a ws
XE_IORECD	305	c	I/O err has occured receiving data from ws
XE_LIBMAN	306	c	I/O err has occ during prog libr management
XE_READWD	307	c	I/O err has occured while reading wsd
XE_ARITHM	308	c	arithmetic err has occured
B.1 Implementation dependent			
XE_2001	-1	c	wsdcap file not found
XE_2004	-4	c	can't find a workstation description table entry for a driver as its ws-type is unknown (adapt u_cgksws.c).
XE_2005	-5	c	no entry on file 'wsdcap' for one of workstation types
XE_2006	-6	c	info on wsdcap file not good organized
XE_2007	-7	c	wsdcap: format on wsdcap incorrect for entry
XE_2008	-8	c	wsdcap: a 2 letter code not recognized
XE_2009	-9	c	wsdcap: not enough space available for entry in wsd
XE_2010	-10	c	can't find Berkeley font file
XE_2011	-11	c	bad header in Berkeley font file
XE_2012	-12	c	bad dispatch table in Berkeley font file
XE_2013	-13	c	wrong magic nr in Berkeley font file
XE_2014	-14	c	Berkeley font file not good organized
XE_2015	-15	c	AVWT wrong defined in cgkswstype.h
XE_2016	-16	c	combination of flags in Screen structure not supported
XE_2020	-20	c	didn't succeed in opening a message file
XE_2030	-30	c	could not open a file for spooler for versatec.
XE_MIUNRD	-31	c	specified Metafile Input could not be read
XE_2040	-40	c	can't find hershey font file
XE_2041	-41	c	bad header in hershey font file
XE_2042	-42	c	bad magic number in hershey font file
XE_2043	-43	c	Hershey font file not good organized
XE_2101	-101	c	func called via driver entry for open_ws met error condition
XE_DIVBYZ	-309	c	tried to divide by 0

3.1. Alphabetic function list.

What follows is a list of all GKS functions and the implementation names.

GKS function	binding to C	return	code in file	level
ACCUMULATE TRANSF MATRIX	acc_trmat()	Tmat *	acc_trmat.c	1a
ACTIVATE WORKST	activate()	Ercode	activate.c	0a
ASSOCIATE SEGMENT WITH WS	sendseg()	Ercode	sendseg.c	2a
AWAIT EVENT	await()	Ercode	await.c	0c
CELL ARRAY	cellarray()	Ercode	cellarray.c	0a
CLEAR WORKST	clear()	Ercode	clear.c	0a
CLOSE GKS	close_gks()	Gks *	open_gks.c	0a
CLOSE SEGMENT	closeg()	Ercode	newseg.c	1a
