**[Open Declaration](eclipse-open:%E2%98%82=Opcat/src%3CexportedAPI%7BRelativeConnectionPoint.java%E2%98%83RelativeConnectionPoint)exportedAPI.RelativeConnectionPoint**

A point representing a relative location on some graphical object. A point composed from two slots - side and param. Size is one of constants defined in OpcatConstants interface : N\_BORDER, S\_BORDER, W\_BORDER or E\_BORDER. Param is float number between 0 and 1 that defines a relative location of this point In order to get absolute point of (in coordinates of above graphical object) you should use getAbsoluteConnectionPoint method of IConnectionEdgeInstance or IXConnectionEdgeInstance interfaces.

From: gui/opdGraphics/opdBaseComponents/OpdBaseComponent

\*\*

\* <p>

\* <code>OpdBaseComponent</code> is a superclass of all OPD components except

\* links and non funadamental relations. It implements all common graphic

\* activities such as resizing, dragging, mouse event handlers and defines

\* interfaces for common properties.

\* <p>

\* For writing your own custom OPD graphic element you have only to inherit this

\* OpdBaseComponent and implement following methods that describe your component

\* properties<br>

\* public abstract void paint(Graphics g); - paints the component<br>

\* public abstract void callPropertiesDialog(); - calls the Dialog Window which

\* allows to edit component's properties. This method is invoked when user

\* performs double click on the component.<br>

\* public abstract boolean inResize(int pX, int pY); - Test if the point in

\* resizing area i.e. if user drags mouse in this area it resizes the component.

\* <br>

\* public abstract boolean inMove(int pX, int pY); - Test if the point in

\* moviing area i.e. if user drags mouse in this area it moves the component.<br>

\* public abstract int whichBorder(int pX, int pY); - Returns one of constants

\* which determines on which border point (pX, pY) lays.<br>

\*/

**public** **abstract** **class** OpdBaseComponent **extends** BaseGraphicComponent **implements**

IXNode {

OpdProject –

\* This class represents user's project.

\* <p>

\* This class manages all needed for the project: graphics, internal data

\* Unfolding and Zooming in, repository manager ,printing.

\* </p>

Default formatting settings

**private** **void** initConfig() {

**this**.config.setProperty("ThingWidth", **new** Integer(120));

**this**.config.setProperty("ThingHeight", **new** Integer(70));

**this**.config.setProperty("MinimalThingWidth", **new** Integer(60));

**this**.config.setProperty("MinimalThingHeight", **new** Integer(35));

**this**.config.setProperty("FundamentalRelationWidth", **new** Integer(36));

**this**.config.setProperty("FundamentalRelationHeight", **new** Integer(31));

**this**.config.setProperty("StateWidth", **new** Integer(60));

**this**.config.setProperty("StateHeight", **new** Integer(35));

**this**.config.setProperty("OPDWidth", **new** Integer(1300));

**this**.config.setProperty("OPDHeight", **new** Integer(860));

**this**.config.setProperty("DraggerSize", **new** Integer(36));

**this**.config.setProperty("ThingFont",

**new** Font("OurFont", Font.PLAIN, 16));

**this**.config.setProperty("URLFont", **new** Font("OurFont", Font.BOLD

+ Font.ITALIC, 16));

**this**.config.setProperty("StateFont",

**new** Font("OurFont", Font.PLAIN, 12));

**this**.config

.setProperty("LineFont", **new** Font("OurFont", Font.PLAIN, 11));

**this**.config.setProperty("LabelFont",

**new** Font("OurFont", Font.PLAIN, 11));

**this**.config

.setProperty("LinkFont", **new** Font("OurFont", Font.PLAIN, 12));

**this**.config.setProperty("SmallFont",

**new** Font("OurFont", Font.PLAIN, 11));

**this**.config.setProperty("BackgroundColor", **new** Color(230, 230, 230));

// kind of gray

**this**.config.setProperty("TextColor", Color.black);

**this**.config.setProperty("UrlColor", Color.RED);

**this**.config.setProperty("ObjectColor", **new** Color(0, 110, 0));

// kind of green

**this**.config.setProperty("ProcessColor", **new** Color(0, 0, 170));

// kind of blue

**this**.config.setProperty("StateColor", **new** Color(91, 91, 0));

// kind of brown

**this**.config.setProperty("NormalSize", **new** Integer(5));

**this**.config.setProperty("CurrentSize", **new** Integer(4));

}

gui/projectStructure/GraphicalRelationRepresentation

\* This class describes some "triangle" (graphical component representing

\* several fundamental relations). We need this class because to one "triangle"

\* user connects fundamental relation that don't related logically. So this

\* class holds data structure that contains instances of all fundamental

\* relations connected via this "triangle". This class also holds the

\* informatiom about the "tringle" itself. It compound from following components -

\* <br>

\* relation - OpdFundamentalRelation that represents graphically "triangle"

\* itself.<br>

\* source - OpdConnectionEdge which is source for all fundamental relation

\* connected via this "triangle".<br>

\* sourceDragger - AroundDragger which is connected to the source.<br>

\* line - OpdLine which connects sourceDragger to the "triangle".

