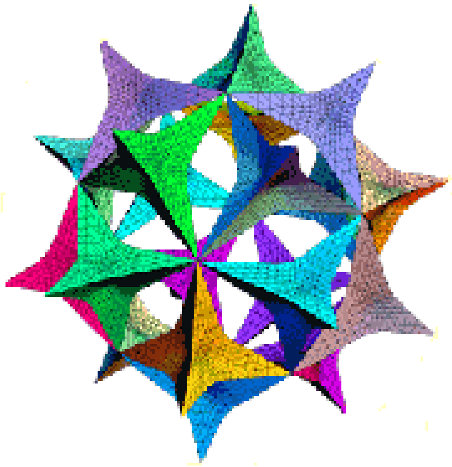


Easy *Math* Solution

`GDSIIExporter` (Version 1.0)

Context

Thank you for your interest in ***Easy Math Solution*** products.

This document does not pretend to be a manual of the GDSII stream format or some kind training course. The document is useful for the users of *GDSIIExporter* (component) to build their own projects with GDSII format exporting features.
To get the complete manual of the format and sample files, please visit http://www.xs4all.nl/~kholwerd/interface/bnf/gdsformat.html. Also, see web site of Cadense Systems Inc., http://www.cadence.com.

Introduction

The GDSII stream format is a standard file format for transferring or archiving 2D graphical design data. The GDSII is most commonly used format for electron beam lithography, photo mask production and optical circuit design.

Easy Math Solution has developed and implemented *GDSIIExporter* as static and dynamic link library, as procedural and object oriented version. It allows using *GDSIIExporter* in C/C++, Delphi, VB, C#, VB.NET and other languages and systems that can understand and implement the dll's calls. Also, one of *GDSIIExporter* version is developed as a managed dll and can be used in .NET environment.

The main characteristics of *GDSIIExporter*:

- easy in use,
- procedural and OOP implementation,
- exception mechanism provided,
- has extended the basic primitives to contain circle, arcs, sector.

Try the DEMO version of *GDSIIExporter* and check it out if it's useful for your project.
Please, review our web site www.e-MathSolution.com regularly for new products and services.
For more detail information, please contact us at:
info@e-MathSolution.com

Price and licenses

As it was mentioned earlier, *GDSIIExporter* component has different implementation. In this chapter we describe tags, which represent variety of the implementations, and type of the licenses.

So, *GDSIIExporter* is implemented as:

- Static Link Library (lib), both procedural and object oriented version;
- Dynamic Link Library (dll), procedural version;
- Managed Dynamic Link Library (dll), .NET version.

Each type of the implementation is tagged on purpose to distinguish different products and to avoid mistakes in the Order Form.
A tag consists of 3 fields separated by point:

xxxxx.xxx.xxx

1st position is a name of the component. Here is GDSII.
2nd position is a binary implementation: lib or dll.
3rd position is a programming technique: PRC describes procedural, OOP – object oriented, or NET for .NET.

License options include Home, Academic, and Professional. Before selecting a license type, please read this part thoroughly.

Home License

With this type of License, *GDSIITranslator* can be used for personal projects only. It covers software development process with *GDSIITranslator* on personally owned computer(s) without any type of a commercial use and distribution.

Academic License

This License is for non-commercial use with in an educational institution and for educational process only.

Professional License

GDSIITranslator can be included royalty-free in commercially distributed projects as well as non-commercial.

Table below presents prices on different implementations of *GDSIITranslator* and licenses.

All prices are given in US dollars

Tag \ License	Home	cademic	rofessional	Use
GDSII.lib.PRC				VC++
GDSII.lib.OOP				VC++
GDSII.dll.PRC				VC++, Delphi, VB, C++Builder, .NET…
GDSII.dll.NET				.NET

*To get the price for *GDSIITranslator*, please, visit: www.e-MathSolution.com.
Want to get more information, please send request to license@e-MathSolution.com .

Interface

Both, procedural and object oriented versions have practically the same interface with a very little difference. Here, we describe procedural version of *GDSIIExporter*.
The interface was developed according to correspond to well-known Buchus Naur Form, which is logical representation of the GDSII format. First of all let's see what Buchus Naur Form is. The form uses the following symbols:

Symbol Name	**Symbol**	**Meaning**
Double Colon	::	"Is composed of."
Square brackets	[ ]	An element which can occur zero or one time.
Braces	{ }	Choose one of the elements within the braces.
Braces with an asterisk	{ }*	The elements within the braces can occur zero or more times.
Braces with a plus	{ }+	The elements within braces must occur one or more times.
Angle brackets	< >	These elements are further defined as separate entities in the syntax list.
Vertical bar	|	Or

The following is the Bachus Naur Form of the GDSII format; the words in capital are the names of *RECORDS*

<stream format> ::=	HEADER BGNLIB [LIBDIRSIZE] [SRFNAME] [libsecur] libname [reflibs] [fonts] [attrtable] [generations] [<FormatType>] UNITS {<structure>}* ENDLIB
<FormatType> ::=	FORMAT | FORMAT {MASK}+ ENDMASKS
<structure> ::=	BNGSTR STRNAME [STRCLASS] {<element>}* ENDSTR
<element> ::=	{<boundary> | <path> | <SREF> | <AREF> | <text> | <node> | <box>} {<property>}* ENDEL
<boundary> ::=	BOUNDARY [EFLAGS] [PLEX] LAYER DATATYPE XY
<path> ::=	PATH [EFLAGS] [PLEX] LAYER DATATYPE [PATHTYPE] [WIDTH] [BGNEXTN] [ENDEXTN] XY
<SREF> ::=	SREF [EFLAGS] [PLEX] SNAME [<strans>] XY
<AREF> ::=	AREF [EFLAGS] [PLEX] SNAME [<strans>] COLROW XY
<text> ::=	TEXT [EFLAGS] [PLEX] LAYER <textbody>
<node> ::=	NODE [EFLAGS] [PLEX] LAYER NODETYPE XY
<box> ::=	BOX [EFLAGS] [PLEX] LAYER BOXTYPE XY
<textbody> ::=	TEXTTYPE [PRESENTATION] [PATHTYPE] [WIDTH] [<strans>] XY STRING
<strans> ::=	STRANS [MAG] [ANGLE]
<property> ::=	PROPATTR PROPVALUE

The interface function signature corresponds to Bachus Naur Form that helps better understand and use *GDSIIExporter*.

The GDSII format is a binary format that is platform independent, because it uses internally defined formats and types (real, integers, string length, date and time records etc.). So, it is very important to use a right type of the function arguments to send a data to *GDSIIExporter.*
To describe the interface function signature, here are used C/C++ basic types that can be represented by equivalent types in other languages and systems. The types **__int16**, **__int32**, **double** have length 2 byte, 4 byte, 8 byte correspondently and **TCHAR** represents the C/C++ null terminated string.

void GDS_BeginLib(const TCHAR *fileName, double dbUserUnit, double dbUnitInMeter);

void GDS_EndLib(void);

The functions should be used when GDSII format file are started and ended. This is logical “brackets” and must be called only once for each new GDSII file.
The functions create records, which can be represented in Bachus Naur Form as:

```
HEADER BGNLIB LIBNAME UNITS
[{BGNSTR STRNAME [{<element>}+] ENDSTR}+]
ENDLIB
```

`[{BGNSTR STRNAME [{<element>}+] ENDSTR}+]` – the structures and elements. An element portion of a stream file: <boundary> | <path> | <sref> | <aref> | <text> | <node> | <box> ENDEL. The structures and elements can be inserted in the file by other interface functions (see below).

The arghuments:
filename – name of the GDSII format file;
dbUserUnit - the size of a database unit in user units;
dbUnitInMeter - the size of a database unit in meters.
For example, if you create a library with the default units (user unit = 1 micron and 1000 database units per user unit), the first number is .001, and the second number is 1E-9. Typically, the first number is less than 1, since you use more than 1 database unit per user unit. To calculate the size of a user unit in meters, divide the second number by the first.

Example:

```
int main(int argc, char* argv[])
{
        TCHAR *src1="ems.gds";
        TCHAR *src2="test.gds";

        GDS_BeginLib(src1, 1e-3, 1e-9);
                        // [{BGNSTR STRNAME [{<element>}+] ENDSTR}+]
        GDS_EndLib();
        ……
        GDS_BeginLib(src2, 1e-3, 1e-9);
                        // [{BGNSTR STRNAME [{<element>}+] ENDSTR}+]
        GDS_EndLib();
}
```

void GDS_BeginStructure(const TCHAR* strName);

void GDS_EndStructure(void);

The functions are logical “brackets” for structure element. The GDSII format presumes zero or more structures in the library. In Bachus Naur Form the element can be represented as:

BGNSTR STRNAME [STRCLASS] [{<element>}+] ENDSTR

Each structure has two header records and one tail record that sandwich an arbitrary list of elements. The first structure header is the BGNSTR record, which contains the creation date and the last modification date. Following that is the STRNAME record, which names the structure using any alphabetic or numeric characters, the dollar sign, or the underscore. Then the structure is opened and any of the elements can be listed.
The last record of the structure is ENDSTR. Following it must be another BGNSTR or the end of the library, ENDLIB.

The arguments:
strName – name of the structure. Up to 32 characters in GDSII, A-Z, a-z, 0-9, _, ?, and $ are all legal characters.
The functions create records, which can be represented in Bachus Naur Form as:

```
BNGSTR STRNAME {<element>}* ENDSTR
```

{<element>}* - element portion of a stream file: <boundary> | <path> | <sref> | <aref> | <text> | <node> | <box> ENDEL.

Example:

```
int main(int argc, char* argv[])
{
        TCHAR *src="ems.gds";

        GDS_BeginLib(src1, 1e-3, 1e-9);
                GDS_BeginStructure(“str1”);
                        // {<element>}*
                GDS_EndStructure();

                GDS_BeginStructure(“str2”);
                        // {<element>}*
                GDS_EndStructure();
        GDS_EndLib();
}
```

```
void GDS_Path(__int32 *pointsX, __int32 *pointsY,
                 __int16 pointsNum, __int32 width,
                 __int16 layer, __int16 datatype,
                  __int16 pathtype, __int16 elflags, __int32
                 plex);
```

The function creates a PATH element. In Bachus Naur Form the element can be represented as:

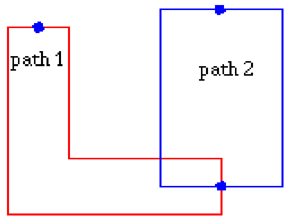
```
PATH [ELFLAGS][PLEX] LAYER DATATYPE [PATHTYPE]
[WIDTH] XY
```

A path is an open figure with a nonzero width that is typically used to place wires. This element is initiated with a PATH record followed by the optional ELFLAGS and PLEX records. The LAYER record must follow to identify the desired path material. Also, a DATATYPE record must appear and an XY record to define the coordinates of the path. From two to 200 points may be given in a path.
Prior to the XY record of a path specification there may be two optional records called PATHTYPE and WIDTH. The PATHTYPE record describes the nature of the path segment ends, according to its parameter value. If the value is 0, the segments will have square ends that terminate at the path vertices. The value 1 indicates rounded ends and the value 2 indicates square ends that overlap their vertices by one-half of their width. The width of the path is defined by the optional WIDTH record.

