INSTRUMIND THINKCOMPOSER

Product Manual

Document Version: 1.5.13.1127
Website: http://www.instrumind.com
E-Mail: contact@instrumind.com

Content

Overview

Welcome to ThinkComposer!

This manual will introduce you about the conception of the product and its management. It is divided in two main parts: First, the *Base Model* part, which describes the documents and the rich-content objects you create, plus their meaningful typification. And second, the *Application Guide* part, about the program components, its behavior and how to take advantage of its features.

Context

The ThinkComposer product is created in a context encompassing the next set of factors:

- **Intended Users**: Professionals, academic people, students and any person or teamwork who are used to visual tools or require graphic means to do their job.

- **Work performed**: Intellectual discovery or creation, research, management activities, analysis of problems, design of solutions, organization of ideas, knowledge representation or others alike.

- **What they need to show, store and share**:
    - Symbols, connectors, boundaries and their visual styles.
    - The meaning (semantics) of these graphic objects.
    - Detailed and structured information, plus attachments.
    - Multiple perspectives and levels of depth, not only a flat main view.

Vision

ThinkComposer enables its users in the creation of a new kind of Concept Maps, Mind Maps, Models or general purpose Diagrams, which provide enhancements over classic implementations:

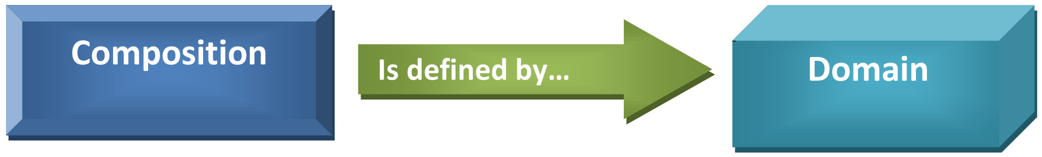
- **Composability**: Each object (node, symbol or idea) can be composed of a whole diagram within it. This way, users can go beyond the initial view and reach multiple levels/layers of expression.
- **Rich-Content**: Each object can have multiple details, being them text, images, attached files, external links or structured information (custom-fields and tables).
- **Meta-Definitions**: The content of the diagrams (ideas/nodes, their connections and details) are based on particular descriptors which typify, delineate and rule their appearance, information and behavior.

With the provided features, ThinkComposer **can be adapted to change and evolve** together with the specialized business/field of its users.

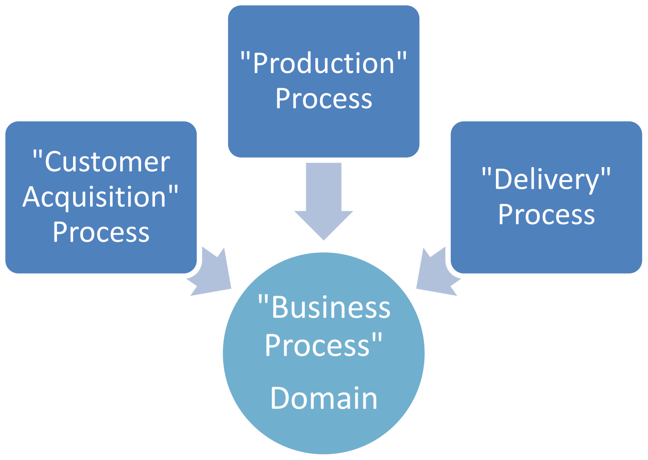
Base Model

Working Documents

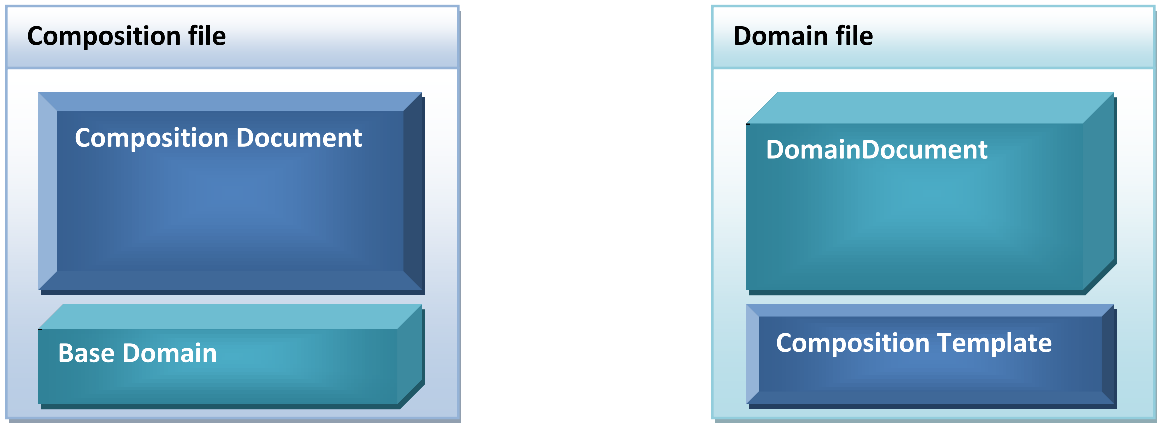
ThinkComposer bases its work on two main document types: *Compositions* and *Domains*. A Composition contains information belonging to a particular business or field of knowledge (user's world), and a Domain defines the meaning of that particular area of interest.



Multiple Compositions can be defined by one Domain. In the next example a "Business Process" Domain serves as the basis for three Compositions describing the processes of some departments in a company.



Documents of both types are stored in their own file types and can live separately. That is because a Composition file also contains the base Domain within it. In the other hand, Domains can exist alone or include a Composition, for optional use as Template, in the creation of new Compositions.



Common Objects Properties

Most ThinkComposer objects have the next properties for identification or documentation:

Name/Title: Name or Title of the object.

Summary: Summary of the object.

Pictogram: Graphic representation of the object. It is like an icon, however can be larger to be used as symbol.

Tech-Name: Technical-Name of the object. It should be unique and is intended for machine-level usage as code, identifier or name for files/tables/programs. It can contain only characters allowed for file names.

Tech-Spec: Technical-Specification of the object. It is intended as a machine-level representation for computation (i.e. for use as script, template, formula or other kind of expression).

Global ID: Global unique identifier of the object.

Description: Detailed description (rich) text of the object.

Versioning: Stores versioning information such as Creator, Last-Modifier, the respective change dates, Annotation and Version Number, plus a Version Sequence number which is automatically incremented in each change (considers also changes in the nested composite objects).

Note: You can set Versioning available for Ideas while declaring an Idea Definition (on the Domain), marking it as "Versionable".

File Types

The next table shows the document types in relation to their file types:

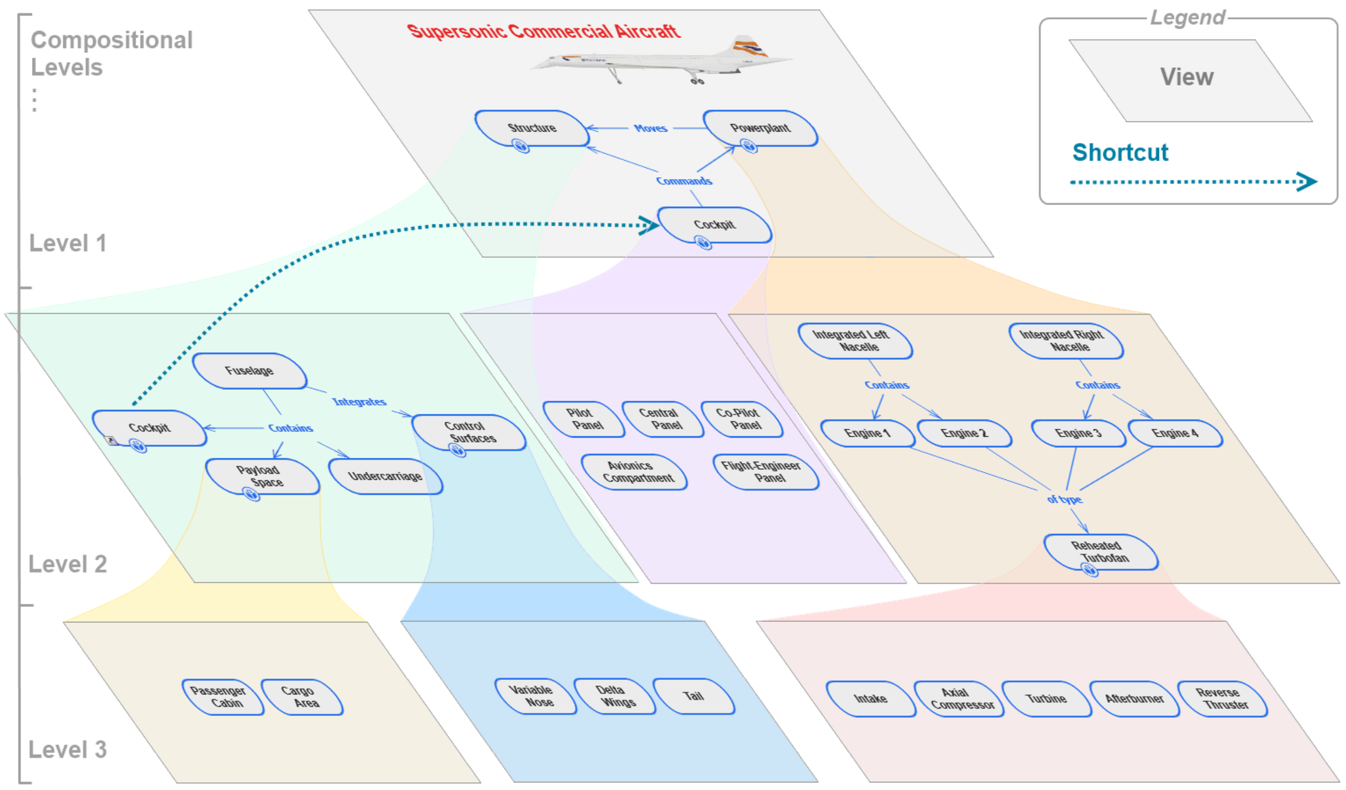
Document Type	Icon	File Extension	Mime-Type (for Windows file associations)
Composition		.tcom	application/x-instrumind-thinkcomposer-composition
Domain		.tdom	application/x-instrumind-thinkcomposer-domain

Compositions

A *Composition* is a rich-content document, identified and documented by a set of *Common Object Properties*, for expressing an analysis, design, creation, knowledge or general thinking within a field/business *Domain* using interrelated *Ideas* (*Concepts* or *Relationships*) for that purpose.

