TMS FNC Blox
DEVELOPERS GUIDE

December 2019

Web: http://www.tmssoftware.com
Email: info@tmssoftware.com

Index

Availability

Supported frameworks and platforms
- VCL Win32/Win64
- FMX Win32/Win64, MacOS-X, iOS, Android
- LCL Win32/Win64, Mac OS-X, iOS, Android, numerous Linux variants including Raspbian
- WEB: Chrome, Edge, Firefox, …

Supported IDE's
- Delphi XE7 and C++ Builder XE7 or newer releases
- Lazarus 1.4.4 with FPC 2.6.4 or newer official releases.

Important Notice: TMS FNC Blox requires TMS FNC Core (separately available at the My Products page)

Overview

TMS FNC Blox is a set of cross-platform, cross-framework components for Delphi and C++ Builder to easily add feature-rich and user friendly diagramming, flowcharting & graphing capabilities to your applications. TMS FNC Blox provides TTMSFNCBloxControl component, a panel-like control where user can build diagrams by inserting blocks, lines and link them together. The TTMSFNCBloxSelector component is also provided to allow insert blox by dragging and dropping them on the TTMSFNCBloxControl canvas. The blocks can be customized by changing dozens of available properties. User can change shapes of blocks, shadow, bitmaps, among other features. Blocks can be rotated and resized. TMS FNC Blox provides an open architecture to allow users to building their own blocks by inheriting from TTMSFNCBloxBlock class and registering them by using OnRegisterElements event.

Among the many features that make TMS FNC Blox the ultimate cross-platform and cross-framework diagramming and flowcharting component set are:

-High-quality (anti-aliasing) drawing of blocks and lines
-Open architecture for building custom blocks and lines inherited from base classes
-Ready-to-use flowchart, arrow and electric blocks
-Linking system allow customizable link points and information retrieval of connected blocks
-Block gradient, shadow and bitmap
-Full block customization: pen, brush, color, selection color, minimum width and height
-Block text customization: horizontal and vertical alignment, font, word wrap, clipping
-Customizable link points in blocks
-Full line (link) customization: pen, source arrow shape, target arrow shape
-Arc & bezier lines, polygon objects
-Block rotation supported (including text, bitmap and gradient)
-Separate TTMSFNCBloxToolBar component for easy editing with no line of code (needs TMS FNC UI Pack)
-Snap grid
-Rulers
-Saving / loading to/from file and stream
-Zoom in / out
-Panning
-Helper classes TTMSFNCBloxBlockDrawer for easy custom drawing on custom blocks
-Clipboard operations, object deletion and inserting, zooming, and more.

Getting Started

Quick start - using blox control and toolbar components

The main component in TMS FNC Blox is the TTMSFNCBloxControl component. The TTMSFNCBloxControl component is a visual control which holds and displays the diagram or flow chart, and allows editing of it. Additionally, the TTMSFNCBloxSelector component that displays all the available blocks in a single categorized list can be used to quickly insert blocks by dragging and dropping them on to the blox control.

Another helpful component is the TTMSFNCBloxToolBar. Although TTMSFNCBloxControl component does not require the TTMSFNCBloxToolBar component to work, the toolbar component is very useful to quickly get started, since it allows easy insertion and editing of blocks without requiring a single line of code. (Please note that the TTMSFNCBloxToolBar needs to be installed manually via a separate package that requires the TMS FNC UI Pack dependency, more information about this can be found in the “Installing the TTMSFNCBloxToolBar component” chapter)

Getting started with TMS FNC Blox is as quick as dropping both TTMSFNCBloxControl and TTMSFNCBloxToolBar/TTMSFNCBloxSelector components in a form, and linking the toolbar to the blox control, by setting the BloxControl property of the TTMSFNCBloxToolBar/TTMSFNCBloxSelector component to make reference to the TTMSFNCBloxControl component. When the TTMSFNCBloxToolBar/TTMSFNCBloxSelector detects a TTMSFNCBloxControl on the form, the BloxControl will automatically be assigned with the first instance it encounters during creation.

With this setting, and with no line of code, you will have a running application with already provides diagramming and flowcharting capabilities. Just choose an object from the toolbar/selector and click the blox control to insert objects, and you can start editing, resizing, moving and deleting them. Below is a screenshot of the TTMSFNCBloxToolBar (left), the TTMSFNCBloxSelector (middle) and the TTMSFNCBloxControl (right).

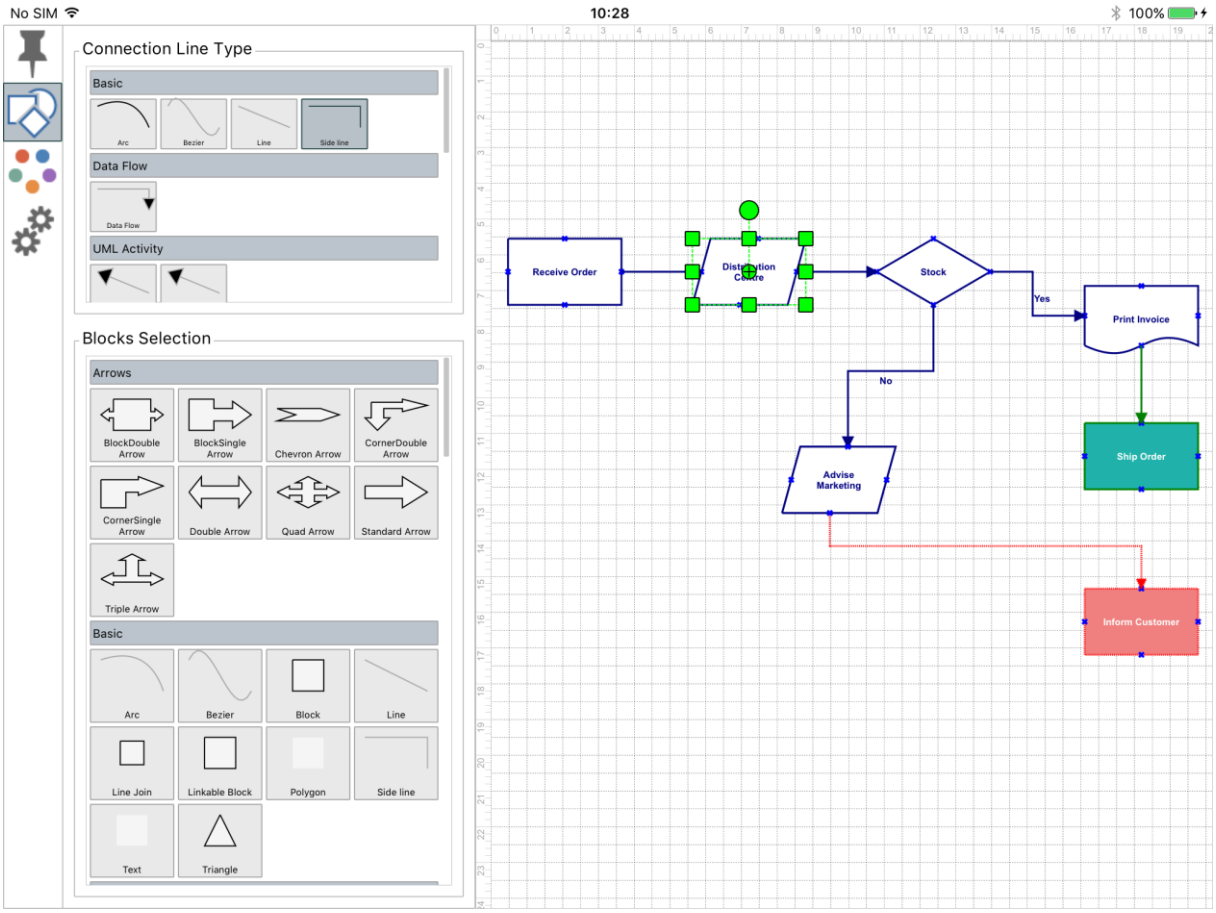


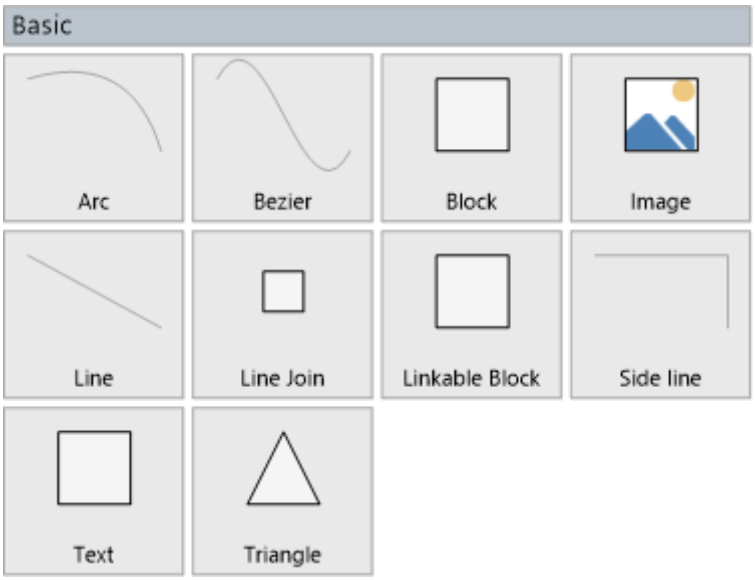
Diagram Objects

There are two types of diagram objects: blocks and lines. All blocks descend from TTMSFNCBloxBlock class, and all lines descend from TTMSFNCBloxLine block. TTMSFNCBloxElement class is the ancestor for both classes.

TTMSFNCBloxElement
- TTMSFNCBloxLine
- TTMSFNCBloxBlock

Basic objects

By default, TMS FNC Blox provides some basic blocks and lines, which descend from both TTMSFNCBloxBlock and TTMSFNCBloxLine classes.