CLOSE WORKST	close_ws()	Wss *	open_ws.c	0a
COPY SEGMENT TO WS	copyseg()	Ercode	copyseg.c	2a
CREATE SEGMENT	newseg()	Seg *	newseg.c	1a
DEACTIVATE WORKST	deactivate()	Ercode	activate.c	0a
DELETE SEGMENT	zapseg()	Ercode	zapseg.c	1a
DELETE SEGM FROM WORKST	delseg()	Ercode	delseg.c	1a
EMERGENCY CLOSE GKS	emergency()	Ercode	emergency.c	0a
ERROR HANDLING	er_hand()	Ercode	er_hand.c	0a
ERROR LOGGING	er_logg()	Ercode	er_hand.c	0a
ESCAPE	escape()	Ercode	escape.c	0a
EVALUATE TRANSF. MATRIX	ev_trmat()	Tmat *	ev_trmat.c	1a
FILL AREA	fillarea()	Ercode	fillarea.c	0a
FLUSH DEVICE EVENTS	flsh_ev()	Ercode	flsh_ev.c	0c
GEN. DRAWING PRIMITIVE	g_draw()	Ercode	g_draw.c	0a
GET CHOICE	no GET funcs			0c
GET ITEM TYPE FROM GKSM	get_itmtp()		get_itmtp.c	0a
GET LOCATOR	no GET funcs			0c
GET PICK	no GET funcs			0c
GET STRING	no GET funcs			1c
GET STROKE	no GET funcs			0c
GET VALUATOR	no GET funcs			0c
INITIALISE CHOICE	init_cho()	Ercode	init_cho.c	0b
INITIALISE LOCATOR	init_loc()	Ercode	init_loc.c	0b
INITIALIZE PICK	init_pik()	Ercode	init_pik.c	1b
INITIALISE STRING	init_str()	Ercode	init_str.c	0b
INITIALISE STROKE	init_ske()	Ercode	init_ske.c	0b
INITIALISE VALUATOR	init_val()	Ercode	init_val.c	0b
INSERT SEGMENT	insseg()	Ercode	insseg.c	2a
INTERPRET ITEM	intrp_itm()	Ercode	intrp_itm.c	0a
MESSAGE	message()	Ercode	message.c	1a
OPEN GKS	open_gks()	Ercode	open_gks.c	0a
OPEN WORKST	open_ws()	Ercode	open_ws.c	0a
POLYLINE	polyline()	Ercode	polyline.c	0a
POLYMARKER	polymark()	Ercode	polymark.c	0a
READ ITEM FROM GKSM	rd_itm()	Ercode	rd_itm.c	0a
REDRAW ALL SEGM ON WS	redraw()	Ercode	redraw.c	1a
RENAME SEGMENT	renameseg()	Ercode	renameseg.c	1a
REQUEST CHOICE	req_cho()	Bool	req_cho.c	0b
REQUEST LOCATOR	req_loc()	Bool	req_loc.c	0b
REQUEST PICK	req_pik()	Bool	req_pik.c	1b
REQUEST STRING	req_str()	Bool	req_str.c	0b
REQUEST STROKE	req_ske()	Bool	req_ske.c	0b
REQUEST VALUATOR	req_val()	Bool	req_val.c	0b
SAMPLE CHOICE	smp_cho()	Ercode	smp_cho.c	0c
SAMPLE LOCATOR	smp_loc()	Ercode	smp_loc.c	0c
SAMPLE PICK	smp_pik()	Ercode	smp_pik.c	1c
SAMPLE STRING	smp_str()	Ercode	smp_str.c	0c

