Diagrams for UML and Function Blocks drawn with OpenOffice or LibreOffice

Dr. Hartmut Schorrig
www.vishia.org
2024-01-26
# Table of Contents

1 Open/Libre Office for Graphical programming......................................................... 4

2 Join FBlock Diagrams and UML-Class Diagrams.................................................... 5

3 Approaches, basic considerations............................................................................. 6
   3.1 Question of sizes and grid snapping in diagrams............................................. 6
   3.2 Using figures with style sheets for elements.................................................... 10
   3.3 Connectors of LibreOffice for References between classes........................... 11
   3.4 Connect Points for more complex references.................................................. 12
   3.5 Diagrams with cross reference Xref................................................................. 13

4 Working flow creating your own diagrams............................................................. 14

5 Overview capabilities of the UFBgl......................................................................... 16
   5.1 All Kind of Elements with there Style Sheets.................................................. 16
   5.2 Function Block and class diagram thinking in one diagram............................... 18
   5.3 Show same FBlocks multiple times in different perspectives............................ 18
   5.4 More as one page for the FBlock or class diagram.......................................... 19
   5.5 Using events instead sample times in FBlock diagrams..................................... 20
   5.6 Storing the textual representation of UFBgl in IEC61499................................. 22
   5.7 Source code generation from the graphic......................................................... 23

6. Details..................................................................................................................... 24
   6.1 Data types.......................................................................................................... 24
      6.1.1 One letter for the base type:................................................................. 24
      6.1.2 Unspecified types:.................................................................................... 25
      6.1.3 Array data type specification................................................................. 26
      6.1.4 Container type specificatio....................................................................... 26
      6.1.5 Structured type on data flow................................................................. 27
      6.1.6 Data type forward and backward propagation......................................... 28
   6.2 Expressions inside the data flow........................................................................ 30
      6.2.1 Expression parts as input.......................................................................... 30
      6.2.2 Factor and constants on inputs............................................................... 31
      6.2.3 Any expression in FBexpr........................................................................ 31
      6.2.4 Output possibilities.................................................................................... 32
      6.2.5 FBexpr as data collect and access............................................................ 33
### Table of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>View 40%</td>
<td>7</td>
</tr>
<tr>
<td>Figure 2</td>
<td>View 100%</td>
<td>7</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Example for a Module Diagram</td>
<td>8</td>
</tr>
<tr>
<td>Figure 4</td>
<td>Text alignment in connection elements</td>
<td>10</td>
</tr>
<tr>
<td>Figure 5</td>
<td>block crossing reference</td>
<td>12</td>
</tr>
<tr>
<td>Figure 6</td>
<td>Using a connection point for aggregations</td>
<td>12</td>
</tr>
<tr>
<td>Figure 7</td>
<td>UMLdiagramXrefExample.png Cross Reference usage</td>
<td>13</td>
</tr>
<tr>
<td>Figure 8</td>
<td>UMLallConnections.png</td>
<td>16</td>
</tr>
<tr>
<td>Figure 9</td>
<td>OFB/DflowStructData1.png</td>
<td>27</td>
</tr>
<tr>
<td>Figure 10</td>
<td>OFB/DflowStructData1.png</td>
<td>27</td>
</tr>
</tbody>
</table>
One of the advantages of textual programming is: You can visit your program code with any desired editor, such as Notepad++, or VIM on Linux or just a powerful Integrated Development Environment. For development of course, compiler tool suites are necessary. But to discuss content, behavior, look what's happen you need only standard tools. For long time maintenance it means it may be sufficient only to have the source code itself, if maintenance actions can be done by parametrization (with given Operation and Monitoring tools), or for update the program you need only the compilation tools or possible use newer versions of compilation tools which are compatible.

If you use graphical programming, then the graphical sources can be viewed often only with the original tools which may be vendor specific, need licenses to use etc. Sometimes older source files cannot be opened with newer (currently in use) versions of the tools. It means only for view what is contained in your device you need a specific tool. Additional often code changes are sophisticated in the tool chain, needs specific knowledge (about set options etc.).

This may be one reason that textual programming is preferred, though for the graphical programming it was rumored also for more as 20 years, it would be replace the textual programming because of some advantages.

That's why graphical programming is the playground for some big tool providers, whereas different approaches are given with the tools which are not compatible. Whereas textual programming is also familiar for common software, sometimes Open Source.

The second reason to favor textual programming is: The sources are immediately comparable with simple text diff tools. And the third reason is: Tools are interchangeable, the source is always understandable as text source.

Now, to favor the graphical programming, this paper offers the idea and shows approaches related with usable software for content evaluation to use a common graphical draw tool for the graphical programming, which is usable for everybody without effort, which is compatible also with some other tools and which is enough powerful to use. For that LibreOffice and also OpenOffice was tested to draw the diagrams, and a translator to evaluate the content was written (just in progress). This concept is presented here.

Some basic ideas are:

- Use Style Sheets to designate semantic information to graphical blocks,
- Evaluate it reading information from the odg file, it is a simple zip file containing XML information
- Translate the content to other graphic formats for the specific tool or make the own code generation.

A second approach of this work is: For graphical programming the familiar idea to use Function Block Diagrams (FBD) to present functional content are combined with important features of the UML class diagrams. All in all the Function Blocks (FBlocks) are seen as instances of classes, which is self evident often for code implementation (in C++) but also in C where Object Oriented classes can be implement with struct data and the appropriate operations for this data. It means the FBlock Diagrams are advanced with UML features of class diagrams.

And also, UML class diagrams (without the FBlock idea) can be drawn and translated also with this approach.
The **Unified Modeling Language** (UML) was created in the beginning of the 1990\textsuperscript{th} based on different existing modeling approaches, firstly by Grady Booch, Ivar Jacobson and James Rumbaugh \cite{1}. Another contribution to UML comes from David Harel \cite{2} who had development state machine technology firstly introduced with his own tool "Statemate" and then applied to the UML tool Rhapsody (original from I-Logix, now IBM).

The focus of UML was also code generation for particular devices, but also the approach of commonly describing of systems which can be applied to particular software, with focus of Object Orientation.

In opposite, the technology for Function Block Diagrams (FBD) inclusively code generation for particular usual firstly automation devices was created already in the 1960\textsuperscript{th} with the IEC 61131 Norm for "Programmable Logic Controllers". It was also similar used for some other approaches such as LabVIEW \cite{3} or simulation tools. Especially Simulink from Mathworks \cite{4} is used here for some comparisons with the here shown technology. This tools has its basics in the 1980\textsuperscript{th}.

Both approaches, the UML and the FBD tools are designated as "model driven development". But there are not related. The FBD tools does not use diagrams from the UML, and it is usual not seen as "Object Oriented" and the UML seems not accept a diagram kind which is firstly for a particular software or device and not for a commonly described system.

Usual the code generation is familiar from the FBD tools. In UML code generation generates only the frames of the classes respectively instances, it is not so frequently used.

The FBD tools focus only to the functional aspect of a device or a software. The operation system and managing to properly run the software (organization of threads, hardware access etc.) is usual done by specific settings (for example the "hardware config" part of configuration for automation devices with the Siemens TIA portal). The system itself is hard coded given and does not need an elaborately description presentation.

In opposite, the UML focuses to the whole system. For example the operation system itself is a "component", which is presented with interactions etc. in the component diagram. Also some hardware components.

In this manner the here presented combination of the UML Class and the FBBlock diagram is only a part of a possible "UML 3.0". It does not replace all basics from UML, of course. It is only a contribution for this imagined UML 3.0.

How to name this combination of a FBBlock and Class Diagram ... Let's use the abbreviation FBcD. The "c" means either "class" or may be also "connection" for the UML like connections between FBBlocks as supplement to the known data flow for ordinary FBBlock Diagrams. A textual representation of the same content should be named FBcL as Function Block connection Language. The focus to the UML is not presented in this abbreviation, but UML is familiar and recognizable.