- TTMSFNCBloxArc: a TTMSFNCBloxLine descendant, configured as a curved line.
- TTMSFNCBloxBezier: a TTMSFNCBloxLine descendant, configured as a Bezier-like line.
- TTMSFNCBloxBlock: a basic diagram block. Descends from TTMSFNCBloxElement and published all of its properties, so this block is full-featured, allow setting colors, shadows, gradients, pictures, linkpoints, shapes, … .
- TTMSFNCBloxImageBlock: a TTMSFNCBloxBlock descendant which sets some properties to look more like an image block. (TTMSFNCBloxBlock aslo provides imaging capabilities, along all other blox objects)
- TTMSFNCBloxLine: a single line (one segment).
- TTMSFNCBloxLineJoin: a TTMSFNCBloxBlock descendant which is just a link point container. You can attach lines to this block.
- TTMSFNCBloxLinkableBlock: a TTMSFNCBloxBlock descendant that adds 4 default link points.
- TTMSFNCBloxSideLine: a TTMSFNCBloxLine descendant, configured with several perpendicular segments. The number of segments is calculated automatically.
- TTMSFNCBloxTextBlock: a TTMSFNCBloxBlock descendant which sets some properties to look more like a text block (TTMSFNCBloxBlock also provides text capabilities, along all other blox objects).
- TTMSFNCBloxTriangle: a TTMSFNCBloxBlock descendant, shaped as a triangle.

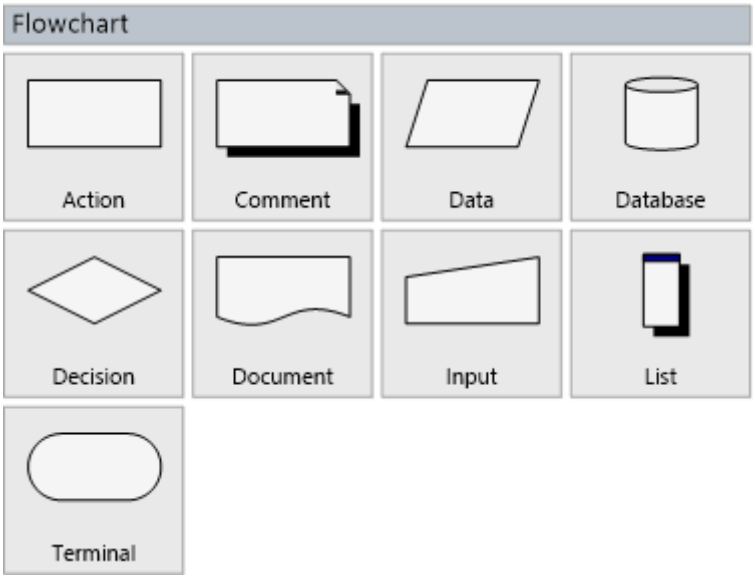
Extra Objects

TMS FNC Blox provides extra objects to be used inside the TTMSFNCBloxControl component. These extra objects are included for instant usage, but also as samples that demonstrate how to extend TMS FNC Blox by creating more blocks. The extra objects in TMS FNC Blox are grouped in the following categories:

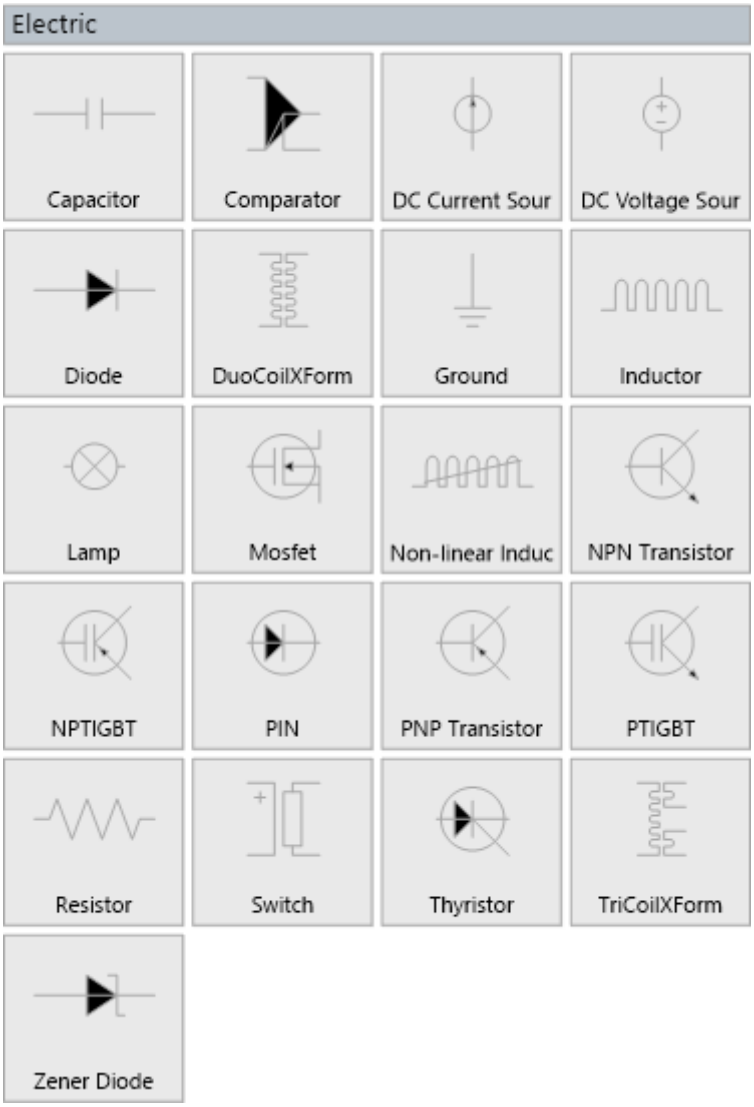
- Flowchart blocks: Provides some basic blocks for flowcharting diagrams such as action, start, end, decision …
- Electric blocks: Provides some basic blocks for electric diagrams such as resistor, capacitor, voltage source, ground …
- Arrow blocks: Provides some arrow-shaped blocks.
- DFD blocks: Provides some data flow diagram blocks.

- UML blocks: Provides some Unified Modeling Language blocks.

All extra objects are registered by the TTMSFNCBloxControl and appear in the TTMSFNCBloxSelector component at runtime. Below is an overview of the types that are registered for each category.

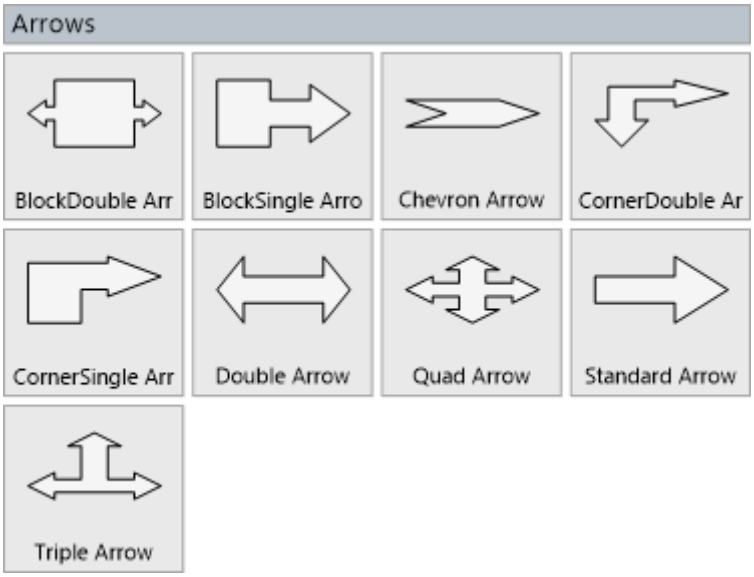


- TTMSFNCBloxFlowActionBlock
- TTMSFNCBloxFlowCommentBlock
- TTMSFNCBloxFlowDataBlock
- TTMSFNCBloxFlowDatabaseBlock
- TTMSFNCBloxFlowDecisionBlock
- TTMSFNCBloxFlowDocumentBlock
- TTMSFNCBloxFlowInputBlock
- TTMSFNCBloxFlowListBlock
- TTMSFNCBloxFlowTerminalBlock



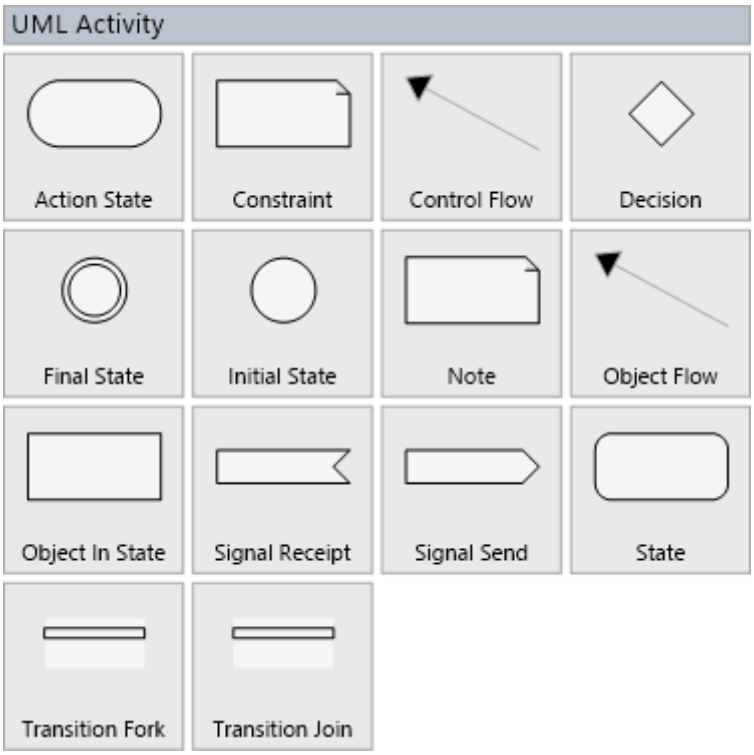
- TTMSFNCBloxCapacitorBlock
- TTMSFNCBloxComparatorBlock
- TTMSFNCBloxDCCurrentSourceBlock
- TTMSFNCBloxDCVoltageSourceBlock
- TTMSFNCBloxDiodeBlock
- TTMSFNCBloxDuoCoilXFormBlock
- TTMSFNCBloxGroundBlock
- TTMSFNCBloxInductorBlock
- TTMSFNCBloxLampBlock
- TTMSFNCBloxMosfetBlock
- TTMSFNCBloxNonLinearInductorBlock
- TTMSFNCBloxNPNTransistorBlock
- TTMSFNCBloxNPTIGBTBlock
- TTMSFNCBloxPINBlock
- TTMSFNCBloxPNPTransistorBlock
- TTMSFNCBloxPTIGBTBlock
- TTMSFNCBloxResistorBlock
- TTMSFNCBloxSwitchBlock
- TTMSFNCBloxThyristorBlock