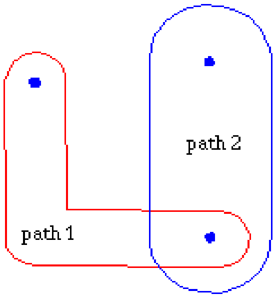
The arguments:
pointsX, pointsY – arrays of the XY coordinates to be connected by a segment;
pointsNum – number of points in the arrays *pointsX* and *pointsY.* Path elements may have a minimum of 2 and a maximum of 200 coordinates. The max points can be changed up on customer request;
width - [WIDTH], specifies the width of a path or text lines in database units. A negative value for width means that the width is absolute, that is, the width is not affected by the magnification factor of any parent reference. If omitted, zero is assumed;
layer – LAYER, the value of the layer must be in the range of 0 to 255;
datatype – `DATATYPE,` the value of the datatype must be in the range of 0 to 255;
pathtype - [PATHTYPE], value that describes the type of path endpoints. The value is

- 0 for square-ended paths that end flush with their endpoints
- 1 for round-ended paths
- 2 for square-ended paths that extend a half-width beyond their endpoints

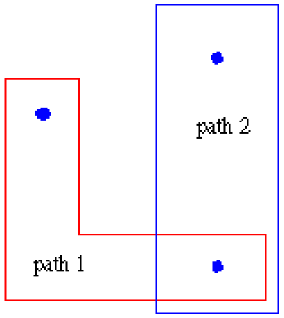
If not specified, a Path-type of 0 is assumed. The following picture shows the path types:



Pathtype 0 produces a square-ended path, ending flush with the digitized endpoints. This is the default pathtype if none is specified



Pathtype 1 produces a round-ended path. The two ends are semicircular with center at the digitized endpoints.



Pathtype 2 produces a square-ended path. The ends of the pathextend beyond the digitized end-points by one-half the path width.

Predefined constants:

```
// PATHTYPE
const   __int16 PATHTYPE0=0;
const   __int16 PATHTYPE1=1;
const   __int16 PATHTYPE2=2;
```

elflags - [ELFLAGS], bit 15 (the rightmost bit)specifies Template data. Bit 14 specifies External data (also referred to as Exterior data). All other bits are currently unused and must be cleared to 0. If this record is omitted, all bits are assumed to be 0. The following shows an ELFLAGS record. For additional information on Template data, consult the GDSII Reference Manual.
Predefined constants:

```
//ELFLAGS
const   __int16 ELFLAGS_0=0;
const   __int16 ELFLAGS_TEMPLATE=1;
const   __int16 ELFLAGS_EXTERIOR=2;
```

plex - [PLEX], a unique positive number which is common to all elements of the plex to which this element belongs. The head of the plex is flagged by setting the seventh bit; therefore, plex numbers should be small enough to occupy only the right-most 24 bits.
Predefined constant:

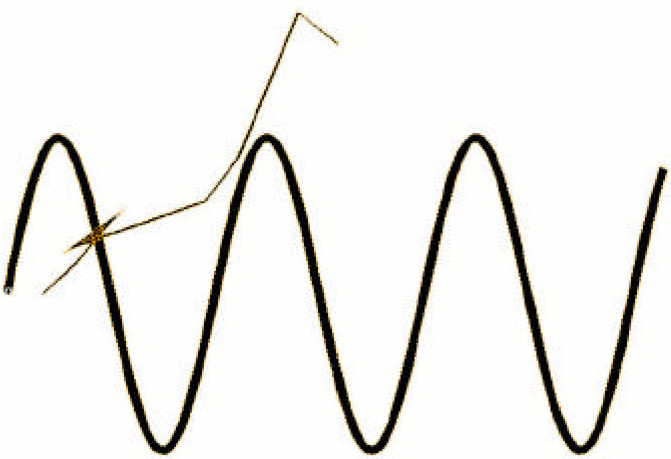
```
const __int32 PLEX_HEAD=0x1000000
```

Example:

```
#define pt 199

int main(int argc, char* argv[])
{
   int x1[pt];
   int y1[pt];
```

```
    char *src="ems.gds";
    int x[]={110,210,327,200,600,700,880,1000};
    int y[]={0,100,250,150,300,440,900,800};

    //To build SIN function
    double k;
    for(int i=0; i<pt; i++)
    {
            k=i/10.;
            x1[i]=i*10;
            y1[i]=long(sin(k)*500);
    }

GDS_BeginLib(src, 1e-3, 1e-9);
    GDS_BeginStructure("str1");
      GDS_Path( x, y, 8 ,10, 1,63, PATHTYPE1, 0, 0);
      GDS_Path( x1, y1, pt , 30, 1, 63, PATHTYPE1,
      ELFLAGS_TEMPLATE,0);
    GDS_EndStructure();
GDS_EndLib();

return 0;
}
```

Result:



```
void GDS_Arc(__int32 centerX, __int32 centerY, __int32
              radius, double startAng, double endAng,
              __int16 segmentNum, __int32 width,
              __int16 layer, __int16 datatype,
              __int16 pathtype, __int16 elflags, __int32 plex);
```

The function is an extension of the GDSII stream format and builds an arc. GDS_Arc uses PATH record and as result uses similar arguments (see GDS_Path).

The arguments:

centerX, centerY –coordinates of an arc center;

radius – radius of an arc;

startAng, endAng - start angle and end angle of an arc in degree;

segmentNum – a number of segments to draw an arc. The number is limited by 200 (see max pints for PATH).

width - [WIDTH], specifies the width of a path or text lines in database units. A negative value for width means that the width is absolute, that is, the width is not affected by the magnification factor of any parent reference. If omitted, zero is assumed;

layer – LAYER, the value of the layer must be in the range of 0 to 255;

datatype – `DATATYPE,` the value of the datatype must be in the range of 0 to 255;

pathtype - [PATHTYPE], value that describes the type of path endpoints. The value is

- 0 for square-ended paths that end flush with their endpoints
- 1 for round-ended paths
- 2 for square-ended paths that extend a half-width beyond their endpoints

If not specified, a Path-type of 0 is assumed (see GDS_Path for more details).

Predefined constants:

```
// PATHTYPE
const   __int16 PATHTYPE0=0;
const   __int16 PATHTYPE1=1;
const   __int16 PATHTYPE2=2;
```

elflags - [ELFLAGS], bit 15 (the rightmost bit)specifies Template data. Bit 14 specifies External data (also referred to as Exterior data). All other bits are currently unused and must be cleared to 0. If this record is omitted, all bits are assumed to be 0. The following shows an ELFLAGS record. For additional information on Template data, consult the GDSII Reference Manual.

Predefined constants:

```
//ELFLAGS
const   __int16 ELFLAGS_0=0;
const   __int16 ELFLAGS_TEMPLATE=1;
const   __int16 ELFLAGS_EXTERIOR=2;
```

plex - [PLEX], a unique positive number which is common to all elements of the plex to which this element belongs. The head of the plex is flagged by setting the seventh bit; therefore, plex numbers should be small enough to occupy only the right-most 24 bits.

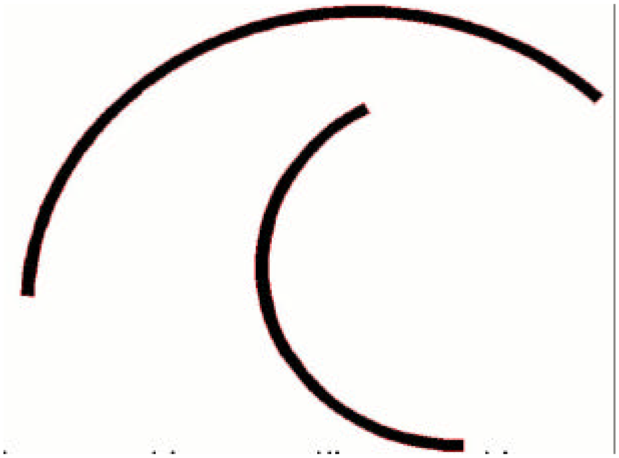
Predefined constant:

```
const  __int32 PLEX_HEAD=0x1000000
```

Example:

```
int main(int argc, char* argv[])
{
        char *src="ems.gds";

GDS_BeginLib(src, 1e-3, 1e-9);
    GDS_BeginStructure("str1");
        GDS_Arc(100, 150, 1000, 45, 178, 135,40, 0, 0, PATHTYPE1,0,0);
        GDS_Arc(400, 300, 600, 120, 270, 100,40, 0, 0, PATHTYPE2,0,0);
    GDS_EndStructure();
GDS_EndLib();

        return 0;
}
```

Result:



void GDS_Boundary(__int32 *pointsX, __int32 *pointsY, __int16 pointsNum, __int16 layer,__int16 datatype, __int16 elflags, __int32 plex);

The function creates a BOUDARY element. In Bachus Naur Form the element can be represented as:

```
BOUNDARY[ELFLAGS][PLEX]LAYER DATATYPE XY
```

The boundary element defines a filled polygon. It begins with a BOUNDARY record, has an optional ELFLAGS and PLEX record, and then has required LAYER, DATATYPE, and XY records.

The ELFLAGS record, which appears optionally in every element, has two flags in its parameter to indicate template data (if bit 16 is set) or external data (if bit 15 is set). This record should be ignored on input and excluded from output. Note that the GDS II integer has bit 1 in the leftmost or most significant position so these two flags are in the least significant bits.

The PLEX record is also optional to every element and defines element structuring by aggregating those that have common plex numbers. Although a 4-byte integer is available for plex numbering, the high byte (first byte) is a flag that indicates the head of the plex if its least significant bit (bit 8) is set.

The LAYER record is required to define which layer is to be used for this boundary. The meaning of these layers is not defined rigorously and must be determined for each design environment and library.

The DATATYPE record contains unimportant information and its argument can be zero.

The XY record contains anywhere from four to 200 coordinate pairs that define the outline of the polygon. The record length defines the number of points in this record. Note that boundaries must be closed explicitly, so the first and last coordinate values must be the same.

The arguments:

pointsX, *pointsY* – arrays of the XY coordinates to be connected by a segment;

pointsNum – number of points in the arrays *pointsX* and *pointsY*. Boundary elements may have a minimum of 4 and a maximum of 200 coordinates. The max points can be changed up on customer request;

layer – LAYER, the value of the layer must be in the range of 0 to 255;

datatype – `DATATYPE,` the value of the datatype must be in the range of 0 to 255;

elflags - [ELFLAGS], bit 15 (the rightmost bit)specifies Template data. Bit 14 specifies External data (also referred to as Exterior data). All other bits are currently unused and must be cleared to 0. If this record is omitted, all bits are assumed to be 0. The following shows an ELFLAGS record. For additional information on Template data, consult the GDSII Reference Manual.
Predefined constants:

```
//ELFLAGS
const   __int16 ELFLAGS_0=0;
const   __int16 ELFLAGS_TEMPLATE=1;
const   __int16 ELFLAGS_EXTERIOR=2;
```

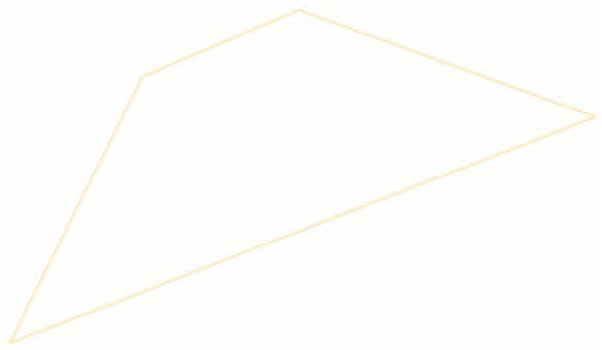
plex - [PLEX], an unique positive number which is common to all elements of the plex to which this element belongs. The head of the plex is flagged by setting the seventh bit; therefore, plex numbers should be small enough to occupy only the right-most 24 bits. Predefined constant:

```
const __int32 PLEX_HEAD=0x1000000
```

Example:

```
int main(int argc, char* argv[])
{
        char *src="ems.gds";
        int x[]={110,210,327,550};
        int y[]={0,200,250,170};

GDS_BeginLib(src, 1e-3, 1e-9);
    GDS_BeginStructure("str1");
        GDS_Boundary( x, y, 4 ,1, 61, 0, 0);
    GDS_EndStructure();
GDS_EndLib();

        return 0;
}
```

Result:



void GDS_Circle(__int32 centerX, __int32 centerY, __int32 radius, __int16 segmentNum, __int16 layer, __int16 datatype, __int16 elflags, __int32 plex);

The function is an extension of the GDSII stream format and builds a circle. GDS_Circle uses BOUNDORY record and as result uses similar arguments (see GDS_Boundary).