GKS function	binding to C	return	code in file	level
SAMPLE STROKE	smp_ske()	Ercode	smp_ske.c	0c
SAMPLE VALUATOR	smp_val()	Ercode	smp_val.c	0c
SELECT NORM. TRANSFORM.	sel_cntran()	Ercode	sel_cntran.c	0a
SET ASPECT SOURCE FLAGS	s_asflags()	Ercode	s_asflag.c	0a
SET CHARACTER EXP. FACTOR	s_ch_ef()	Ercode	s_ch_ef.c	0a
SET CHARACTER HEIGHT	s_ch_ht()	Ercode	s_ch_ht.c	0a
SET CHARACTER SPACING	s_ch_sp()	Ercode	s_ch_sp.c	0a
SET CHARACTER UP VECTOR	s_ch_up()	Ercode	s_ch_up.c	0a
SET CHOICE MODE	cho_mode()	Ercode	cho_mode.c	0b
SET CLIPPING INDICATOR	s_clip()	Ercode	s_clip.c	0a
SET COLOUR REPRESENT.	col_rep()	Ercode	col_rep.c	0a
SET DEFERRAL STATE	s_defer()	Ercode	s_defer.c	1a
SET DETECTABILITY	s_detect()	Ercode	s_detect.c	1b
SET FILLAREA COLOUR INDEX	s_fa_ci()	Ercode	s_fa_ci.c	0a
SET FILLAREA INDEX	s_fa_i()	Ercode	s_fa_i.c	0a
SET FILLAREA INTERIOR STYLE	s_fa_is()	Ercode	s_fa_is.c	0a
SET FILLAREA REPRESENT.	fa_rep()	Ercode	fa_rep.c	1a
SET FILLAREA STYLE INDEX	s_fa_si()	Ercode	s_fa_si.c	0a
SET HIGHLIGHTING	s_hilite()	Ercode	s_hilite.c	1a
SET LINETYPE	s_lt()	Ercode	s_lt.c	0a
SET LINEWIDTH SCALE FACTOR	s_lw()	Ercode	s_lw.c	0a
SET LOCATOR MODE	loc_mode()	Ercode	loc_mode.c	0b
SET MARKER SIZE SCALE FAC.	s_msz()	Ercode	s_msz.c	0a
SET MARKER TYPE	s_mt()	Ercode	s_mt.c	0a
SET PATTERN REFERENCE PT	s_patpt()	Ercode	s_patpt.c	0a
SET PATTERN REPRESENT.	pa_rep()	Ercode	pa_rep.c	1a
SET PATTERN SIZE	s_patsiz()	Ercode	s_patsiz.c	0a
SET PICK IDENTIFIER	s_pikid()	Ercode	s_pikid.c	1b
SET PICK MODE	pik_mode()	Ercode	pik_mode.c	1b
SET POLYLINE COLOUR INDEX	s_pl_ci()	Ercode	s_pl_ci.c	0a
SET POLYLINE INDEX	s_pl_i()	Ercode	s_pl_i.c	0a
SET POLYLINE REPRESENTATION	pl_rep()	Ercode	pl_rep.c	1a
SET POLYMARKER COL. INDEX	s_pm_ci()	Ercode	s_pm_ci.c	0a
SET POLYMARKER INDEX	s_pm_i()	Ercode	s_pm_i.c	0a
SET POLYMARKER REPR.	pm_rep()	Ercode	pm_rep.c	1a
SET SEGMENT PRIORITY	s_segpri()	Ercode	s_segpri.c	1a
SET SEGMENT TRANSFORM.	transeg()	Ercode	transeg.c	1a
SET STRING MODE	str_mode()	Ercode	str_mode.c	0b
SET STROKE MODE	ske_mode()	Ercode	ske_mode.c	0b
SET TEXT ALIGNMENT	s_tx_al()	Ercode	s_tx_al.c	0a
SET TEXT COLOUR INDEX	s_tx_ci()	Ercode	s_tx_ci.c	0a
SET TEXT FONT AND PRECISION	s_tx_fp()	Ercode	s_tx_fp.c	0a
SET TEXT INDEX	s_tx_i()	Ercode	s_tx_i.c	0a
SET TEXT PATH	s_tx_pt()	Ercode	s_tx_pt.c	0a
SET TEXT REPRESENTATION	tx_rep()	Ercode	tx_rep.c	1a
SET VALUATOR MODE	val_mode()	Ercode	val_mode.c	0b
SET VIEWPORT	s_viewport()	Ercode	s_viewport.c	0a
SET VIEWP. INPUT PRIORITY	s_vip()	Ercode	s_vip.c	0b
SET VISIBILITY	s_segvis()	Ercode	s_segvis.c	1a
SET WINDOW	s_window()	Ercode	s_window.c	0a
SET WORKST VIEWPORT	s_w_view()	Ercode	s_w_view.c	0a
SET WORKST WINDOW	s_w_wind()	Ercode	s_w_wind.c	0a
TEXT	text()	Ercode	text.c	0a
UPDATE WORKST	update()	Ercode	update.c	0a
WRITE ITEM TO GKSM	wr_itm()	Ercode	wr_itm.c	0a