Ideas are visually expressed in *Views* with *Symbols*, *Connectors* and *Shortcuts* to them. Plus, through its *Composite-Content*, Ideas can span multiple composability depth levels. The Composition is said to be "flat" when only the main/root view is present.

Consider the next image, showing a multilevel Composition structured to model an "Aircraft":



Note: This is just a representative image, of a multilevel Composition structure, showing its composite Ideas as nested diagrams.

In the example you can see the parts of the aircraft and how they are composed. Plus, a *Shortcut* is established at different levels between two parts (the cockpit, which is also exposed as part of the fuselage), this allows the reuse of Ideas by referencing them instead of making copies.

Views

A View is the visual surface on which the diagrams are created. Every Composition has an initial main View, which is the root of the arbitrary compositional hierarchy it may have. It has the standard *Common Objects Properties* and capabilities of pan (scroll), zoom (scale), show a grid to allow the easy editing of its content, plus optional background brush/color and image.

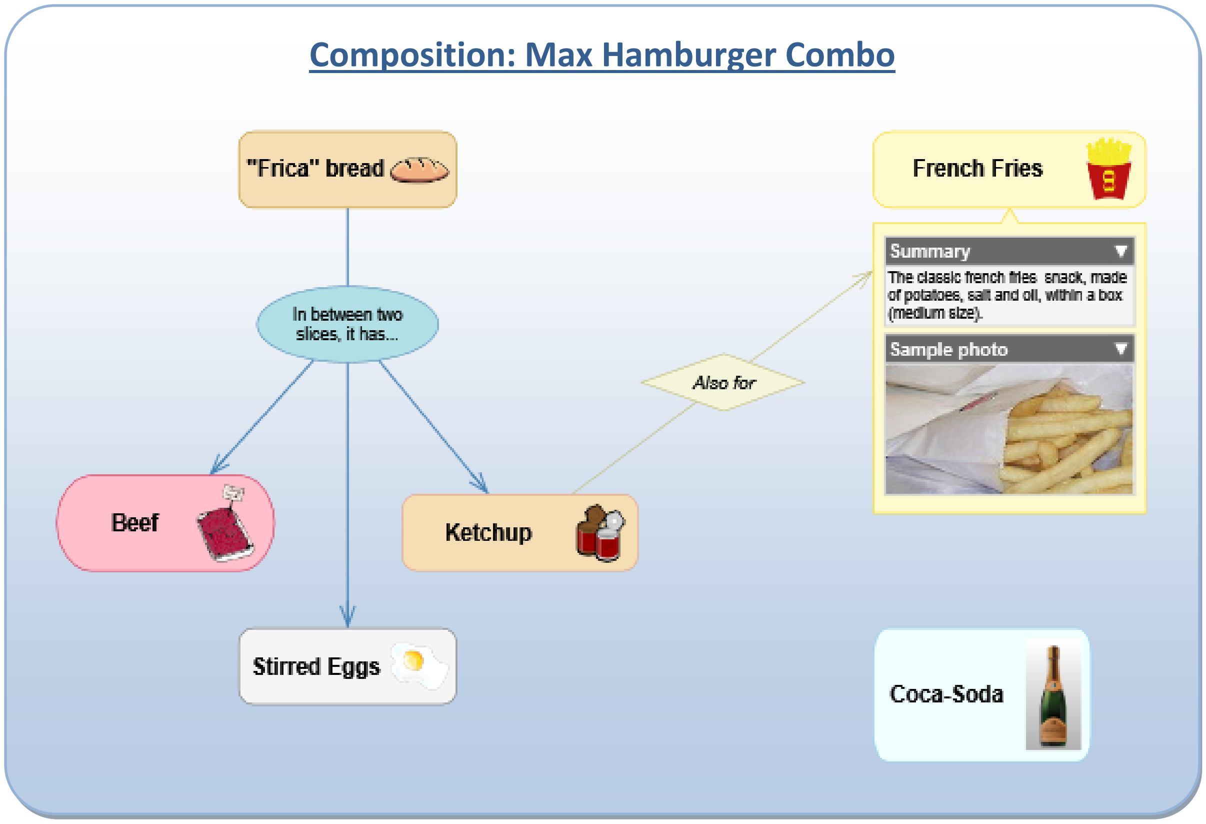
Ideas

An *Idea* is either a *Concept* or a *Relationship*, being a common abstraction to refer to them. They have a set of properties about their visual appearance, the information they contain and their interrelations. Idea types are defined by declarative objects called *Idea Definitions*, being them either *Concept Definitions* or *Relationship Definitions* depending on the intended usage. The underlying definition of an Idea determines its initial visual appearance and the details it can contain (attachments, links and tables).

As other objects, all Ideas have the *Common Object Properties* available to identify and document them.

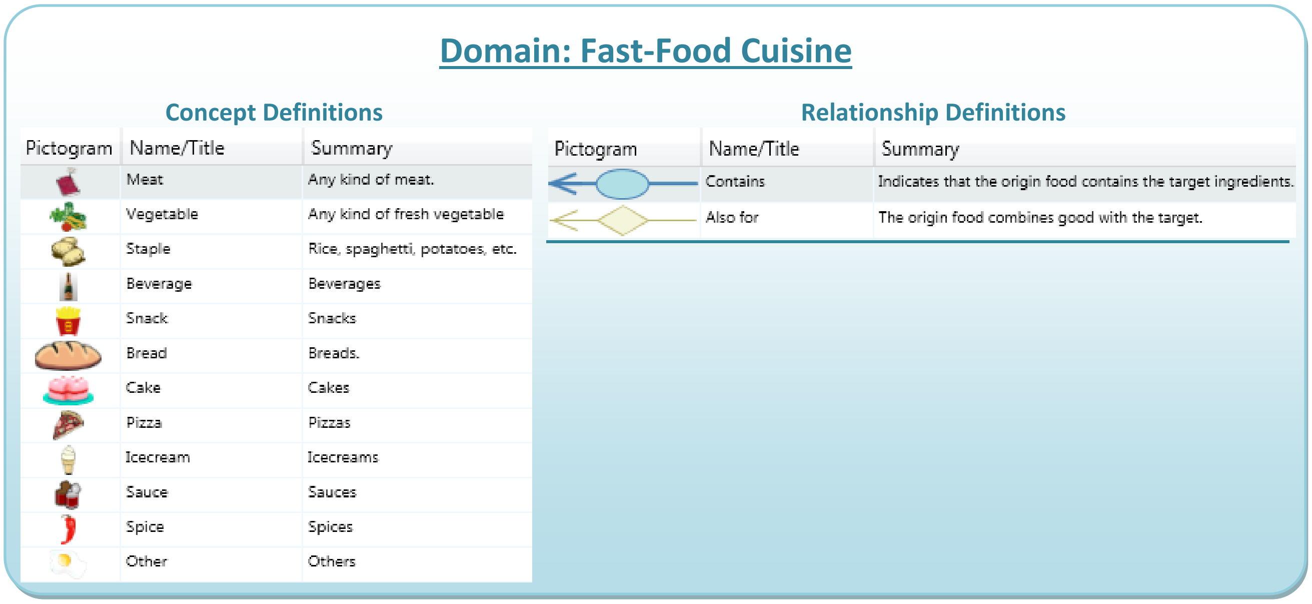
By having all these properties under a common base, *Concepts* and *Relationships* can be as simple or powerful depending on the needs of the particular situation. They can be extended with more details or with a composite-content hierarchy under each one. This is a strong difference with other tools, where concepts/topics are no more than simple symbols and relationships/associations are no more than just lines.

Consider the next Composition diagram example, where a fast food "Combo" product is briefly explained with its ingredients:


Composition: Max Hamburger Combo
"Frica" bread
In between two slices, it has...
Beef
Ketchup
Stirred Eggs
Also for
French Fries
Summary
The classic french fries snack, made of potatoes, salt and oil, within a box (medium size).
Sample photo
Coca-Soda

As you can see, each Concept includes a Pictogram (or Icon) and one, the "French Fries", also show a couple of Details: Its Summary property and a "Sample Photo" image. Plus, the Relationships describe how the ingredients are combined.

That “Max Hamburger Combo” Composition is based on the next illustrated Domain:



Domain: Fast-Food Cuisine

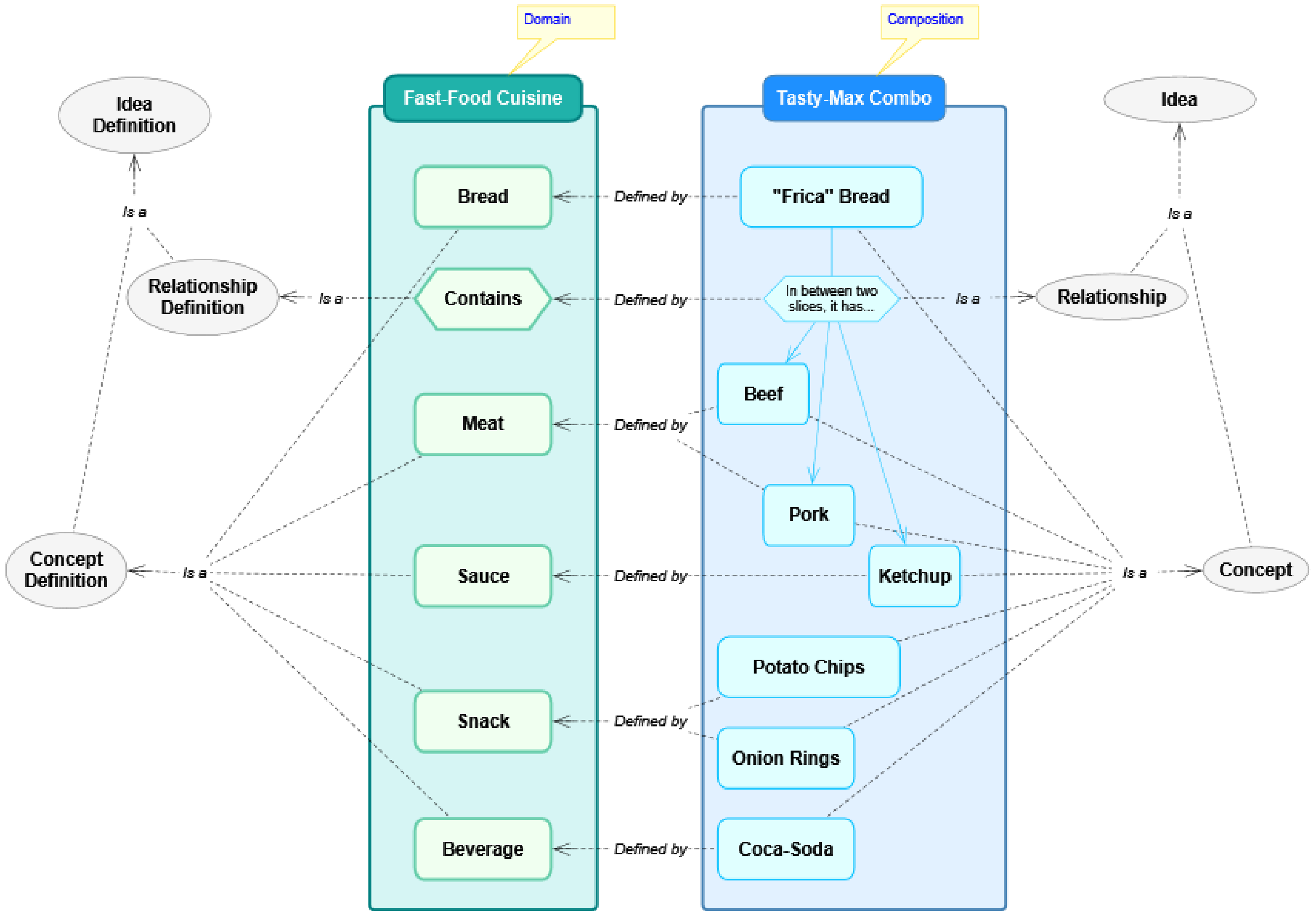
Concept Definitions

Pictogram	Name/Title	Summary
	Meat	Any kind of meat.
	Vegetable	Any kind of fresh vegetable
	Staple	Rice, spaghetti, potatoes, etc.
	Beverage	Beverages
	Snack	Snacks
	Bread	Breads.
	Cake	Cakes
	Pizza	Pizzas
	Icecream	Icecreams
	Sauce	Sauces
	Spice	Spices
	Other	Others

Relationship Definitions

Pictogram	Name/Title	Summary
	Contains	Indicates that the origin food contains the target ingredients.
	Also for	The origin food combines good with the target.

Now, the next diagram explains how the exemplified *Idea Definitions* and *Ideas* are related:

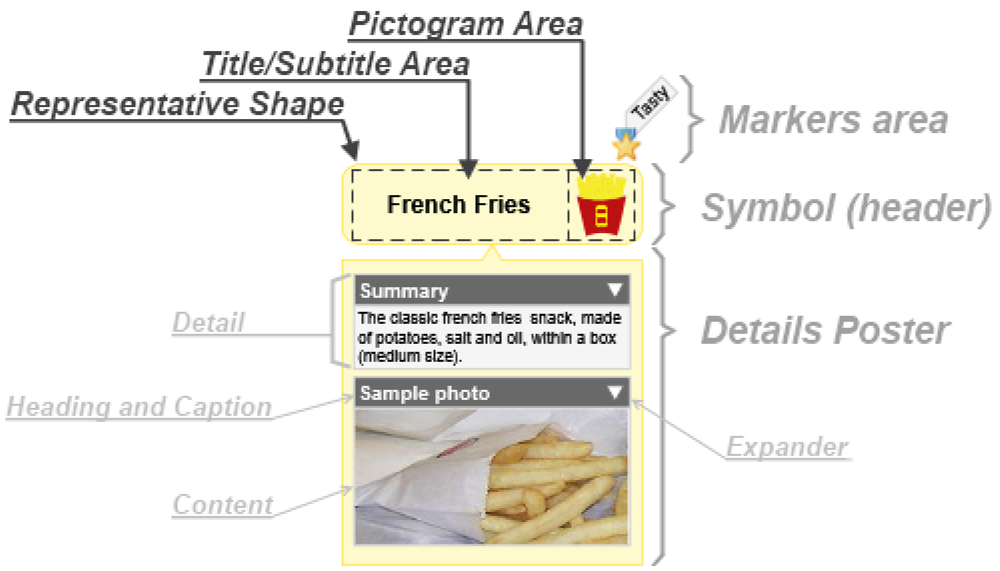


Then, in the *Idea Definitions* section, that vision will be discussed again in further detail.

Symbols

Ideas can have a *Symbol* as its whole *Visual Representation* when they are *Concepts* (the Symbol is the "Body" of the Concept), or as part of it when they are *Relationships* (the Symbol is the "Central/Main-Symbol" of the Relationship, which is in the middle of its Connectors).

The Symbol structure has the parts shown in the next illustration.



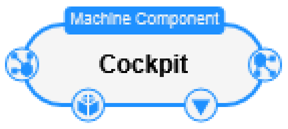
The parts are:

Markers area: This zone shows any Markers that the symbol may have appended.

Symbol: This is the body of the Symbol, composed of the Representative Shape plus *Title/Subtitle* and the *Pictogram* areas. Also, is called "Header" to differentiate it respect the "Details Poster".

Details Poster: Is an expandable/collapsible area to show Details, also can expose a mini-view of the *Composite-Content* inside the represented Idea. It can be "hanging" (with a triangular visual appendix pointing to the symbol) or directly joined. Each Detail area has a Content zone exposing whatever is stored or referenced, plus a Heading zone to show the Caption entitling it and an expander for the Content.

Also, Symbols may expose its *Idea Definition* name on top o it, plus some *Indicators* as visual cues about hidden content. They are shown in the next illustration.



The current Symbol **Indicators**, aside from the Shortcut one, are:

- **Related Origin Ideas**: Appears when hiding/collapsing related Ideas, which are origins of this one, plus their intermediate Relationships.
- **Related Target Ideas**: Appears when hiding/collapsing related Ideas, which are targeted by this one, plus their intermediate Relationships.

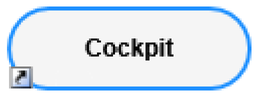
Composite-Content: Appears when the Idea has been extended with other Ideas composing it; therefore exists an underlying View as the root for that Composite-Content.

Detailed Content: Appears when the Idea has Details present (Attachments, Links, Tables or Custom-Fields).

Shortcuts

A Shortcut is a reference from a local Symbol to a target Idea in a remote area, such as a different View where that target is located. Also, it can be located in the same View of the target, but at a distant position (for avoiding long connectors crossing the View), or as intentional visual duplicate (called “visual synonyms” in other tools).

You can identify a Shortcut Symbol by the “↗” link indicator icon located in its bottom left corner, such as in the next example.

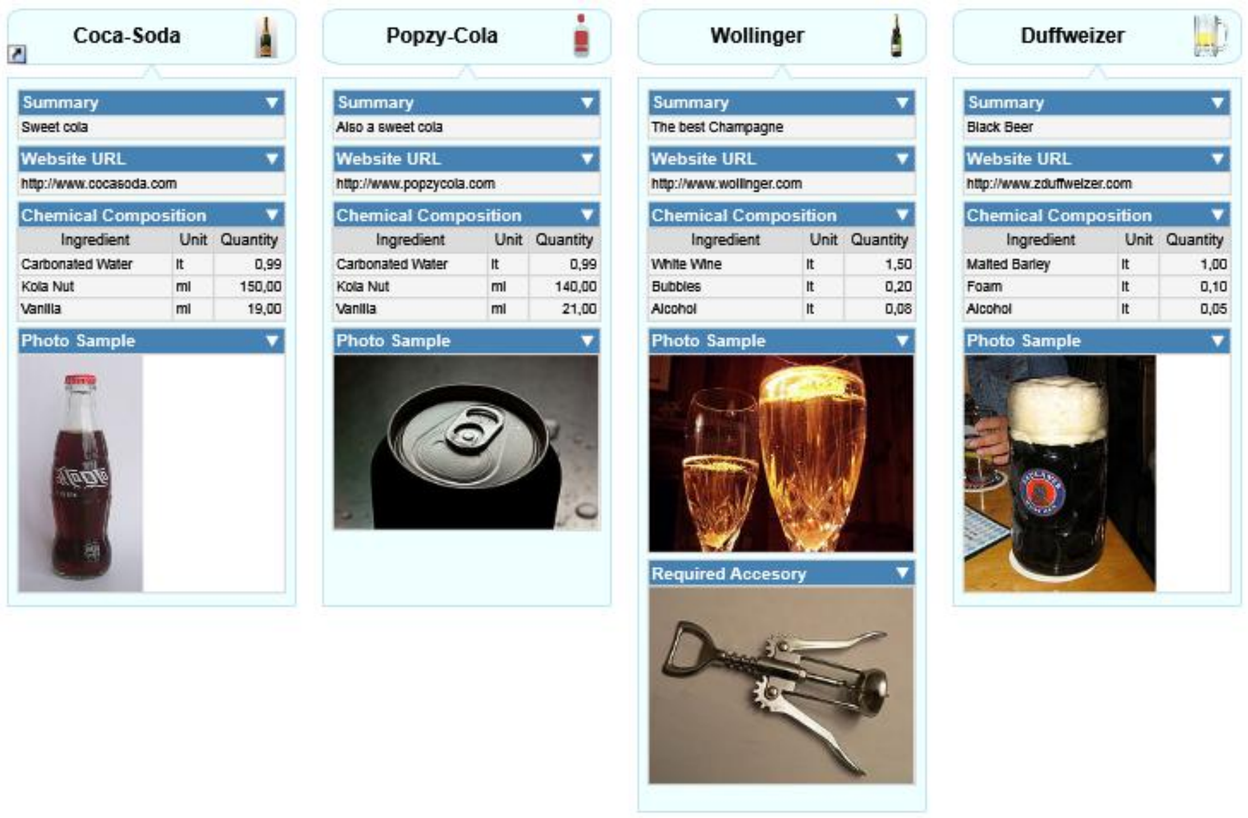


It is a good practice to have Shortcuts instead of, for example, making copies of an Idea, because the represented semantic content remains the same.

Details

A Detail is a piece of content appended to an Idea to enrich it. The available kinds of Details are: *Attachments*, *Links* and *Tables*, plus *Custom Fields* which are a simplified Table. They can be *Designated* either transversally, for the *Idea Definition* (Domain) level when it is required to appear for all Ideas of the same kind, or locally, at the Idea (Composition) level, when is for non-anticipated cases.