The arguments:

centerX, *centerY* –coordinates of a circle center;

radius – radius of a circle;

segmentNum – a number of segments to draw a circle. The number is limited by 200 (see max pints for BOUNDARY).

layer – LAYER, the value of the layer must be in the range of 0 to 255;

datatype – `DATATYPE,` the value of the datatype must be in the range of 0 to 255;

elflags - [ELFLAGS], bit 15 (the rightmost bit)specifies Template data. Bit 14 specifies External data (also referred to as Exterior data). All other bits are currently unused and must be cleared to 0. If this record is omitted, all bits are assumed to be 0. The following shows an ELFLAGS record. For additional information on Template data, consult the GDSII Reference Manual.
Predefined constants:

```
//ELFLAGS
const   __int16 ELFLAGS_0=0;
const   __int16 ELFLAGS_TEMPLATE=1;
const   __int16 ELFLAGS_EXTERIOR=2;
```

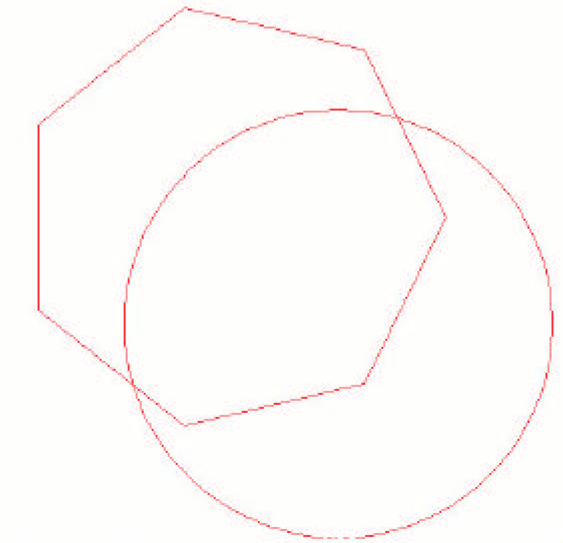
plex - [PLEX], an unique positive number which is common to all elements of the plex to which this element belongs. The head of the plex is flagged by setting the seventh bit; therefore, plex numbers should be small enough to occupy only the right-most 24 bits. Predefined constant:

```
const __int32 PLEX_HEAD=0x1000000
```

Example:

```
int main(int argc, char* argv[])
{
        char *src="ems.gds";
GDS_BeginLib(src, 1e-3, 1e-9);
        GDS_BeginStructure("str1");
                GDS_Circle(1500, 1000, 1000, 150, 0,0,0,0);
                GDS_Circle(1000, 1500, 1000, 7, 0,0,0,0);
        GDS_EndStructure();
GDS_EndLib();

                return 0;
}
```

Result:



```
void GDS_Sector(__int32 centerX, __int32 centerY,
                __int32 radius, double startAng, double endAng,
                __int16 segmentNum, __int16 layer,
                 __int16 datatype, __int16 elflags, __int32
                plex);
```

The function is an extension of the GDSII stream format and builds a sector. GDS_Sector uses BOUNDORY.

The arguments:

centerX, centerY –coordinates of the sector center;

radius – radius of a sector;

startAng, endAng - start angle and end angle of an arc in degree;

segmentNum – a number of segments to draw a circle. The number is limited by 200 (see max pints for BOUNDARY).

layer – LAYER, the value of the layer must be in the range of 0 to 255;

datatype – `DATATYPE,` the value of the datatype must be in the range of 0 to 255;

elflags - [ELFLAGS], bit 15 (the rightmost bit)specifies Template data. Bit 14 specifies External data (also referred to as Exterior data). All other bits are currently unused and must be cleared to 0. If this record is omitted, all bits are assumed to be 0. The following shows an ELFLAGS record. For additional information on Template data, consult the GDSII Reference Manual.

Predefined constants:

```
//ELFLAGS
const   __int16 ELFLAGS_0=0;
const   __int16 ELFLAGS_TEMPLATE=1;
const   __int16 ELFLAGS_EXTERIOR=2;
```

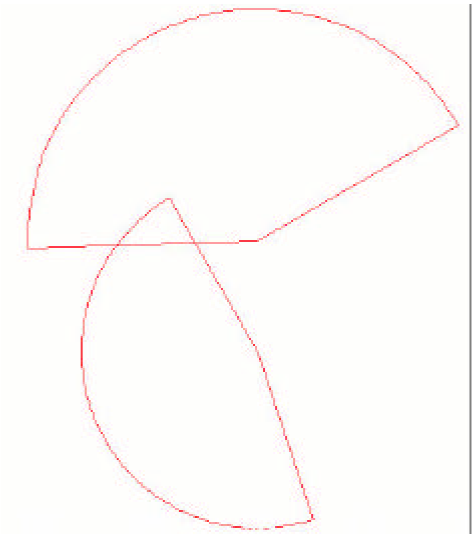
plex - [PLEX], an unique positive number which is common to all elements of the plex to which this element belongs. The head of the plex is flagged by setting the seventh bit; therefore, plex numbers should be small enough to occupy only the right-most 24 bits. Predefined constant:

```
const  __int32 PLEX_HEAD=0x1000000
```

Example:

```
int main(int argc, char* argv[])
{
                char *src="ems.gds";

GDS_BeginLib(src, 1e-3, 1e-9);
        GDS_BeginStructure("str1");
                GDS_Sector(1000, 1500, 1050, 30, 185,100, 0,0,0,0);
                GDS_Sector(1000, 1000, 800, 120, 290,199, 0,0,0,0);
        GDS_EndStructure();
GDS_EndLib();
}
```

Result:



```
void  GDS_Box(__int32 x1, __int32 y1, __int32 x2, __int32 y2,
              __int16 layer, __int16 boxtype,
              __int16 elflags, __int32 plex);

void  GDS_BoxWH(__int32 x, __int32 y, __int32 width,
                __int32 height,  __int16 layer,
                 __int16 boxtype,  __int16 elflags,
                __int32 plex);

void GDS_Square(__int32 x, __int32 y, __int32 side,
                __int16 layer, __int16 boxtype,  __int16
                elflags, __int32 plex);
```

The function creates a BOX element. In Bachus Naur Form the element can be represented as:

```
BOX [ELFLAGS] [PLEX] LAYER BOXTYPE XY
```

Optional ELFLAGS and PLEX records follow the BOX record, a mandatory LAYER record, a BOXTYPE record with a zero argument, and an XY record. The XY must contain five points that describe a closed, four-sided box. Unlike the boundary, this is not a filled figure.

The arguments:

x1, y1, x2, y2 – diagonal coordinates;

layer – LAYER, the value of the layer must be in the range of 0 to 255;

boxtype – `BOXTYPE,` the value of the boxtype must be in the range of 0 to 255;

elflags - [ELFLAGS], bit 15 (the rightmost bit)specifies Template data. Bit 14 specifies External data (also referred to as Exterior data). All other bits are currently unused and must be cleared to 0. If this record is omitted, all bits are assumed to be 0. The following shows an ELFLAGS record. For additional information on Template data, consult the GDSII Reference Manual.

Predefined constants:

```
//ELFLAGS
const   __int16 ELFLAGS_0=0;
const   __int16 ELFLAGS_TEMPLATE=1;
const   __int16 ELFLAGS_EXTERIOR=2;
```

plex - [PLEX], an unique positive number which is common to all elements of the plex to which this element belongs. The head of the plex is flagged by setting the seventh bit; therefore, plex numbers should be small enough to occupy only the right-most 24 bits. Predefined constant:

```
const __int32 PLEX_HEAD=0x1000000
```

GDS_BoxWH is an extension of the GDSII stream format and builds a box. GDS_BoxWH uses BOX record.

x, y – coordinates of the init point (the left-bottom point);

width, height – width and height of the box;

GDS_Square is an extension of the GDSII stream format and builds a square. GDS_Square uses BOX record.

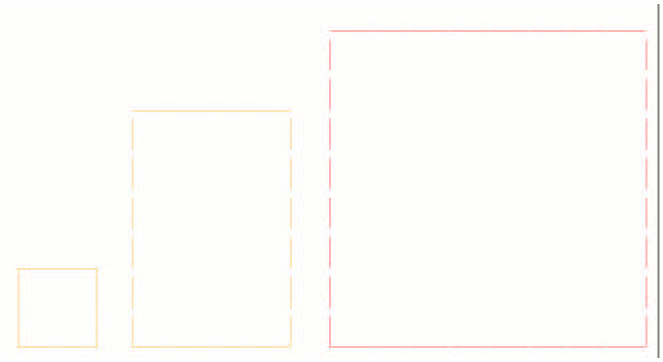
x, y – coordinates of the init point (the left-bottom point);

side – length of the square side;

Example:

```
int main(int argc, char* argv[])
{
        char *src="ems.gds";

GDS_BeginLib(src, 1e-3, 1e-9);
    GDS_BeginStructure("str1");
        GDS_Box(105, 105,  205, 205,0,10, 0, 0);
        GDS_BoxWH(250, 105,  200, 300,0,10, 0, 0);
        GDS_Square(500, 105, 400,0,0,0,0);
    GDS_EndStructure();
GDS_EndLib();
}
```

Result:



```
void GDS_Text(__int32 pointX, __int32 pointY,
              const TCHAR *msg, __int16 layer,
               __int16 txtType, __int16 presentation,
              __int16 pathtype, __int32 width,
              __int16 strans, double mag, double angle,
              __int16 elflags,     __int32 plex);
```

The function creates TEXT element. In Bachus Naur Form the element can be represented as:

```
TEXT [ELFLAGS] [PLEX] LAYER TEXTTYPE
[PRESENTATION] [PATHTYPE] [WIDTH] [STRANS
[MAG] [ANGLE]] XY STRING
```

Messages can be included in a circuit with the TEXT record. The optional ELFLAGS and PLEX follow with the mandatory LAYER record after that. A TEXTTYPE record must then appear. An optional PRESENTATION record specifies the font in bits 11 and 12, the vertical presentation in bits 13 and 14 (0 for top, 1 for middle, 2 for bottom), and the horizontal presentation in bits 15 and 16 (0 for left, 1 for center, 2 for right). Optional PATHTYPE, WIDTH, STRANS, MAG, and ANGLE records may appear to affect the text. The last two records are required: an XY with a single coordinate to locate the text and a STRING record to specify the actual text.