What else: The **event connection** between FBBlocks are also used here as important part. Events are familiar in UML for state machines. An Event connection is also used in FBBlock Diagrams with the standard IEC61499 \cite{5} for automation devices as a basically feature. Also in Simulink events are designated and used for "triggered subsystems" as well as for state machines. But events are familiar also in UML for State Charts, and should be familiar in Object Orientation.
3 Approaches, basic considerations

This chapter shows how capabilities of LibreOffice are used to draw diagrams.

3.1 Question of sizes and grid snapping in diagrams

Commercial tools for graphical programming have often not a proper answers to this question. Often sizes are scalable in any kind, as the user want to have. Grid snapping is sometimes supported or not, and, sometimes sophisticated algorithm are implemented which avoids lines through blocks and make instead mad ways around all blocks. LibreOffice is here more friendly, it let the user decide about the connection path. This may be only a marginalia.

Let's think about font sizes and grid, requirements:

- In a usual document a proper font size is 11 pt, as written also here. A smaller font (9 pt, 6 pt) is not suitable for reading because of the recognizability of the words and their contexts, it is only for read the package leaflet of medical products.

- A diagram should have place in a document on a A4 or size-B page (~ 18 cm text width). It means the size of a proper view is ~18 x 10..12 cm. Using a whole side in landscape orientation may have a size of 25 x 17 cm, but in landscape mode the document must be rotated only for this page, this is not suitable for reading a PDF document on the screen.

- A diagram has two tasks:
  a) Documentation
  b) Base for the software

For the approach b) the diagram may be well editable as a whole on a large screen, for example with resolution 2650 x 1200 pixel. To document this complex diagram it can be shown in landscape orientation in a document, or better: It should be reduced in size to fit on a normal page in portrait format. Details are then no longer legible, but important things and orientation should be shown in larger font. Then the overview can be explained and details can be shown as part from exact the same diagram in a higher resolution.

- A common and contradictory question for diagrams is: How comprehensive should it be. Should it contain only one block and some less aggregated ones? Or should it contain the whole truth of a module? The answer of this question depends on the available size for presentation. There should not be too less content.

The UML has the advantage that you can use more as one class diagrams to explain the same class in different contexts. That is a very great advantage and it should be usable also for some Function Block presentations! (Not yet in professional tools). This helps to decide how many content a diagram should contain.

- The readability of a word which is isolated of a sentence, an identifier of a block or line or such one is given also with a smaller font size than 11 pt, especially if it is present in bold font or maybe also in a non proportional font (as for programming language source code). Because in proportional fonts often important small characters such as “il” are to small and bad visible

- For positioning a proper grid size and the possibility of positioning with cursor keys (!) is essential. LibreOffice has the property that the step size for the cursor key is anytime 1 mm, independent of other settings. It's
There is a specific property of LibreOffice: The step width by moving with cursor keys is normally 1 mm. You can do fine adjusting in combination with the Alt-key, but this is too fine. If also a grid fine spacing with snap points of 1 mm is selected (a 5 mm grid with 5 fine divisions), then the placing is very proper. All elements are placed in a 1 mm grid, the 1 mm is enough fine for details and enough raw to simple snap in the grid points.

From that, the idea comes to have a standard size of small elements of 2 mm. The mid point is also in 1 mm grid snapping raster. You can have a near distance of lines of 1 mm, well obviously.

To show enough content in a diagram you may use an A3 paper in landscape orientation. On a larger monitor (2560 or 3280 pixel width) it is editable in entire page mode. The diagram has a width of ~40 cm.

The same content is presented here right side in original magnification. The font size of 6 pt for the most elements is just readable. It is Consolas bold. The type names of the classes are Arial 8 pt, the name of ClassA is Arial 14 pt. This is a 1:1 presentation, drawn in portrait A4 it is really 1/1 site width.

It means you can have an overview, but you don't see some details in the documentation.

Parts of the same diagram can be shown in original size, then all is readable.

You should place different approaches of the same module in more as one diagram. This is definitely supported by UML, and should also be usable for function block presentations. In
commercial tools such as Simulink it is not possible, but here it is.

As living example look on the following Class-Object-diagram:

**Figure 3: Example for a Module Diagram**

This diagram should be well readable in normal view of a pdf viewer. The font and size of the names is consolas 6 pt bold. The original draw area is the width of a A4 page. The pixel solution is 1351 x 480, results from a Zoom of 200 % on a 1980 pixel width monitor.

The diagram shows a coherence of different blocks to build a synchronized clock enable (ce) in a FPGA. You see two receiver (Rx) modules, which are combined with a third module, with equal light-brown colors. Its a selection of the active input. The output of this third module has the same interface type `RxClkSync.Inp_ifc` as the module in the mid. Both are selected from the red right module. With less explanations the coherence should be understandable.
3.2 Using figures with style sheets for elements

The original UML class diagram has the following approach:

- A class is a rectangle box containing the type name of the class.
- Some data or operations may be named inside the class box, it does not need to be completely.
- All relations to other classes are shown with references to the other classes. This references are often non directed, but sometimes only in a specific direction marked with a little arrow on end. This relations are associations, aggregations, compositions, inheritance, dependencies.

The last point is not mapped to the languages which presents the software which is presented by the UML diagrams. Because: The fact that a class has an aggregation to any other class is a property of this class, not a property of relations between this classes. It is exactly the same as for data. A data element has a type, and a reference has also a type, the type of the referenced class. It least the name of a reference is only a property of the class, it is not a property of the relation between the classes.

For that reason the shown relations to other classes are assigned to the class itself. They are existing also if there is no connection. Then, of course in the implementation it's a null or nil pointer. Or it is just not shown here in this diagram, instead shown in another diagram, but nevertheless shown as element of the class. Look on the images on the page before. There are some not connected aggregations, which may have a meaning on explanation to the diagram.

The elements for connections are named pin. This is similar as in Function Block Diagrams where the data connections are presented also as pins.

In UML a port is known. This is also a pin, see Error: Reference source not found page Error: Reference source not found.

This pins are simple small figures with a fixed size, known from UML as the diamond (filled / non filled) for Composition and Aggregation, or they are simple rectangles. The elements contains a text, which is the identifier for the element or also the type specification. The text is written outside on the element itself by using the Libre Office property, that a text can

*Figure 4: Text alignment in connection elements*
3.2 Using figures with style sheets for elements

Exceed the bounds of the element's graphic. More as that, the left or right margin of the text is set to a value greater or equal the size of the element, and in this kind the text is written outside, left or right next to the element.

Now, an element or more precise a connection to the class is shown as this small figure. But not the figure itself characteristics the element for code generation, instead the associated style sheet. The look of the figure can be changed, should not be changed, it is for human. But the style sheet marks the semantic of the figure, the kind of the element. The settings in the style sheet, especially the size of the text, can be overridden by direct formatting. This is for larger fonts explained in the chapter before and shown in Figure 1: View 40% page 7. The style sheet should not be changed by the user. It is defined for this kind of diagrams.

Style sheets are a proven concept for text writing. The direct formatting approach can be also used to a style sheet formatting approach, and both can be combined. A style sheet allows change a formatting style for all designated elements (paragraphs, parts of text etc.) to achieve a uniform presentation. It is an advantage that is often not enough known. That are common statements.

3.3 Connectors of LibreOffice for References between classes

The connectors as known from LibreOffice are the proper possibility to connect blocks, which are classes or objects. The connection can be done for the class itself, or for one one of the elements.

You can use connectors with orthogonal lines, or straight or curve connectors as if you want.

LibreOffice assigns four connection points ("glue points") to each element by itself. This is sufficient for elements of the class. It is very simple to connect for example the end point of a diamond of an aggregation with the mid of a port as destination of the aggregation, or also with any other class if the whole class is referenced.

For the larger class block with maybe more connections on different positions you can add some more glue points.

Using connectors between elements in your graphic, the connection remains stable if you move some blocks. You may adjust the inflection points (more precise the mid points between inflection). Some commercial tools such as Simulink try to adjust connections between blocks by itself by sophisticated algorithm, which should avoid lines crossing blocks, and make instead mad ways around all blocks only to avoid crossing a free but reserved area for a name of a block. LibreOffice is here more friendly, it does nothing by itself, only move the connection as necessary, and let the user decide about the outfit of the connection path.