3.2. Inquiry functions.

The inquiry functions are implemented as macros. devno stands for a device number (up from 1).

GKS inquiry function	C call	return	lev
INQ CHOICE DEVICE STATE	inq_wss(ws, ws_devs[CHOICE] [devno - 1])	Idevice	0b
INQ CLIPPING	inq_gkss(gk_clip)	Bool	0a
	inq_gkss(gk_curnt->nt_view)	Nrect	0a
INQ COLOUR FACILITIES	inq_wsd(wsd, wd_ncix)	Int	0a
	inq_wsd(wsd, wd_colav)	Bool	0a
	inq_wsd(wsd, wd_ncol)	Int	0a
INQ COLOUR REPRESENT	inq_wss(wss, ws_colr[col_index])	Colour	0a
INQ CURR. NORM. TRANSF. NR	inq_gkss(gk_curnt)	Ntran *	0a
INQ CURR. INDIV. ATT. VALUES	inq_gkss(gk_lrlt)	Ltype	0a
	inq_gkss(gk_lrlw)	Lwidth	0a
	inq_gkss(gk_lrci)	Cindex	0a
	inq_gkss(gk_mrmt)	Mtype	0a
	inq_gkss(gk_mrms)	Msize	0a
	inq_gkss(gk_mrci)	Cindex	0a
	inq_gkss(gk_txfp)	FoPr	0a
	inq_gkss(gk_chef)	Charef	0a
	inq_gkss(gk_chsp)	Charsp	0a
	inq_gkss(gk_txci)	Cindex	0a
	inq_gkss(gk_fais)	Istyle	0a
	inq_gkss(gk_fasi)	Sindex	0a
	inq_gkss(gk_faci)	Cindex	0a
	inq_gkss(gk_asf)	Int	0a
INQ CURR. PRIM. ATT. VALUES	inq_gkss(gk_line)	Bindex	0a
	inq_gkss(gk_mark)	Bindex	0a
	inq_gkss(gk_text)	Bindex	0a
	inq_gkss(gk_chht)	Charht	0a
	inq_gkss(gk_chup)	Wc	0a
	inq_gkss(gk_chwd)	Wc	0a
	inq_gkss(gk_chbs)	Wc	0a
	inq_gkss(gk_path)	Path	0a
	inq_gkss(gk_txal)	Talign	0a
	inq_gkss(gk_area)	Bindex	0a
	inq_gkss(gk_ptwd)	Wc	0a
	inq_gkss(gk_ptht)	Wc	0a
	inq_gkss(gk_ptref)	Wc	0a
	inq_gkss(gk_pickid)	Pickid	0a
INQ DEFAULT CHOICE DEVICE ST	inq_wsd(wsd, wd_ddescr[CHOICE] [devno - 1])	Iddescr	0b
INQ DEFAULT DEFERRAL STATE	inq_wsd(wsd, wd_defr)	Defmode	1a
	inq_wsd(wsd, wd_waitr)	Bool 6)	1a
INQ DEFAULT LOCAT DEVICE ST	inq_wsd(wsd, wd_ddescr[LOCATOR] [devno - 1])	Iddescr	0b
INQ DEFAULT PICK DEVICE DATA	inq_wsd(wsd, wd_ddescr[PICK] [devno - 1])	Iddescr	1b
INQ DEFAULT STRING DEVICE ST	inq_wsd(wsd, wd_ddescr[STRING] [devno - 1])	Iddescr	0b
INQ DEFAULT STROKE DEVICE ST	inq_wsd(wsd, wd_ddescr[STROKE] [devno - 1])	Iddescr	0b
INQ DEFAULT VALUAT DEVICE ST	inq_wsd(wsd, wd_ddescr[VALUATOR][devno-1])	Iddescr	0b
INQ DISPLAY SPACE SIZE	inq_wsd(wsd, wd_rsize)	Dc	0a
	inq_wsd(wsd, wd_isize)	Ic	0a
INQ DYNAMIC MODIF OF SEG ATTR	inq_wsd(wsd, wd_dmar)	Int	1a
INQ DYNAMIC MODIF OF WS ATTR	inq_wsd(wsd, wd_dmas)	Int	1a
INQ FILLAREA FACILITIES	inq_wsd(wsd, wd_nistyl)	Int	0a
	inq_wsd(wsd, wd_istyl)	Istyle *	0a
	inq_wsd(wsd, wd_nhatst)	Int	0a
	inq_wsd(wsd, wd_hatst)	Sindex *	0a
	inq_wsd(wsd, wd_narea)	Int	0a
INQ FILLAREA REPRESENT	inq_wss(ws, ws_area[b_index - 1])	AreaRep	1a
INQ GEN. DRAWING PRIMITIVE	use "inquire list of gdp's"		0a