The arguments:

pointX, pointY – init coordinates of the text to print;

msg – text to be printed;

layer – LAYER, the value of the layer must be in the range of 0 to 255;

txtType- the value of the texttype must be in the range 0 to 255;

presentation - bits 10 and 11, taken together as a binary number, specify the font (00 means font 0, 01 means font 1, 10 means font 2, and 11 means font 3). Bits 12 and 13 specify the vertical justification (00 means top, 01 means middle, and 10 means bottom). Bits 14 and 15 specify the horizontal justification (00 means left, 01 means center, and 10 means right). Bits 0 through 9 are reserved for future use and must be cleared. If this record is omitted, then top-left justification and font 0 are assumed. The following shows a PRESENTATION record.

```
//Presentation bits
//Horizontal justification
const   __int16 PRESENT_HLEFT=0x0;
const   __int16 PRESENT_HCENTER=0x1;
const   __int16 PRESENT_HRIGHT=0x2;

//Vertical justification
const   __int16 PRESENT_VTOP=0x0;
const   __int16 PRESENT_VMIDDLE=0x4;
const   __int16 PRESENT_VBOTTOM=0x8;

//Font number
const   __int16 PRESENT_FONT0=0x0;
const   __int16 PRESENT_FONT1=0x10;
const   __int16 PRESENT_FONT2=0x20;
const   __int16 PRESENT_FONT3=0x30;
```

pathtype – see GDS_Path;

width - [WIDTH], specifies the width of the text lines in database units. A negative value for width means that the width is absolute, that is, the width is not affected by the magnification factor of any parent reference. If omitted, zero is assumed;

strans – [STRANS], bit 0 (the leftmost bit) specifies reflection. If bit 0 is set, the element is reflected about the X-axis before angular rotation. For an Aref, the entire array is reflected, with the individual array members rigidly attached. Bit 13 flags absolute magnification. Bit 14 flags absolute angle. Bit 15 (the rightmost bit) and all remaining bits are reserved for future use and must be cleared. If this record is omitted, the element is assumed to have no reflection, non-absolute magnification, and non-absolute angle. The following shows a STRANS record.

```
//STRANS
const   __int16 STRANS_0=0;
const   __int16 STRANS_ABSMAG=0x4;
const   __int16 STRANS_ABSANG=0x2;
const   __int16 STRANS_REFLECT=0x8000;
```

mag – [MAG], contains a double-precision real number (8 bytes), which is the magnification factor. If this record is omitted, a magnification factor of one is assumed.

angle – [ANGLE], angular rotation factor in CCW direction. If omitted, the default is 0. The angle of rotation is measured in degrees.

elflags - [ELFLAGS], bit 15 (the rightmost bit)specifies Template data. Bit 14 specifies External data (also referred to as Exterior data). All other bits are currently unused and must be cleared to 0. If this record is omitted, all bits are assumed to be 0. The following shows an ELFLAGS record. For additional information on Template data, consult the GDSII Reference Manual.

Predefined constants:

```
//ELFLAGS
const   __int16 ELFLAGS_0=0;
const   __int16 ELFLAGS_TEMPLATE=1;
const   __int16 ELFLAGS_EXTERIOR=2;
```

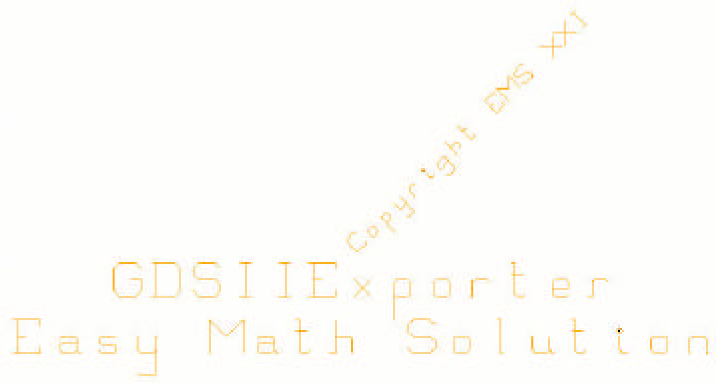
plex - [PLEX], an unique positive number which is common to all elements of the plex to which this element belongs. The head of the plex is flagged by setting the seventh bit; therefore, plex numbers should be small enough to occupy only the right-most 24 bits. Predefined constant:

```
const __int32 PLEX_HEAD=0x1000000
```

Example:

```
int main(int argc, char* argv[])
{
        char *src="ems.gds";

GDS_BeginLib(src, 1e-3, 1e-9);
    GDS_BeginStructure("str1");
        GDS_Text( 100, 100, "Easy Math Solution", 1, 4,
                PRESENT_HCENTER |PRESENT_VBOTTOM,
                PATHTYPE1, 0, STRANS_0, 0.1, 0, 0, 0);
        GDS_Text( 100, 300, "GDSIIExporter", 1, 4,
                PRESENT_HCENTER |PRESENT_VBOTTOM,
                PATHTYPE1, 0, STRANS_0, 0.1, 0, 0, 0);
        GDS_Text( 100, 500, "Copyright EMS XXI", 1, 4,
                PRESENT_HLEFT |PRESENT_VBOTTOM, PATHTYPE1,
                0, STRANS_0, 0.05, 45, 0, 0);
    GDS_EndStructure();
GDS_EndLib();
return 0;
}
```

Result:



```
void GDS_Sref(const TCHAR *strName, __int32 x, __int32 y,
              __int16 strans, double mag, double angle,
              __int16 elflags, __int32 plex);
```

The function creates a SREF structure (structure reference element). In Bachus Naur Form the element can be represented as:

```
SREF [ELFLAGS] [PLEX] SNAME [STRANS [MAG]
[ANGLE]] XY
```

Hierarchy is achieved by allowing structure references (instances) to appear in other structures. The SREF record indicates a structure reference and is followed by the optional ELFLAGS and PLEX records. The SNAME record then names the desired structure and an XY record contains a single coordinate to place this instance. It is legal to make reference to structures that have not yet been defined with STRNAME. Prior to the XY record there may be optional transformation records. The STRANS record must appear first if structure transformations are desired. Its parameter has bit flags that indicate mirroring in *x* before rotation (if bit 1 is set), the use of absolute magnification (if bit 14 is set), and the use of absolute rotation (if bit 15 is set). The magnification and rotation amounts may be specified in the optional MAG and ANGLE records. The rotation angle is in counterclockwise degrees.

The arguments:

strName – a name of structure to be referenced.

x,y – a point coordinate where the structure to be printed.

strans – [STRANS], bit 0 (the leftmost bit) specifies reflection. If bit 0 is set, the element is reflected about the X-axis before angular rotation. For an Aref, the entire array is reflected, with the individual array members rigidly attached. Bit 13 flags absolute magnification. Bit 14 flags absolute angle. Bit 15 (the rightmost bit) and all remaining bits are reserved for future use and must be cleared. If this record is omitted, the element is assumed to have no reflection, non-absolute magnification, and non- absolute angle. The following shows a STRANS record.

```
//STRANS
const   __int16 STRANS_0=0;
const   __int16 STRANS_ABSMAG=0x4;
const   __int16 STRANS_ABSANG=0x2;
const   __int16 STRANS_REFLECT=0x8000;
```

mag – [MAG], contains a double-precision real number (8 bytes), which is the magnification factor. If this record is omitted, a magnification factor of one is assumed.

angle – [ANGLE], angular rotation factor in CCW direction. If omitted, the default is 0. The angle of rotation is measured in degrees.

elflags - [ELFLAGS], bit 15 (the rightmost bit)specifies Template data. Bit 14 specifies External data (also referred to as Exterior data). All other bits are currently unused and must be cleared to 0. If this

record is omitted, all bits are assumed to be 0. The following shows an ELFLAGS record. For additional information on Template data, consult the GDSII Reference Manual.
Predefined constants:

```
//ELFLAGS
const   __int16 ELFLAGS_0=0;
const   __int16 ELFLAGS_TEMPLATE=1;
const   __int16 ELFLAGS_EXTERIOR=2;
```

plex - [PLEX], a unique positive number which is common to all elements of the plex to which this element belongs. The head of the plex is flagged by setting the seventh bit; therefore, plex numbers should be small enough to occupy only the right-most 24 bits.
Predefined constant:

```
const  __int32 PLEX_HEAD=0x1000000
```

Example:

```
int main(int argc, char* argv[])
{
        char *src="ems.gds";

GDS_BeginLib(src, 1e-3, 1e-9);

    GDS_BeginStructure("str3");
        GDS_Sref("str1", 1000, 1000,0,1,30,ELFLAGS_TEMPLATE,0);
    GDS_EndStructure();

    GDS_BeginStructure("str1");
        GDS_Text( 100, 100, "Easy Math Solution", 1, 4,
                PRESENT_HCENTER |PRESENT_VBOTTOM,
                PATHTYPE1, 0, STRANS_0, 0.1, 0, 0, 0);
        GDS_Text( 100, 300, "GDSIIExporter", 1, 4,
                PRESENT_HCENTER |PRESENT_VBOTTOM,
                PATHTYPE1, 0, STRANS_0, 0.1, 0, 0, 0);
        GDS_Text( 100, 500, "Copyright EMS XXI", 1, 4,
                PRESENT_HLEFT |PRESENT_VBOTTOM, PATHTYPE1,
                0, STRANS_0, 0.05, 45, 0, 0);
    GDS_EndStructure();

GDS_EndLib();
return 0;
}
```

```
void  GDS_Aref( const TCHAR *strName, __int32 *x,
              __int32 *y, __int16 col, __int16 row,
              __int16 strans, double mag, double angle,
              __int16 elflags, __int32 plex);
```

The function creates a AREF structure (structure array reference element). In Bachus Naur Form the element can be represented as:

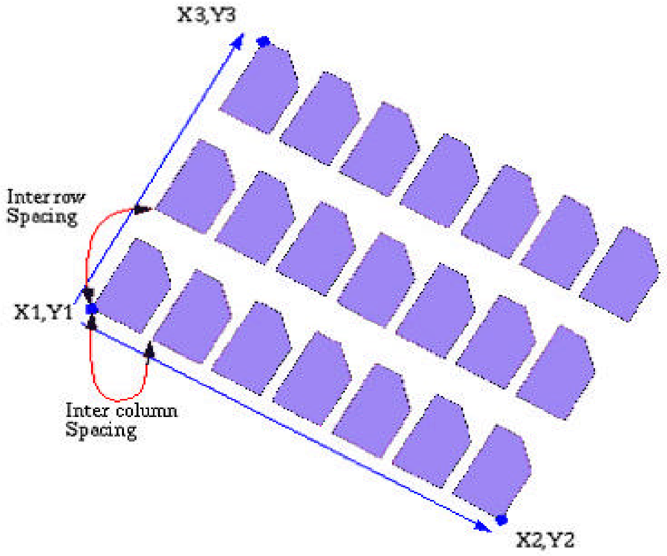
```
AREF [ELFLAGS] [PLEX] SNAME  [STRANS [MAG]
[ANGLE]] COLROW XY
```

For convenience, an array of structure instances can be specified with the AREF record. Following the optional ELFLAGS and PLEX records comes the SNAME to identify the structure being arrayed. Next, the optional transformation records STRANS, MAG, and ANGLE give the orientation of the instances. A COLROW record must follow to specify the number of columns and the number of rows in the array. The final record is an XY with three points: the coordinate of the corner instance, the coordinate of the last instance in the columnar direction, and the coordinate of the last instance in the row direction. From this information, the amount of instance overlap or separation can be determined. Note that flipping arrays (in which alternating rows or columns are mirrored to abut along the same side) can be implemented with multiple arrays that are interlaced and spaced apart to describe alternating rows or columns.