A connector as reference between blocks should have also a Style Sheet. If the connected elements are well dedicated by Style Sheets, you can use the ofRef style for all connectors. It produces a small arrow on the end, and a line width of 0.2 mm, nor more.

But there is also a possibility using connectors as in UML. The connectors have especially the start arrow outfit as in UML necessary (diamond for aggregation). Then you can use for the connected elements the common style ofPinLeft or ofPinRight which does not specify the kind of the element. The connector specifies it. That is the originally approach of UML, also possible here (but not recommended). Both are supported by code generation.
3.4 Connect Points for more complex references

Libre Office seems to be have the disadvantage that additional inflection points on orthogonal connectors are not possible. Look on the example right side. The connection from \texttt{aggr2} to \texttt{port2} through ClassF is not nice.

The solution is shown also image. From \texttt{aggr1} to \texttt{port1} two connection lines are concatenated. The first line is of type (style) \texttt{ofrConnPoint}, its without arrow on end. Both lines together appears as one line, with proper inflection points.

Another question is: Having aggregations or other references with one destination and more sources. In UML often there are drawn parallel. But it is more consequentially to use a connection point as it is known from any electrical circuit scheme and also from Function Block Diagrams for data flow. The difference is only: Data flow and electrical schemes has one source and more destination. An aggregation has one destination and can have more sources. The reference line to the connection point is either a simple \texttt{ofRef} which has an arrow on its end, or it is the same as in the image above for concatenation of reference lines, with style or type \texttt{ofrConnPoint}. 

![Figure 5: block crossing reference](image)

![Figure 6: Using a connection point for aggregations](image)
3.5 Diagrams with cross reference Xref

Image Cross Reference usage

The cross reference or usual nominated as Xref is an often used symbol to replace too much lines in one graphic, or also to make connections to several sheets of a graphic. The last one should not be in focus here, because on graphic sheet presents one aspect, spread one diagram over several sheets is not familiar for UML or also Function Block Diagrams.

You may use a Xref for signals and connections, which are well known from name, and which have basically connection meanings (such as “reset”) and may be connected to more as one block.

- The figure for the Xref can have any form, but should use the given form (copy it from template). The Style Sheet should be either ofbXrefLeft or ofbXrefRight, whereby the difference is only the text alignment to left or right.

- The name in the Xref symbol should be a mnemonic name for the functionality, valid for this diagram. Here it is a combination of the type of the port and part of name, maybe proper.

- The line from a block to the Xref should be the same type (here a simple ofRef) as without Xref.

- The line from the Xref to the block should have usual the same type, but this is not evaluated. Because the type of connection can be also composition or association here, the type for the association is used here, it is not specified to the aggregation or composition with the filled or non filled diamond.

You can use Xref connections for all line types. The evaluation of the graphic builds a list for all Xref by name per sheet, and checks the connections.
4 Working flow creating your own diagrams

First you should load and open the template file from

https://www.vishia.org/SwEng/oofb.wrk/src/UML_FB_DiagramTemplates/odg/UML_FB_DiagramsTemplate.otg

To create a new empty UML class or Function Block diagram you should save this template file under your specific location/name.odg. You should delete the content, the style sheets are not deleted.

Reopen the template file, you need it to copy figures and elements from.

If you have your own file with content but maybe an older version of style sheets, you may copy the style sheets immediately with zip: The odg or otg file is a zip file format. Add the extension .zip and unzip it (simple us the Total Commander). It contains a styles.xml. Replace the styles.xml in your own file (with zip extension). Remove the zip extension and reopen it in Libre Office. It should work. Do not forget to make a backup copy. This is a non documented way, but it seems to be stable since many years. It works also for OpenOffice in different versions.

Look for Grid and Snap

- Open "Tools - Options", select "Libre Office Draw" and then first "General". Look for the measurement unit, it should be "cm".
- Then open "Libre Office Draw" and "Grid", look for the proper grid settings (recommended 0.5 cm and 5 Subdivisions because the natural cursor step width is 1 mm. Select "Snap to grid". This is strong recommended, because you have a lot of work for unsnapped blocks and some small inflection points in orthogonal reference lines.

If you have copied from the template, it should be proper.

Create a class or function block:

- Create a simple rectangle in your diagram and assign the style sheet ofnClass. The it gets the yellow color with brown border. Alternatively you can copy a class block from the template.
- Create a simple rectangle and write first your class name into it (press F2 to write text in a selected rectangle). The assign the style sheet ofnClassTypeName to it. Now move the rectangle into the class box, usual (not necessary, but recommended) to the top right border. You should not place the name exact in the mid, it makes a little bit trouble by selecting the correct glue point for the class rectangle.

Alternatively you can copy the rectangle from the template.

- Maybe write some data or operations into your class block in the same kind, either by copying from the template, or also by creating simple rectangles and assign the style.

Copy connection elements

- Then you may copy connections (aggregations etc.). For this you should use the template, copy the correct element in your diagram. On paste it lands on exact the same position as in the template, its on the top spread of the page. You can use the cursor keys to shift it to your destination firstly, so long it is selected. Sometimes the landing position is inside any other stuff, this is a little bit confusing. Unfortunately Libre Office does not paste a figure on the cursor position (as other tools do). It would be more proper.
- You can copy more connection points
from the same type also from other ones in your diagram of course, it is usual faster.

- On copying and moving the figures the landing position should be any time in the 1 mm-Grid. Sometimes it may be wrong, you see it on small inflection points and obviously misplaced positions. Then you can press F4, correct the position to even mm. If you have activating snapping, all will be proper after such an adjustment (till a next non obviously positioning which may be also caused by accidentally size changes).

**Group and ungroup**

It may be seen as recommended to group associated elements, the FBBlock frame, all pins and names. Then it is more simple to move a FBBlock. Group is necessary for evaluation of coherent elements in current versions, but should be no more necessary in future. Then grouping is only optional for your work with the graphic.

This is basically and your daily work.

- You can catch all content of a class block inclusively the connection elements with selecting with left pressed mouse. Then you should select "Shape-Group-Group" from the menu. It should be set to ctrl-G per default.

- After grouping you can move your class block simple with the mouse or by cursor keys as a whole. All Elements are moved together and are stable in the class block.

  The grouping is also recommended and necessary for content evaluation. You should anyway have a grouped situation.

- But in the grouping state you cannot connect the elements of the class with a connector. Only the glue points of the group itself are accessible, and that is not desired.

It means before connecting, and also before shifting some elements you should ungroup. This is in menu "Shape-Group-Group". You should assign the hot key strg-sh-G to ungroup to have a fast work flow. Sometimes also strg-U may be used. Originally this is not assigned. Use menu "tools - customize" and then keyboard. Search "ungroup" and assign the desired key.

You need very often Group and Ungroup.

**Small problems with movement**

The elements have a height of 2 mm and often only a size of 2 x 2 mm. If you select it, Libre Office shows drag points to change the size, but because the size is not changeable, also a "non possible" symbol. The space for movement is small in the mid of this points. 2 x 2 mm is the smallest size where movement is possible on a 1920-pixel screen with full size width. This is a little bit stupid.

But you can also move with cursor keys.

Using a higher zoom factor (200 % is recommended) ameliorates this situation. Usual you don't need to see your page margins.

**Hint: Bring to Back / bring to Front**

The rectangle of a class should have a transparency. Then you see also elements which are arranged below (in the back) in relation to the class rectangle. But to work with, the inner elements should be in front and the class rectangle should be in back. Use the menu entry "Shape - Arrange" or the context menu with "Arrange" to adjust it.
5 Overview capabilities of the UFBgl

What do the diagram contents mean?

This chapter should discuss some presentations in the ObjectOriented Function Block diagrams in relation to the UML standards and some quasi Standards used for Function Block. Function block representation and UML should not be a contradiction. It should be thought together for the future.