- TTMSFNCBloxTriCoilXFormBlock
- TTMSFNCBloxZenerDiodeBlock

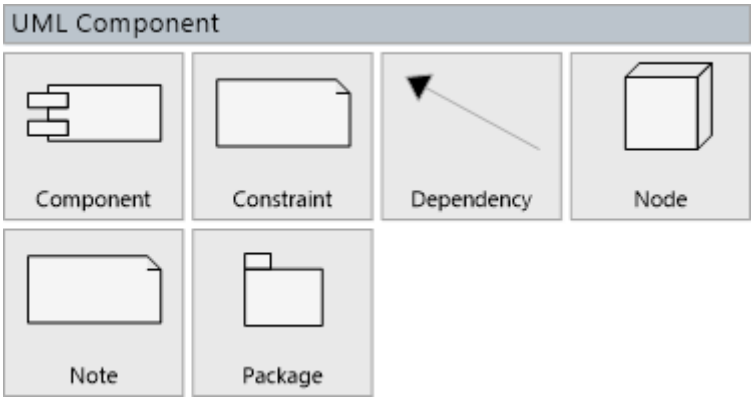


- TTMSFNCBloxBlockDoubleArrowBlock
- TTMSFNCBloxBlockSingleArrowBlock
- TTMSFNCBloxChevronArrowBlock
- TTMSFNCBloxCornerDoubleArrowBlock
- TTMSFNCBloxCornerSingleArrowBlock
- TTMSFNCBloxDoubleArrowBlock
- TTMSFNCBloxQuadArrowBlock
- TTMSFNCBloxStandardArrowBlock
- TTMSFNCBloxTripleArrowBlock

- TTMSFNCBloxDFDDataFlowLine
- TTMSFNCBloxDFDDataStoreBlock
- TTMSFNCBloxDFDInterfaceBlock
- TTMSFNCBloxDFDProcessBlock

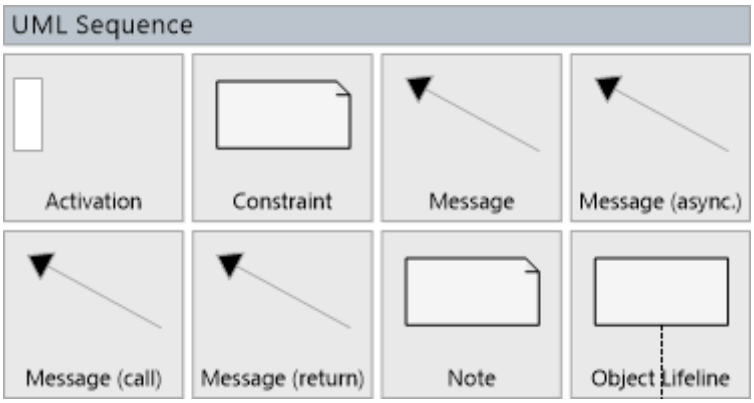


- TTMSFNCBloxUMLActionStateBlock
- TTMSFNCBloxUMLConstraintBlock
- TTMSFNCBloxUMLGenericLine (Control Flow)
- TTMSFNCBloxUMLDecisionBlock
- TTMSFNCBloxUMLFinalStateBlock
- TTMSFNCBloxUMLInitialStateBlock
- TTMSFNCBloxUMLNoteBlock
- TTMSFNCBloxUMLGenericLine (Object Flow)
- TTMSFNCBloxUMLObjectInStateBlock
- TTMSFNCBloxUMLSignalReceiptBlock
- TTMSFNCBloxUMLSignalSendBlock
- TTMSFNCBloxUMLStateBlock
- TTMSFNCBloxUMLTransitionForkBlock
- TTMSFNCBloxUMLTransitionJoinBlock

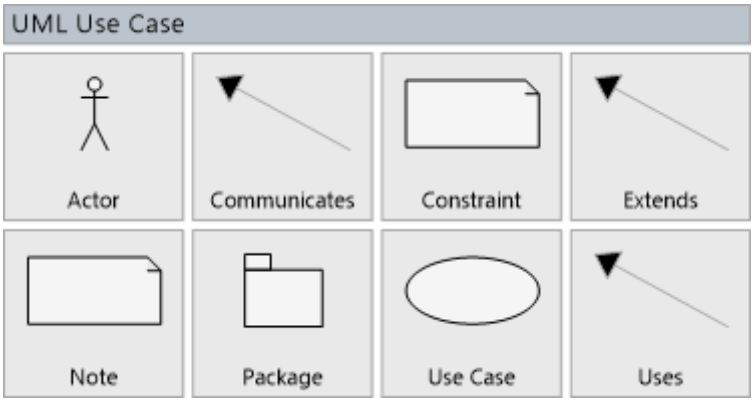


- TTMSFNCBloxUMLComponentBlock
- TTMSFNCBloxUMLContraintBlock

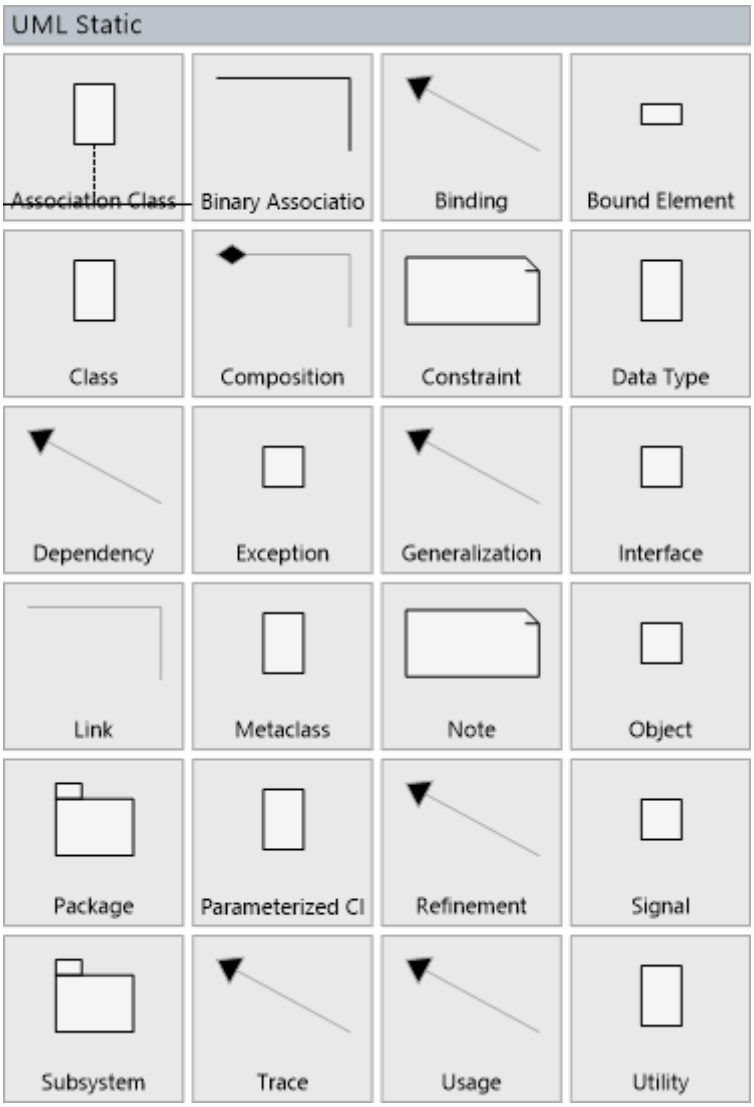
- TTMSFNCBloxUMLGenericLine (Dependency)
- TTMSFNCBloxUMLNodeBlock
- TTMSFNCBloxUMLNoteBlock
- TTMSFNCBloxUMLPackageBlock



- TTMSFNCBloxUMLActivationBlock
- TTMSFNCBloxUMLConstraintBlock
- TTMSFNCBloxUMLGenericLine (Message)
- TTMSFNCBloxUMLGenericLine (Message async)
- TTMSFNCBloxUMLGenericLine (Message call)
- TTMSFNCBloxUMLGenericLine (Message return)
- TTMSFNCBloxUMLNoteBlock
- TTMSFNCBloxUMLObjectLifelineBlock



- TTMSFNCBloxUMLActorBlock
- TTMSFNCBloxUMLGenericLine (Communicates)
- TTMSFNCBloxUMLConstraintBlock
- TTMSFNCBloxUMLGenericLine (Extends)
- TTMSFNCBloxUMLNoteBlock
- TTMSFNCBloxUMLPackageBlock
- TTMSFNCBloxUMLUseCaseBlock
- TTMSFNCBloxUMLGenericLine (Uses)



- TTMSFNCBloxUMLAssociationClassBlock
- TTMSFNCBloxUMLAssociation
- TTMSFNCBloxUMLGenericLine (Binding)
- TTMSFNCBloxUMLBoundElementBlock
- TTMSFNCBloxUMLClassBlock
- TTMSFNCBloxUMLComposition
- TTMSFNCBloxUMLConstraintBlock
- TTMSFNCBloxUMLDataTypeBlock
- TTMSFNCBloxUMLGenericLine (Dependency)
- TTMSFNCBloxUMLExceptionBlock
- TTMSFNCBloxUMLGenericLine (Generalization)
- TTMSFNCBloxUMLInterfaceBlock
- TTMSFNCBloxUMLLink
- TTMSFNCBloxUMLMetaclassBlock
- TTMSFNCBloxUMLNoteBlock
- TTMSFNCBloxUMLObjectBlock
- TTMSFNCBloxUMLPackageBlock
- TTMSFNCBloxUMLParamClassBlock
- TTMSFNCBloxUMLGenericLine (Refinement)

- TTMSFNCBloxUMLSignalBlock
- TTMSFNCBloxUMLSubsystemBlock
- TTMSFNCBloxUMLGenericLine (Trace)
- TTMSFNCBloxUMLGenericLine (Usage)
- TTMSFNCBloxUMLUtilityBlock

Controlling visual editing of diagram

The toolbar component makes it easier to allow visual editing of diagram: end-user just click the toolbar to choose a block, and then click on the blox control to insert the selected block. However there are several methods in the TTMSFNCBloxControl component that allows you to control visual editing of the diagram. In summary, TTMSFNCBloxControl implements the concept of “state”, so depending on the state of the TTMSFNCBloxControl component the end-user is able to do something with the current diagram.