The arguments:

strName – a name of the structure to be referenced.

x,y – an array of the coordinates where the structure array to be printed. An Aref has exactly three coordinates. In an Aref, the first coordinate is the array reference point (origin point). The other two coordinates are already rotated, reflected as specified in the STRANS record (if specified). So in order to calculate the inter column and inter row spacing, the coordinates must be mapped back to their original position, or the vector length (x1,y1-> x3,y3) must be divided by the number of row etc. . The second coordinate locates a position, which is displaced from the reference point by the inter-column spacing times the number of columns. The third coordinate locates a position, which is displaced from the reference point by the inter-row spacing times the number of rows. For an example of an array lattice see the next picture.



col, row – columns and rows for an AREF. Two 2 byte integers. The first is the number of columns. The second is the number of rows. Neither may exceed 32767.

strans – [STRANS], bit 0 (the leftmost bit) specifies reflection. If bit 0 is set, the element is reflected about the X-axis before angular rotation. For an Aref, the entire array is reflected, with the individual array members rigidly attached. Bit 13 flags absolute magnification. Bit 14 flags absolute angle. Bit 15 (the rightmost bit) and all remaining bits are reserved for future use and must be cleared. If this record is omitted, the element is assumed to have no reflection, non-absolute magnification, and non- absolute angle. The following shows a STRANS record.

```
//STRANS
const   __int16 STRANS_0=0;
const   __int16 STRANS_ABSMAG=0x4;
const   __int16 STRANS_ABSANG=0x2;
const   __int16 STRANS_REFLECT=0x8000;
```

mag – [MAG], contains a double-precision real number (8 bytes), which is the magnification factor. If this record is omitted, a magnification factor of one is assumed.

angle – [ANGLE], angular rotation factor in CCW direction. If omitted, the default is 0. The angle of rotation is measured in degrees.

elflags - [ELFLAGS], bit 15 (the rightmost bit)specifies Template data. Bit 14 specifies External data (also referred to as Exterior data). All other bits are currently unused and must be cleared to 0. If this record is omitted, all bits are assumed to be 0. The following shows an ELFLAGS record. For additional information on Template data, consult the GDSII Reference Manual.

Predefined constants:

```
//ELFLAGS
const   __int16 ELFLAGS_0=0;
const   __int16 ELFLAGS_TEMPLATE=1;
const   __int16 ELFLAGS_EXTERIOR=2;
```

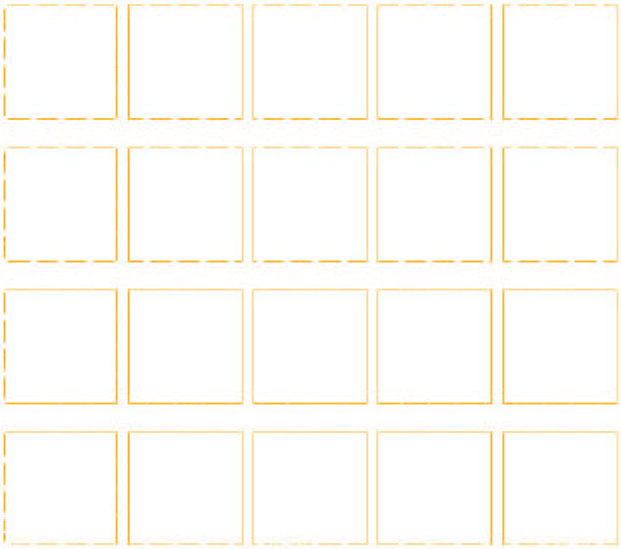
plex - [PLEX], an unique positive number which is common to all elements of the plex to which this element belongs. The head of the plex is flagged by setting the seventh bit; therefore, plex numbers should be small enough to occupy only the right-most 24 bits. Predefined constant:

```
const __int32 PLEX_HEAD=0x1000000
```

Example:

```
int main(int argc, char* argv[])
{
        char *src="ems.gds";
        int xar[]={0,1100,0};
        int yar[]={0,0,1000};

GDS_BeginLib(src, 1e-3, 1e-9);

    GDS_BeginStructure("str2");
        GDS_Aref("ur11",  xar, yar,5,4,0,1,0,0,0);
    GDS_EndStructure();

    GDS_BeginStructure("ur11");
        GDS_Box(0,0,200,200, 0,10, 0, 0);
    GDS_EndStructure();

GDS_EndLib();
return 0;
}
```

Result:



```
void  GDS_Node(__int32 *pointsX, __int32 *pointsY,
        __int16 pointsNum, __int16 layer, __int16 nodeType,
        __int16 elflags,  __int32 plex);
```

The function creates a NODE structure. In Bachus Naur Form the element can be represented as:

```
NODE [ELFLAGS] [PLEX] LAYER NODETYPE XY
```

Electrical nets may be specified with the NODE record. The optional ELFLAGS and PLEX records follow and the required LAYER record is next. A NODETYPE record must appear with a zero argument, followed by an XY record with one to 50 points that identify coordinates on the electrical net. The information in this element is not graphical and does not affect the manufactured circuit. Rather, it is for other CAD systems that use topological information.

The arguments:

pointsX, pointsY – arrays of the XY;

pointsNum – number of points in the arrays *pointsX* and *pointsY.* Node elements may have a minimum of 1 and a maximum of 50 coordinates;

layer – LAYER, the value of the layer must be in the range of 0 to 255;

nodeType - the value of the nodetype must be in the range of 0 to 255;

elflags - [ELFLAGS], bit 15 (the rightmost bit)specifies Template data. Bit 14 specifies External data (also referred to as Exterior data). All other bits are currently unused and must be cleared to 0. If this record is omitted, all bits are assumed to be 0. The following shows an ELFLAGS record. For additional information on Template data, consult the GDSII Reference Manual.

Predefined constants:

```
//ELFLAGS
const   __int16 ELFLAGS_0=0;
const   __int16 ELFLAGS_TEMPLATE=1;
const   __int16 ELFLAGS_EXTERIOR=2;
```

plex - [PLEX], a unique positive number which is common to all elements of the plex to which this element belongs. The head of the plex is flagged by setting the seventh bit; therefore, plex numbers should be small enough to occupy only the right-most 24 bits. Predefined constant:

```
const __int32 PLEX_HEAD=0x1000000
```

Example:

```
int main(int argc, char* argv[])
{
        char *src="ems.gds";

GDS_BeginLib(src, 1e-3, 1e-9);

    GDS_BeginStructure("str2");
        GDS_Node(x,y,4, 0,0,0,0);
    GDS_EndStructure();

GDS_EndLib();
return 0;
}
```

Error Handling. Examples.

Some languages provide built-in support for handling anomalous situations, known as "exceptions," which may occur during the execution of a program. With exception handling, the program can communicate unexpected events to a higher execution context that is better able to recover from such abnormal events. These exceptions are handled by code that is outside the normal flow of control.

The problems are raised when we use exception mechanism within dll and try catches error outside of the dll that is very important in parsing and calculating the math expression. Especially if an application which is using dll was developed with another language or within another environment.

GDSIIExporter uses exception mechanism to locate error and pass message of the error that occurs.

But for Delphi, VB, C#, VB.NET or even Borland C++ application to get an error message through exception mechanism from a VC++ dynamic link library is obviously a challenge.

There was developed mechanism to catch error message using a **Callback** function and raise native exception inside calling program.

For that purposes in the interface has been included function:

```
void __stdcall SetErrorHandle(TpfErrorHandler);
```

with argument

```
typedef void (__stdcall*TpfErrorHandler)(const TCHAR*errMsg);
```

pointer to the function with one argument `const TCHAR *`, a string with error message.

In case if programmer wants to provide his own exception mechanism, he must implement a function that throws exception with error message from *GDSIIExporter*, for example:

```
void __stdcall RiaseException (const TCHAR*err)
{
        throw Exception(err);
}
```

Calling `SetErrorHandle(RiaseException)`, programmer establishes mechanism to redirect all error messages from *GDSIIExporter* to `void RiaseException (const TCHAR*err)` function and as result, all errors can be easy caught in

```
Try{
...}
catch(Exception &e)
{...}
```

block inside of the calling function.

To make this clear, let's illustrate, how different languages and systems provide the initialization and the error handling using GDSII.dll.PRC version of *GDSIIExporter*.

Example 1: C/C++

For using of GDSIIExporter.dll, head file GDSIIExporter.h must be included in a project:

```
//GDSIIExporter.h
//Copyright © 2001-2005 Easy Math Solution. All Rights Reserved
//http://www.e-MathSolution.com
//For details contact us: info@e-MathSolution.com
#ifndef GDSIIEXPORTEREX_H
#define GDSIIEXPORTEREX_H

#include <TCHAR.H>
#include<windows.h>

#include <string>
#include "GDSIIConst.h"

//The type definitions
typedef void (__stdcall*TpfErrorHandler)(const TCHAR* err);

typedef void (__stdcall *TGDS_Path)(__int32 *pointsX, __int32 *pointsY,
            __int16 pointsNum,   __int32 width, __int16 layer, __int16 datatype,
            __int16 pathtype, __int16 elflags, __int32 plex);

typedef void (__stdcall *TGDS_Boundary)(__int32 *pointsX, __int32 *pointsY,
            __int16 pNum,  __int16 layer,__int16 datatype,  __int16 elflags,
            __int32 plex);

typedef void (__stdcall *TGDS_Circle)(__int32 centerX, __int32 centerY,
            __int32 radius, __int16 segmentNum, __int16 layer,
            __int16 datatype,__int16 elflags, __int32 plex);

typedef void (__stdcall *TGDS_Arc)(__int32 centerX, __int32 centerY,
            __int32 radius,  double startAng, double endAng,
            __int16 segmentNum, __int32 width, __int16 layer, __int16 datatype,
            __int16 pathtype, __int16 elflags, __int32 plex);

typedef void (__stdcall *TGDS_Sector)(__int32 centerX, __int32 centerY,
            __int32 radius,  double startAng, double endAng,
            __int16 segmentNum, __int16 layer, __int16 datatype, __int16 elflags,
            __int32 plex);

typedef void (__stdcall *TGDS_BoxWH)(__int32 x, __int32 y, __int32 width,
            __int32 height, __int16 layer, __int16 boxtype,  __int16 elflags,
            __int32 plex);

typedef void (__stdcall *TGDS_Box)(__int32 x1, __int32 y1, __int32 x2, __int32 y2,
            __int16 layer, __int16 boxtype, __int16 elflags, __int32 plex);

typedef void (__stdcall *TGDS_Square)(__int32 x1, __int32 y1, __int32 side,
            __int16 layer, __int16 boxtype,    __int16 elflags, __int32 plex);

typedef void (__stdcall *TGDS_Sref)(const TCHAR *strName, __int32 x, __int32 y,
            __int16 strans, double mag, double angle,
            __int16 elflags, __int32 plex);

typedef void (__stdcall *TGDS_Aref)( const TCHAR *strName, __int32 *x,
            __int32 *y, __int16 col, __int16 row, __int16 strans, double mag,
            double angle, __int16 elflags, __int32 plex);

typedef void (__stdcall *TGDS_BeginStructure)(TCHAR* strName);
typedef void (__stdcall *TGDS_EndStructure)(void);

typedef void (__stdcall *TGDS_BeginLib)(const TCHAR *fileName, double
            dbUserUnit, double dbUnitInMeter);
typedef void (__stdcall *TGDS_EndLib)(void);

typedef void (__stdcall *TGDS_Text)(__int32 pointsX, __int32 pointsY,
            const TCHAR *msg, __int16 layer, __int16 txtType,
            __int16 presentation, __int16 pathtype, __int32 width, __int16
            strans,double mag, double angle, __int16 elflags,  __int32 plex);

typedef void (__stdcall *TGDS_Node)(__int32 *pointsX, __int32 *pointsY,
            __int16 pointsNum, __int16 layer, __int16 nodeType, __int16 elflags,
            __int32 plex);

typedef void (__stdcall *TSetErrorHandle)(TpfErrorHandler);


//External variables
extern      TGDS_Path                   GDS_Path;
extern      TGDS_Boundary               GDS_Boundary;
extern      TGDS_Circle                 GDS_Circle;
extern      TGDS_Arc                    GDS_Arc;
extern      TGDS_Sector                 GDS_Sector;
extern      TGDS_BoxWH                  GDS_BoxWH;
extern      TGDS_Box                    GDS_Box;
extern      TGDS_Square                 GDS_Square;
extern      TGDS_Sref                   GDS_Sref;
extern      TGDS_Aref                   GDS_Aref;
extern      TGDS_BeginStructure         GDS_BeginStructure;
extern      TGDS_EndStructure           GDS_EndStructure;
extern      TGDS_BeginLib               GDS_BeginLib;
extern      TGDS_EndLib                 GDS_EndLib;
extern      TGDS_Text               GDS_Text;
extern      TGDS_Node               GDS_Node;
extern      TSetErrorHandle         SetErrorHandle;

//////////////////////////////////////////////////////////////////////
//Load & UnLoad GDSIIExporter.dll
bool LoadGDSIIExporter(void);
void UnLoadGDSIIExporter(void);

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//          Exception class
class GDSIIException
{
public:
          GDSIIException(const TCHAR *msg){ error.assign(msg); }
          const TCHAR * getError( void ) const { return error.c_str(); }
private:
          std::basic_string<TCHAR> error;
};
#endif
```