5.1 All Kind of Elements with their Style Sheets

The next image shows all given template elements. It is the content of the file https://www.vishia.org/SwEng/oofb.wrk/src/UML_FB_DiagramTemplates/odg/UML_FB_DiagramsTemplate.otg

![Figure 8: UMLallConnections.png](https://www.vishia.org/SwEng/oofb.wrk/src/UML_FB_DiagramTemplates/odg/UML_FB_DiagramsTemplate.png)

Right side you see some style sheets. You can use this image (given in the file UML_FB_DiagramsTemplate.otg) to pick an element, copy it to clipboard and insert it in your graphic. The style sheets are copied by opening this file and save it with your name. Unfortunately LibreOffice does not allow loading style sheets from another given odg document, only by copying the original one (see also https://ask.libreoffice.org/t/how-can-i-import-styles-from-other-draw-documents/8834).

But you can copy the internal style.xml file from the UML_FB_DiagramsTemplate.otg zip archive. This is a simple, proven workflow that has not been recommended as often, but it works:

* Copy the original UML_FB_DiagramsTemplate.otg file to UML_FB_DiagramsTemplate.otg.zip
* Open the zip file by a unzip tool.
* Copy the internal style.xml for your own.
* Make a backup from your own *.odg file only to have it for trouble.
* Rename your own *.odg file to *.odg.zip and open it with a zip tool.

* Replace the internal style.xml with the style.xml from the template.

* Rename your own *.odg.zip file back to *.odg

* Check if all is proper. It should be.

For dealing with zip content using the Total Commander is a good decision.

The class in the mid with **name: ClassType** contains all connection elements for the concept described in 3.2 Using figures with style sheets for elements page 10. The identifier of the style sheet is here used also as name.

The class left **ClassType name** contains simple connection elements of the base style **ofPinRight** and **ofPinLeft**, but using connections with the specific type. Their style names are shown here as pin names.

The type name of a class is marked with the style **ofnClassTypeName**. A class can have also an instance name, then it is an Object or a Function Block. A super class cannot have a name, also not a composited class. Because there are defined by its relation to the using class. A Function Block presents a class which is instantiated by a here usual not shown main class of a module as composite. Then its name is written either in the element **ofnClassTypeName** as first, then with colon, or after the class type name. This writing style is also possible for all names of the elements. The type can be written their also, shown only for the aggregation left bottom **ofRefAggr: SuperClassX**. The type of a connection (a reference) may be sometime interesting. In UML a reference should refer always the class which is the reference type. But in this here used combined class-object-Diagram (objects are Function Blocks) sometimes the reference goes to the used instance with a derived type, and the reference type should be shown.

The style **ofnClassObjName** is the other possibility to show the name of a Function Block or just class object. Then only the name is written here, parallel to the **ofnClassTypeName** which may/should only contain the class type name. In the left **ClassTypeB name2** both is used.

The internal data of a class can be shown, as usual in UML, with the style **ofnData**. The designation about private, public, protected should be written with a first character - + # as usual in UML. Writing the type of the data is recommended. The operations can be written with their argument names, if it is more informational. The operation itself, its body, should be define anyway in a programming code and not with a diagram. The association between the shown operation in a diagram and the real operation is only for documentation, should not be formalistic.

*For the documentation blocks the style** ofbComment should be assigned. A requirement is presented also usual in UML with a short identifier. It is written in a ofbRequirement rectangle block. The connector between ofbComment and ofbRequirement has the style ofRefDocu. If you copy this connectors from the template, you get also the style reference.*

This diagram contains also data and event flow. This is described in chapter Error: Reference source not found.
5.2 Function Block and class diagram thinking in one diagram

One of the basic ideas of this approach is just, join UML thinking and FBlock thinking. UML presents in class diagrams relations between classes. A class is an abstraction of implementation. The implementation uses instances (of classes).

In opposite Function Block Diagrams works only with instances. A "class" is an unused word in this way of thinking. But using a Function Block type from a Library is "instantiation of a class", the library block type is the class.

![OrthBandpassFilter.odg.png](OrthBandpassFilter.odg.png)

The Figure 1 OrthBandpassFilter.odg.png shows primary a Function Block Diagram (FBlock diagram). The green parts are the input and output pins of the module. Some FBlocks presents expressions, these are with dashed lines. The other FBlocks presents instances (each three from the same type) which are connected with data flow.

But from the Bandpass FBlock to the BpParam FBlock there are an aggregation. That shows two things:

a) There is an aggregation from the type (class) Bandpass to the class BpParam. This is a relation of a class diagram.

b) The aggregation from h1 is initialized to refer h1p, as also h2 refers h2p and h3 refers h3p. This is a property of the FBlock instances.

The relation shown with the aggregation can be seen also as data flow, but in the opposite direction. Initially the address of the h1p FBlock is provided to the h1 FBlock, to refer it, adequate for h2 and h3. Hence, the diagram contains information about class (or type) relations as class diagram and information about instance relations as Function Block Diagram with data flow.

5.3 Show same FBlocks multiple times in different perspectives

There is an interesting and important principle using in UML class diagrams. A class can be presented in more as one perspective in several diagrams, and also more as one time in one diagram. The class is presented by its name, it is able to found
in the repository of the UML data. The diagrams plays only the role of presentation of the class with its properties just in several perspective.

In opposite, traditional Function Block Diagrams shows one FBlock as one instance. Often the FBlock does not need a specific name, then it is automatically named.

This approach uses the principle, showing also a FBlock in several perspectives, in opposite to traditional FBlock diagrams, but similar as UML. It means, on FBlock as one instance can be shown more as one time in the same diagram or in several diagrams related to the same module. The FBlock is dedicated by its name. Drawing a second FBlock with the same name is the same instance.

This principle enables showing complex large FBlocks in several perspectives. Different connections are shown on different places, also the same connection can be shown more as one. For example inputs of one functionality of a FBlock are shown on one page with focus of that input signals, other input signals are shown on a second page, and the output connections and processing are shown on a third one. Also the connections are unique dedicated by its pin name on the named FBlock with the named type. This offers more overview. The dispersion of one FBlock connectivity in several views may be seen as disadvantage, it becomes confusing. But notice, there are search operations and evaluations of the graphic which gives an overview to find all locations of the same FBlock instance. The idea is newly for FBlock diagrams, look for its advantage.

Now this idea is also usable for the class description idea: Any FBlock instance is dedicated by its type. The type is the class type. All occurrences of the same type of Flocks are properties of its class. Also FBlock with only the type name, without instance name presents the class properties. The sum of all is the property. This is true for the type of a c FBlock which is a class as also for the connectivity of an instance of a FBlock in several graphic presentations.

5.4 More as one page for the FBlock or class diagram

The chapter above 5.2 Function Block and class diagram thinking in one diagram allows simple to disperse a diagram over a lot of pages (as necessary) because the same FBlock instance can be shown for example with its input signal wiring, and on another page with its output signals, or group of signals. This allows formally descriptions more near to explanations. One Image (one side) should present one aspect. Which – this is document- or explanation oriented. Data flow connections can also be joined by Xref blocks.

OrthBandpassFilter: Main page

Figure 2 ofbTitle-1.png

Any page need have a title block, of style ofbTitle. It contains the name of the module and a short text what it contains.

The pages can contain several modules. The association of module diagrams to files.odg is an important topic. If you have related modules, you can store all it in one file. On the other hand it is possible to have more as one file for one module. This should only be regarded while translation the module.
5.5 Using events instead sample times in FBBlock diagrams

Usual for FBBlock diagrams sample times are familiar. It follows from the basic approach that the FBBlock connections are executed cyclically. That is so in some applications, for example industrial automation control. But sometimes events also play a role. In ordinary automation control often this is regarded by polling (quest of input signals) in a cyclically kind, because their basic operation system supports firstly cycles. The importance of events was often not the focus when such systems were created, although events were common and well-known in other areas of software technology. For example Simulink works basically with “sample times” but has specific opportunities (“triggered subsystem”) to deal with events.

Well, the assignment of signals and FBBlocks to events includes working with sampling times, but also triggered operations. More as that, the event flow presents better as a data flow the execution order of FBBlocks. Only using the data flow sometimes it is not well as necessary predicted. If the execution order is internal information (the user does not see it unless you study the generated source code), then uncertainties remain.

