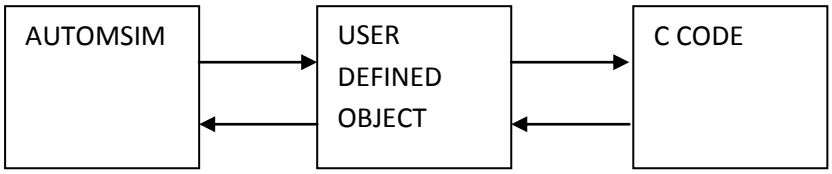
AUTOMSIM C user defined objects

Main concept

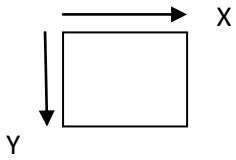
The purpose of AUTOMSIM C user defined objects is to give to the user a powerful tool for creating his own objects. This new version of user defined objects use an C compiler to dynamically generate the code associated to the object. This technology permits to create objects which have the same power of the standard AUTOMSIM objects: full functionalities, full execution speed.



The C code have to define some functions which will be called from AUTOMSIM through the user defined objects (by example, functions for drawing the object). C code may also call AUTOMSIM functions through the user defined objects. Standard C and Windows API functions may also be called from the C code.

Coordinates

When using coordinates in C code (eg. for drawing object) these coordinates are always relative to the top left corner of the object.



C skeleton

When adding a new C user defined object into an AUTOMSIM folder, a skeleton with some function prototypes is automatically created.

Drawing functions called by AUTOMSIM

ud_draw

This function is called from AUTOMSIM to render the object. Object should call drawing functions to render the object. Drawing functions that the object can call are Windows SDK API functions like (eg. MoveToEx, LineTo, etc) with an extra argument "modifier". This extra parameter is used to manage rotation and mirroring of the object.

```
void ud_draw(HDC hdc,HPEN staticpen,HPEN mobilpen,HPEN connectionpen,int modifier);
```

hdc: handle to device context (must be used when calling drawing functions)

staticpen: pen handle for drawing static parts of object

mobilepen: pen handle for drawing mobile parts of the objects (eg. rod and piston for a cylinder)

connectiopen: pen used for drawing mechanical connections of an objects (eg. sensor detection zone)

modifier: value to pass to drawing functions (see below "drawing functions called from the C code")

ud_visudraw

This function is called from AUTOMSIM after ud_draw when rendering an object when the dynamic display is activated. Typically, this function is used to draw stats (eg. pressure color in a part of a pneumatic object).

```
void ud_visudraw(HDC hdc,HPEN truepen,HPEN falsepen,HPEN unknowpen,int modifier);
```

hdc: handle to device context (must be used when calling drawing functions)

truepen: a pen to be used to draw a true stat

falsepen: a pen to be used to draw a false stat

unknowpen: a pen to be used to draw an unknown stat

Drawing functions called from the C code

Hers is the list of drawing functions available:

void MoveToEx_(HDC hdc,int modifier,int x,int y,POINT *p);

void LineTo_(HDC hdc,int modifier,int x,int y);

void Polygon_(HDC hdc,int modifier,int *point,int len);

void TextOut_(HDC hdc,int modifier,int x,int y,char *lp,unsigned len);

int DrawText_(HDC hdc,int modifier,LPCTSTR lp,int nCount,LPRECT lpRect,UINT format);

void Rectangle_(HDC hdc,int modifier,int x1,int y1,int x2,int y2);

void Ellipse_(HDC hdc,int modifier,int x1,int y1,int x2,int y2);

void Chord_(HDC hdc,int modifier,int x1,int y1,int x2,int y2,int x3,int y3,int x4,int y4);

void Arc_(HDC hdc,int modifier,int x1,int y1,int x2,int y2,int x3,int y3,int x4,int y4);

void RoundRect_(HDC hdc,int modifier,int x1,int y1,int x2,int y2,int dx,int dy);

void SetArcDirection_(HDC hdc,int modifier,int mode);

Please see the Windows SDK API reference for more information on these functions.