The consts are stored in file GDSIIConst.h

```
//GDSIIConst.h
//Copyright © 2001-2005 Easy Math Solution. All Rights Reserved
//http://www.e-MathSolution.com
//For details contact us: info@e-MathSolution.com
#ifndef GDSIICONST_H
#define GDSIICONST_H

//////////////////////////////////////////////////////////////////////
//Presentation bits
//Horizontal justification
const   __int16 PRESENT_HLEFT=0x0;
const   __int16 PRESENT_HCENTER=0x1;
const   __int16 PRESENT_HRIGHT=0x2;

//Vertical justification
const   __int16 PRESENT_VTOP=0x0;
const   __int16 PRESENT_VMIDDLE=0x4;
const   __int16 PRESENT_VBOTTOM=0x8;

//Font number
const   __int16 PRESENT_FONT0=0x0;
const   __int16 PRESENT_FONT1=0x10;
const   __int16 PRESENT_FONT2=0x20;
const   __int16 PRESENT_FONT3=0x30;


//////////////////////////////////////////////////////////////////////
// PATHTYPE
const   __int16 PATHTYPE0=0;
const   __int16 PATHTYPE1=1;
const   __int16 PATHTYPE2=2;
const   __int16 PATHTYPE4=4;


//////////////////////////////////////////////////////////////////////
//ELFLAGS
const   __int16 ELFLAGS_0=0;
const   __int16 ELFLAGS_TEMPLATE=1;
const   __int16 ELFLAGS_EXTERIOR=2;

//////////////////////////////////////////////////////////////////////
//STRANS
const   __int16 STRANS_0=0;
const   __int16 STRANS_ABSMAG=0x4;
const   __int16 STRANS_ABSANG=0x2;
const   __int16 STRANS_REFLECT=0x8000;

//////////////////////////////////////////////////////////////////////
//PLEX
const   __int32 PLEX_HEAD=0x1000000;
#endif
```

Also, source file GDSIIExporter.cpp, which comes with package too, should be included in project:

```
//GDSIIExporterEx.cpp
//Copyright © 2001-2005 Easy Math Solution. All Rights Reserved
//http://www.e-MathSolution.com
//For details contact us: info@e-MathSolution.com

#include "GDSIIExporterEx.h"

//Function variables
TGDS_Path                      GDS_Path;
TGDS_Boundary                  GDS_Boundary;
TGDS_Circle                    GDS_Circle;
TGDS_Arc                       GDS_Arc;
TGDS_Sector                    GDS_Sector;
TGDS_BoxWH                     GDS_BoxWH;
TGDS_Box                       GDS_Box;
TGDS_Square                    GDS_Square;
TGDS_Sref                      GDS_Sref;
TGDS_Aref                      GDS_Aref;
TGDS_BeginStructure            GDS_BeginStructure;
TGDS_EndStructure              GDS_EndStructure;
TGDS_BeginLib                  GDS_BeginLib;
TGDS_EndLib                    GDS_EndLib;
TGDS_Text                      GDS_Text;
TGDS_Node                      GDS_Node;
TSetErrorHandle                SetErrorHandle;

HINSTANCE hDLL=NULL;


//////////////////////////////////////////////////////////////////////
//CallBack function for errors handling
#if defined(__BORLANDC__)
void __stdcall ErrorHandler(const TCHAR*err)
{
     throw GDSIIException(err);
}
#endif


//////////////////////////////////////////////////////////////////////
//Load GDSIIExporter and to get address of the interface functions
bool LoadGDSIIExporter(void)
{
   if((hDLL = LoadLibrary(_T("GDSIIExporter.dll")))==NULL)
        return false;

GDS_Path=(TGDS_Path)GetProcAddress(hDLL,_T("GDS_Path"));
GDS_Boundary=(TGDS_Boundary)GetProcAddress(hDLL,_T("GDS_Boundary"));
GDS_Circle=(TGDS_Circle)GetProcAddress(hDLL,_T("GDS_Circle"));
GDS_Arc  =(TGDS_Arc)GetProcAddress(hDLL,_T("GDS_Arc"));
GDS_Sector=(TGDS_Sector)GetProcAddress(hDLL,_T("GDS_Sector"));
GDS_BoxW=(TGDS_BoxWH)GetProcAddress(hDLL,_T("GDS_BoxWH"));
GDS_Box =(TGDS_Box)GetProcAddress(hDLL,_T("GDS_Box"));
GDS_Square=(TGDS_Square)GetProcAddress(hDLL,_T("GDS_Square"));
GDS_Sref =(TGDS_Sref)GetProcAddress(hDLL,_T("GDS_Sref"));
GDS_Aref =(TGDS_Aref)GetProcAddress(hDLL,_T("GDS_Aref"));
GDS_BeginStructure=(TGDS_BeginStructure)GetProcAddress(hDLL,_T("GDS_B
eginStructure"));
GDS_EndStructure=(TGDS_EndStructure)GetProcAddress(hDLL,_T("GDS_EndS
tructure"));
GDS_BeginLib=(TGDS_BeginLib)GetProcAddress(hDLL,_T("GDS_BeginLib"));
GDS_EndLib=(TGDS_EndLib)GetProcAddress(hDLL,_T("GDS_EndLib"));
GDS_Text=(TGDS_Text)GetProcAddress(hDLL,_T("GDS_Text"));
GDS_Node=(TGDS_Node)GetProcAddress(hDLL,_T("GDS_Node"));
SetErrorHandle=(TSetErrorHandle)GetProcAddress(hDLL,_T("SetErrorHandle"));


    if( !GDS_Path ||  !GDS_Boundary ||   !GDS_Circle ||   !GDS_Arc ||
      !GDS_Sector || !GDS_BoxWH || !GDS_Box || !GDS_Square ||
      !GDS_Sref || !GDS_Aref ||!GDS_BeginStructure || !GDS_EndStructure ||
       !GDS_BeginLib || !GDS_EndLib ||!GDS_Node ||!GDS_Text ||
           !SetErrorHandle )
                return false;

#if defined(__BORLANDC__)
    SetErrorHandle(ErrorHandler);
#endif

    return true;
}

//////////////////////////////////////////////////////////////////
//
void UnLoadGDSIIExporter(void)
{
          if(hDLL){
                    SetErrorHandle(NULL);
    FreeLibrary(hDLL);
          }
}
```

Calling function looks like:

```
#include "gdsIIExporterEx.h"
#include <iostream>

int main(int argc, char* argv[])
{
          char *src="ems.gds";
          int x[]={110,210,327,200,600,700,880,1000};
          int y[]={0,100,250,150,300,440,900,800};

          int xar[]={0,1000,0};
          int yar[]={0,0,1000};

    if(!LoadGDSIIExporter())
        exit(1);

 try{
GDS_BeginLib(src, 1e-3, 1e-9);

     GDS_BeginStructure("str3");
           GDS_Sref("str1", 100, 100,0,1,30,ELFLAGS_TEMPLATE,0);
      GDS_EndStructure();

     GDS_BeginStructure("str2");
          GDS_Aref("ur11",  xar, yar,5,4,0,1,0,0,0);
     GDS_EndStructure();

     GDS_BeginStructure("ur11");
          GDS_Box(0,0,100,100, 0,10, 0, 0);
     GDS_EndStructure();

     GDS_BeginStructure("str1");
          GDS_Box(100, 100,  200, 200,0,10, 0, 0);
          GDS_BoxWH(150, 150,  500, 500,0,10, 0, 0);

          GDS_Path( x, y, 8 ,10, 1,63, PATHTYPE1, 0, PLEX_HEAD);
          GDS_Text( 100, 100, "Easy Math Solution", 1, 4,
                    PRESENT_HCENTER |PRESENT_VBOTTOM,
                    PATHTYPE1, 0, STRANS_0, 0.1, 0, 0, 0);

          GDS_Circle(1500, 1000, 1000, 4, 0,0,0,0);
          GDS_Circle(1000, 1500, 1000, 7, 0,0,0,0);
          GDS_Arc(100, 150, 1000, 45, 178, 5,40, 0, 0, PATHTYPE1,0,0);
          GDS_Sector(1000, 1500, 1050, 120, 290,199, 0,0,0,0);
          GDS_Square(500, 500, 400,0,0,0,0);
     GDS_EndStructure();

GDS_EndLib();

    UnLoadGDSIIExporter();

 }catch(GDSIIException &e)
    {
    std::cout<<e.getError();
    }

          return 0;
}
```