The UFBgl tool allows the automatic derivation of the event flow from the data connections (data flow). The event flow is shown in the textual representation of the graphic and can be viewed or analyzed. It is also possible to determine a specific event connection in the graphic by the user.

Follow another example:

The Figure 3 OFB/DataFlowPID4.png is an example primary as Function Block diagram with a data flow. The event flow shown in gray is not necessary to be drawn. Here it is only shown in gray what is automatically generated. But the event pins should be determined as shown (drawn black). With the given event pins the data are related to the events, instead to “sample times”. Here the x ist related to step, and the w to stepslow. The reference value w comes from another sample time or just with another event. The data flow from x to the output yCtrl is given, hence yCtrl is related to the step event chain and it is delivered with the stepO output event. The value stored in the w1 variable is a “state value” set with the stepSlow event and only used, similar as after a “Rate Transition” in Simulink.
But this image has also an Aggregation from the PID controller FBlock to its Parameter FBlock. This is UML. In Runtime, the address of the parameter instance is delivered to the ctrl: PID one time on initializing the system. It means that is a data flow from ctrlp_ Param_PID to ctrl: PID revers to the aggregation line.

The green blocks of style ofbMdlPins are responsible to determine the module pins from/to outer or just the type of the module. Each ofbMdlPins block is responsible to associate event-data relations (as also familiar in IEC61499 diagrams), but additionally the update pin is also associated here:

It means that the input variable x is bind to the input event step. It presents the step() operation (should be called cyclically in the step or sample time). Because the x is forwarded by data flow to the ctrl: PID, also the event step is forwarded. Due to the interface definition of the PID type the input dwx is associated to the PID event input step. Hence the data flow x → ctrl.dwx determines also an event flow from step → ctrl.step.

The role of “update” comes from the mealy and moore automate thinking for logic and it is also familiar in numeric solutions for control: All values are first prepared. Preparation uses always the values from the step time before (or in binary logic preparation of D inputs of Flipflops uses only values of the Q outputs of the clock cycle before). That is the ordinary role of the step event.

The update event now realizes the switch of all state values (or clock for Q in Flipflop logic) from the old to the current step to use for the next step. In a sample or step time of a controlling logic first all modules executes the prepare event which is here named step. If all parts have been prepared, then the update comes. This assures exactly working for solutions of differential equations and typically for controller theory, it is the Euler principle for numerical integration.

A FBlock can also propagate output values with the prepare event, it depends from the functionality. In Simulink as similar solution an input of an S-Function can be designated as ssSetInputPortDirectFeedThrough(port,1) if it influences an output or not (set to 0, default).

In this example shown the output y.ctrl is set newly with the ctrl.upd event. Hence an event connection between ctrl.upd and upd of the module accompanies the data flow from ctrl.y to the modules yCtrl output. The relation between step, stepO, upd, updO in the PID FBlock type is clarified by the class definition of PID.

Next you see a code snippet of the textual representation of this module in IEC61499, see next chapter:

```
FUNCTION_BLOCK CtrlExample
  EVENT_INPUT
    param WITH Td, Tn, Tsd, kP;
    run;
    stslow WITH w;
    ...
  END_EVENT
  EVENT_OUTPUT
    stepO WITH yCtrl;
    ...
  END_EVENT
  VAR_INPUT
    Td : REAL;
    Tn : REAL;
    ...
  VAR_OUTPUT
    yCtrl : REAL;
  END_VAR
  FBS
    ctrl : PIDf_Ctrl_emC;
    ctrlp : Param_PID;
    w1 : Expr_FBUMLgl( expr:='+';;' );
    wxd : Expr_FBUMLgl( expr:='-+';;' );
    yCtrl1 : Expr_FBUMLgl( expr:='+; ... 
  END_FBS
  EVENT_CONNECTIONS
    run TO ctrlp.run;
    stslow TO w1.prep;
    updslow TO w1.upd;
    step TO wxd.prep;
  END_CONNECTIONS
  DATA_CONNECTIONS
    Td TO ctrlp.Td; (*dtype: F *)
    Tn TO ctrlp.Tn; (*dtype: F *)
```
5.6 Storing the textual representation of UFBgl in IEC61499

It is interesting and promising that the widely proven FBlock programming in the IEC61131 standard for industrial automation control (tools such as Siemens Simatic FBD in TIA-Portal or Beckhoff Codesys) has been further developed to the IEC61499 standard. This development was started in ~2006, Also Siemens was one of the driver in that time. The IEC61131 is used since many years for automation programming. The IEC61499 is standardized and used, but not from the global meaningful players, they only monitors this development. The reason (in my mind and experience) is not disadvantages of IEC61499, it is more a too widely usage, supporting and maintenance of the long term existing IEC61131.

The IEC61499 has introduced an event coupling between function blocks. This determines the stepping order better than the ordinary net lists in IEC61131, but it allows also to distribute the implementation of one Function Block Diagram over several automation stations. Event connections between distant stations forces automatically network communication implementation and assures the correct order of execution in the dispersed station, without additional effort. That’s the advantage for automation programming. But the more universal character of event coupling inclusively state machine thinking can also basically used for embedded control programming.

CPU) defines a statement order. Different event chains are related to operations, which can be called either cyclically (for step time driven thinks) or also from the state behavior or independent for example on user accesses.

But the drawing of the event connections in a IEC61499 diagram is an additional effort. The image shows an example with event coupling for simple data relations with the graphical edition tool 4diac. In most cases an event flow (chain) is also determined by the data flow. Evaluation of the data flow results in an event connection, which should not be drawn manually. It is automatically detected during the evaluation of the graphic, and stored in the data model. Only if dedicated event relations are necessary, the events should be drawn in graphic.

The IEC61499 standard is used to store the content of UFBgl diagrams in textual form. This allows also a proper comparability if details in the diagrams are changed. That is a high importance to use this tooling in the development of software, a proper traceability of changes is necessary. With pure graphics, this is often not properly supported, one of the reasons for the still widespread use of textual programming.

It is also possible to read this stored IEC61499 textual files for processing for sub modules, and for code generations, as well as reading IEC61499 fbd files from other tools to merge here.

Figure 4 4diacTestcg_Fork1.png

A chain of events in the same implementation platform (same thread in a
5.7 Source code generation from the graphic

As is usual with some FBBlock graphics, code generation from the graphic is a prerequisite for being able to work productively with it. This chapter should only give an overview. Refer for more opportunities in chapter ToDo.

The evaluation of the graphic is done with a Java command line process as (shortened):
```
java -cp tools/vishiaBase.jar;
    ... tools/vishiaFBcL.jar
    ... org.vishia.fbcl.Ufbconv
    ... src/UFBglExmpl/odg/OrthBandpassFilter.odg
```

This reads the graphic, writes anyway an IEC61499 fbd file, and writes here C-language header and implementing code.

The graphic is shown (as part, one page) in Figure 1 OrthBandpassFilter.odg.png. The generated code looks like (shortened):
```
/**Generated by org.vishia.fbcl. made by ...
 ifndef HGUARD_OrthBandpassFilter
 define HGUARD_OrthBandpassFilter
 include <emC\Ctrl\OrthBandpass_Ctrl_emC.h>

typedef struct OrthBandpassFilter_T {
    struct { // Locale struct for all din
        float x; // OrthBandpassFilter.x
        float x2;
        float fq;
    } din;

    struct { // Locale struct for all dout
        bool initOk;
        ...
    } dout;

    float_complex xdab; // Expression xdab

    OrthBandpassF_Ctrl1_emC_s h1; // h1
    Param_OrthBandpassF_Ctrl1_emC_s h1p; // h1p
    OrthBandpassF_Ctrl1_emC_s h2; // h2
    ...
} OrthBandpassFilter_s;

void step_OrthBandpassFilter ();
void upd_OrthBandpassFilter ();
...#endif
```

The implementation file is generated as:
```
/**Operation step(...)*/
void step_OrthBandpassFilter ( OrthBandpassFilter_s* thiz , float x, float x2 ) {
    // --> x1.prep otx:evChainExprSetvar
    float_complex x1;
    x1.re = x; // Y D otx:evChainExprSetvar
    x1.im = 0; // Y D otx:evChainExprSetvar
    thiz->xdab.re = ( x1.re - ( thiz->h1.ya ...
    thiz->xdab.im = ( x1.im - ( thiz->h1.yabz.im
        + thiz->h3.yabz.im));
    step_OrthBandpassF_Ctrl1_emC(&thiz->h1,
        thiz->xdab);
    ...
```

There are some stuff which is regarded beside the event flow and hence the execution order. The types of all elements are forward and backward propagated. For the here used complex data types the operations are duplicated respectively specific functions are created, and so on.