Consider the next example:



Coca-Soda

Summary: Sweet cola

Website URL: http://www.cocasoda.com

Chemical Composition

Ingredient	Unit	Quantity
Carbonated Water	lt	0,99
Kola Nut	ml	150,00
Vanilla	ml	19,00

Photo Sample

Popzy-Cola

Summary: Also a sweet cola

Website URL: http://www.popzycola.com

Chemical Composition

Ingredient	Unit	Quantity
Carbonated Water	lt	0,99
Kola Nut	ml	140,00
Vanilla	ml	21,00

Photo Sample

Wollinger

Summary: The best Champagne

Website URL: http://www.wollinger.com

Chemical Composition

Ingredient	Unit	Quantity
White Wine	lt	1,50
Bubbles	lt	0,20
Alcohol	lt	0,08

Photo Sample

Required Accesory

Duffweizer

Summary: Black Beer

Website URL: http://www.zduffweizer.com

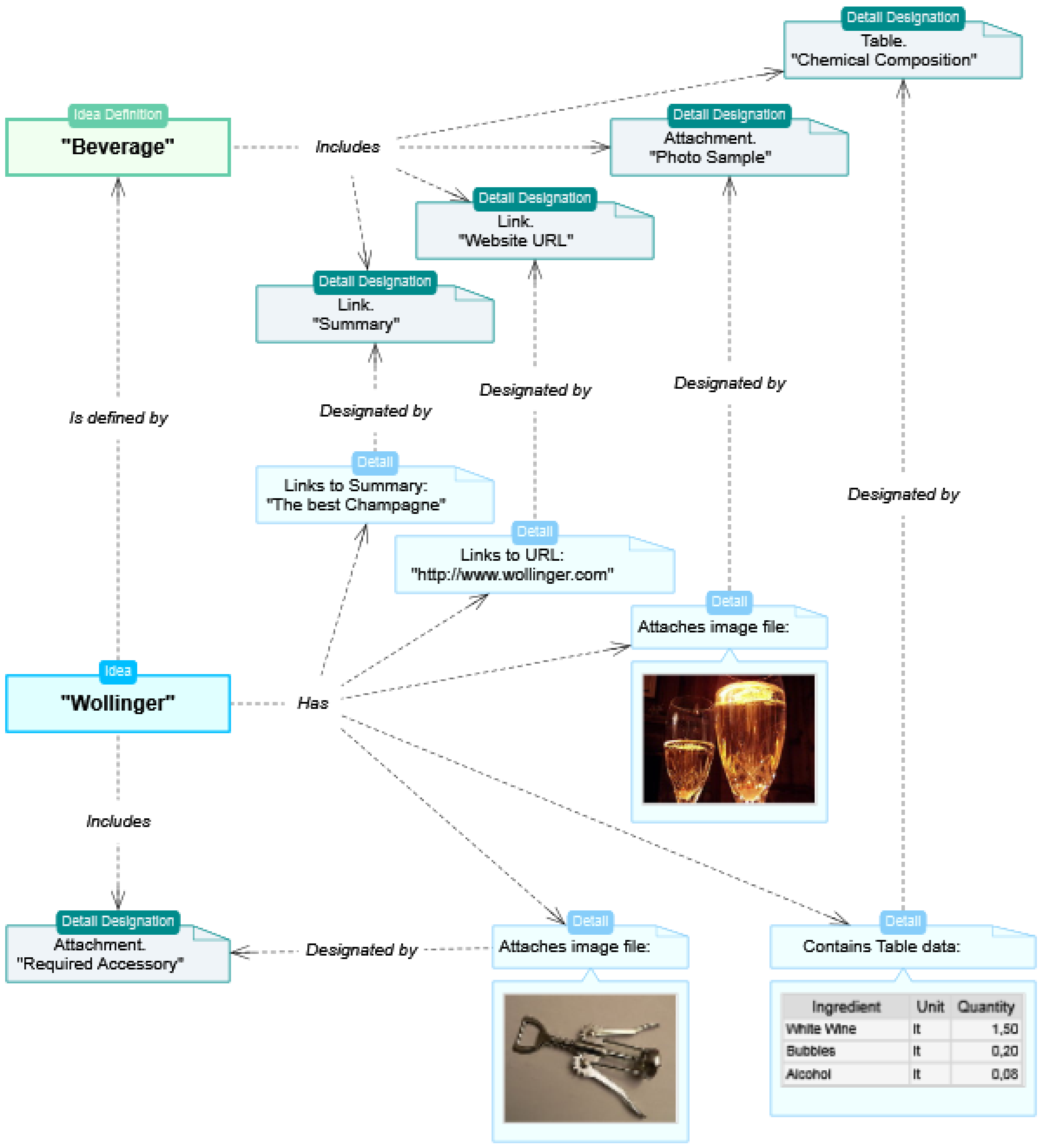
Chemical Composition

Ingredient	Unit	Quantity
Malted Barley	lt	1,00
Foam	lt	0,10
Alcohol	lt	0,05

Photo Sample

As you can see, all symbols have the same Details (defined on their common Idea Definition): A Link to the “Summary” property, other Link with the “Website URL”, a Table with the “Chemical Composition” and an Attachment with the “Photo Sample”. In addition, the “Wollinger” beverage Idea has also a specific Attachment called “Required Accessory”, which only belongs to that particular Idea.

For a better understanding of how Details are implemented, consider the next diagram.



Ingredient	Unit	Quantity
White Wine	lt	1,50
Bubbles	lt	0,20
Alcohol	lt	0,08

By designating Details at the Idea Definition level, you create a standardized set of details which each Idea, created from that Definition type, must have.

In the next sections, the currently available detail types will be exposed plus the detail designations.

Attachments

An Attachment is an arbitrary embedded object, obtained from external source, such as: image, video, data file, etc. Currently, only Images and Text can be displayed when shown in the Symbol's Details Poster, however any file type can be stored as attachment.

Links

A Link detail (not to be confused with *Relationship Link*) is a reference or pointer to an object, which may be reachable from the tool by invoking an external one. There are two types of links:

- **External**: References an object outside the Composition, such as a Web site/page, Folder or File.
- **Internal**: Points to a property of the Idea being detailed, therefore the Symbol can show properties other than Name/Title or Tech-Name on its Details Poster.

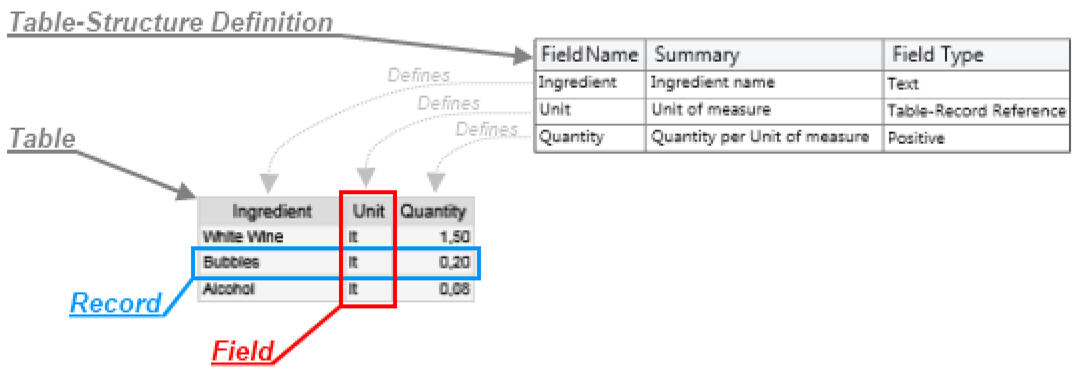
By default, the Summary property is linked to every Idea Definition, just to satisfy the common need of showing it in the diagram.

Tables

A Table is a container of structured information, composed of one or more *Records* (also called "rows") of the same type. Each Record has a set of *Fields* (also called "columns") which store individual data items.

Note: The terms "Record" and "Field" were chosen, instead of "Row" and "Column", to avoid confusion when visually transposing a Table on the diagram (where rows become columns and vice versa).

The structure of a Table is specified by a *Table-Structure Definition*, where each Field is named, typified and described with other attributes. That vision is expressed in the next diagram.



Field Name	Summary	Field Type
Ingredient	Ingredient name	Text
Unit	Unit of measure	Table-Record Reference
Quantity	Quantity per Unit of measure	Positive

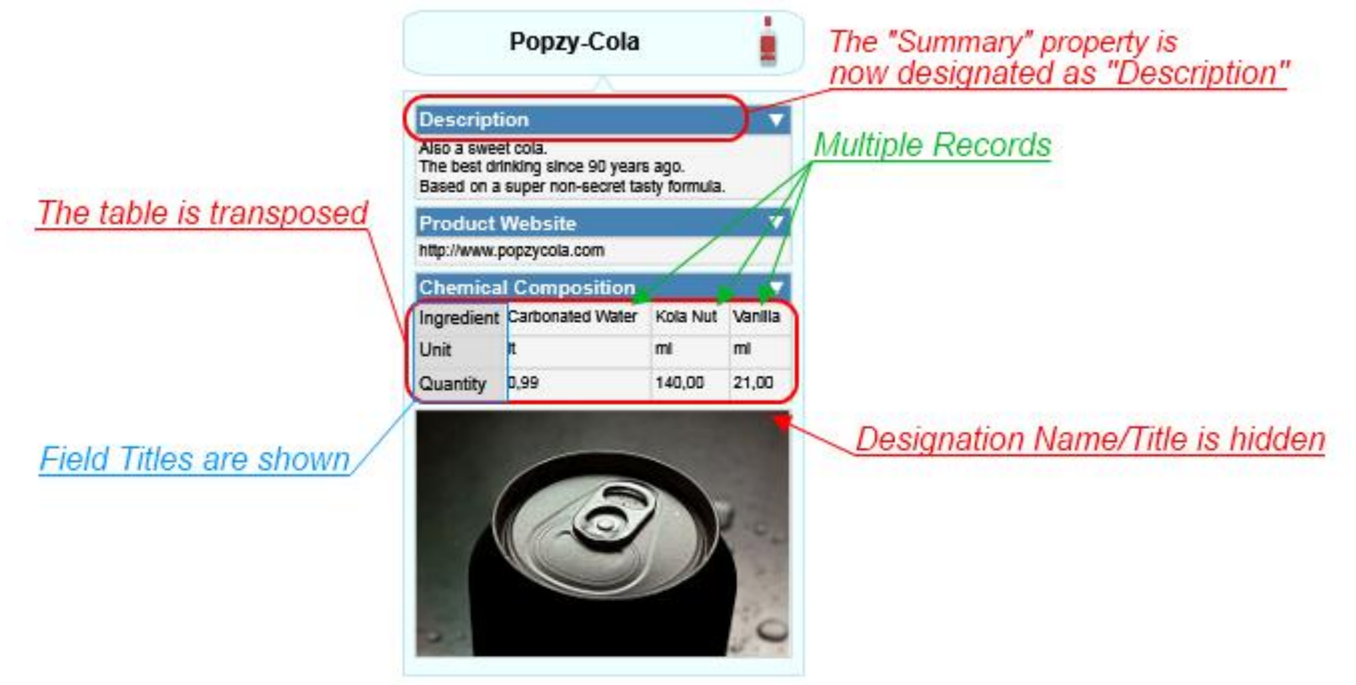
Ingredient	Unit	Quantity
White Wine	lt	1,50
Bubbles	lt	0,20
Alcohol	lt	0,08

Custom-Fields

All Ideas have a limited set of internal properties (or Fields), such as Name/Title and Summary. However, you can create your own "Custom" Fields according to your specific needs. They are defined in the same way as Table Structures and are shown and edited as a single-record Table (see the "*Editing Custom-Fields*" section).

