

uclaShape Documentation

Detailed Description

In general, uclaShape API is an extension of oaShape in OpenAccess. It is implemented by boost library 1.45.0. and OpenAccess. It allows users to do basic operations between layers that OpenAccess API does not provide. Also the API is divided into two modes, one is OVERWRITE mode, meaning that all the original shapes in the given layer will be overwritten, and the other is APPEND mode, meaning that all the original shapes will remain the same, but new shapes will be appended to the given layer. However if the given layer doesn't exist, both modes will create new layer to store the results.

Functions

(all function calls can be found inside the *namespace:uclaShape* in uclaShape.h)

FLAG=0 -> APPEND MODE (default)

FLAG=1 -> OVERWRITE MODE

void clear(oaLPPHeader* lppHeader);	Taking the oaLPPHeader object from OpenAccess and making the object empty of geometry
void clear(oaBlock* block, oaLayerNum layerNum, oaPurposeNum purposeNum);	Overloading function performs the same functionality as above, but taking oaLayerNum and oaPurposeNum from OpenAccess as inputs instead. (note: if the layerNum and purposeNum are not valid, nothing will be done!)
oaBoolean logic_operation(oaBlock* block, oaLPPHeader *lppHeader1, oaLPPHeader *lppHeader2, oaLPPHeader *lppHeader3, oaLayerOp op, int flag=0);	Taking the block object,oaLPPHeader object 1 and 2 from OpenAccess.Return True and store the results into the given oaLPPHeader object 3. (note: oaLayerOp op can be AND, NOT, OR and XOR)

<pre> oaBoolean logic_operation(oaBlock* block, oaLPPHeader *lppHeader1, oaLPPHeader *lppHeader2, oaLayerNum layerNum, oaPurposeNum purposeNum, oaLayerOp op, int flag=0); </pre>	<p>Overloading function performs the same functionality as above, but taking oaLayerNum and oaPurposeNum from OpenAccess as inputs instead.</p>
<pre> oaBoolean equivalence(oaLPPHeader *lppHeader1, oaLPPHeader *lppHeader2); </pre>	<p>Taking two oaLPPHeader objects from OpenAccess. Return TRUE if they are identical, Otherwise, return FALSE.</p>
<pre> oaBoolean equivalence(oaBlock* block1, oaLayerNum layerNum1, oaPurposeNum purposeNum1, oaBlock* block2, oaLayerNum layerNum2, oaPurposeNum purposeNum2); </pre>	<p>Overloading function performs the same functionality as above, but taking oaLayerNum and oaPurposeNum from OpenAccess as inputs instead.</p>
<pre> oaBoolean empty(oaLPPHeader* lppHeader); </pre>	<p>Taking the oaLPPHeader object from OpenAccess. Return TRUE if the object is empty of geometry. Otherwise, return FALSE.</p>
<pre> oaBoolean empty(oaBlock* block, oaLayerNum layerNum, oaPurposeNum purposeNum); </pre>	<p>Overloading function performs the same functionality as above, but taking oaLayerNum and oaPurposeNum from OpenAccess as inputs instead.</p>
<pre> oaDouble area(oaLPPHeader *lppHeader); </pre>	<p>Taking the oaLPPHeader object from OpenAccess. Return the sum of individual areas inside the object.</p> <p>(note: if the oaLPPHeader object is not valid, it will return -1 to indicate error)</p>
<pre> oaDouble area(oaBlock* block, oaLayerNum layerNum, oaPurposeNum purposeNum); </pre>	<p>Overloading function performs the same functionality as above, but taking oaLayerNum and oaPurposeNum from OpenAccess as inputs instead.</p>