Opcat Constants – ID numbers for various attributes of things within Opcat

src/exportedAPI/OpcatConstants

gives “linkType” values in xml representation

**package** exportedAPI;

/\*\*

\* OpcatConstants contains different constants used in Opcat2 system

\*/

**public** **class** OpcatConstants {

/\*\*

\* Represents public scope

\*/

**public** **final** **static** String *PUBLIC* = "0";

/\*\*

\* Represents protected scope

\*/

**public** **final** **static** String *PROTECTED* = "1";

/\*\*

\* Represents private scope

\*/

**public** **final** **static** String *PRIVATE* = "2";

/\*\*

\* OPM object type

\*/

**public** **static** **final** **int** *OBJECT* = 100;

/\*\*

\* OPM process type

\*/

**public** **static** **final** **int** *PROCESS* = 101;

/\*\*

\* OPM state type

\*/

**public** **static** **final** **int** *STATE* = 102;

/\*\*

\* OPM note type

\*/

**public** **static** **final** **int** *NOTE* = 103;

/\*\*

\* OPM Specialization fundamental relation type

\*/

**public** **static** **final** **int** *SPECIALIZATION\_RELATION* = 201;

/\*\*

\* OPM Exfibition fundamental relation type

\*/

**public** **static** **final** **int** *EXHIBITION\_RELATION* = 202;

/\*\*

\* OPM Instantiation fundamental relation type

\*/

**public** **static** **final** **int** *INSTANTINATION\_RELATION* = 203;

/\*\*

\* OPM Aggregation fundamental relation type

\*/

**public** **static** **final** **int** *AGGREGATION\_RELATION* = 204;

/\*\*

\* OPM Uni Direction general relation type

\*/

**public** **static** **final** **int** *UNI\_DIRECTIONAL\_RELATION* = 205;

/\*\*

\* OPM Bi Direction general relation type

\*/

**public** **static** **final** **int** *BI\_DIRECTIONAL\_RELATION* = 206;

/\*\*

\* OPM Agent link type

\*/

**public** **final** **static** **int** *AGENT\_LINK* = 305;

/\*\*

\* OPM Instrument link type

\*/

**public** **final** **static** **int** *INSTRUMENT\_LINK* = 303;

/\*\*

\* OPM Result link type

\*/

**public** **final** **static** **int** *RESULT\_LINK* = 306;

/\*\*

\* OPM Consumption link type

\*/

**public** **final** **static** **int** *CONSUMPTION\_LINK* = 301;

/\*\*

\* OPM Effect link type

\*/

**public** **final** **static** **int** *EFFECT\_LINK* = 302;

/\*\*

\* OPM Condition link type

\*/

**public** **final** **static** **int** *CONDITION\_LINK* = 304;

/\*\*

\* OPM Invocation link type

\*/

**public** **final** **static** **int** *INVOCATION\_LINK* = 307;

/\*\*

\* OPM Consumption Event link type

\*/

**public** **final** **static** **int** *CONSUMPTION\_EVENT\_LINK* = 310;

/\*\*

\* OPM Instrument Event link type

\*/

**public** **final** **static** **int** *INSTRUMENT\_EVENT\_LINK* = 308;

/\*\*

\* OPM Exception link type

\*/

**public** **final** **static** **int** *EXCEPTION\_LINK* = 309;

/\*\*

\* Checks if given type is a structural relation

\* **@param** type one of the types above

\* **@return** true if given type is structural relation, false otherwise

\*/

**public** **static** **boolean** isRelation(**int** type) {

**return**

((*SPECIALIZATION\_RELATION* == type) ||

(*EXHIBITION\_RELATION* == type) ||

(*INSTANTINATION\_RELATION* == type) ||

(*AGGREGATION\_RELATION* == type) ||

(*UNI\_DIRECTIONAL\_RELATION* == type) ||

(*BI\_DIRECTIONAL\_RELATION* == type));

}

/\*\*

\* Checks if given type is a procedural event link

\* **@param** type one of the types above

\* **@return** true if given type is procedural link, false otherwise

\*/

**public** **static** **boolean** isEventLink(**int** type) {

**return**

((*INVOCATION\_LINK* == type) ||

(*INSTRUMENT\_EVENT\_LINK* == type) ||

(*EXCEPTION\_LINK* == type) ||

(*CONSUMPTION\_EVENT\_LINK* == type));