Another function is available to transform a rectangle defined by 2 points regarding the "modifier" parameter.

void rotaterect_(int modifier,int *x1,int *y1,int *x2,int *y2);

Other standard Windows API functions may be called from the C code, by example SetTextColor should be called to change text color.

Object size definition

ud_getsize

This function is called from AUTOMSIM to retrieve the size of the object in number of pixels. The values must be multiple of 10.

```
void ud_getsize(unsigned *width,unsigned *height)
```

Connections definition

ud_getconnections

```
void ud_getconnections(int *nconnections,int *connectionstechnology,int *connectionx,int *connectiony)
```

This function is called from AUTOMSIM to retrieve the connections list. For each connection, technology and position must be returned.

nconnections: must be set with the number of connections

connectionstechnology: for each connection, one of the following type: ASI_TECHNOLOGY_PNEU (for pneumatic), ASI_TECHNOLOGY_ELEC (for electric), ASI_TECHNOLOGY_HYDR (for hydraulic)

connectionx, connectiony: position of the connection (must be a multiple of 10).

Processing

ud_run

Called from AUTOMSIM to allow the object to process.

```
void ud_run(int iEllapsed,double *matrixin,double *matrixout,int *matrixwrite);
```

iEllapsed: time ellapsed since the last call in ms

matrixin: value of each connection (read access)

matrixout: value to write to each connection

matrixwrite: mode for writing, one of this value:

ASI_MATRIX_NOTHING: don't write

ASI_MATRIX_WRITE: write the value defined with matrixout

ASI_MATRIX_CONNECT: connect with another connection (the connection number is written in matrixout)

ASI_MATRIX_BLOCK: block the connection (only available for pneumatic and hydraulic: plug)

Mouse event

ud_getactionzones

```
void ud_getactionzones(int modifier,int *nactionzones,int *rect,int *id);
```

Called by AUTOMSIM to retrieve areas which will generate mouse events when the user clicks on them.

modifier: defines the rotation and mirroring function

nactionzones: must be filled with the number of areas

rect: must be filled with the rectangles defining each areas

id: must be filled with an id for each area

ud_mouseevent

```
void ud_mouseevent(int event,int zone);
```

Called by AUTOMSIM when an user click on the object.

event: one of the following values:

0 double click

1 left mouse button up

2 left mouse button down

3 right mouse button up

4 right mouse button sown

zone: id as defined with the ud_getactionzones function

Mechanical connections

ud_getmecconnections

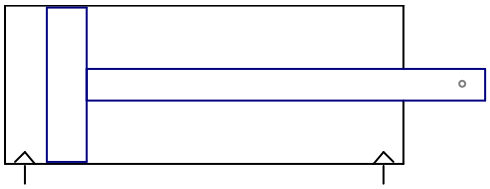
Called from AUTOMSIM to retrieve the mechanical connections.

```
void ud_getmecconnections(int *nconnections,RECT *rectconn);
```