The code generation is controlled by textual template files using the java class OutTextPreparer, see

Any user can proved its own templates for code generation, can copy the originals and modify, or can write its own template for other languages or only specific style guides. For pure C language an object oriented style is used of course to represent the instances of classes. classes are presented by struct {} with its associated operations with a thiz reference to the own struct. This can be encapsulated also by C++.
6. Details

6.1 Data types

In the Error: Reference source not found the input \( x : F \) is designated as float input with the letter \( F \). This is very space-saving but still obvious. Other tools sometimes have only a “Pin dialog” where the type can be selected and can optional show the type in the graphic, but then all types destroying the overview. The idea only using one character should be seen as proper, the number of types used are not too much. This is for the standard usual numeric types. The type of aggregations are determined by the destination class. A type name can be given additionally if necessary.

The problem on numeric and basic types is: There are a lot of designations in different programming languages and usages, but they are similar. A second approach is: Also regard non full deterministic types.

6.1.1 One letter for the base type:

- **D F J I S B** that are the standard numeric types which are also known with this same char in Java as return value of `java.lang.Class.getName()` for the primitive types `double`, `float`, `long` (64 bit), `int` (32 bit), `short` (16 bit) and `byte` (8 bit). They have its adequate in C++ with `int64_t`, `int32_t`, `int16_t` and `int8_t` for the integers. In IEC61499 they are named `LREAL`, `REAL`, `LINT`, `DINT`, `INT`, `SINT`.

- **Q U W V** are the unsigned types in C++ `uint64_t`, `uint32_t`, `uint16_t` and `uint8_t`. In IEC61499 they are named `ULINT`, `UDINT`, `UINT`, `USINT`. In Java there is not a counterpart, the larger signed types should be used. The used characters should have their mnemonic in “Quad word”, “Unsigned” instead \( I = \text{int32} \), “Word” usual in some systems for 16 bit and \( V \), it is near \( W \).

- **q u w v** are the counterparts of unsigned,

- **d f j i s** That are the complex types as counterpart to the real types. Complex types are fundamentally for numeric solutions, but they are not standardized in any language. General this types are structured types. For IEC61499 code generation they are named `CLREAL`, `CREAL`, `CLINT`, `CDINT`, `CINT`.

- **C c** is for one character and a String. Unfortunately the letter s or S is already used for “short” and T or t for “Time”. Whether a character has 8 or 16 bit (ASCII, UTF8, UTF16) is clarified on implementing level.

- **Z** is for boolean, the same as in Java `Class.getName()`. What is a boolean, it should be clarified. How is a boolean presented in machine level: This is not a problem of the graphic, depends on implementing stuff. A boolean may be also possible to represent only by one bit in a bitfield. In IEC61499 it is named `BOOL`.

- **T** is for a current time (relative) due to the usage in IEC61499 and IEC61131 as `TIME`. How many milli or nanoseconds is represented by one step, it should be clarified on implementing level.

- **t** is an absolute time stamp adequate to `DATE_AND_TIME` in IEC61499 / 61131. The
format of the absolute time stamp should be clarified for the implementation. Often it is the seconds after Jan 1th, 1970 (as in UNIX), or better seconds and nanoseconds after a dedicated base year. It is important that it is a continues value of seconds.

- **a h** is a value of the date only, the day, and the time of day or the question which hour. As mnemonic. It is also implementing specific how it is presented in machine code. It is supported also as continues value. For the human interface it is always processable as human readable format, which can also regard time zones etc or country specific presentations. This stuff should not be mixed in a core application.

### 6.1.2 Unspecified types:

Some FBtype uses unspecified types, because they are available for more or all numeric types, or the type is checked and used really on runtime. In C/++ this is often designated as `void*` also as pointer to basic numeric types. In Java there is the `Object` class as common representation of all types. But the main approach is: The type should be specified by forward or backward declaration in the graphic model by data connections.

- **N** presents any numeric type. This is formally also an unsigned type, whereby using unsigned for numerics is sometimes a prone of error. It is compatible to `DFJISBQUWV`

- **n** presents a complex numeric type, compatible to `DFJIS`

- **M** is any numeric presentation, not complex one and not bit values. It is `NT`

- **E** is a non referenced type.

- **L** is a referenced type. In IEC61499 and 61131 it is named `ANY_DERIVED` and distinguished from the `ANY_ELEMENTARY`. It does mean a structured type or also an enumeration defined there with `TYPE ... END TYPE`. All of them can be present by an aggregation to a FBBlock which contains the appropriate values. The `L` follows the `Class.getName()` in Java for the `Object` type. It is especially any reference type to a class type (a pointer) similar as the `void*` in C/++.

- **A** is a really unspecified type. This is also if the type specifier is not given.

IEC61499 and also the automation system programming language IEC61131 knows the following definition of types:

```plaintext
<table>
<thead>
<tr>
<th>ANY</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANY_DERIVED</td>
<td>L</td>
</tr>
<tr>
<td>ANY_ELEMENTARY</td>
<td>E</td>
</tr>
<tr>
<td>ANY_MAGNITUDE</td>
<td>M</td>
</tr>
<tr>
<td>ANY_NUM</td>
<td>N</td>
</tr>
<tr>
<td></td>
<td>ANY_REAL</td>
</tr>
<tr>
<td></td>
<td>LREAL</td>
</tr>
<tr>
<td></td>
<td>REAL</td>
</tr>
<tr>
<td></td>
<td>ANY_INT</td>
</tr>
<tr>
<td></td>
<td>LINT, DINT, INT, SINT</td>
</tr>
<tr>
<td></td>
<td>UINT, UDINT, UINT, USINT</td>
</tr>
<tr>
<td>TIME</td>
<td>T</td>
</tr>
<tr>
<td>ANY_BIT</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td>LWORD, DWORD, WORD, BYTE</td>
</tr>
<tr>
<td>BOOL</td>
<td>Z</td>
</tr>
<tr>
<td>CHAR</td>
<td>C</td>
</tr>
<tr>
<td>ANY_STRING</td>
<td>C</td>
</tr>
<tr>
<td>STRING</td>
<td></td>
</tr>
<tr>
<td>WSTRING</td>
<td></td>
</tr>
<tr>
<td>(not specified)</td>
<td></td>
</tr>
<tr>
<td>ANY_DATE</td>
<td>H</td>
</tr>
<tr>
<td>DATE_AND_TIME</td>
<td>T</td>
</tr>
<tr>
<td>DATE, TIME_OF_DAY</td>
<td>a h</td>
</tr>
</tbody>
</table>
```

See IEC 61131-3 Second edition 2003-01, Reference number IEC 61131-3:2003(E), page 32. The type `CHAR C` was later defined in IEC61131.

Complex types, not defined in IEC61499

```plaintext
| ANY_CNUM             | n |
| ANY_CREAL            | g |
| CLREAL               | f |
| CREAL                | d |
| ANY_CINT             | k |
| CLINT, CDINT, CINT   | jis |
```
6.1.3 Array data type specification

Arrays with one dimension and a determines length are defined by a simple number after the one-char-type, such as F3 for a float[3] array. This is a concise simple style which needs less space in the graphic.