Detail Designations

A Detail Designation is the assignment or declaration of a detail, for a particular purpose, using a proper name and a set of attributes about its visualization. Consider the next example:



Also a sweet cola.
The best drinking since 90 years ago.
Based on a super non-secret tasty formula.

http://www.popzycola.com

Ingredient	Carbonated Water	Kola Nut	Vanilla
Unit	lt	ml	ml
Quantity	0,99	140,00	21,00

The available properties for designate the details are:

- **Designator**: Gives the Name/Title to the detail, plus Summary and Pictogram that describe it.
- **Structure**: Relevant only for Table details, assigns the Structure which constitutes it.
- **Is Displayed**: Indicates that the detail will be shown in the Details Poster. This can be changed with the expander (presented as a little triangle icon) shown together with the Name/Title.
- **Show Title**: Indicates that the Name/Title will be shown over the detail in the Details Poster.
- **Properties for Table details**…
  - **Multi-Record**: Indicates whether the grid will show multiple records, else only the first one.
  - **Layout**: Visual style of the grid. Options are:
    - Conventional: Shows records on rows (one upon the other, in the vertical axis) and fields on columns (side by side, in the horizontal axis).
    - Simple: Shows a simple list of records on rows (one upon the other, in the vertical axis) and just the assigned Label-Fields.
    - Transposed: Shows fields on rows (one upon the other, in the vertical axis) and records on columns (side by side, in the horizontal axis).
- **Field Titles**: Indicates whether to display the Field Titles.

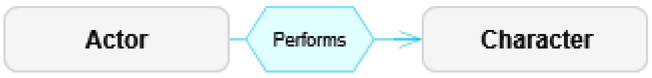
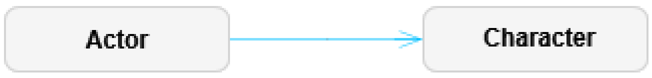
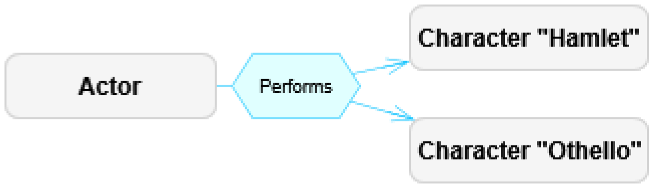
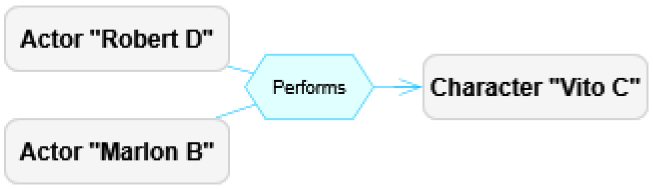
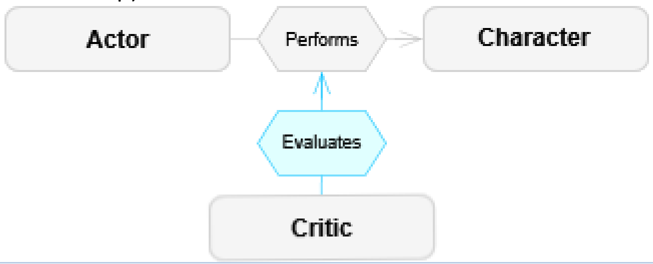
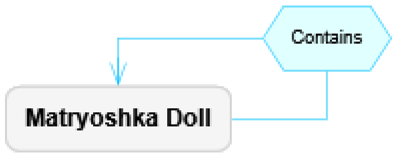
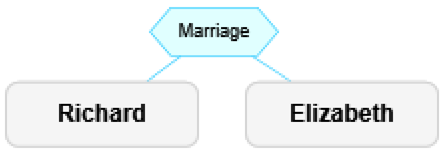
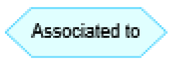
Concepts

A Concept is an *Idea* subtype, which constitutes a concrete object that can be associated to others through *Relationships*. It is visually represented by a *Symbol* shaping its body. Each Concept type is described by a Concept Definition. Concepts have all the properties and features of Ideas, plus a few ones about Automatic Creation (see the "Creating Concepts" section). The main difference with Relationships is that Concepts are designed to represent individual objects, not to link them.

Relationships

A Relationship is an *Idea* subtype, which constitutes an association between *Ideas*, connected using *Links*, forming a nexus. Its visual representation has one Central or Main *Symbol* shaping its body, which maybe hidden, and as many *Connectors* as needed in order to represent the Links to/from/with the related Ideas. Each Relationship type is described by a Relationship Definition.

Relationships can link Ideas in several combinations. A few cases are exemplified in the next diagrams.

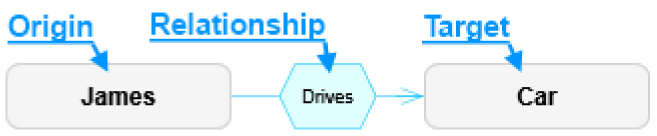
Concept to Concept: 	Concept to Concept (Hiding Central/Main Symbol):  This is a "Simple" Relationship, because one Idea can be related to only one other.
Concept to multiple Concepts:  Actor Performs Character "Hamlet" Character "Othello" Here the "Character" concepts are said to be *Companions* among themselves (sometimes called *Siblings*), because both are Targets of the same Relationship.	Multiple Concepts to one Concept:  Actor "Robert D" Actor "Marlon B" Performs Character "Vito C" Here the "Actor" concepts are said to be *Companions* among themselves (sometimes called *Siblings*), because both are Origins of the same Relationship.
Concept to Relationship (which also could be Relationship to Relationship): 	Concept to itself (Self/Auto-Reference): 
Concept related to Concept, **without direction**: 	Unlinked/isolated Relationship:  This can happen when copying a Relationship, for later append the links to the related Ideas.

Note: The graphic symbols and connectors used, plus their colors and styles, are just for this example and may vary depending in the definitions of the underlying Domain and/or the particular settings the user wants to make.

Directionality

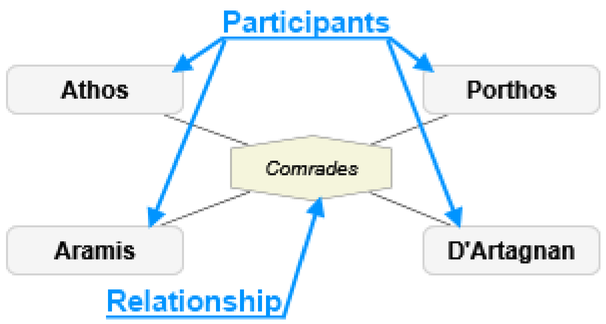
Relationships, depending in their ability to relate Ideas with a sense of direction, can be of these kinds:

- **Directional**: Most Relationships are of this type. They relate Ideas from an origin to a target, or with a similar sense of direction. Example:



Typical Directional relationships are: “Is a”, “As”, “Belongs to”, “Goes To”, “Sends”, “Controls”, “Associated with”, “Represents”, “Implements”, “Better than”, “Builds”, etc.

- **Non-Directional**: They relate Ideas with a sense of participation or other common affiliation, without direction. Example:



Typical Non-Directional Relationships are (for when relating same level Ideas): “Members”, “Players”, “Marriage”, “Siblings”, etc.

Relationship Links

A Link, or *Role Based Link* (not to be confused with Link *Detail*), is a part of a Relationship which makes a nexus to a related Idea, by implementing a *Link-Role*. A Relationship between N Ideas has at least N Links, even when linking the same Idea twice (for Self/Auto-Referencing Relationships).

Depending on the *Directionality* of the Relationship, the Link can implement…

- For Directional Relationship: An **Origin** Link-Role **from** the Idea that originates that Relationship, and implement a **Target** Link-Role **to** the Idea that Relationship points.

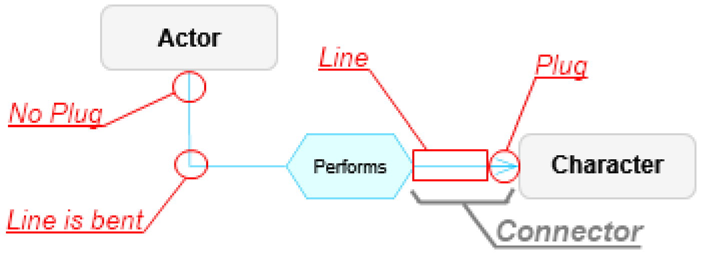
- For Non-Directional Relationship: Only a **Participant** Link-Role for the Ideas **sharing** that Relationship.

If the Relationship is not defined as **Simple**, there can exist more than one Link per Link-Role (i.e.: multiple Origins or multiple Targets). Else, only two links can exist: a single origin and a single target.

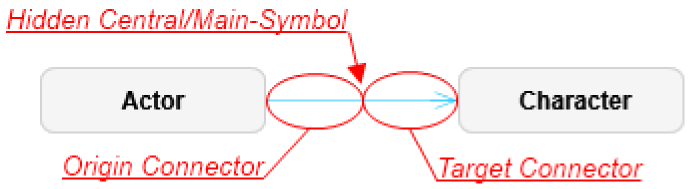
Also, a Link can have an optional **Descriptor**, with the usual *Common Object Properties*, in order to let the user describe it with further detail.