States

The TTMSFNCBloxControl component can be in any of these states:

- Browsing: end-user is able to select, move, resize and rotate blocks. This is the default state.
- Inserting: end-user is able to insert a block in the TTMSFNCBloxControl component.
- Panning: end-user is panning the diagram (dragging inside the TTMSFNCBloxControl component).
- Zooming: end-user is zooming the diagram in or out.

These are the basic four states. They define what end-user actions do in the diagram, in terms of visual editing. For example, what happens if the end-user left-clicks the diagram, keeps the left button of the mouse, finger pressed, moves the mouse, finger some pixels and then release the button, finger? It depends on the state of the TTMSFNCBloxControl component. If the state is "browsing", then the end-user action will select some diagram objects. If state is inserting, it will insert an object in the TTMSFNCBloxControl component. If state is panning, it will move the diagram some pixels in the mouse, finger direction. If state is zooming, it will zoom the diagram.
The default state is browsing. So, to put the diagram in the other four states, you can use some key methods.

Inserting blocks – TTMSFNCBloxControl insert mode (state)

Call StartInsertingElement method to put the TTMSFNCBloxControl component in insert mode. After calling this method, any mouse, or touch operation like clicking or dragging will insert a new object. The object to be inserted is the first parameter of the StartInsertElement, and must be identified by its control id. The second parameter (AKeepInserting) specifies if the TTMSFNCBloxControl component should go back to browsing mod after the object is inserted (false), or if the TTMSFNCBloxControl component will continue in the insert mode (true), allowing the same object to be inserted multiple times. At any time, the TTMSFNCBloxControl component can be put back in browsing mode by calling CancelInsertingBlock method.

```
procedure TForm1.NewBlockButtonClick(Sender: TObject);
begin
  TMSFNCBloxControl1.Presenter.StartInsertingElement('TTMSFNCBloxBlock');
//Insert the TDiagramBlock object once
end;

procedure TForm1.NewTextButtonClick(Sender: TObject);
begin

TMSFNCBloxControl1.Presenter.StartInsertingElement('TTMSFNCBloxTextBlock',
true); //Insert the TTextBlock object multiple times
```

```
end;

procedure TForm1.CancelInsertButtonClick(Sender: TObject);
begin
  TMSFNCBloxControl1.Presenter.CancelInsertingBlock;
end;

procedure TForm1.NewBlockButtonClick(Sender: TObject);
begin
  TMSFNCBloxControl1.Presenter.StartInsertingElement('TTMSFNCBloxBlock');
//Insert the TDiagramBlock object once
end;

procedure TForm1.NewTextButtonClick(Sender: TObject);
begin

TMSFNCBloxControl1.Presenter.StartInsertingElement('TTMSFNCBloxTextBlock',
true); //Insert the TTextBlock object multiple times
end;

procedure TForm1.CancelInsertButtonClick(Sender: TObject);
begin
  TMSFNCBloxControl1.Presenter.CancelInsertingBlock;
end;
```

Panning mode (state)

To put the TTMSFNCBloxControl component in panning mode, just call StartPanning method. To bring the TTMSFNCBloxControl component back to browsing state, call CancelPanning.

```
procedure TForm1.StartPanningButtonClick(Sender: TObject);
begin
  TMSFNCBloxControl1.Presenter.StartPanning;
end;

procedure TForm1.CancelPanningButtonClick(Sender: TObject);
begin
  TMSFNCBloxControl1.Presenter.CancelPanning;
end;
```

Zooming mode (state)

To put the TTMSFNCBloxControl component in zooming mode, call StartZooming. You must inform if the zoom is in (more zoom) or out (less zoom). To bring the TTMSFNCBloxControl component back to browsing state, call CancelZooming.

```
procedure TForm1.ZoomInButtonClick(Sender: TObject);
begin
  TMSFNCBloxControl1.Presenter.StartZooming(zsZoomIn);
end;

procedure TForm1.ZoomOutButtonClick(Sender: TObject);
begin
  TMSFNCBloxControl1.Presenter.StartZooming(zsZoomOut);
end;

procedure TForm1.CancelZoomButtonClick(Sender: TObject);
```

```
begin
  TMSFNCBloxControl1.Presenter.CancelZooming;
end;
```

Loading and saving a diagram

Saving and loading a diagram is a very simple task, and you can do that just by using the methods SaveToFile and LoadFromFile.

```
TMSFNCBloxControl1.SaveToFile('MyBlox1.blox');
TMSFNCBloxControl1.LoadFromFile('MyBlox2.blox');
```

As an alternative, you can load/save from/to a stream. This can be useful for saving diagrams in different storages, such as databases.

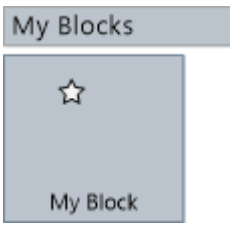
```
TMSFNCBloxControl1.SaveToStream(AStream);
TMSFNCBloxControl1.LoadFromStream(AStream);
```

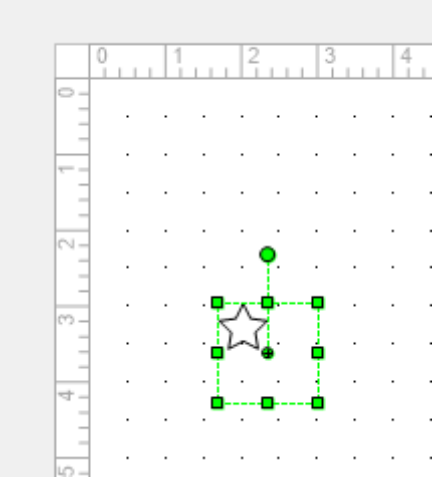
Registering new blocks

To register new blocks, you can use the OnRegisterElements event from TTMSFNCBloxControl and use the RegisterElement call located in the *.TMSFNCUIRegistration unit (* can be FMX. VCL. or LCL depending on the chosen framework). After registering the new block the selector / toolbar will automatically pick up the new category and block. If for some reason the selector does not refresh itself or doesn't show the newly registered block, call Rebuild on both the toolbar and the selector. Below is a sample of how to add a new block.

```
type
  TMyBlock = class(TTMSFNCBloxBlock)
  protected
    procedure GetBlockPath(APath: TTMSFNCBloxPath; {%H-}ADrawer:
TTMSFNCBloxBlockDrawer); override;
  end;

implementation

procedure TForm1.TMSFNCBloxControl1RegisterElements(Sender: TObject);
begin
  RegisterElement(TMyBlock, '', 'My Block', 'My Blocks');
end;

{ TMyBlock }

procedure TMyBlock.GetBlockPath(APath: TTMSFNCBloxPath;
  ADrawer: TTMSFNCBloxBlockDrawer);
var
  poly: TTMSFNCBloxPointArray;
  an1, an2, sin1, sin2, cos1, cos2, inr, outr: Single;
  I: Integer;
begin
  inherited;
  APath.Reset;
  an1 := PI / 5.0;
  an2 := 2.0 * an1;

  sin1 := Sin(an1);
  sin2 := Sin(an2);
```

```
  cos1 := Cos(an1);
  cos2 := Cos(an2);

  if Width > Height then
  begin
    outr := Height / 2;
    inr := Height / 4;
  end
  else
  begin
    outr := Width / 2;
    inr := Width / 4;
  end;

  SetLength(poly, 11);

  for I := 0 to length(poly) - 1 do
  begin
    poly[i] := TMSFNCBloxPoint(Width / 2, Height / 2);
  end;

  poly[0].Y := poly[0].Y - outr;
  poly[1].X := poly[1].X + inr * sin1;
  poly[1].Y := poly[1].Y - inr * cos1;
  poly[2].X := poly[2].X + outr * Sin2;
  poly[2].Y := poly[2].Y - outr * cos2;
  poly[3].X := poly[3].X + inr * sin2;
  poly[3].Y := poly[3].Y + inr * cos2;
  poly[4].X := poly[4].X + outr * sin1;
  poly[4].Y := poly[4].Y + outr * cos1;
  poly[5].Y := poly[5].Y + inr;
  poly[6].X := poly[6].X + poly[6].X - poly[4].X;
  poly[6].Y := poly[4].Y;
  poly[7].X := poly[7].X + poly[7].X - poly[3].X;
  poly[7].Y := poly[3].Y;
  poly[8].X := poly[8].X + poly[8].X - poly[2].X;
  poly[8].Y := poly[2].Y;
  poly[9].X := poly[9].X + poly[9].X - poly[1].X;
  poly[9].Y := poly[1].Y;
  poly[10] := poly[0];

  APath.AddPolygon(poly);
end;
```





Installing the TTMSFNCBloxToolBar component

The TTMSFNCBloxToolBar component is not installed by default, which means it needs to be installed separately, via a separate package. The TTMSFNCBloxToolBar relies on the TMS FNC UI Pack, with a minimum version of 1.7.4.0. Depending on the framework you are working with, you'll need to install a package following the guidelines below.

1) Install the latest versions of the TMS FNC Core, TMS FNC UI Pack and TMS FNC Blox

FMX

2) Create a new package called FMXTMSFNCBloxBridgePkgD*.dproj
3) Add the units FMX.TMSFNCBloxToolBar.pas and FMX.TMSFNCBloxToolBarReg.pas
4) Build your package which automatically prompts for adding the dependencies to TMS FNC Blox and TMS FNC UI Pack FMX versions (FMXTMSFNCBloxPkgD*.dcp and FMXTMSFNCUIPackPkgD*.dcp)
5) Install the package

VCL

2) Create a new package called VCLTMSFNCBloxBridgePkgD*.dproj
3) Add the units VCL.TMSFNCBloxToolBar.pas and VCL.TMSFNCBloxToolBarReg.pas
4) Build your package which automatically prompts for adding the dependencies to TMS FNC Blox and TMS FNC UI Pack VCL versions (VCLTMSFNCBloxPkgD*.dcp and FMXTMSFNCUIPackPkgD*.dcp)
5) Install the package

LCL

2) Create a new package called LCLTMSFNCBloxBridgePkg.dproj
3) Add the units LCLTMSFNCBloxToolBar.pas and LCLTMSFNCBloxToolBarReg.pas
4) Check "Register Unit" for the LCLTMSFNCBloxToolBarReg.pas file in the file properties section of the package window
5) Add a dependency to LCLTMSFNCBloxPkg and LCLTMSFNCUIPackPkg
6) Install the package

* The full package name depends on the chosen IDE: XE7 (RAD Studio XE7), XE8 (RAD Studio XE8), XE9 (RAD Studio 10 Seattle), XE10 (RAD Studio 10.1 Berlin) or XE11 (RAD Studio 10.2 Tokyo)