<pre> oaBoolean bloat(oaBlock* block, oaLPPHeader* lppHeader, int num, int flag=0); </pre>	<p>Taking the oaLPPHeader object from OpenAccess. Return TRUE if the object is expanded by the input number. Otherwise, return FALSE.</p>
<pre> oaBoolean bloat(oaBlock* block, oaLPPHeader* lppHeader, oaLayerNum layerNum, oaPurposeNum purposeNum, int num, int flag=0); </pre>	<p>Overloading function performs the same functionality as above, but taking oaLayerNum and oaPurposeNum from OpenAccess as inputs instead.</p>
<pre> oaBoolean shrink(oaBlock* block, oaLPPHeader* lppHeader, int num, int flag=0); </pre>	<p>Taking the oaLPPHeader object from OpenAccess. Return TRUE if the object is shrunk by the input number. Otherwise, return FALSE.</p>
<pre> oaBoolean shrink(oaBlock* block, oaLPPHeader* lppHeader, oaLayerNum layerNum, oaPurposeNum purposeNum, int num, int flag=0); </pre>	<p>Overloading function performs the same functionality as above, but taking oaLayerNum and oaPurposeNum from OpenAccess as inputs instead.</p>
<pre> oaBoolean scale_up(oaBlock* block, oaLPPHeader* lppHeader, int num, int flag=0); </pre>	<p>Taking the oaLPPHeader object from OpenAccess. Return TRUE if the object is scaled up by the input number. Otherwise, return FALSE.</p>
<pre> oaBoolean scale_up(oaBlock* block, oaLPPHeader* lppHeader, oaLayerNum layerNum, oaPurposeNum purposeNum, int num, int flag=0); </pre>	<p>Overloading function performs the same functionality as above, but taking oaLayerNum and oaPurposeNum from OpenAccess as inputs instead.</p>
<pre> oaBoolean scale_down(oaBlock* block, oaLPPHeader* lppHeader, int num, int flag=0); </pre>	<p>Taking the oaLPPHeader object from OpenAccess. Return TRUE if the object is scaled down by the input number. Otherwise, return FALSE.</p>

<pre> oaBoolean scale_down(oaBlock* block, oaLPPHeader* lppHeader, oaLayerNum layerNum, oaPurposeNum purposeNum, int num, int flag=0); </pre>	<p>Overloading function performs the same functionality as above, but taking oaLayerNum and oaPurposeNum from OpenAccess as inputs instead.</p>
<pre> oaBoolean keep(oaBlock* block, oaLPPHeader* lppHeader, oaUInt4 min_area, oaUInt4 max_area, oaUInt4 min_width, oaUInt4 max_width, oaUInt4 min_height, oaUInt4 max_height); </pre>	<p>Taking the oaLPPHeader object from OpenAccess. Return TRUE if it keeps the area that satisfied the arguments. Otherwise, return FALSE.</p>

How to Use in C++

First of all, we have to stream Layout to OpenAccess by the command strm2oa

Ex:

```
>>strm2oa -lib Layout0 -gds layout0.gds -cell TOPCELL -view layout -overwrite
```

and the command stream OpenAccess back to Layout is oa2strm

Ex:

```
>>oa2strm -lib Layout0 -gds layout0_output1.gds -cell TOPCELL -view layout
```

Then, we can include the header file logic_function.h. Here is a simple sample code:

```

#include "logic_operation.h"
...
int main(int argc, char* argv[]){
    ....
    view = oaDesign::open(libName, cellName, viewName, 'a');
    oaTech    *techFile = view->getTech();
    // *view is the design.....
    // Iterate over all the layer purpose pairs used by 'view'

```

```
oalter<oaLPPHeader> headers(view->getTopBlock()->getLPPHeaders());
```

```
oaLPPHeader *lppHeader1 = headers.getNext();  
oaLPPHeader *lppHeader2 = headers.getNext();
```

```
// USING THE NAMESPACE :uclaShape  
// Flag =1 represents we are using the overwrite option  
// By calling the function below, it will overwrite lppHeader1 and return //the new result  
layer
```

```
logic_op::logic_operation(view-  
>getTopBlock(),lppHeader1,lppHeader2,lppHeader1,oacNotLayerOp,1);
```

```
....
```

```
}
```

For more information, you may want to look at the sample code in the same directory called listLPP.cpp. In addition, there is a sample makeFile as well.

Lastly, if you find any bugs or have any comments, please feel free to email me!