Example 2: Delphi

Within Delphi environment the component initialization is done by initialization section of the unit. Just include source file GDSIIExporter.pas in your project:

```
//GDSIIExporter.pas
//Copyright © 2001-2005 Easy Math Solution. All Rights Reserved
//http://www.e-MathSolution.com
//For details contact us: info@e-MathSolution.com

unit GDSIIExporter;

interface
uses  SysUtils;

type
    p__int32 =^Longint;
    __int16 =  Smallint;
    __int32 =  Longint;
    TCHAR   =  PChar;
    TpFuncError=^TFuncErr;
    TFuncErr=procedure(er:string);

//Interface of GDSIIExporter.dll
procedure GDS_Path(pointsX:p__int32; pointsY:p__int32; pointsNum:__int16;
                    width:__int32; layer:__int16; datatype:__int16;
                    pathtype:__int16; elflags:__int16; plex:__int32 );stdcall;
procedure GDS_Boundary(pointsX:p__int32; pointsY:p__int32; pNum:__int16;
                    layer:__int16; datatype:__int16; elflags:__int16;
                    plex:__int32 ); stdcall;
procedure GDS_Circle(centerX:__int32; centerY:__int32; radius:__int32;
                    segmentNum:__int16; layer:__int16; datatype:__int16;
                    elflags:__int16; plex:__int32 );stdcall;
procedure GDS_Arc(centerX:__int32; centerY:__int32; radius:__int32;
                    startAng:double; endAng:double; segmentNum:__int16;
                    width:__int32; layer:__int16; datatype:__int16;
                    pathtype:__int16; elflags:__int16; plex:__int32 );stdcall;
```

```
procedure GDS_Sector(centerX:__int32; centerY:__int32; radius:__int32;
                    startAng:double; endAng:double; segmentNum:__int16;
                    layer:__int16; datatype:__int16; elflags:__int16;
                    plex:__int32 );stdcall;
procedure GDS_BoxWH(x:__int32; y:__int32; width:__int32; height:__int32;
                    layer:__int16; boxtype:__int16;
                    elflags:__int16; plex:__int32 ); stdcall;
procedure GDS_Box(x1:__int32; y1:__int32; x2:__int32; y2:__int32;
                    layer:__int16; boxtype:__int16;  elflags:__int16;
                    plex:__int32 ); stdcall;
procedure GDS_Square(x1:__int32; y1:__int32; side:__int32;
                     layer:__int16; boxtype:__int16; elflags:__int16;
                    plex:__int32 );stdcall;
procedure GDS_Sref(strName:TCHAR; x:__int32; y:__int32;
                    strans:__int16; mag:double; angle:double;
                     elflags:__int16; plex:__int32 ); stdcall;
procedure GDS_Aref(strName:TCHAR; x:p__int32; y:p__int32;
                    col:__int16; row:__int16; strans:__int16; mag:double;
                    angle:double; elflags:__int16; plex:__int32 );stdcall;
procedure GDS_BeginStructure(strName:TCHAR); stdcall;
procedure GDS_EndStructure;stdcall;
procedure GDS_BeginLib(fileName:TCHAR; dbUserUnit:double;
                    dbUnitInMeter:double );stdcall;
procedure GDS_EndLib;stdcall;
procedure GDS_Text(pointsX:__int32; pointsY:__int32;  msg:TCHAR;
                      layer:__int16; txtType:__int16; presentation:__int16;
                    pathtype:__int16; width:__int32; strans:__int16;
                    mag:double; angle:double; elflags:__int16;
                    plex:__int32 );stdcall;
procedure GDS_Node(pointsX:p__int32; pointsY:p__int32; pointsNum:__int16;
                    layer:__int16; nodeType:__int16; elflags:__int16 ;
                    plex:__int32 );stdcall;
procedure SetErrorHandle(errF:TpFuncError);stdcall;


const//////////////////////////////////////////////////
//Presentation bits
//Horizontal justification
PRESENT_HLEFT: word =$0;
PRESENT_HCENTER: word =$1;
PRESENT_HRIGHT: word =$2;

//Vertical justification
PRESENT_VTOP: word =$0;
PRESENT_VMIDDLE: word =$4;
PRESENT_VBOTTOM: word =$8;

//Font number
PRESENT_FONT0: word =$0;
PRESENT_FONT1: word =$10;
PRESENT_FONT2: word =$20;
PRESENT_FONT3: word =$30;


////////////////////////////////////////////////////////////////////////
// PATHTYPE
PATHTYPE0: word =0;
PATHTYPE1: word =1;
PATHTYPE2: word =2;
PATHTYPE4: word =4;


////////////////////////////////////////////////////////////////////////
//ELFLAGS
ELFLAGS_0: word =0;
ELFLAGS_TEMPLATE: word =1;
ELFLAGS_EXTERIOR: word =2;
```

```
////////////////////////////////////////////////////////////////////////
//STRANS
STRANS_0: word =0;
STRANS_ABSMAG: word =$4;
STRANS_ABSANG: word =$2;
STRANS_REFLECT: Word    =$8000;

////////////////////////////////////////////////////////////////////////
//PLEX
PLEX_HEAD: __int32 =$1000000;


implementation

procedure GDS_Path; external 'GDSIIExporter.dll';
procedure GDS_Boundary; external 'GDSIIExporter.dll';
procedure GDS_Circle; external 'GDSIIExporter.dll';
procedure GDS_Arc; external 'GDSIIExporter.dll';
procedure GDS_Sector; external 'GDSIIExporter.dll';
procedure GDS_BoxWH; external 'GDSIIExporter.dll';
procedure GDS_Box; external 'GDSIIExporter.dll';
procedure GDS_Square; external 'GDSIIExporter.dll';
procedure GDS_Sref; external 'GDSIIExporter.dll';
procedure GDS_Aref; external 'GDSIIExporter.dll';
procedure GDS_BeginStructure; external 'GDSIIExporter.dll';
procedure GDS_EndStructure; external 'GDSIIExporter.dll';
procedure GDS_BeginLib; external 'GDSIIExporter.dll';
procedure GDS_EndLib; external 'GDSIIExporter.dll';
procedure GDS_Text; external 'GDSIIExporter.dll';
procedure GDS_Node; external 'GDSIIExporter.dll';
procedure SetErrorHandle; external 'GDSIIExporter.dll';


//Native function to raise exception
procedure RiaseException(err:PChar);stdcall;
begin
    raise   Exception.Create(err);
end;


initialization
    //Init the error handle
    SetErrorHandle( @RiaseException );
end.
```

Calling function looks like:

```
program Test;
{$APPTYPE CONSOLE}
uses
  SysUtils,
  GDSIIExporter in 'GDSIIExporter.pas';

const ptNum:integer=199;
const xar: array[0..2] of __int32=(0,1000,0);
const yar: array[0..2] of __int32=(0,0,1000);
var
  src:PChar;
  x:array [0..199] of __int32;
  y:array [0..199] of __int32;
  k:double;
  i:word;

begin
```

```
src:='ems.gds';

for i:=0  to 199 do begin
            k:=i/10.0;
            x[i]:=i*10;
            y[i]:=round(sin(k)*500);
end;

try
GDS_BeginLib(src, 1e-3, 1e-9);

     GDS_BeginStructure('str3');
            GDS_Sref('str1', 100, 100,0,1,30,ELFLAGS_TEMPLATE,0);
      GDS_EndStructure();

     GDS_BeginStructure('str2');
            GDS_Aref('ur11',  @xar, @yar,5,4,0,1,0,0,0);
      GDS_EndStructure();

      GDS_BeginStructure('ur11');
            GDS_Box(0,0,100,100, 0,10, 0, 0);
      GDS_EndStructure();


      GDS_BeginStructure('str1');
            GDS_Path( @x, @y, ptNum ,30, 1,63,
                        PATHTYPE1,ELFLAGS_TEMPLATE,0);
            GDS_Box(100, 100,  200, 200,0,10, 0, 0);
            GDS_BoxWH(150, 150,  500, 500,0,10, 0, 0);
          GDS_Text( 100, 100, 'Easy Math Solution', 1, 4,
              PRESENT_HCENTER or PRESENT_VBOTTOM, PATHTYPE1, 0,
              STRANS_0, 0.2, 0, 0, 0);
            GDS_Circle(1500, 1000, 1000, 4, 0,0,0,0);
            GDS_Circle(1000, 1500, 1000, 7, 0,0,0,0);
            GDS_Arc(100, 150, 1000, 45, 178, 5,40, 0, 0, 1,0,0);
            GDS_Sector(1000, 1500, 1050, 120, 290,199, 0,0,0,0);
            GDS_Square(500, 500, 400,0,0,0,0);
         GDS_EndStructure();
GDS_EndLib();

 except
      on E:Exception do writeln('Exception: '+e.Message);
     end;
end.
```

.NET

The interface for .NET has some little differences. The source code below shows GDSIIExporter class and the methods signature:

```
namespace EMS{

public __gc class GDSIIException: public
System::Exception
{
public:
GDSIIException(System::String *_err):
               System::Exception(_err) {}
};

public __gc class GDSIIExporter
{
public:
GDSIIExporter(void);
~GDSIIExporter(void);

void        BeginLib(const System::String *fileName, double dbUnit, double
            dbUnitInMeter);
 void       EndLib( void );
 void       BeginStructure(const System::String *strName ) ;
 void       EndStructure( void ) ;
//Elements
 void       Box( __int32 x1, __int32 y1, __int32 x2, __int32 y2, __int16 layer,
            __int16 boxtype, __int16 elflags, __int32 plex);
 void       BoxWH( __int32 x, __int32 y, __int32 w, __int32 h, __int16 layer,
            __int16 boxtype, __int16 elflags, __int32 plex);
 void Square(__int32 x, __int32 y, __int32 side, __int16 layer, __int16 boxtype,
            __int16 elflags, __int32 plex );

 void       Boundary(__int32 pointsX __gc[], __int32 pointsY __gc[],
            __int16 layer, __int16 datatype, __int16 elflags, __int32 plex );
 void       Circle(__int32 centerX, __int32 centerY,__int32 radius, __int16
            segmentNum ,__int16 layer, __int16 datatype, __int16 elflags,
            __int32 plex);
 void       Arc(__int32 centerX, __int32 centerY, __int32 radius,   double startAng,
            double endAng, __int16 segmentNum,__int32 width, __int16 layer,
            __int16 datatype, __int16 pathtype, __int16 elflags, __int32 plex);

 void       Sector(__int32 centerX, __int32 centerY,__int32 radius,
            double startAng, double endAng,__int16 segmentNum ,__int16 layer,
            __int16 datatype, __int16 elflags,  __int32 plex);

 void        Sref(const System::String *strName, __int32 x, __int32 y, __int16
            strans, double mag, double angle, __int16 elflags, __int32 plex);
void        Aref(const System::String *strName, __int32 pointsX __gc[], __int32
            pointsY __gc[], __int16 col, __int16 row, __int16 strans, double mag,
            double angle0, __int16 elflags, __int32 plex);

void        Path( __int32 pointsX __gc[], __int32 pointsY __gc[],     __int32 width ,
            __int16 layer , __int16 datatype , __int16 pathtype , __int16 elflags,
            __int32 plex) ;

void        Text(__int32 pointsX, __int32 pointsY, const System::String *msg,
            __int16 layer,  __int16 txtType, __int16 presentation, __int16 pathtype,
            __int32 width, __int16 strans, double mag, double angle,
            __int16 elflags, __int32 plex);
void        Node(__int32 pointsX __gc[], __int32 pointsY __gc[], __int16 layer,
            __int16 nodeType, __int16 elflags,  __int32 plex);

 ///////////////////////////////////////////////////////////////////////////////
//Presentation bits
//Horizontal justification
static const  __int16 PRESENT_HLEFT=0x0;
static const  __int16 PRESENT_HCENTER=0x1;
static const  __int16 PRESENT_HRIGHT=0x2;

//Vertical justification
static const  __int16 PRESENT_VTOP=0x0;
static const  __int16 PRESENT_VMIDDLE=0x4;
static const  __int16 PRESENT_VBOTTOM=0x8;

//Font number
static const  __int16 PRESENT_FONT0=0x0;
static const  __int16 PRESENT_FONT1=0x10;
static const  __int16 PRESENT_FONT2=0x20;
static const  __int16 PRESENT_FONT3=0x30;

/////////////////////////////////////////////////////////////////////////////
// PATHTYPE
static const  __int16 PATHTYPE0=0;
static const  __int16 PATHTYPE1=1;
static const  __int16 PATHTYPE2=2;
static const   __int16 PATHTYPE4=4;

///////////////////////////////////////////////////////////////////
//ELFLAGS
static const   __int16 ELFLAGS_0=0;
static const   __int16 ELFLAGS_TEMPLATE=1;
static const   __int16 ELFLAGS_EXTERIOR=2;

///////////////////////////////////////////////////////////////////
//STRANS
static const   __int16 STRANS_0=0;
static const   __int16 STRANS_ABSMAG=0x4;
static const   __int16 STRANS_ABSANG=0x2;
static const   __int16 STRANS_REFLECT=0x8000;

///////////////////////////////////////////////////////////////////
//PLEX
static const   __int32 PLEX_HEAD=0x1000000;
};  //class GDSIIExporter
}  //namespace EMS
```