}

/\*\*

\* North component border

\*/

**public** **static** **final** **int** *N\_BORDER* = 1;

/\*\*

\* North-east component corner

\*/

**public** **static** **final** **int** *NE\_BORDER* = 2;

/\*\*

\* North-west component corner

\*/

**public** **static** **final** **int** *NW\_BORDER* = 3;

/\*\*

\* South component border

\*/

**public** **static** **final** **int** *S\_BORDER* = 4;

/\*\*

\* South-east component corner

\*/

**public** **static** **final** **int** *SE\_BORDER* = 5;

/\*\*

\* South-west component corner

\*/

**public** **static** **final** **int** *SW\_BORDER* = 6;

/\*\*

\* West component border

\*/

**public** **static** **final** **int** *W\_BORDER* = 7;

/\*\*

\* East component border

\*/

**public** **static** **final** **int** *E\_BORDER* = 8;

/\*\*

\* Some point that is not in the border

\*/

**public** **static** **final** **int** *NOT\_IN\_BORDER* = 0;

gui.opdGraphics.GraphicUtils.calculateConnectionPoint

**public** **static** RelativeConnectionPoint calculateConnectionPoint(

Connectable mainItem, Connectable secondItem) {

Point sItemCenter;

Point tmpPoint = **new** Point(secondItem.getX(), secondItem.getY());

tmpPoint = SwingUtilities.*convertPoint*(secondItem.getParent(),

tmpPoint, mainItem.getParent());

sItemCenter = **new** Point(

(**int**) ((tmpPoint.getX() + (**double**) secondItem.getActualWidth() / 2) - (mainItem

.getX() + (**double**) mainItem.getActualWidth() / 2)),

(**int**) ((mainItem.getY() + (**double**) mainItem.getActualHeight() / 2) - (tmpPoint

.getY() + (**double**) secondItem.getActualHeight() / 2)));

**double** k;

**if** (sItemCenter.getX() == 0) {

k = sItemCenter.getY() / 0.0001;

} **else** {

k = sItemCenter.getY() / sItemCenter.getX();

}

**if** (k == 0.0) {

k = 0.0001;

}

**double** a = mainItem.getActualWidth() / 2.0;

**double** b = mainItem.getActualHeight() / 2.0;

**if** (mainItem **instanceof** OpdProcess) {

**double** x = Math.*sqrt*((a \* a \* b \* b) / (b \* b + (a \* a \* k \* k)));

**if** ((sItemCenter.getY() >= 0) && (sItemCenter.getX() >= 0)) {

**return** **new** RelativeConnectionPoint(OpcatConstants.*N\_BORDER*,

(x + a) / mainItem.getActualWidth());

}

**if** ((sItemCenter.getY() >= 0) && (sItemCenter.getX() <= 0)) {

**return** **new** RelativeConnectionPoint(OpcatConstants.*N\_BORDER*,

(a - x) / mainItem.getActualWidth());

}

**if** ((sItemCenter.getY() <= 0) && (sItemCenter.getX() >= 0)) {

**return** **new** RelativeConnectionPoint(OpcatConstants.*S\_BORDER*,

(x + a) / mainItem.getActualWidth());

}

**if** ((sItemCenter.getY() <= 0) && (sItemCenter.getX() <= 0)) {

**return** **new** RelativeConnectionPoint(OpcatConstants.*S\_BORDER*,

(a - x) / mainItem.getActualWidth());

}

}

**double** x1 = Math.*abs*(b / k);

**double** y1 = Math.*abs*(a \* k);

**if** ((sItemCenter.getY() >= 0) && (sItemCenter.getX() >= 0)) {

**if** (x1 <= a) {

**return** **new** RelativeConnectionPoint(OpcatConstants.*N\_BORDER*,

(x1 + a) / (2 \* a));

} **else** {

**return** **new** RelativeConnectionPoint(OpcatConstants.*E\_BORDER*,

(b - y1) / (2 \* b));

}

}

**if** ((sItemCenter.getY() >= 0) && (sItemCenter.getX() <= 0)) {

**if** (x1 <= a) {

**return** **new** RelativeConnectionPoint(OpcatConstants.*N\_BORDER*,

(a - x1) / (2 \* a));

} **else** {

**return** **new** RelativeConnectionPoint(OpcatConstants.*W\_BORDER*,

(b - y1) / (2 \* b));

}

}

**if** ((sItemCenter.getY() <= 0) && (sItemCenter.getX() >= 0)) {

**if** (x1 <= a) {

**return** **new** RelativeConnectionPoint(OpcatConstants.*S\_BORDER*,

(x1 + a) / (2 \* a));

} **else** {

**return** **new** RelativeConnectionPoint(OpcatConstants.*E\_BORDER*,

(b + y1) / (2 \* b));

}

}

**if** ((sItemCenter.getY() <= 0) && (sItemCenter.getX() <= 0)) {

**if** (x1 <= a) {

**return** **new** RelativeConnectionPoint(OpcatConstants.*S\_BORDER*,

(a - x1) / (2 \* a));

} **else** {

**return** **new** RelativeConnectionPoint(OpcatConstants.*W\_BORDER*,

(b + y1) / (2 \* b));

}

}

**return** **new** RelativeConnectionPoint(OpcatConstants.*S\_BORDER*, 0.5);

}