Connectors

A Connector is the visual representation of a Link. It is formed by a **Line**, straight or bent, and a **Plug** joined to the connected Symbol, typically pointing from/to the Symbol's center (not considering its *Details Poster*) unless using precise pointing. The next diagram shows its structure:



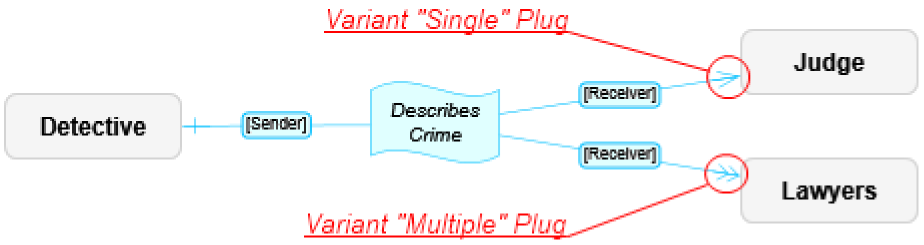
In “Simple” Relationships (those that can link an Idea to only one other) which have their Central/Main-Symbol hidden, they appear like a single line as in the next diagram:



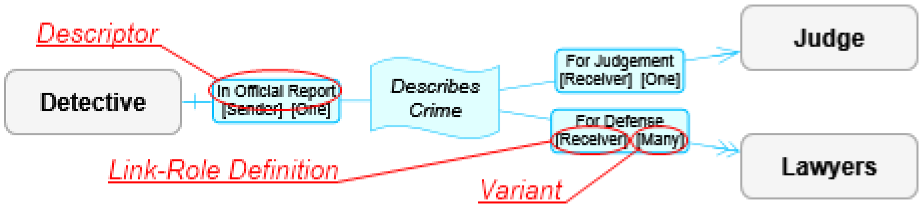
A Connector can also show *Labels* over it, with information about what it represents (the underlying Link-Role, its Definition and Variants). Learn more about that in the *Working with diagram Views* section.

Link-Roles

A Link-Role is the purpose of the nexus expressed by a Link, based on its *Link-Role Definition* and Directionality, plus **Variants** for specialize further more that nexus. For example, a Relationship called “Sending Message” could have an Origin Link-Role called “Sender” and a Target Link-Role called “Receiver”. Also, the Target Link-Role called “Receiver” can have two Variants: “Single” and “Multiple”. This can be better seen in the next diagram:



Notice the *Labels* shown with the name of the used Link-Role Definitions of each Link, “Sender” and “Receiver” in this case, enclosed in square brackets. They also can show the name of the Variant and a possible Descriptor:

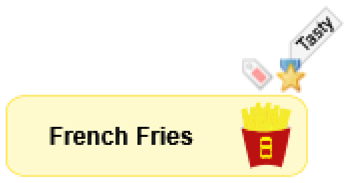


The way of declaring Link-Roles and associate Variants with the available Plugs, is explained in the *Link-Role Definitions* section.

Markers

A Marker (sometimes called "Marking") is a simple visual classifier, such as a "Tag" or "Sticker", defined by a *Marker Definition*. It can be just a graphic icon or also can contain a Descriptor, with the usual *Common Object Properties*, to detail its meaning.

The next diagram shows two Markers appended to an Idea's Symbol, where:



-The "🏷" (Red Tag) Marker is used to denote a "Hot" product, as defined by its Marker Definition, and has no Descriptor in this particular Idea.

- The "⭐" (Gold Star Aware) Marker is used to denote a "Top" product and has a Descriptor indicating that is "Tasty" (shown in a label for its Title/Name).

Complements

A Complement is a visual object for enhancing, by appending text and graphics, the objects of a diagram View.

They **are not a relevant semantic objects** like an Idea (Concept or Relationship), therefore cannot be linked to other objects, have details or any other kind of rich-content despite some of them may be visually attached to Symbols.

The currently available Complements are exposed in the next sections.

Legend

A Legend Complement shows the Idea types used in the target diagram View, plus some Domain information as exposed in the next lines:

Domain: Name/Title of the underlying Domain on which the Composition is based.

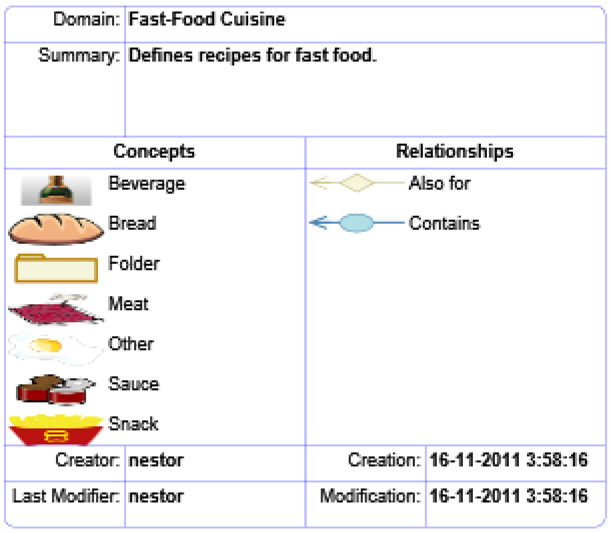
Summary: Summary of the Domain.

Concepts: Used Concepts on the target View.

Relationships: Used Relationships on the target View, including multiple Role Variants.

Versioning Information:
Creator: User who created the Domain.
Creation: Instant of the Domain creation.
Last Modifier: Last user who changed the Domain.
Modification: Instant of the last Domain change.

Sample



Domain:	Fast-Food Cuisine		
Summary:	Defines recipes for fast food.		
Concepts		Relationships	
	Beverage		Also for
	Bread		Contains
	Folder		
	Meat		
	Other		
	Sauce		
	Snack		
Creator:	nestor	Creation:	16-11-2011 3:58:16
Last Modifier:	nestor	Modification:	16-11-2011 3:58:16

Info-Card

An Info-Card Complement shows information about the Composition and target View. It has:

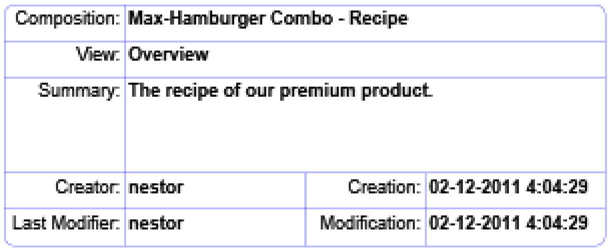
Composition: Name/Title of the Composition.

View: Name/Title of the target View.

Summary: Summary of the Composition.

Versioning Information:
Creator: User who created the Domain.
Creation: Instant of the Domain creation.
Last Modifier: Last user who changed the Domain.
Modification: Instant of the last Domain change.

Sample



Composition:	Max-Hamburger Combo - Recipe		
View:	Overview		
Summary:	The recipe of our premium product.		
Creator:	nestor	Creation:	02-12-2011 4:04:29
Last Modifier:	nestor	Modification:	02-12-2011 4:04:29

Image

An Image Complement is a free "floating" image used as Illustration or Background.

If you need to link to/from an Image, **better use a Concept**, assign its Pictogram and set the "Use Pictogram as Symbol" on its *Symbol Format Definition*.

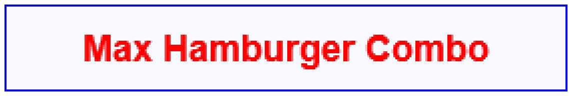
Note: Images can only be manipulated from their borders, because they can be used as background for other visual objects. See the *Working with Complements* section to learn more.

Sample

Text

A Text Complement is a free text, with formatting attributes, used as Title or Label.

Sample

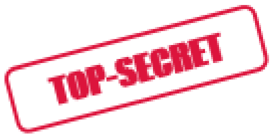


Max Hamburger Combo

Stamp

A Stamp Complement is like a Text Complement, but rotated and enclosed in a rounded-rectangle in order to appear like a rubber stamp.

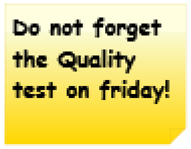
Sample



Note

A Note Complement is also like a Text Complement, but enclosed in a "sticky note" shape.

Sample

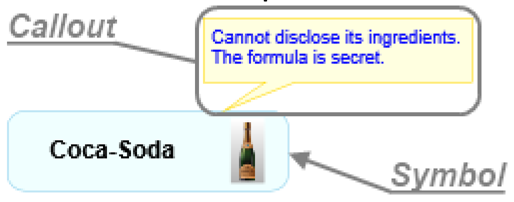


Do not forget the Quality test on friday!

Callout

A Callout is like a Note, but attached to a Symbol with a pointing appendix.

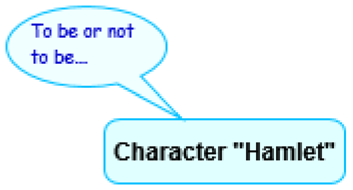
Sample



Quote

A Quote is like a Callout, but representing something said by the pointed Symbol.

Sample

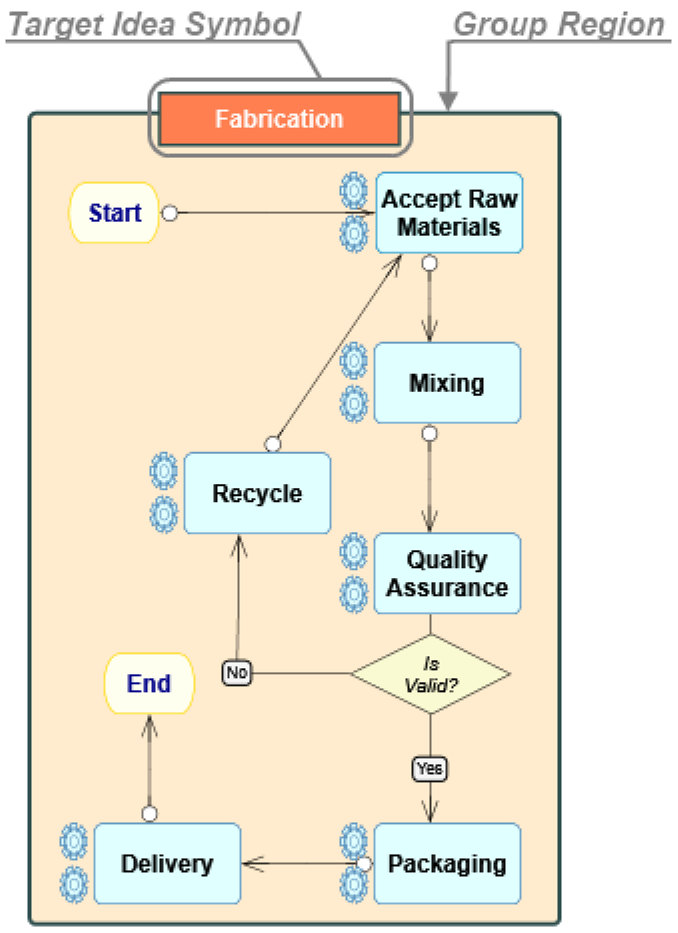


Group Region

A Group Region is a rectangular area, appended under a target Idea Symbol, which serves as a background grouping boundary of visual objects dependents on that Idea.