Working with TTMSFNCBloxControl programmatically

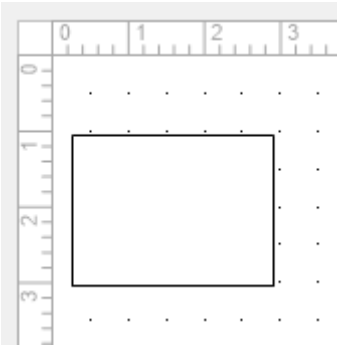
Inserting objects in TTMSFNCBloxControl

To insert objects in TTMSFNCBloxControl, you must:

1. Create the object, instantiating the class of the object of choice.
2. Set the properties you want to change
3. Add the object to the blox collection property of the TTMSFNCBloxControl

Example:

```
procedure TForm1.AddBlockButtonClick(Sender: TObject);
var
  b: TTMSFNCBloxBlock;
begin
  b := TTMSFNCBloxBlock.Create;
  b.Left := 10;
  b.Top := 40;
  b.Height := 75;
  b.Width := 100;
  TMSFNCBloxControl1.Blox.Add(b);
end;
```



Removing an object from TTMSFNCBloxControl

To remove an object from the TTMSFNCBloxControl, simply remove the block from the blox list.

```
TMSFNCBloxControl1.Blox.Remove(MyBlock);
```

Alternatively, you can remove the currently selected objects in the TTMSFNCBloxControl with the following code:

```
TMSFNCBloxControl1.Presenter.DeleteSelecteds;
```

Adding a line between two blocks (linking blocks)

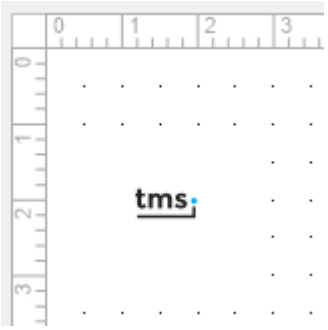
A line start (or end) can be attached to a link point. To link two blocks, you must attach the start of a line to a block linkpoint, and attach the end of the line to another block linkpoint. Below is an example that demonstrates this.

```
procedure TForm1.AddLineButtonClick(Sender: TObject);
var
  l: TTMSFNCBloxLine;
begin
  l := TTMSFNCBloxLine.Create;
  l.SourceLinkPoint.AnchorLink := SomeBlock.LinkPoints[0];
  l.TargetLinkPoint.AnchorLink := AnotherBlock.LinkPoints[1];
  TMSFNCBloxControl1.Blox.Add(l);
end;
```

Using an picture as the block shape

If you want to use a picture as the block shape, you need to load the picture, and also set the shape to none, so the shape block is not drawn, only the picture. Example:

```
procedure TForm1.AddBlockButtonClick(Sender: TObject);
var
  b: TTMSFNCBloxBlock;
begin
  b := TTMSFNCBloxBlock.Create;
  b.Left := 10;
  b.Top := 40;
  b.Height := 75;
  b.Width := 100;
  b.Shape := NoShape;
  b.Picture.LoadFromFile('MyImage.png');
  TMSFNCBloxControl1.Blox.Add(b);
end;
```



Creating linkpoints in a block

To create linkpoints, the LinkPoints collection must be used.Each linkpoint is a collection item. The position of the linkpoint must be set relatively to the rect specified by the Drawer.OriginalRect property. By default, OriginalRect is (0, 0, 100, 100), making it easy to define link points positions in terms of % of block width/height. The following example, from the Flowchart blocks unit, shows how to define four link points for the block, at the top, left, right and bottom sides of the block, in the middle of each line segment.

```
procedure TTMSFNCBloxFlowChartBlock.UpdateLinkPoints;
begin
  LinkPoints.Clear;
  with Drawer.OriginalRect do
  begin
    LinkPoints.AddLink((Right - Left) / 2, Top, aoUp);
    LinkPoints.AddLink((Right - Left) / 2, Bottom, aoDown);
```

```
    LinkPoints.AddLink(Left, (Bottom - Top) / 2, aoLeft);
    LinkPoints.AddLink(Right, (Bottom - Top) / 2, aoRight);
  end;
end;
```

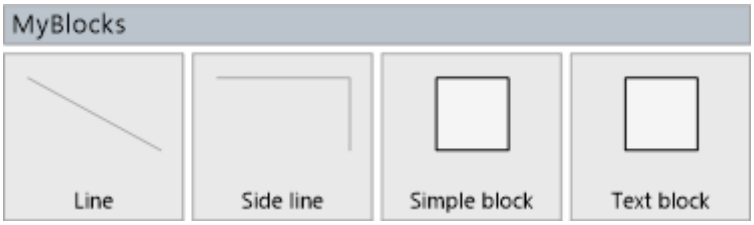
Accessing TextCells of a line object

The line objects has at least one text cell defined by default. If you want to access a text cell of a TTMSFNCBloxLine object, for example, you can use this code:

```
var
  t: TTMSFNCBloxTextCell;
begin
  t := LinkLine.TextCells[0];
end;
```

Changing category of block

Calling RegisterElement for any existing block will re-register it. So it's possible to change the category or caption for the block. The following code moves all the blocks to the "MyBlocks" category.




```
RegisterElement(TTMSFNCBloxBlock, '', 'Simple block', 'MyBlocks');
RegisterElement(TTMSFNCBloxLine, '', 'Line', 'MyBlocks');
RegisterElement(TTMSFNCBloxTextBlock, '', 'Text block', 'MyBlocks');
RegisterElement(TTMSFNCBloxSideLine, '', 'Side line', 'MyBlocks');
TMSFNCBloxSelector1.Rebuild;
```

Prevent a line from being detached from a block

To prevent a line from being detached from a block, simply set the RequireConnections property to true on the line object.

```
MyLine1.RequiresConnections := True;
```

Properties, Methods and Events

Below is a list of the most important properties methods and events for TTMSFNCBloxControl, TTMSFNCBloxToolBar, TTMSFNCBloxSelector and the base classes TTMSFNCBloxBlock and TTMSFNCBloxLine.

Properties for TTMSFNCBloxControl

AntiAliasing	Turn AntiAliasing on or off, to improve quality or speed. By default AntiAliasing is true
AutoScroll	Automatically scrolls to the selected block when the selected block is moved outside the visual area of the TTMSFNCBloxControl
Blox	A list of blox, used to add newly created blox to the TTMSFNCBloxControl
DefaultUnit	The default measuring unit which is applied to the rulers and snap grid. Values are duMili, duCenti and duInch
Fill	The background fill of the TTMSFNCBloxControl
Interaction	Interaction properties to determine the way the TTMSFNCBloxControl responds to keyboard mouse and touch actions
Interaction → CanMoveOutOfBounds	Sets a property to force blocks withing the boundaries of the TTMSFNCBloxControl when moving or inserting the blocks
Interaction → KeyActions	Actions that can be executed via the keyboard such as Escape, Move, Page, Resize, Select, Delete
Interaction → ReadOnly	Completely disables interaction with the TTMSFNCBloxControl
Interaction → SelectionMode	Use SelectionMode property to determine how the selection will behave in the Blox. Default value is slmMultiple. slmMultiple: In multiple selection, each object is displayed selected individually. A selecion rectangle is displayed for each object, and also its handles are displayed individually. slmGroup: In multiple selection, all selected objects are displayed as a group, and handles are displayed only for the group. It's a more Visio-like approach
Interaction → TouchElementHandling	Enables or disables touch handling such as panning and zooming on mobile devices.
Interaction → Zooming	Enables or disables zooming via mouse, keyboard or touch.
LeftRuler/TopRuler	The Left and Top ruler properties
LeftRuler/TopRuler → AutoFactor	When the Blox is zoomed in or out, the ruler automatically changes its step. For example, when the zoom is 100%, the step is 1 centimeter (when MeasureUnit property is set to unCenti). If the zoom increases, the Blox can

	automatically change the step to 0.5 centimeter or even less. This happens when AutoFactor property is true. You can disable this behaviour by setting AutoFactor to false. In this case, the step and divisions remain the same, regardless of the zoom factor
LeftRuler/TopRuler → BorderColor	Specifies the color of the border of the ruler
LeftRuler/TopRuler → Color	The background color of the ruler.
LeftRuler/TopRuler → Divisions	Use Divisions property to specify how many divisions will be drawn between one ruler value and next. The Divisions property affects the double of minor ticks drawn in ruler
LeftRuler/TopRuler → Font	The font used when drawing the ruler numbers
LeftRuler/TopRuler → FontColor	Specifies the color of the font of the tickmarks
LeftRuler/TopRuler → Inverse	Specifies whether the ruler must be drawn from left to right, top to bottom or vice versa
LeftRuler/TopRuler → MajorTickLength	Specifies the length of the major tick mark
LeftRuler/TopRuler → MeasureUnit	Specifies the unit of the ruler in millimeters, centimeters or inches
LeftRuler/TopRuler → MinorTickLength	Specifies the length of the minor tick mark
LeftRuler/TopRuler → Offset	Specifies the ruler offset in pixels. If Offset is positive, the ruler will be moved left (starting at a point higher than the zero)
LeftRuler/TopRuler → Size	Specifies the height (for top ruler) or width (for left ruler) of the ruler. Size has a minimum value, depending on the chosen font
LeftRuler/TopRuler → TickColor	Specifies the color of the tickmarks
LeftRuler/TopRuler → Visible	Specifies if the ruler and snap grid lines are visible or not.
LeftRuler/TopRuler → Zoom	Specifies if the ruler zoom factor
Presenter	The presenter is used for programmatic access to the blocks and blox control.
SnapGrid	The settings for the snap grid, used to insert blocks, move and rotate them according to specific measuring and snapping coordinates
SnapGrid → Active	If Active is True, movements done in the TTMSFNCBloxControl follow the snap grid. If active is false, snap grid has no effect in the actions performed by the user
SnapGrid → Force	When Force property is true, the object being inserted in the Blox is automatically snapped to grid. When Force is false, the object can be placed everywhere.
SnapGrid → SizeX	Specifies the horizontal size of snap grid step in Blox coordinates
SnapGrid → SizeY	Specifies the vertical size of snap grid step in Blox coordinates
SnapGrid → SnapToRuler	When SnapToRuler is true, SizeX and SizeY properties are ignored, and the snap grid follows the ruler subdivisions (ticks)
SnapGrid → Stroke	The visual appearance of the snap grid lines and dots
SnapGrid → Style	Specify the visual style of snap grid
SnapGrid → Visible	The visibility of the snap grid