Using simple one dimensional arrays is often necessary in FBBlock graphics, because several values are calculated with the same procedures. It depends from the implementation whether a FBtype can really process a vector, or whether more as one FBBlock is instantiated and called for the vectorized calculation. The graphic should not deal with this implementation detail. For example a FBtype to calculate the complex representation from a 3-phase voltage in a grid has of course an input F:3 for the three phase values, and hence an output f as complex, and also an output F for the so named zero sequence value which is often 0.0.

For expressions there is a simple way to build vectorized values and access to elements:

```plaintext
TODO
```

6.1.4 Container type specification

A container is known in higher programming languages, for example in Java as java.util.List or as sorted container as java.util.Map. Also an array with a non limited size is a container.

In UML the * is familiar to designate an aggregation with more possible destinations. This is also a quest of container: The aggregation (or also association and composition) has a multiplicity. Whereby the possibility to select exactly between 1.. or 0.. or 0..2 members or such is not supported in this granularity. It is possible also to have an array of a dedicated size also for aggregations. But whether this elements are set or they are nil, this should be checked by the implementation.

- Write a * after the type specifier or also on place of the type specifier (name:* it is designated: Any container. The implementing level decides about the implementation of a container. A container refers or contains any number of elements, sorted in order of input. Such a linear container can also implemented by an array in a free size.
- ** after the type designates a sorted container. The sorting key is implementation specific or specific from the creating and using FBBlocks. Often the name of an element is the sorting key (it's a String).
- [99] after the type designates an array with variable size but possible with a given maximal size. [] is a free variable size.
- [1..4] after the type designates an array with this possible range of size. It is similar the number of associations in UML

What about more dimensional arrays ... should be clarified in future. Writing style dimensions separated by comma such as [9,3] or F2,3 for an array of 2 element which each 3 elements. All rows and columns have an equal length. It should also be possible to use [], then the rows and columns or more dimensions can have each any different length, such as arrays in Java language.
6.1.5 Structured type on data flow

A structured type for data inputs and outputs is an instance of a FBtype. This instance comes from the data output provided to the data input. The difference to an aggregation is: The aggregation is a stable connection from one instance to another one, the using FBBlock can access the currently data from the aggregated FBBlock. For that also problems of data consistence (mutual exclusion on access changed data) should be considerate as known in Object Orientation and UML.

The data flow with instances of FBtype presume constant instances, which are not changed after delivering on the data input. This approach comes from the IEC61499. It is often also used in ordinary programming, but not so obviously. The common solution is: The data are binding to the event instance. Or, the event instance contains the data.

Often, for such approaches, dynamic allocated memory is used. This is the simplest form. But for frequently used dynamic memory the problem of defragmentation exists. In Java Runtime Systems this problem is solved by using the Garbage Collector. Another possible solution is: Using only memory blocks with equal sizes.

The other often simple solution is: Using a pool of event data. The event flow is usual deterministic in amount. It doesn't make sense to shoot around with events. An event should be created (using a member of the pool) only if it can also be processed, and if the pool is empty, there are obviously too much events in queues, not processed, and more events are only disturbing. Hence, the pool of event data is often a possible and proper solution for implementation.

Designation of the data type:

![Figure 9: OFB/DflowStructData1.png](image)

The Figure 9 shows two possibilities to dedicate the type of the data flow:

- If you have a connection from a dout or din pin to a class frame of style ofbClass or to a FBBlock frame, style ofbFBBlock without instance name, then this defines the type of the data pin.
- The second possibility is, use the type name after colon.

You can define the data pin type also in an extra diagram:

![Figure 10: OFB/DflowStructData1.png](image)

Here the connection is used as Style ofRefAggr which shows the non filled diamond as in UML. Additional for the type an * is written. This means, as also for other types, The type is a container. Also an array size can be used there, or the ** for a sorted container or [] for an array of not variable size. This is also possible of course for a immediately type specification as in Figure 9 on ClassG.
6.1.6 Data type forward and backward propagation

The input variables of the PID controller do not need this type declaration here, because the type is forwarded. But it is shown nevertheless, gets more clarity for usage. The type of the output variable \texttt{y:F} do also not need to be shown if or because the module is well defined in its interface for explicitly types or for type forwarding.

More step times or calculation events: In this example automatically an event chain is generated from \texttt{stslow} (means a slower step time) to the expression block with the \texttt{w1} variable, and forward to the event output \texttt{stslow0} (not shown here). Because \texttt{w1} of of style \texttt{ofpZout...} it needs updated with the correspond \texttt{updslow} event on the module’s input block. If the value of the \texttt{ofpZout} variable is connected to outputs of the module with also the \texttt{updslow} event, the appropriate data flow will be assigned to this event chain till \texttt{updslow0}.

Data consistence: If the value of the \texttt{ofpZout} variable is used in another event chain, as shown here for built \texttt{dwx}, the stored value of the last calculation (after update) is used. In this case the value comes from another step time or calculation event, just the \texttt{stslow}, and hence consistently data all from this update event can be used. The consistence of the data should be guaranteed by a proper implementation. For example a slower step time can prepare values in with higher calculation effort, but the update of this values is done in a high priority interrupt which cannot be interrupted by another. The update needs only copying of values, or as better solution switch only a pointer to a double buffer system, if the update event is registered for the interrupt. Then the values are always consistent.

old:
You can show data and event pins on classes, but the connections are only sensible between the instances. This is familiar for FBlock diagrams. The type of data pins can be given immediately on the pin (after colon), but can be also forward propagated by a data flow. Simple arithmetic operations do not change the type of source pins and forward the type to the destination pins. Specific operations (for example access to the real and imaginary part of a complex value or to an array element) does not change the numeric type but influences the real/complex or array property of the type. Specific FBlocks can forward the type of inputs to the type of outputs. A backward propagation (as in Simulink) is not designed, because sometimes a mix of forward and backward propagation is more confused by the user. An important property of FBlock diagrams is, that the numeric type of pins in library FBlocks are not determined, instead a type dedication as \texttt{ANY_NUMBER} (in IEC61499) or such can be used. In Simulink it is determined as “\textit{inherit}” type. It means that the types in the usage of the FBlocks depends from its using environment. For code generation either any template should be used (C++) or the FBlock should be existing as variant with all necessary types, or the FBlock implementation is a macro (C language) where the compiler associates the type.
6.2 Expressions inside the data flow

Expressions for data flow are presented by a figure (here a circle, but usual also a rectangle) of the style ofbExpression. This figure can immediately connected by ofRef connectors or simple Default Drawing Style for input and output, whereby the input connector can have a text for the expression.

Figure 5 odg\ExpressionExmp1.png

The name wxd is the text on the circle itself.

This is the simple form. Note, writing a text to a line with some inflection point is a little bit sophisticated in currently LibreOffice versions.

6.2.1 Expression parts as input

The other possibility is using a rectangle box with the style ofbExpression, in the following text referred to as FBexpr. ("Function Block as expression"). The original outfit of the style is a dashed line as border. Small inner rectangle shapes with style ofbExprPart can be used for the expression inputs. They can contain operators and also a factor as constant or as variable. The basic form to add and sub is:

Figure 6 odg\ExpressionExmp2.png

In opposite to the circle with lines, here is enough place and clarity to write a text associated to the expression input. This can be one of the operations known from mathematics and logic in the following groups:

- + - numeric
- * / % - numeric % is modulo. % can be used as unary operator to negate the numeric value.
- & ~ boolean or bit wise AND, ~ is not or negate. At least one input (recommended the first) should have the &
- | v ~ boolean OR, also bit wise OR. The v may be better readable as |, hence recommended.
- ^ ~ is XOR. The ~ for negate this input can also be used. Also == and != can be used for boolean and bits.
- @ == != <> < <= > >= For comparison, whereas more as one inputs can be used. For == this is clarity, all should be equal. Also != or <> means, all are not equal together. Elsewhere the relations are valid in comparison to the input before, or in comparison to the first input if this is designated with @.

Mixing faulty operators cause an error while evaluation the graphic. Look on the following examples:

Figure 7 odg\ExpressionExmpCombi.png

The Figure 7 shows a combinatoric, the expression is

\[
y_4 = -((-x_1 + x_2) / (-x_3) * x_4 ) + x_5;
\]

The last expression block has the – as ofbExprPart immediately near the circle which is an ofbExpression. This is an alternative instead write the – on the line. But of course in the translated source expression line the – appears before the representing (...) of the expression before.