When the target Idea Symbol is moved, the Group Region is moved along with it. However, the Group Region can be resized individually. In the *Working with Complements* section it is further detailed.

Sample

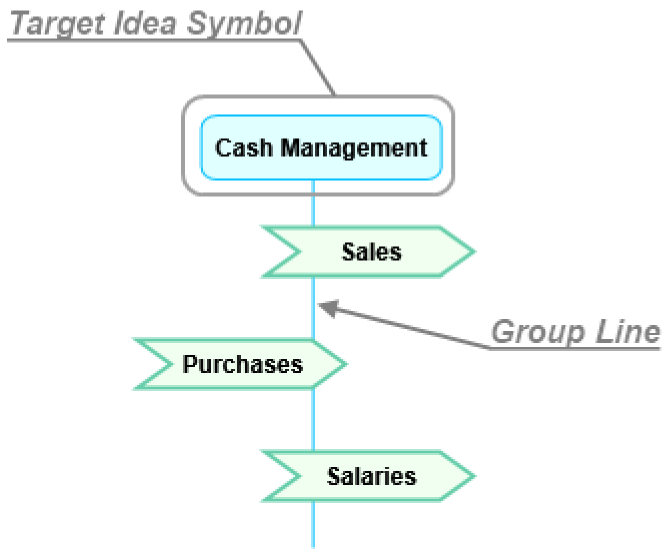


Group Line

Similar to a Group Region, a Group Line is a line appended under a target Idea Symbol, which serves as a background thin area to group by-intersection visual objects dependents on that Idea.

When the target Idea Symbol is moved, the Group Line is moved along with it. However, the Group Line can be resized individually and its axis (vertical or horizontal) can be changed. In the *Working with Complements* section it is further detailed.

Sample

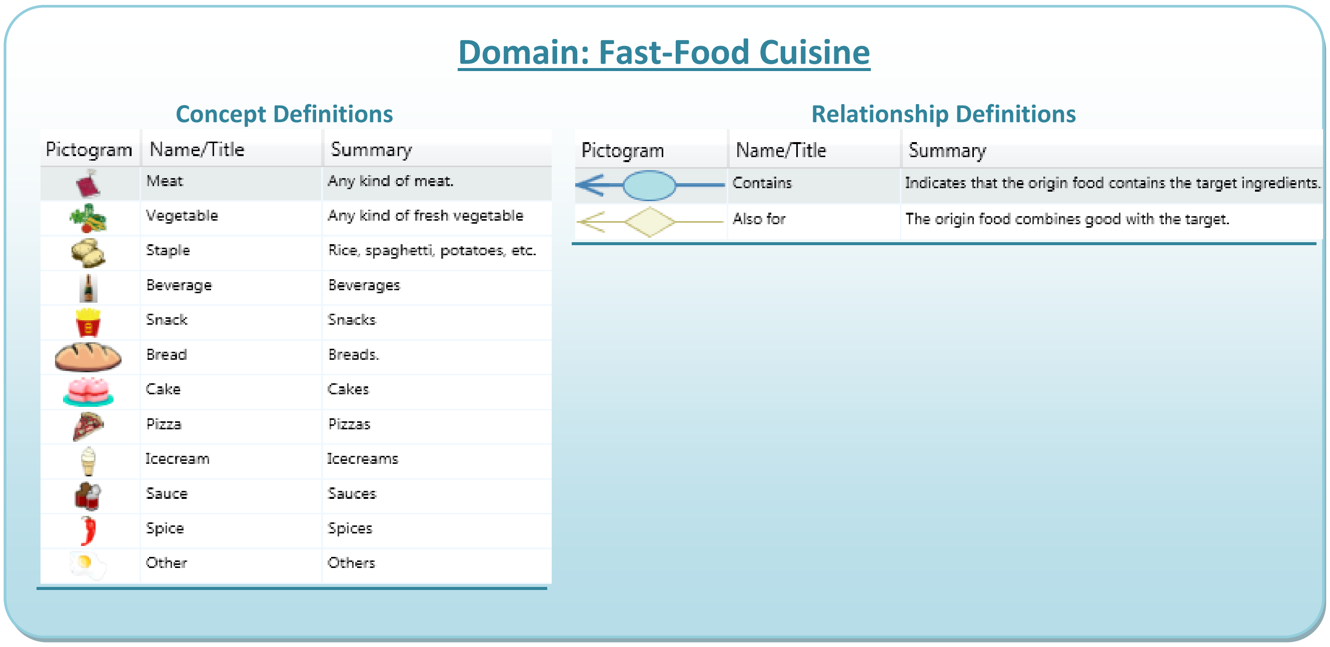


Domains

A *Domain* is a declarative document, containing a set of object definition and rules, describing a particular business or field of knowledge, which later will be expressed in *Composition* documents. These definitions declare graphic properties, information schemas, rules of associativity and other useful descriptors, therefore forming a simple yet expressive *Custom Language*.

Users can create their own Domains or use the preexisting ones provided with the product, including the most little one, the "Basic Domain", which only has one Concept and one Relationship.

Now, consider the next example: Register a recipe for a classic fast-food meal.



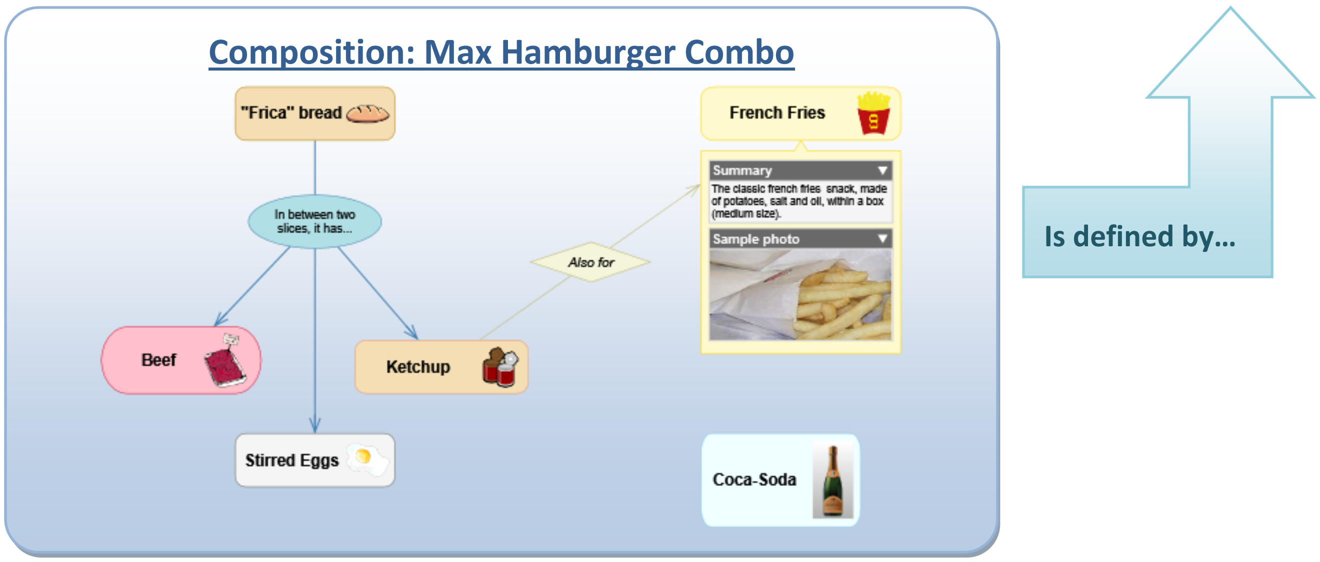
Domain: Fast-Food Cuisine

Concept Definitions

Pictogram	Name/Title	Summary
	Meat	Any kind of meat.
	Vegetable	Any kind of fresh vegetable
	Staple	Rice, spaghetti, potatoes, etc.
	Beverage	Beverages
	Snack	Snacks
	Bread	Breads.
	Cake	Cakes
	Pizza	Pizzas
	Icecream	Icecreams
	Sauce	Sauces
	Spice	Spices
	Other	Others

Relationship Definitions

Pictogram	Name/Title	Summary
	Contains	Indicates that the origin food contains the target ingredients.
	Also for	The origin food combines good with the target.