GKS inquiry function	C call	return	lev
INQ INPUT QUEUE OVERFLOW	inq_gke(ge_event)	Event *	0c
INQ LEVEL OF GKS	inq_gkd(gd_lev)	GksLevl	0a
INQ LIST OF AVAILABLE GDP'S	inq_wsd(wsd, wd_ngdp)	Int	0a
	inq_wsd(wsd, wd_gdp)	Gdp *	0a
INQ LIST OF AVAIL. WS TYPES	inq_gkd(gd_nrwt)	Int	0a
	inq_gkd(gd_avwt)	Wsd **	0a
INQ LIST OF COLOUR INDICES	inq_wss(ws, ws_ncolr)	Int	0a
	last out par not of interest		0a
INQ LIST OF FILL AREA INDICES	inq_wss(ws, ws_narea)	Int	1a
	inq_wss(ws, ws_area)	AreaRep *	1a
INQ LIST OF NORM TRANSF NRS	inq_gkss(gd_ntran)	Ntran **	0a
INQ LIST OF PATTERN INDICES	inq_wss(ws, ws_npatt)	Int	1a
	inq_wss(ws, ws_patt)	PattRep *	1a
INQ LIST OF POLYLINE INDICES	inq_wss(ws, ws_nline)	Int	1a
	inq_wss(ws, ws_line)	LineRep *	1a
INQ LIST OF POLYMARK INDICES	inq_wss(ws, ws_nmark)	Int	1a
	inq_wss(ws, ws_mark)	MarkRep *	1a
INQ LIST OF TEXT INDICES	inq_wss(ws, ws_ntext)	Int	1a
	inq_wss(ws, ws_text)	TextRep *	1a
INQ LOCATOR DEVICE STATE	inq_wss(ws, ws_devs[LOCATOR] [devno - 1])	Idevice	0b
INQ MAX LNGTH OF WS ST TABLES	inq_wsd(wsd, wd_mxpr)	Sint 8)	1a
	inq_wsd(wsd, wd_mxpm)	Sint	1a
	inq_wsd(wsd, wd_mxtx)	Sint	1a
	inq_wsd(wsd, wd_mxfa)	Sint	1a
	inq_wsd(wsd, wd_mxpa)	Sint	1a
	inq_wsd(wsd, wd_mxco)	Sint	1a
INQ MAX. NORM TRANSF. NR	inq_gkd(gd_nntran)	Int 1)	0a
INQ MORE SIMULTAN. EVENTS	inq_gkss(gk_queue)	Quevent *	0c
INQ NAME OF OPEN SEGMENT	inq_gkss(gk_opsg)	Seg *	1a
INQ NORMALIZATION TRANSF.	inq_gkss(gk_ntran[ntr_number])	Ntran *	0a
INQ NR AV LOGIC INP DEVS	inq_wsd(wsd, wd_ndevs[LOCATOR])	Int	0b
	inq_wsd(wsd, wd_ndevs[STROKE])	Int	0b
	inq_wsd(wsd, wd_ndevs[VALUATOR])	Int	0b
	inq_wsd(wsd, wd_ndevs[CHOICE])	Int	0b
	inq_wsd(wsd, wd_ndevs[PICK])	Int	0b
	inq_wsd(wsd, wd_ndevs[STRING])	Int	0b
INQ NR OF SEG PRIO'S SUPP.	inq_wsd(wsd, wd_nprio)	Int	1a
INQ OPERATING STATE VALUE	inq_state()	GksState	0a
INQ PATTERN FACILITIES	inq_wsd(wsd, wd_npatt)	Int	0a
INQ PATTERN REPRESENT	inq_wss(ws, ws_patt[b_index - 1])	PattRep	1a
INQ PICK DEVICE STATE	inq_wss(ws, ws_devs[PICK] [devno - 1])	Idevice	1b
INQ POLYLINE FACILITIES	inq_wsd(wsd, wd_nltyp)	Int	0a
	inq_wsd(wsd, wd_ltyp)	Ltype *	0a
	inq_wsd(wsd, wd_nlwth)	Int	0a
	inq_wsd(wsd, wd_lwth)	Lwidth * 3)	0a
	inq_wsd(wsd, wd_nline)	Int	0a
INQ POLYLINE REPRESENT.	inq_wss(ws, ws_line[b_index - 1])	LineRep	1a
INQ POLYMARKER FACILITIES	inq_wsd(wsd, wd_nmktyp)	Int	0a
	inq_wsd(wsd, wd_mktyp)	Mtype *	0a
	inq_wsd(wsd, wd_nmksz)	Int	0a
	inq_wsd(wsd, wd_mksz)	Msize * 3)	0a
	inq_wsd(wsd, wd_nmark)	Int	0a
INQ POLYMARKER REPRESENT.	inq_wss(ws, ws_mark[b_index - 1])	MarkRep	1a
INQ PREDEF. COLOUR REPR.	inq_wsd(wsd, wd_col[col_index])	Colour	0a
INQ PREDEF. FILL AREA REPR.	inq_wsd(wsd, wd_area[b_index - 1])	AreaRep	0a
INQ PREDEF. PATTERN REPR.	inq_wsd(wsd, wd_patt[b_index - 1])	PattRep	0a
INQ PREDEF. POLYLINE REPR.	inq_wsd(wsd, wd_line[b_index - 1])	LineRep	0a
INQ PREDEF. POLYMARK. REP.	inq_wsd(wsd, wd_mark[b_index - 1])	MarkRep	0a