In the middle FBexpr the * is omitted because it is default, the expression is detected as multiply expression. Also the * on the first input can be omitted because the / is enough concise to determine this
FBexpr as Multiply expression with one operand to divide. The \(-\) after \(-/\) is the unary \(-\) for the \(x3\) input. All of this should be intuitive understandable.

But to reinforce it look on a boolean example:

![Figure 8](ExpressionExmpCombiBoolean.png)

This is\n\[ yb1 = (b1 \& !b2) | !b3; \]

In C/++ Syntax. Because the data types are boolean in C/++ the \(!\) should be used for negation (NOT). If the data types would be \(u\ w v\) then the \(~\) will be proper. The code Input generation designates it automatically.

### 6.2.2 Factor and constants on inputs

But there are more possibilities:

![Figure 9](ExpressionExmpK2const.png)

This figure shows an add expression, but the second input is also multiplied with the variable \(fw\) and the 3\(^{\text{rd}}\) input is a constant with the given value. The variable \(fw\) should be able to find in the state variables of the model. It is wired as the \(k2\) input in the IEC61499 textual presentation. If the pin has an input connected, the constant is the multiplier and assigned to the \(k..\) input. If the pin has no connection, the constant or also a variable is wired to the \(X..\) input. It means one FBexpr supports also multiply its inputs with numeric state variables, which is often proper usable. Also for comparison constant values are proper usable.

### 6.2.3 Any expression in FBexpr

The \(ofpExprOut\) shape or also the text of the \(ofbExpression\) can contain both a function written with parenthesis, for example \(atan2()\) or any expression written in the target language using \(X1\), \(X2\) etc. for the inputs. The source code generation inserts this function or expression either as written or with an adequate derived code, see next. Some functions should be well known for graphical level. Specific maybe complicated functions can be written in the implementation level and called here immediately.

Look on a first basically example:

![Figure 10](ExprAnyX1X2.png)

The \(ofbExpression\) shape or block has not any \(ofpExprPart\) or \(ofpOut\) pins, it is not necessary. Input and outputs are immediately bonded to the expression block. The inputs are counted from top to down, and then right side from top to down, or also from left to right first top, and at last on bottom side, if necessary. The input pins has in this order the names \(X1\) .. \(X99\) so much as given.

While code generation the identifier \(X1\) .. etc. are replaced by the values which are connected on the inputs using the \(code\) template scripts, see chapter 6.2.6 Code template scripts.

Because often target languages such as Java or C/++ are very similar in expression writing, the expression notation in the graphic is compatible with some languages. With an adaption table function names can be replaced for a specific destination language. For example the here shown \(sqrtf()\) is known for C/++ language, for float calculation. For Java source code it can be adapted with \((\text{float})\text{Math.sqrt}()\). This is
done as part of the translation template.

Also for this possibility input \texttt{ofpExprPart} can be used to influence the inputs also with factors, or using constants or negate the input values.

### 6.2.4 Output possibilities

All shown expression examples till now have its outputs on the expression shape. In this kind the expression is not represented with a variable, it is an inline expression. The value is stored or used from the input pin after.

![Figure 11 odg\ExprOutpin.png](image)

This example Figure 11 shows two expressions with a pin symbol on output. A pin symbol forces creation of a pin variable in the generated code. Especially on forking the data flow (using for more as one input) as here for \texttt{xdab} it is sensible. The left output has the style \texttt{ofpDoutRight} which is a normal data output. This forces a stack local variable in the code. But here the variable is necessary to collect the both parts of the complex value. If the expression is only used in one event chain, it is always ok.

The second expression \texttt{xdab} uses a style \texttt{ofbVoutLeft}, and the shape is rotated to 90°. This forces an instance variable in the \texttt{struct} or \texttt{class} of the module. The advantage is, it can be better visited in debugging. The variable can be used also in more as one event chains, which are more as one operations, but the data consistence is not guaranteed here.

The name of the output pin determine the name of the expression. If the output pin has not a name as for \texttt{xdab}, the name of the expression is the text in the \texttt{ofbExpression} shape box.

![Figure 12 odg\ExprOutStateUpd.png](image)

The Figure 12 uses an output with style \texttt{ofpZoutRight}. The letter \texttt{z} is derived from the \url{https://en.wikipedia.org/wiki/Z-transform} which is used for calculation, \texttt{z} is the stored (state) value. Hence it is set with the \texttt{update event}, here \texttt{updSlow}. The image shows the prepare and update events in gray, because there are automatically calculated. The input of the expression is here only one value, \texttt{w}, the expression can have more inputs as shown in the chapter before 6.2.1 Expression parts as input. The expression is calculated with the prepare event, here \texttt{stsSlow}, due to the data flow. But the output of this prepared value, setting of the variable is done with the associated update event, it means after (or before the next) preparation calculation. It means all Zout-variable have the state of the last step for the next preparation. In Simulink these are 1/z Blocks, so named “Unit Delay”, or also so named “Rate transition” FBlocks, from view of another event chain (means another sample time, or another operation in implementation. If the update operations are atomic, non interruptable, then all Zout data are consistent.

The same is given if the text in the \texttt{ofpExpression} shape contains such a text.

![Figure 13 odg\ExprAtan2n.png](image)

Then is is not the name of the FBexpr block, it is the function. Then the expression can have an output variable, where the function result is stored, as described in chapter 6.2.4 Output possibilities.

The function designation can also contain a type for the output, see next chapter, but also type designations for the inputs.
Figure 14  odg/ExprAtan2n.png

The Figure 14 shows an atan2() operation which takes a complex value as input and outputs a scalar number. To translate it, firstly the type letters for maybe non full specified values are replaced by the forward propagate types, for example results in atan2(f)=F. With this text the source code generation searches a proper translation, exact this String is used as identifier for a OutTextPreparer sub script which is then used for code generation. This sub script can be

\[
\begin{align*}
\text{atan2(f)} &= \text{F} \\
\text{atan2f} &\left( \text{cvar.im, cvar.re} \right) \\
\end{align*}
\]

which results in generated code for example to atan2f( cvar.im, cvar.re); which calls the atan2() as given in C/++ destination language.

The designation of the output (here N as any numeric) is important, elsewhere the type propagation forwards the input type to the output. It does not know that the atan2() operation outputs a scalar.

6.2.5 FBexpr as data collect and access

If you look at Figure 11 the you see on input .re and .im. This expression needs an output variable, which collects the real and imagine part and delivers a complex value.

The opposite expression is

Figure 15  odg/ExprOutRelm.png

Here on access (without output variable) the input is accessed as complex value, then the .re part is accessed from this value. It is (strongly) recommended that the input should be connected immediately with an expression output variable as in chapter 6.2 4 Output possibilities or with an output of a FBlock also as ofpVout... or ofpZout... because the variable should be immediately accessed. Then the generated code is more proper.

The same as for .re and .im can be done for elements of an array. The collect (on the ofpExprPart) and the access (on the exprOut) should be written in form [2] where as the 2 is the immediately constant index to the array. But also a variable index is possible, write [ix] where ix is a variable in your module (expression output) similar as the multiply feature in expression parts as input. The size of the array variable on a collect expression should be dedicated, given with the type specifier, see

6.2.6 Type specification in expressions

In the texts of the expression inputs and outputs (ofpExprPart, exprOut and also the pins on output ofpDout..., ofpVout... ofpZout...) the text on the pin can contain a : ...Type designation. This is true also after a variable name (for the out pins) as also for all other possibilities for the expression part and output. The type designation follows chapter 6.1 Data types. The types should be semantically sensible. In this kind the size of an array can be defined, see example:

Figure 16  odg/ExprArray.png

Here the text to the output is wrapped, this is not important. But it ends with :F[3], means it is a float[3] array in C/++ or also Java language. The right expression then accesses the element 1.

6.2.6 Code template scripts

If an expression
exprTerm for an expression before without output variable or adequate due to chapter