Example VB.NET

The example of GDSIExporter component using in VB.NET (.NET version):

```
'Demo.vb :
'GDSIIExporter
'Copyright © 2001-2005 Easy Math Solution. All Rights Reserved
'http://www.e-MathSolution.com
'For details contact us: info@e-MathSolution.com

Module Module1

   Sub Main()
      Dim xar() As Integer = {0, 1100, 0}
      Dim yar() As Integer = {0, 0, 1000}
      Dim x() As Integer = {110, 210, 327, 200, 600, 700, 880, 1000}
      Dim y() As Integer = {0, 100, 250, 150, 300, 440, 900, 800}
      Dim xb() As Integer = {110, 210, 327, 550}
      Dim yb() As Integer = {0, 200, 250, 170}
      Dim gds As EMS.GDSIIExporter

      gds = New EMS.GDSIIExporter()
      Try
         gds.BeginLib("ems.gds", 0.001, 0.000000001)
         gds.BeginStructure("str2")
         gds.Aref("ur11", xar, yar, 5, 4, 0, 1, 0, 0, 0)
         gds.EndStructure()

         gds.BeginStructure("ur11")
         gds.Box(0, 0, 200, 200, 0, 0, 0, 0)
         gds.EndStructure()

         gds.BeginStructure("str1")
         gds.Box(105, 105, 205, 205, 0, 10, 0, 0)
         gds.BoxWH(250, 105, 200, 300, 0, 10, 0, 0)
         gds.Square(500, 105, 400, 0, 0, 0, 0)

           gds.Path(x, y, 10, 1, 63, EMS.GDSIIExporter.PATHTYPE1, 0,
           EMS.GDSIIExporter.PLEX_HEAD)

         gds.Boundary(xb, yb, 1, 61, 0, 0)
         gds.Text(100, 100, "Easy Math Solution", 1, 4, _
         EMS.GDSIIExporter.PRESENT_HCENTER Or
EMS.GDSIIExporter.PRESENT_VBOTTOM, _
           EMS.GDSIIExporter.PATHTYPE1, 0, EMS.GDSIIExporter.STRANS_0,
0.1, 0, 0, 0)

         gds.Text(100, 300, "GDSIIExporter", 1, 4, _
            EMS.GDSIIExporter.PRESENT_HCENTER Or
EMS.GDSIIExporter.PRESENT_VBOTTOM, _
            EMS.GDSIIExporter.PATHTYPE1, 0, EMS.GDSIIExporter.STRANS_0, _
             0.1, 0, 0, 0)
         gds.Text(100, 500, "Copyright EMS XXI", 1, 4, _
          EMS.GDSIIExporter.PRESENT_HLEFT Or
EMS.GDSIIExporter.PRESENT_VBOTTOM, _
           EMS.GDSIIExporter.PATHTYPE1, 0, EMS.GDSIIExporter.STRANS_0, _
           0.05, 45, 0, 0)

         gds.Circle(1500, 1000, 1000, 150, 0, 0, 0, 0)
         gds.Circle(1000, 1500, 1000, 7, 0, 0, 0, 0)

         gds.Arc(100, 150, 1000, 45, 178, 135, 40, 0, 0,
EMS.GDSIIExporter.PATHTYPE1, 0, 0)
         gds.Arc(400, 300, 600, 120, 270, 100, 40, 0, 0,
EMS.GDSIIExporter.PATHTYPE2, 0, 0)
         gds.Sector(1000, 1500, 1050, 30, 185, 100, 0, 0, 0, 0)
         gds.Sector(1000, 1000, 800, 120, 290, 199, 0, 0, 0, 0)
         gds.Square(500, 500, 400, 0, 0, 0, 0)
         gds.EndStructure()

         gds.EndLib()

      Catch e As EMS.GDSIIException
         Console.WriteLine("GDSIIException Handler: {0}", e.ToString())
      End Try

   End Sub

End Module
```

Example C#

The example of GDSIExporter component using in C# (.NET version):

```
// Demo.cs :
//GDSIIExporter version 1.0
//Copyright © 2001-2005 Easy Math Solution. All Rights Reserved
//http://www.e-MathSolution.com
//For details contact us: info@e-MathSolution.com

using System;
using EMS;

namespace Sharp
{
class Class1
{
[STAThread]
static void Main(string[] args)
{
          int []xar=new int[]{0,1100,0};
          int []yar=new int[]{0,0,1000};
          int []x=new int[]{110,210,327,200,600,700,880,1000};
          int []y=new int[]{0,100,250,150,300,440,900,800};
          int []xb=new int[]{110,210,327,550};
          int []yb=new int[]{0,200,250,170};

          GDSIIExporter gds = new GDSIIExporter();
try
{
gds.BeginLib("ems.gds",1e-3, 1e-9);
          gds.BeginStructure("str2");
```

```
                gds.Aref("ur11",  xar, yar,5,4,0,1,0,0,0);
        gds.EndStructure();

        gds.BeginStructure("ur11");
                gds.Box(0,0,200,200, 0,0, 0, 0);
        gds.EndStructure();

        gds.BeginStructure("str1");
                gds.Box(105, 105,  205, 205,0,10, 0, 0);
                gds.BoxWH(250, 105,  200, 300,0,10, 0, 0);
                gds.Square(500, 105, 400,0,0,0,0);

                gds.Path( x, y, 10, 1,63, GDSIIExporter.PATHTYPE1, 0,
                        GDSIIExporter.PLEX_HEAD);

                gds.Boundary( xb, yb, 1, 61, 0, 0);
                gds.Text( 100, 100, "Easy Math Solution", 1, 4,
                        GDSIIExporter.PRESENT_HCENTER
                        |GDSIIExporter.PRESENT_VBOTTOM,
                        GDSIIExporter.PATHTYPE1, 0,
                        GDSIIExporter.STRANS_0,      0.1, 0, 0, 0);
                gds.Text( 100, 300, "GDSIIExporter", 1, 4,
                        GDSIIExporter.PRESENT_HCENTER
                        |GDSIIExporter.PRESENT_VBOTTOM,
                        GDSIIExporter.PATHTYPE1, 0,
                        GDSIIExporter.STRANS_0,      0.1, 0, 0, 0);
                gds.Text( 100, 500, "Copyright EMS XXI", 1, 4,
                        GDSIIExporter.PRESENT_HLEFT
                        |GDSIIExporter.PRESENT_VBOTTOM,
                        GDSIIExporter.PATHTYPE1, 0,
                        GDSIIExporter.STRANS_0, 0.05, 45, 0, 0);

                gds.Circle(1500, 1000, 1000, 150, 0,0,0,0);
                gds.Circle(1000, 1500, 1000, 7, 0,0,0,0);

                gds.Arc(100, 150, 1000, 45, 178, 135,40, 0, 0,
                      GDSIIExporter.PATHTYPE1,0,0);
                gds.Arc(400, 300, 600, 120, 270, 100,40, 0, 0,
                        GDSIIExporter.PATHTYPE2,0,0);
                gds.Sector(1000, 1500, 1050, 30, 185,100, 0,0,0,0);
                gds.Sector(1000, 1000, 800, 120, 290,199, 0,0,0,0);
                gds.Square(500, 500, 400,0,0,0,0);
        gds.EndStructure();

gds.EndLib();
}
catch (GDSIIException e)
{
        Console.WriteLine("GDSIIException Handler: {0}", e.ToString());
}
}
}
}
```

Example VC.NET

The example of GDSIExporter component using in VC.NET (.NET version):

```
#include "stdafx.h"

#using <mscorlib.dll>
#using <GDSIIExporter.dll>
#include <tchar.h>

using namespace System;
using namespace EMS;
int _tmain(void)
{
String *src=S"ems.gds";
String *str2=S"str2";
String *ur11=S"ur11";
String *str1=S"str1";

int x __gc[]={110,210,327,200,600,700,880,1000};
int y __gc[]={0,100,250,150,300,440,900,800};
int xb __gc[]={110,210,327,550};
int yb __gc[]={0,200,250,170};

int xar __gc[]={0,1100,0};
int yar __gc[]={0,0,1000};
GDSIIExporter *gds=new GDSIIExporter;

try{
    gds->BeginLib(src, 1e-3, 1e-9);
        gds->BeginStructure(str2);
                gds->Aref(ur11,  xar, yar, 5, 4, 0,
                1, 0, 0, 0);
        gds->EndStructure();

        gds->BeginStructure(ur11);
                gds->Box(0,0,200,200, 0,0, 0, 0);
        gds->EndStructure();

        gds->BeginStructure(str1);
                gds->Box(105, 105,  205, 205,0,10,
                0, 0);
                gds->BoxWH(250, 105,  200, 300,0,10,
                0, 0);
                gds->Square(500, 105, 400,0,0,0,0);

                gds->Path( x, y, 10, 1,63,
                GDSIIExporter->PATHTYPE1, 0,
                GDSIIExporter->PLEX_HEAD);

                gds->Boundary( xb, yb, 1, 61, 0, 0);
                gds->Text( 100, 100, "Easy Math
                Solution", 1, 4,
                GDSIIExporter->PRESENT_HCENTER
                |GDSIIExporter->PRESENT_VBOTTOM,
                GDSIIExporter->PATHTYPE1, 0,
                GDSIIExporter->STRANS_0, 0.1, 0, 0,
                0);
                gds->Text( 100, 300, GDSIIExporter",
                1, 4,  GDSIIExporter->PRESENT_HCENTER
                |GDSIIExporter->PRESENT_VBOTTOM,
                GDSIIExporter->PATHTYPE1, 0,
                GDSIIExporter->STRANS_0, 0.1, 0, 0,
                0);
                gds->Circle(1500, 1000, 1000, 150,
                0,0,0,0);
                gds->Circle(1000, 1500, 1000, 7,
                0,0,0,0);

                gds->Arc(100, 150, 1000, 45, 178,
                135,40, 0, 0, GDSIIExporter-
                >PATHTYPE1,0,0);
                gds->Arc(400, 300, 600, 120, 270,
                100,40, 0, 0,
                GDSIIExporter->PATHTYPE2,0,0);
                gds->Sector(1000, 1500, 1050, 30,
                185,100, 0,0,0,0);
                gds->Sector(1000, 1000, 800, 120,
                290,199, 0,0,0,0);
                gds->Square(500, 500, 400,0,0,0,0);
        gds->EndStructure();
    gds->EndLib();
}
        catch (GDSIIException* e)
{
Console::WriteLine("GDSIIException Handler: {0}",
e->ToString());
}

     return 0;
}
```

Demo of GDSIIExporter

Demo for GDSIIExporter algorithm is distributed as GDSIIExporter.dll component, which can be used in your own projects.
The *GDSIIExporter.zip* includes a demo version of the algorithm as *GDSIIExporter.dll* component and some project examples in VC++, C++Builder, Delphi.
The demo version of *GDSIIExporter.dll* must be in active folder of the calling application to run properly. And, the functions: GDS_Path, GDS_Boundary, GDS_Arc, GDS_Circle, GDS_Sector are limited by 5 points. These restrictions are defined for the **demo version only**.
If some questions arise or you want to get examples of using of GDSIIExporter in VB, please, don't hesitate and ask us about: support@e-MathSolution.com