GKS inquiry function	C call	return	lev
INQ PREDEF. TEXT REPR.	inq_wsd(wsd, wd_text[b_index - 1])	TextRep	0a
INQ SEGMENT ATTRIBUTES	inq_seg(seg, sg_mat)	Tmat	1a
	inq_seg(seg, sg_vis)	Bool	1a
	inq_seg(seg, sg_hili)	Bool 7)	1a
	inq_seg(seg, sg_pri)	Segpri	1a
	inq_seg(seg, sg_det)	Bool	1a
INQ SET OF ACTIVE WORKST	inq_gkss(gk_actv)	Wss ** 4)	1a
INQ SET OF ASSOCIATED WS	inq_seg(seg, sg_onws)	Wss ** 4)	1a
INQ SET OF OPEN WORKST.	inq_gkss(gk_opws)	Wss ** 4)	0a
INQ SET OF SEG NAMES IN USE	inq_gkss(gk_segs)	Seginst *	1a
INQ SET OF SEG NAMES IN WS	inq_wss(ws, ws_segs)	Seginst *	1a
INQ STRING DEVICE STATE	inq_wss(ws, ws_devs[STRING] [devno - 1])	Idevice	0b
INQ STROKE DEVICE STATE	inq_wss(ws, ws_devs[STROKE] [devno - 1])	Idevice	0b
INQ TEXT FACILITIES	inq_wsd(wsd, wd_nfopr)	Int	0a
	inq_wsd(wsd, wd_fopr)	FoPr *	0a
	inq_wsd(wsd, wd_nchht)	Int	0a
	inq_wsd(wsd, wd_chht)	Charht * 2)	0a
	inq_wsd(wsd, wd_nchef)	Int	0a
	inq_wsd(wsd, wd_chef)	Charef * 2)	0a
	inq_wsd(wsd, wd_ntext)	Int	0a
INQ TEXT REPRESENTATION	inq_wss(ws, ws_text[b_index - 1])	TextRep	1a
INQ VALUATOR DEVICE STATE	inq_wss(ws, ws_devs[VALUATOR][devno-1])	Idevice	0b
INQ WORKST. CATEGORY	inq_wsd(wsd, wd_cat)	Int	0a
INQ WORKST. CLASSIF.	inq_wsd(wsd, wd_rov)	Int	0a
INQ WORKST. CONN. & TYPE	inq_wss(ws, ws_ifile)	File *	0a
	inq_wss(ws, ws_ofile)	File *	0a
	inq_wss(ws, ws_wsd)	Wsd * 5)	0a
INQ WS. DEF. & UPD. STATES	inq_wss(ws, ws_defr)	Defmode	0a
	inq_wss(ws, ws_waitr)	Bool	0a
	inq_wss(ws, ws_empty)	Bool	0a
	inq_wss(ws, ws_newfr)	Bool	0a
INQ WORKST MAX NUMBERS	inq_gkd(gd_nopws)	Int	1a
	inq_gkd(gd_nactv)	Int	1a
	inq_gkd(gd_nwsas)	Int	1a
INQ WORKST. STATE	inq_wss(ws, ws_actv)	Bool	0a
INQ WORKST. TRANSF.	inq_wss(ws, ws_trupd)	Bool	0a
	inq_wss(ws, ws_reqw)	Nrect	0a
	inq_wss(ws, ws_curw)	Nrect	0a
	inq_wss(ws, ws_reqv)	Drect	0a
	inq_wss(ws, ws_curv)	Drect	0a

The following inquiry functions are not implemented as macro's but as functions.