Stroke	The background stroke of the TTMSFNCBloxControl
Zoom	The overall zoom of the TTMSFNCBloxControl

Methods for TTMSFNCBloxControl

BeginUpdate;	Blocks all visual updates. Needs to be paired with EndUpdate;
EndUpdate;	Blocks all visual updates. Needs to be paired with BeginUpdate; When EndUpdate is called, the control is automatically recalculated/repainted
Load;	Loads the saved state of the TTMSFNCBloxControl (saved with the Save method)
LoadFromFile(AFileName: string);	Loads a diagram from a specific file
LoadFromStream(AStream: TStream);	Loads a diagram from a specific stream
Presenter → AddSelected(AElement: TTMSFNCBloxElement);	Adds an element to the selected list
Presenter → AsMember(AElement: TTMSFNCBloxElement): Boolean;	Verifies if the element is a member of a group and if the element group is selected
Presenter → BringSelectedsToFront;	Brings all selected elements to the front
Presenter → BringToFront(AElement: TTMSFNCBloxElement);	Brings a specific element to the front
Presenter → CancelInsertingBlock;	Cancels inserting a block started with one of the StartInsertingElement method overloads
Presenter → CancelPanning;	Cancels panning started with StartPanning
Presenter → CanPaste: Boolean;	Returns a Boolean if blox can be pasted from the clipboard
Presenter → CanRedo: Boolean;	Returns a Boolean if the TTMSFNCBloxControl can perform a Redo action
Presenter → CanUndo: Boolean;	Returns a Boolean if the TTMSFNCBloxControl can perform an Undo action
Presenter → CanvasToClient(APoint: TTMSFNCBloxPoint): TTMSFNCBloxPoint;	Converts a blox coordinate to a pixel coordinate
Presenter → ClearUndoStack;	Clears the undo stack (The undo stack has a maximum of 20 undo actions, for each action that is pushed in the stack, the oldest action is removed)
Presenter → ClientToCanvas(APoint: TTMSFNCBloxPoint): TTMSFNCBloxPoint;	Converts a pixel coordinate to a blox coordinate
Presenter → CopyElementsToClipboard;	Copies the selected elements to the clipboard
Presenter → DeleteElements(restricted: TTMSFNCBloxRestrictions);	Deletes elements in the selected list based on the element restrictions. Restrictions can be one or multiple of the following values crNoMove, crNoResize, crNoRotation, crNoEdit, crNoDelete, crKeepRatio, crNoClipboard, crNoSelect, crNoRotCenterMove, crNoLink
Presenter → DeleteSelecteds;	Delete all the selected elements
Presenter → EndInsertingElement;	Inserts the element at position 0, 0
Presenter → GroupSelectedBlocks: TTMSFNCBloxGroup;	Groups all the selected blocks, creates a TTMSFNCBloxGroup element and returns that group
Presenter → GroupSelection: Boolean;	Returns a Boolean to determine if the

	TTMSFNCBloxControl has groups selected
Presenter → HasSelecteds(AGroup: TTMSFNCBloxGroup): Boolean;	Returns a Boolean whether a group has selected elements
Presenter → IsDisplayingHandles(AElement: TTMSFNCBloxElement): Boolean;	Returns a Boolean whether an element is displaying selection handles
Presenter → Modified;	Calls the OnModified event
Presenter → MoveBlocks(ADeltaX, ADeltaY: Double; AOnlySelected: Boolean = false; AOnlyMovable: Boolean = true);	Moves blocks to a specific coordinate based on the original coordinate shifted with the delta x and y parameters. Optionally move only the selected blocks and blocks that are movable
Presenter → NextRedoAction: string;	Returns the next redo action based on the name added when pushing an action to the undo stack
Presenter → NextUndoAction: string;	Returns the next undo action based on the name added when pushing an action to the undo stack
Presenter → PasteElementsFromClipboard;	Pastes elements from the clipboard when the clipboard has valid blox data
Presenter → PushUndoStack(const AActionName: string);	Pushes an action to the undo stack
Presenter → Redo;	Goes one step forward in the undo stack and reinitializes the TTMSFNCBloxControl with the next state of the blocks
Presenter → Redraw;	Forces a redraw of the blox control
Presenter → RemoveSelected(AElement: TTMSFNCBloxElement);	Removes the element if the element is selected
Presenter → SelectAll;	Selects all elements
Presenter → SelectedBlockCount(AFilter: TTMSFNCBloxElementfilter = cfAll): Integer;	Returns the count of selected blocks based on a specific filter
Presenter → SelectedCount(AFilter: TTMSFNCBloxElementFilter = cfAll): Integer;	Returns the count of selected elements based on a filter
Presenter → SelectedLinkCount: Integer;	Returns the count of selected links
Presenter → SendSelectedsToBack;	Sends all the selected elements to the back
Presenter → SendToBack(AElement: TTMSFNCBloxElement);	Sends a specific element to the back
Presenter → SetDefaultState;	Forces the state of the TTMSFNCBloxControl to msBrowsing if for some reason the state is corrupted due to the combination of multiple actions such as panning, zooming in the TTMSFNCBloxControl or inserting, deleting an element
Presenter → StartInsertingElement(AElement: TTMSFNCBloxEelement; AKeepInserting: Boolean = False);	Starts inserting a created element in the TTMSFNCBloxControl. The AKeepInserting parameter can be used to create a duplicate of that item and keep inserting that item. To cancel or end this operation call CancelInsertBlock or EndInsertingElement
Presenter → StartInsertingElement(AElementClass: TTMSFNCBloxElementClass; AKeepInserting: Boolean = False);	Starts inserting an element based on a specific element class. The behavior is identical to the previous StartInsertingElement method
Presenter → StartInsertingElement(const AElementId: string; AKeepInserting: Boolean = False);	Starts inserting an element based on a specific element id. The behavior is identical to the previous StartInsertingElement method
Presenter → StartPanning;	Starts a panning operation in the TTMSFNCBloxControl
Presenter → Undo;	Goes one step backward in the undo stack and

	reinitializes the TTMSFNCBloxControl with the previous state of the blocks
Presenter → UngroupSelectedBlocks;	Ungroups all selected blocks
Presenter → UnselectAll;	Unselects all selected blocks
RegisterElements;	Forces a re-register of all the supported basic and extra elements and additionally calls the OnRegisterElements event to re-register custom elements
Save;	Saves the diagram
SavedBlox: string;	Returns the state of the diagram as a string
SaveToFile(AFileName: string);	Saves the state of the diagram to a file
SaveToImage(AFileName: string; ABloxOnly: Boolean = True; ALinkPoints: Boolean = False; ABackground: Boolean = True);	Saves the diagram to an image, based on a specific series of parameters to enable disable the background, show or hide the link points and display only the blocks or also the rulers and snap grid
SaveToStream(AStream: TStream);	Saves the diagram to a stream

Events for TTMSFNCBloxControl

OnAfterBackgroundRender	Event called after the background is rendered. With the Renderer.Canvas parameter you are able to add additional drawing to the blox control
OnAfterMove	Event called after an element has been moved
OnAfterRender	Event called after all elements are rendered
OnAfterResize	Event called after an element is resized
OnBeforeRender	Event called before all elements are rendered
OnElementClick	Event called when a specific element is clicked
OnElementDblClick	Event called when a specific element is double clicked
OnElementInsert	Event called when an element is inserted
OnElementMouseDown	Event called when the mouse is down on an element
OnElementMouseMove	Event called when the mouse is moving over an element
OnElementMouseUp	Event called when the mouse is up on an element
OnElementPainted	Event called when an element is done painting
OnElementPainting	Event called when an element is being painted
OnElementRemove	Event called when an element is removed
OnElementResizing	Event called when an element is being resized
OnElementSelect	Event called when an element is selected
OnElementUnselect	Event called when an element is unselected
OnModified	Event called when a modification is made to the blox control
OnRegisterElements	Event called when the blox control is registering all basic an extra elements and you want to register your own custom elements or you want to re-arrange existing elements in different categories
OnSelectionChange	Event called when the selection is changed between elements

Properties for TTMSFNCBloxToolBar

AntiAliasing	Turn AntiAliasing on or off, to improve quality or speed. By default AntiAliasing is true
Appearance	The appearance of the items in the toolbar
Appearance → Fill	The fill of the items in normal state
Appearance → FillDisabled	The fill of the items in disabled state
Appearance → FillDown	The fill of the items in down state
Appearance → FillHover	The fill of the items in hover state
Appearance → FillSelected	The fill of the items in selected state
Appearance → Font	The font of the items
Appearance → HorizontalSpacing	The horizontal spacing between items
Appearance → SeparatorStroke	The stroke of the separator
Appearance → Stroke	The stroke of the items in normal state
Appearance → StrokeDisabled	The stroke of the items in disabled state
Appearance → StrokeDown	The stroke of the items in down state
Appearance → StrokeHover	The stroke of the items in hover state
Appearance → StrokeSelected	The stroke of the items in selected state
Appearance → VerticalSpacing	The vertical spacing between items
BitmapContainer	A reference to an instance of TTMSFNCBitmapContainer, used when assigning a Bitmaps to the items
BloxControl	A reference to an instance of TTMSFNCBloxControl to interact with it and update the toolbar items / panels
EmbeddedItemPosition	The position of the items in embedded mode
EmbeddedMode	Embedded mode displays the panel inside the control at all times instead of the default popup mode
EmbeddedSize	The size of the panels when displaying them in embedded mode
Fill	The fill of the toolbar
Items	The items of the toolbar
Items[Index] → Bitmaps	The bitmaps of the items of the toolbar
Items[Index] → CanDeselect	Enables/disables deselection of an item
Items[Index] → CanSelect	Enables/disables selection of an item
Items[Index] → ColumnSpan	The column span of an item (based on the columns property of the toolbar)
Items[Index] → DisabledBitmaps	The bitmaps of the items of the toolbar in disabled state
Items[Index] → Enabled	Enables/disables an item
Items[Index] → FocusedControl	The control which needs to be focused when the popup is shown after clicking an item, this can be the first control in the PopupControl or any other control that requires focus when first displayed
Items[Index] → Hint	A hint displayed on each item
Items[Index] → Margins	The margins of an item relative to its position and the width and height of an item
Items[Index] → PopupControl	The popup control, displayed when clicking an item
Items[Index] → PopupHeight	The height of the popup when PopupMode is set to ipmCustom
Items[Index] → PopupMode	The mode of the popup, which can be to match

	the control width or height (depending on the orientation), the size of the PopupControl itself or a custom size (PopupWidth and PopupHeight properties)
Items[Index] → PopupOrientation	The orientation of the popup relative to the toolbar item
Items[Index] → PopupWidth	The width of the popup when PopupMode is set to ipmCustom
Items[Index] → RowSpan	The row span of an item (based on the rows property of the toolbar)
Items[Index] → Separator	Configures an item to be displayed as a separator. A separator is non-interactable and displays a thin line based on the SeparatorStroke under appearance
Items[Index] → SeparatorHeight	The height of the separator
Items[Index] → Visible	Shows or hides an item
Mode	The mode of the toolbar, to be display horizontally or vertically
SelectedItemIndex	The index of the item that is currently selected
Stroke	The stroke of the toolbar