nconnections: must be filled with the number of mechanical connections

rectconn: must be filled with rectangle of each connections

Example



```
#include "windows.h"
#include "asi.h"

// Object size
unsigned cx=200;
unsigned cy=80;

// rod pos between 0 and 100
unsigned rodpos=0;

// pistion thick
unsigned pthick=20;

// rod len
unsigned rlen=200;

// rod thick
unsigned rthick=16;

// pressure threshold
double tp=400;
```

```
// Draw object
void ud_draw(HDC hdc,HPEN staticpen,HPEN mobilpen,HPEN connectionpen,int modifier)
{
// Draw static part
SelectObject(hdc,staticpen);
MoveToEx_(hdc,modifier,cx,cy/2-rthick/2,NULL);
LineTo_(hdc,modifier,cx,0);
LineTo_(hdc,modifier,0,0);
LineTo_(hdc,modifier,0,cy);
LineTo_(hdc,modifier,cx,cy);
LineTo_(hdc,modifier,cx,cy/2+rthick/2);

// Draw left connection position
MoveToEx_(hdc,modifier,5,cy-1,NULL);
LineTo_(hdc,modifier,10,cy-6);
LineTo_(hdc,modifier,15,cy);

MoveToEx_(hdc,modifier,10,cy+1,NULL);
LineTo_(hdc,modifier,10,cy+10);

// Draw right connection position
MoveToEx_(hdc,modifier,cx-5,cy-1,NULL);
LineTo_(hdc,modifier,cx-10,cy-6);
LineTo_(hdc,modifier,cx-15,cy);

MoveToEx_(hdc,modifier,cx-10,cy+1,NULL);
LineTo_(hdc,modifier,cx-10,cy+10);
```

```
// Draw moving part
unsigned pos;
pos=(cx-pthick-2-40)*rodpos/100;
SelectObject(hdc,mobilpen);
// Piston
MoveToEx_(hdc,modifier,1+pos+20,1,NULL);
LineTo_(hdc,modifier,1+pos+20,cy-1);
LineTo_(hdc,modifier,1+pos+20+pthick,cy-1);
LineTo_(hdc,modifier,1+pos+20+pthick,1);
LineTo_(hdc,modifier,1+pos+20,1);
// Rod
MoveToEx_(hdc,modifier,1+pos+20+pthick,cy/2-rthick/2,NULL);
LineTo_(hdc,modifier,1+pos+20+pthick+rlen,cy/2-rthick/2);
LineTo_(hdc,modifier,1+pos+20+pthick+rlen,cy/2+rthick/2);
LineTo_(hdc,modifier,1+pos+20+pthick,cy/2+rthick/2);
LineTo_(hdc,modifier,1+pos+20+pthick,cy/2-rthick/2);

// Draw mechanical detection zone
pos+=20+pthick+rlen-10;
SelectObject(hdc,connectionpen);
Ellipse_(hdc,modifier,pos-2,cy/2-2,pos+2,cy/2+2);
}


// Draw object (visu mode)
void ud_visudraw(HDC hdc,HPEN truepen,HPEN falsepen,HPEN unknowpen,int modifier)
```

```
{

}

// Draw get connections

void ud_getconnections(int *nconnections,int *connectionstechnology,int *connectionx,int *connectiony)

{

*nconnections=2;

connectionstechnology[0]=ASI_TECHNOLOGY_PNEU;

connectionx[0]=10;

connectiony[0]=cy+10;

connectionstechnology[1]=ASI_TECHNOLOGY_PNEU;

connectionx[1]=cx-10;

connectiony[1]=cy+10;


}

// Process

void ud_run(int iEllapsed,double *matrixin,double *matrixout,int *matrixwrite)

{

if(matrixin[0]-matrixin[1]>tp)

 { // Exit

 if(rodpos+iEllapsed<100)

   {

   rodpos+=iEllapsed;

   // decrease input pressure

   matrixout[0]=matrixin[0]-100;

   matrixwrite[0]=ASI_MATRIX_WRITE ;

   // increase output pressure

   matrixout[1]=matrixin[1]+100;
```

```
   matrixwrite[1]=ASI_MATRIX_WRITE ;
   }
  else rodpos=100;
  }
if(matrixin[1]-matrixin[0]>tp)
  { // Return
  if((int)rodpos>iEllapsed)
   {
   rodpos-=iEllapsed;
   // Decrease input pressure
   matrixout[1]=matrixin[1]-100;
   matrixwrite[1]=ASI_MATRIX_WRITE ;
   // increase output pressure
   matrixout[0]=matrixin[0]+100;
   matrixwrite[0]=ASI_MATRIX_WRITE ;
   }
  else rodpos=0;
 }
}

// Get object size
void ud_getsize(unsigned *width,unsigned *height)
{
*width=cx;
*height=cy;
}
// Define mouse events
```

```
void ud_getactionzones(int modifier,int *nactionzones,int *rect,int *id)
{
}
// Receive mouse event
void ud_mouseevent(int event,int zone)
{
}

// Return active mechanical detection zone
void ud_getmecconnections(int *nconnections,RECT *rectconn)
{
unsigned pos;
pos=(cx-pthick-2-40)*rodpos/100;
*nconnections=1;
pos+=20+pthick+rlen-10;
rectconn[0].left=pos-2;
rectconn[0].top=cy/2-2;
rectconn[0].right=pos+2;
rectconn[0].bottom=cy/2+2;
}
```