GKS function	binding to C	return	code in file	level	comment
INQ PIXEL	pixel()	Ercode	pixel.c	0a	
INQ PIXEL ARRAY	pxl_arr()	Ercode	pxl_arr.c	0a	
INQ PIXEL ARRAY DIM	pxl_siz()	Ercode	pxl_siz.c	0a	
INQ TEXT EXTENT	textext()	Ercode	textext.c	0a	only for Berk. fonts

1) returns array size
2) returns pointer to 2 values
3) returns pointer to 3 values
4) A list of pointers is returned, followed by a NULL pointer.
5) is type field
6) TRUE means suppressed
7) FALSE means NORMAL
8) for these entries a value -1 means infinite

4. C-GKS example programs

What follows are some examples of C-GKS application programs.

The following programme opens the GKS system, opens and activates the aed512 workstation, and draws a simple polyline.

```
#include "cgks.h"

main()
{
        Gks         *gkss;
        Wss         *wssaed;
        Wc          p[2];

        gkss = open_gks((Erhandle)NULL, (Erarea *)NULL, (Size)NULL,"mess");
        wssaed = open_ws(NULL, NULL, aed512);
        activate(wssaed);

        p[0].w_x = 0.0; p[0].w_y = 0.0;
        p[1].w_x = 1.0; p[1].w_y = 1.0;
        polyline(2, p);

        deactivate(wssaed);
        close_ws(wssaed);
        close_gks();
}


/*
 * The following C-GKS programme is a translation of example 3 from the GKS 7.4 document
 */
#include "cgks.h"

Wstype      refreshvectordisplay, drumplotter;
            /* in a real application these declarations must be external*/
Gks         *gkss;
Wss         *wssvect, *wssplot, *wssmfin, *wsswiss;
Size        lrec;
Itmrecord   datrec;
Itmtp       item_type;
Int         i;
Real        timeout;
Value       value, scale, angle;
Event       *event;
Drect       drect_val, drect_loc;
Drecord     drecord_val, drecord_loc;
Tmat        *matrix, *matac;
Wc          fixed, shift, pt2_1;
Real        scale_facs[2];
Locate      locate1, locate2;
Seg         *segname;
Pickid      pickid;
Ntran       *trans_1;

main()
{
        gkss = open_gks(NULL, NULL, NULL,"exmpl3.mess");

        wssvect = open_ws("", "", refreshvectordisplay);
        wssplot = open_ws("", "/dev/tty12", drumplotter);
        wssmfin = open_ws("../mf.exmpl3", "", mi);
        wsswiss = open_ws("", "", wis);
```

```
activate(wssvect);
activate(wsswiss);

do {
        get_itmtp(wssmfin, &item_type, &lrec);
        datrec = (Itmrecord)calloc(lrec+1, sizeof(char));
        rd_itm(wssmfin,  lrec+1, datrec);
        intrp_itm(item_type, lrec, datrec);
        free(datrec);
} while (item_type != M_CLWK);

close_ws(wssmfin);
deactivate(wsswiss);

trans_1 = gkss->gk_ntran[0];

do {
        pik_mode(wssvect, 1, EVENT, TRUE);
        loc_mode(wssvect, 1, EVENT, TRUE);

        for (i = 0; i < 2; i++) {
           timeout = 8 * 60 * 60;
           event = await(timeout);

           if (event != NULL) {
               if (event->ev_class == PICK) {
                       segname = event->ev_data.ev_pik.pik_seg;
                       pickid  = event->ev_data.ev_pik.pik_pid;
                       pik_mode(wssvect, 1, REQUEST, TRUE);
               }
               else { /* event->ev_class == LOCATE */
                       locate1 = event->ev_data.ev_loc;