Methods for TTMSFNCBloxToolBar

AddCategoryItem(ABitmap: TTMSFNCBitmap): TTMSFNCCategoryListItem;	Adds a new toolbar item with an existing TTMSFNCBitmap instance
AddCategoryItem(ABitmapName: string): TTMSFNCCategoryListItem;	Adds a new toolbar item with a bitmap name from the referenced TTMSFNCBitmapContainer
AddCategoryItem: TTMSFNCCategoryListItem;	Adds a new toolbar item
BeginUpdate	Blocks all visual updates. Needs to be paired with EndUpdate;
ClosePopup	Closes an active popup, displayed after clicking the item that has a PopupControl attached
EndUpdate	Blocks all visual updates. Needs to be paired with BeginUpdate; When EndUpdate is called, the control is automatically recalculated/repainted
InitializeDefault	Initializes the toolbar with a default appearance and a set of items, based on the necessary tools needed to configure and manipulate diagrams displayed in the TTMSFNCBloxControl
InsertCategoryItem(AIndex: Integer): TTMSFNCCategoryListItem;	Inserts a new toolbar item at a specific index
InsertCategoryItem(AIndex: Integer; ABitmap: TTMSFNCBitmap): TTMSFNCCategoryListItem;	Inserts a new toolbar item at a specific index with an existing TTMSFNCBitmap instance
InsertCategoryItem(AIndex: Integer; ABitmapName: string): TTMSFNCCategoryListItem;	Inserts a new toolbar item at a specific index with a bitmap name from the references TTMSFNCBitmapContainer
Rebuild	Completely rebuilds the toolbar, re-initializes the registered elements and calls InitializeDefault
XYToItem(X, Y: Single): Integer;	Returns the item index of a specific item under X and Y coordinates

Events for TTMSFNCBloxToolBar

When assigning an event that executes an action on a block, line or blox control, you are responsible for setting the correct properties. The default behavior is not executed as soon as an event is assigned. This has been designed in this way that you are able to provide a different meaning/purpose to the action that has been executed.

OnAfterDraw	Event called after all elements of the toolbar has been drawn
OnApplyAlignCenter	Event called when the align center button has been clicked
OnApplyAlignLeft	Event called when the align left button has been clicked
OnApplyAlignRight	Event called when the align right button has been clicked
OnApplyBold	Event called when the bold button has been clicked
OnApplyFillColor	Event called when a fill color is selected
OnApplyFillColorTo	Event called when a fill color to is selected
OnApplyFillOrientation	Event called when the fill orientation is selected
OnApplyFillType	Event called when the fill type is selected
OnApplyFontName	Event called when the font name is selected
OnApplyFontSize	Event called when the font size is selected
OnApplyItalic	Event called when the italic button has been clicked
OnApplyPictureMode	Event called when the picture mode is changed
OnApplySnapToGrid	Event called when the snap to grid button is clicked
OnApplyStrikeOut	Event called when the strike out button has been clicked
OnApplyStrokeColor	Event called when a stroke color is selected
OnApplyStrokeStyle	Event called when a stroke style is selected
OnApplyStrokeWidth	Event called when a stroke width is selected
OnApplyText	Event called when the text changes
OnApplyTextColor	Event called when the text color is applied
OnApplyUnderline	Event called when the underline button has been clicked
OnBeforeDraw	Event called before all elements of the toolbar has been drawn
OnCopy	Event called when the copy button has been clicked
OnCut	Event called when the cut button has been clicked
OnDeletePicture	Event called when the delete picture button has been clicked
OnItemAfterDrawBackground	Event called after the toolbar item background has been drawn
OnItemAfterDrawBitmap	Event called after the toolbar item bitmap has been drawn
OnItemAfterDrawContent	Event called after the toolbar item content has been drawn
OnItemBeforeDrawBackground	Event called before the toolbar item background has been drawn
OnItemBeforeDrawBitmap	Event called before the toolbar item bitmap has

	been drawn
OnItemBeforeDrawContent	Event called before the toolbar item content has been drawn
OnItemClick	Event called when an item is clicked
OnItemSelected	Event called when an item is selected
OnOpenFile	Event called when the open file button is clicked
OnOpenPicture	Event called when the open picture button is clicked
OnPaste	Event called when the paste button is clicked
OnRedo	Event called when the redo button is clicked
OnRotateLeft	Event called when the rotate left button is clicked
OnRotateRight	Event called when the rotate right button is clicked
OnSaveFile	Event called when the save button is clicked
OnUndo	Event called when the undo button is clicked
OnZoomChanged	Event called when the zoom factor has changed

Properties for TTMSFNCBloxSelector

AntiAliasing	Turn AntiAliasing on or off, to improve quality or speed. By default AntiAliasing is true
Appearance	The appearance of the items in the selector
Appearance → Fill	The fill of an item in normal state
Appearance → FillDisabled	The fill of an item in disabled state
Appearance → FillDown	The fill of an item in down state
Appearance → FillHover	The fill of an item in hover state
Appearance → FillSelected	The fill of an item in selected state
Appearance → Font	The font of an item
Appearance → HorizontalSpacing	The horizontal spacing between items
Appearance → ItemHeight	The height of an item
Appearance → ItemWidth	The width of an item
Appearance → SeparatorFill	The fill of a separator item
Appearance → SeparatorFont	The font of a separator item
Appearance → SeparatorStroke	The stroke of a separator item
Appearance → Stroke	The stroke of an item in normal state
Appearance → StrokeDisabled	The stroke of an item in disable state
Appearance → StrokeDown	The stroke of an item in down state
Appearance → StrokeHover	The stroke of an item in hover state
Appearance → StrokeSelected	The stroke of an item in selected state
Appearance → VerticalSpacing	The vertical spacing between items
BloxControl	A reference to an instance of TTMSFNCBloxControl to interact with it and update the toolbar items / panels
Columns	The amount of columns displayed in the selector control
Items	The items collection
Items[Index] → CanDeselect	Enables/disables deselection of an item
Items[Index] → CanSelect	Enables/disables selection of an item
Items[Index] → ColumnSpan	The column span of an item based on the Columns property of the selector
Items[Index] → Empty	Adds an item to be taken into account in the calculation, but ignored in the drawing and

	interaction of the selector
Items[Index] → Enabled	Enables/disables item interaction
Items[Index] → Hint	Displays a hint on an item
Items[Index] → Margins	The margins of an item
Items[Index] → RowSpan	The row span of an item based on the Rows property of the selector
Items[Index] → Separator	Displays the item as a separator based on the separator height. A separator can also display text with a different font
Items[Index] → SeparatorHeight	The height of a separator
Items[Index] → SeparatorLine	Displays the separator as a line or as a rectangle with a fill and stroke and text
Items[Index] → Visible	Shows or hides an item
Modes	The modes property is used to allow displaying lines or blocks or both
Rows	The amount of rows in the selector
SelectedItemIndex	The index of the current selected item

Methods for TTMSFNCBloxSelector

BeginUpdate	Blocks all visual updates. Needs to be paired with EndUpdate;
EndUpdate	Blocks all visual updates. Needs to be paired with BeginUpdate; When EndUpdate is called, the control is automatically recalculated/repainted
Rebuild	Completely rebuilds the toolbar, re-initializes the registered elements and re-configures the default appearance of the selector
XYToItem(X, Y: Single): Integer;	Returns an item at a specific X and Y coordinate

Events for TTMSFNCBloxSelector

OnAfterDraw	Event called after all the elements of the selector are drawn
OnBeforeDraw	Event called before all the elements of the selector are drawn
OnItemAfterDrawBackground	Event called after the background of an item is drawn
OnItemAfterDrawContent	Event called after the content of an item is drawn
OnItemAfterDrawText	Event called after the text of an item is drawn
OnItemBeforeDrawBackground	Event called before the background of an item is drawn
OnItemBeforeDrawContent	Event called before the content of an item is drawn
OnItemBeforeDrawText	Event called before the text of an item is drawn
OnItemClick	Event called when an item is clicked
OnItemSelected	Event called when an item is selected

Properties for TTMSFNCBloxBlock

Alignment	Alignment property to specify the horizontal alignment for the block text

Angle	The angle of the block in degrees
Bottom	A readonly property to get the bottom coordinate of the block, in Blox coordinates
BoundsRect	BoundsRect property contains information about the block position: top and left coordinates, width and height of the block
ClipText	When ClipText property is True, the text drawn in the block is restricted to its shape (more specifically, to the clipping region specified by GetBlockClipRegion method). If ClipText is False, the text is not clipped
Drawer	Exposes an instance for a TMSFNCBloxBlockDrawerEx class to help in the drawing of the block
Fill	Fill property contains information about block background appearance
Handles	Contains information about the handles of the element
Height	Use the Height property to read or change height of the block
Id	Contains the ID of the Blox element, as specified in the Blox element registering procedure (TTMSFNCBloxControl.RegisterElements method and TTMSFNCBloxControl.OnRegisterElements event-)
Left	Specifies the left coordinate of block, in Blox coordinates
LinkPoints	Contains information about the link points of the element
MinHeight	MinHeight property specifies the minimum height allowed for the block. The user will not be able to resize a block with a lower height than indicated by MinHeight property
MinWidth	MinWidth property specifies the minimum width allowed for the block. The user will not be able to resize a block with a lower width than indicated by MinWidth property
Picture	The Picture property contains the background picture of the block. If Picture property is empty, no picture is displayed.
PictureMode	PictureMode defines the way the picture is rendered inside the element shape
Right	A readonly property to get the right coordinate of the block, in Blox coordinates
RotationCenter	Contains the position of the rotation center of the block, in percentage (relative) coordinates. So, a value of (50, 50) will put the rotation center in the middle of the block
RotationStep	RotationStep contains the step, in degrees, used for block rotation while end-user is rotating the block. Default step is 5. You can set higher steps (30, 45, 90 for example) to only allow specific positions while rotating the block
Shadow	Shadow property contains information about

	block shadow. By using TTMSFNCBlockShadow object, you can set shadow visibility, position and color
Shape	Specifies the Shape of the block
Strings	Use Strings property to read/write the text displayed by the block
Stroke	Use the Stroke property to change the border of the element
TextCells	TextCells property holds the collection of text cells of the element
Top	Specifies the top coordinate of block, in Blox coordinates
VertAlign	VertAlign property specifies the vertical alignment for the block text
Width	Use the Height property to read or change height of the block
WordWrap	Use WordWrap property to specify if block text will be wrapped or not