                       trans_1 = locate1.loc_nt;
                       loc_mode(wssvect, 1, REQUEST, TRUE);
               }
           }
        }

        sel_cntran(trans_1);

        value = 1;
        drect_val.d_ll.d_x = 0.95; drect_val.d_ll.d_y = 1.0;
        drect_val.d_ur.d_x = 0.95; drect_val.d_ur.d_y = 1.0; /*ll==ur??*/
        drecord_val.val_rec.val_min = 0.0;
        drecord_val.val_rec.val_max = 10.0;
        init_val(wssvect, 1, &value, 1, &drect_val, &drecord_val);

        value = 0;
        drect_val.d_ll.d_x = 0.90; drect_val.d_ll.d_y = 0.95;
        drect_val.d_ur.d_x = 0.95; drect_val.d_ur.d_y = 1.0;
        drecord_val.val_rec.val_min = 0.0;
        drecord_val.val_rec.val_max = 3.14;
        init_val(wssvect, 2, &value, 1, &drect_val, &drecord_val);

        drect_loc.d_ll.d_x = 0.0; drect_loc.d_ll.d_y = 1.0;
        drect_loc.d_ur.d_x = 0.0; drect_loc.d_ur.d_y = 1.0;
        init_loc(wssvect, 1, &locate1, 2, &drect_loc, &drecord_loc);

        val_mode(wssvect, 1, SAMPLE, TRUE);
        val_mode(wssvect, 2, SAMPLE, TRUE);
        loc_mode(wssvect, 1, SAMPLE, TRUE);
        cho_mode(wssvect, 1, EVENT, TRUE);

        fixed.w_x = fixed.w_y = shift.w_x = shift.w_y = 0.0;
        scale_facs[0] = scale_facs[1] = 1.0;
        matrix = ev_trmat(&fixed.w_x, &shift.w_x, 0.0, scale_facs, FALSE);
```

```
            do {
                smp_val(wssvect, 1, &scale);
                smp_val(wssvect, 2, &angle);
                smp_loc(wssvect, 1, &locate2);
                if (locate1.loc_nt != locate2.loc_nt) break;

                pt2_1.w_x = locate2.loc_pt.w_x - locate1.loc_pt.w_x;
                pt2_1.w_y = locate2.loc_pt.w_y - locate1.loc_pt.w_y;
                scale_facs[0] = scale_facs[1] = angle;
                matac = acc_trmat(matrix, &locate1.loc_pt.w_x,
                        &pt2_1.w_x, angle, scale_facs, FALSE);

                transeg(segname, matac);
                timeout = 0.0;
                event = await(timeout);
            } while (event == NULL || event->ev_class != CHOICE);

            val_mode(wssvect, 1, REQUEST, TRUE);
            val_mode(wssvect, 2, REQUEST, TRUE);
            loc_mode(wssvect, 1, REQUEST, TRUE);

            copyseg(wssplot, segname);
            cho_mode(wssvect, 1, REQUEST, TRUE);
        } while (event->ev_data.ev_cho != 2);

        deactivate(wssvect);
        close_ws(wssvect);
        close_ws(wsswiss);

        close_gks();
}
```

References

1. *GKS, Functional Specification of the Graphical Kernel System - ISO IS 7942 Information Processing Systems - Computer Graphics*, 1985.
2. G. Enderle, K. Kansy, and G. Pfaff, *Computer Graphics Programming GKS The Graphics Standard*, Springer Verlag (1984).
3. F.R.A. Hopgood, D.A. Duce, J.R.Gallop, and D.C. Sutcliffe, *Introduction to the Graphical Kernel System GKS*, Academic Press (1983).
4. D.S.H. Rosenthal and P. ten Hagen, "GKS in C," *Eurographics '82*, North Holland (1982).
5. P.J.W. ten Hagen and M.M. de Ruiter, "Segment Grouping, an Extension to The Graphical Kernel System," Report CS-R8623 , Centre for Mathematics and Computer Science, Amsterdam (1986).