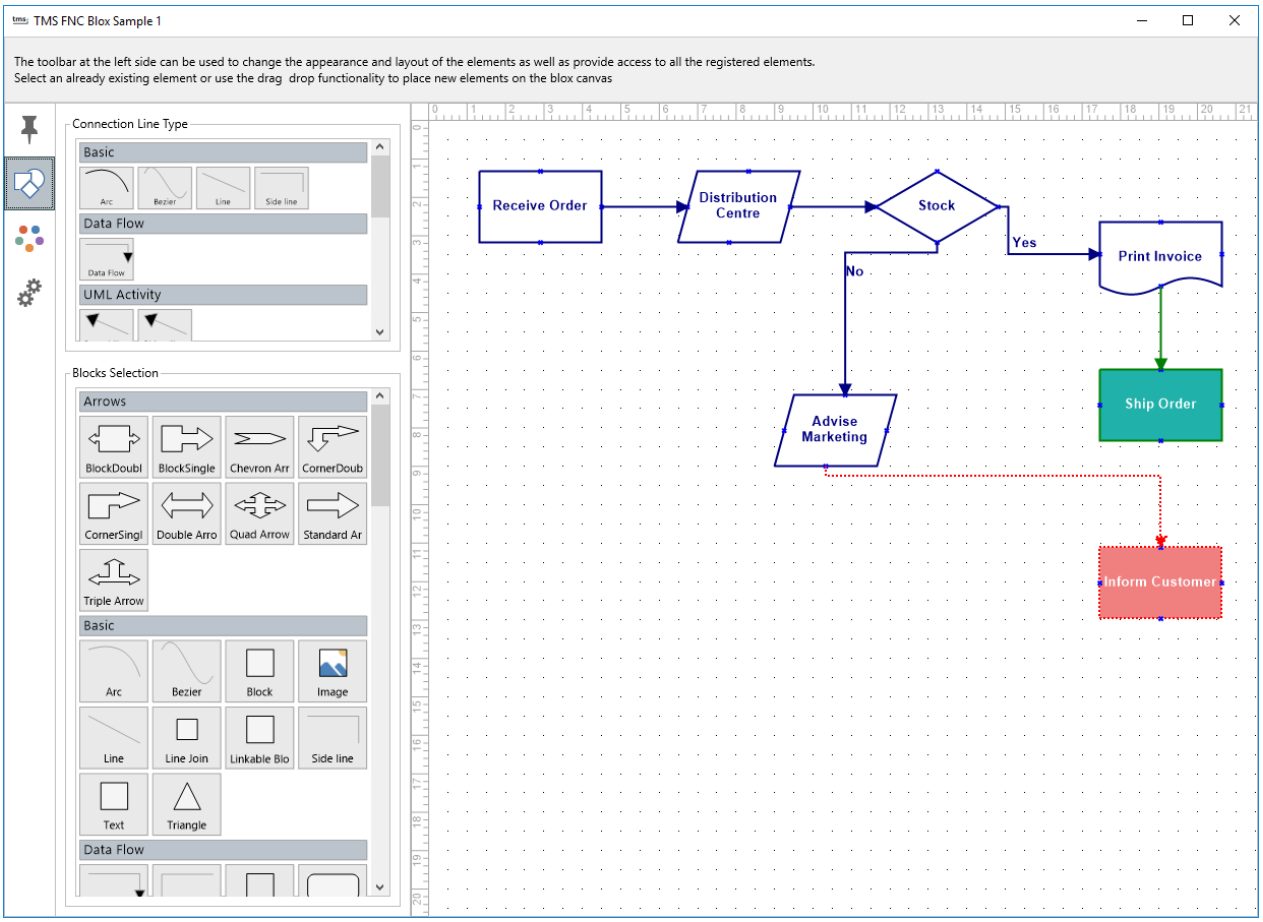
Properties for TTMSFNCBloxLine

Angle	The angle of the block in degrees
Handles	
Id	Contains the ID of the Blox element, as specified in the Blox element registering procedure (TTMSFNCBloxControl.RegisterElements method and TTMSFNCBloxControl.OnRegisterElements event-)
LineStyle	Use LineStyle to define which line will be drawn. You can switch from one line to another. Default is lsLine
LinkPoints	
Points	The Points property contains information about the line points. The Points property is used to draw the whole line: a line segment is drawn from first point to the second, then from the second to third, etc
RequiresConnections	When RequiresConnections is true, the Blox will require the line to be connected to a block. Both start and end points must be connected. The end-user will not able to insert, move or resize a line unless its start and end points are connected to a block
SourceArrow	SourceArrow property specifies the shape for the start point of line
SourceHandle	A readonly property that returns the source handle linked via the SourceLinkPoint property
SourceLinkPoint	SourceLinkPoint contains information about the link point associated with the start point of the line. Use SourceLinkPoint to read or specify an anchor for the line start point
Stroke	Use the Stroke property to change the border of the element
TargetArrow	TargetArrow property specifies the shape for

	the end point of line
TargetHandle	A readonly property that returns the target handle linked via the TargetLinkPoint property
TargetLinkPoint	TargetLinkPoint contains information about the link point associated with the end point of the line. Use TargetLinkPoint to read or specify an anchor for the line end point
TextCells	TextCells property holds the collection of text cells of the element

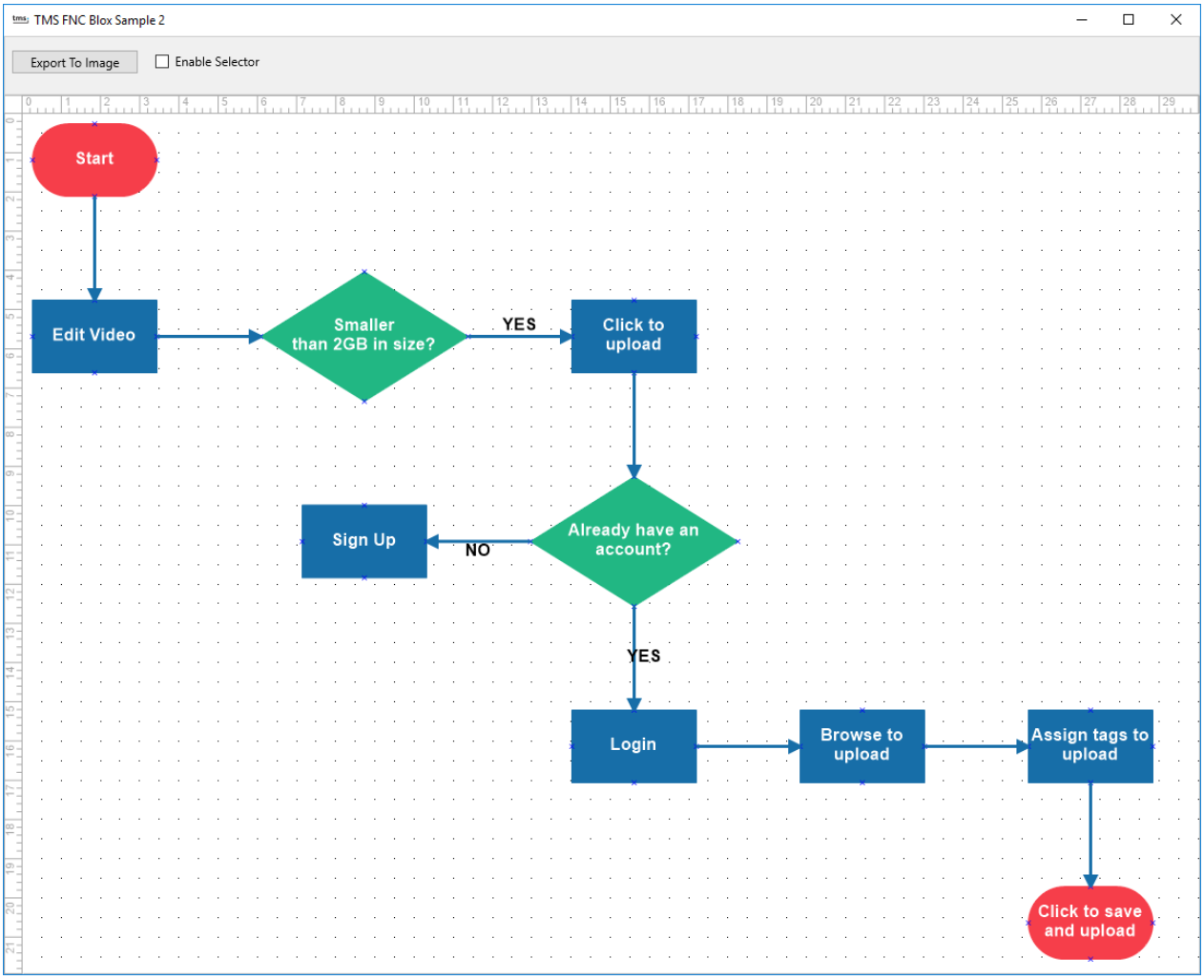
Demos

Sample 1



Sample 1 demonstrates how to create and connect the TTMSFNCBloxToolBar (available separately) to the TTMSFNCBloxControl as well as shows you how to programmatically setup a diagram.

Sample 2



Sample 2 demonstrates how to use the TTMSFNCBloxSelector and its drag & drop capabilities as well as exporting to an image. The diagram shown in the sample is also programmatically built up.