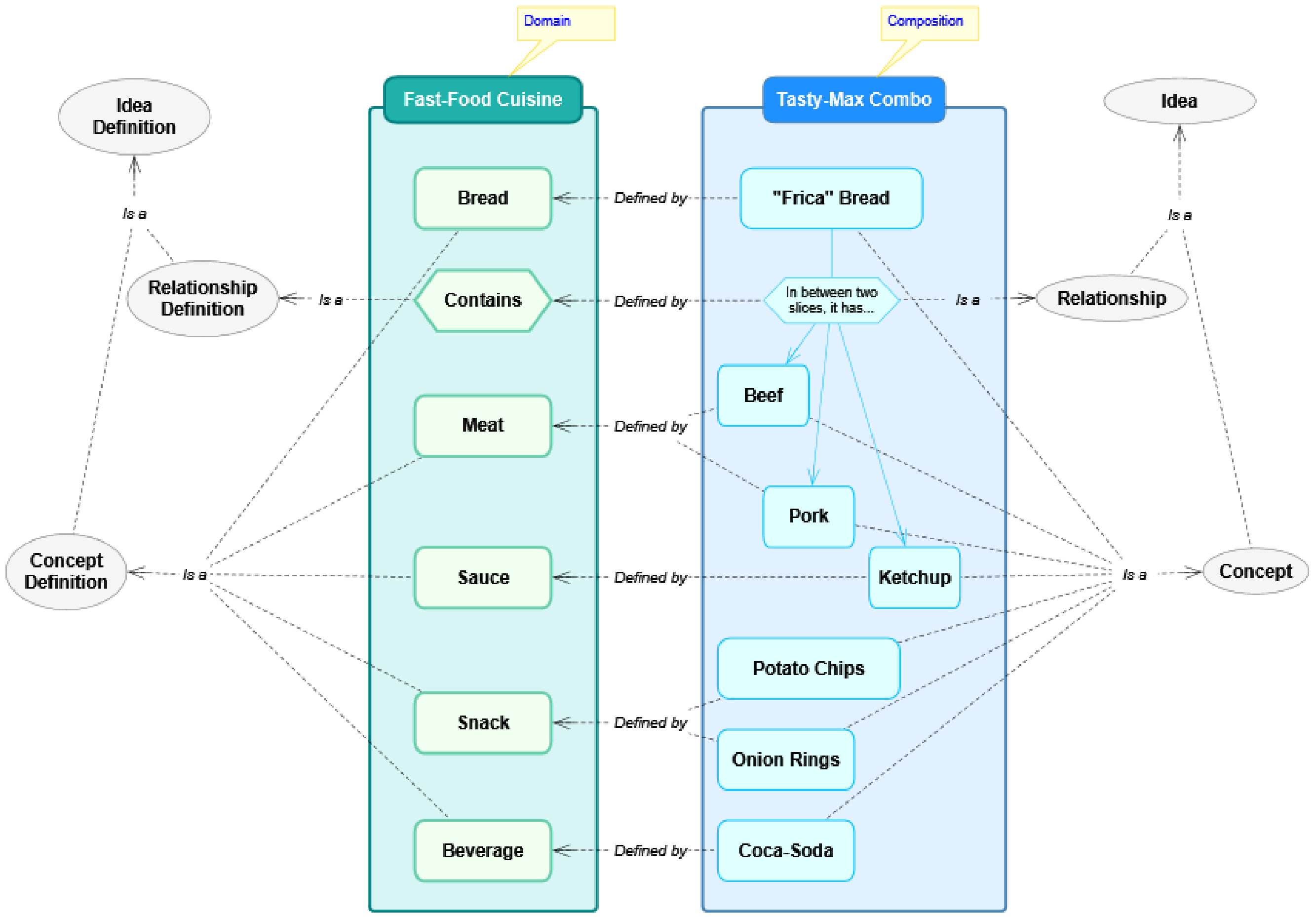
The classic french fries snack, made of potatoes, salt and oil, within a box (medium size).

With these simple ingredients and combinations definitions, belonging to the world (specialized domain) of food, we can document recipes and meals.

Idea Definitions

An *Idea Definition* is either a *Concept Definition* or a *Relationship Definition*; it defines set of properties about types of *Ideas (Concepts or Relationships)* that will be created in *Compositions*.

Reconsider the next sample diagram, shown previously in the *Ideas* section, explaining how *Idea Definitions* and *Ideas* relate.



As it can be seen, the Composition objects (the "Tasty-Max Combo" in this case) are individual *Concepts* or connecting *Relationships*, each one based on its declarative *Concept Definition* or *Relationship Definition*, respectively, belonging them to the base Domain ("Fast-Food Cuisine"). Therefore, your Ideas can have a semantic description, as simple or complete as you want, not being only symbols and lines flying around.

Both, *Concepts Definitions* and *Relationships Definitions* share a set of common features and capabilities, so for example the objects they describe (the final *Concepts* and *Relationships*) can be composed (have a whole diagram inside them), have details (attachments, links and tables) and be shown with customizable visual styles. The next sections will discuss about them.

Properties

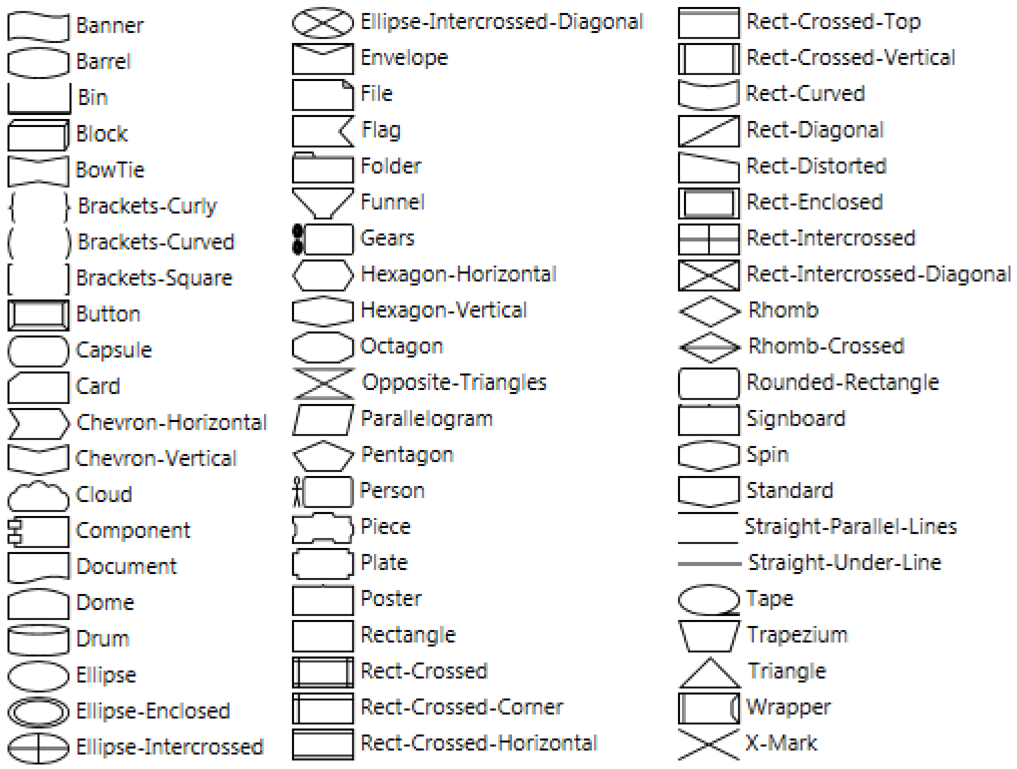
In addition to the standard *Common Object Properties*, Idea Definitions have the next properties:

Is Composable: Indicates that the derived Ideas can be composed of others, within a whole view/diagram contained inside.

Is Versionable: Indicates whether the defined Ideas can maintain versioning information.

Representative Shape: Geometric shape illustrating the Idea definition, to be exposed as the visual symbol of the represented Ideas (either the body symbol for Concepts or the central symbol for Relationships).

The currently available shapes are:



Note: A “<None>” shape also exists for use as non-existent symbol (usual in classic Concept Mapping relationships).

Precise Connect by default: Indicates to connect from/to precise aimed positions inside the Symbol, by default, else from/to the Symbol center. Anyway, you always can perform a precise connect by pressing [Ctrl] while creating a Relationship or re-linking one.

Has Group Region: Indicates whether the derived Ideas are created with a Group Region complement (a boundary) appended.

Has Group Line: Indicates whether the defined Ideas are created with a Group Line complement (like a 'life line') appended.

Can Automatically Create Related Concepts: Indicates whether the Ideas of this type will automatically create related Concepts in editing, by pressing [Enter], [Tab] or dropping Concept Definitions over them (so, you must "bring to front" the target if it is under other one).

Can Group Intersecting Objects: Indicates whether the Ideas of this type will group objects intersecting its symbol or Group Region (which, by default, group objects fully inside them).

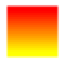
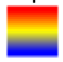
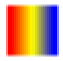
Can Automatically Create Grouped Concepts: Indicates whether the Ideas of this type will automatically create grouped Concepts when linking a Relationship into an appended Group Region/Line.

Automatic Grouped Concept Definition: Definition of the Concept to be automatically created onto an appended Group Region/Line.

Cluster: Cluster to which this Idea-Definition is associated (used for better organization/grouping of the Definitions).

Brushes

A Brush is a style for painting visual content applying colors, gradients and other attributes. Currently, the available properties are:

Property	Sample
Multiplicity: Determines the quantity of colors used, which can be one of these… ○ *None*: The brush is absent, therefore transparent. ○ *Single color*: Only one solid color. ○ *Double gradient colors*: Two colors with intermediate nuances between them. ○ *Triple gradient colors*: Three colors with intermediate nuances between them.	Single color (red) Double gradient colors (red to yellow):  Triple gradient colors (red to yellow to blue): 
Orientation: For gradient colors, it indicates whether the painting is in the horizontal or vertical axis.	Axis changed to vertical: 
Transparency/Opacity level: Specifies the percentage of transparency or opacity to apply.	Transparency/Opacity set to 50%:

Text-Formats

A Text-Format is a set of attributes defining the way text is drawn as visual content. Currently, the available properties are:

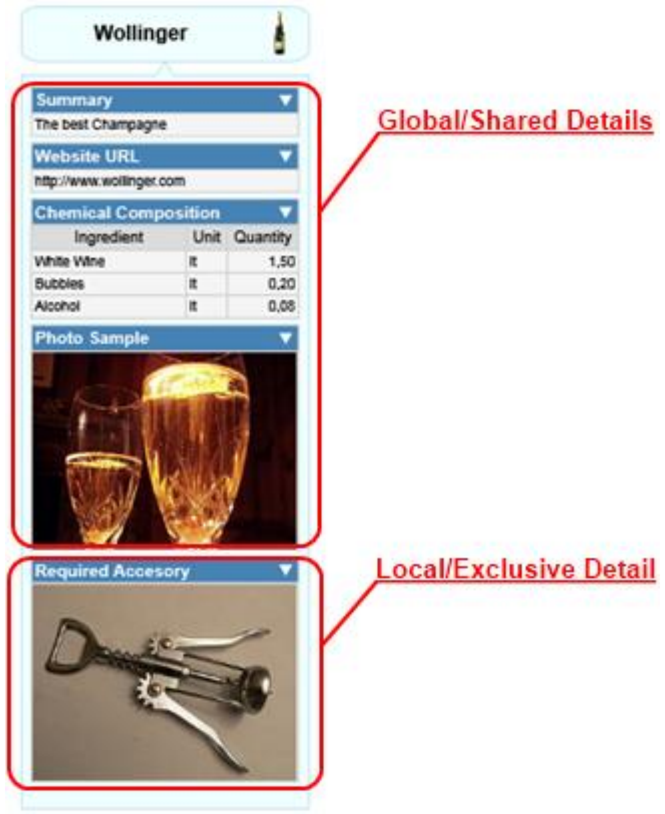
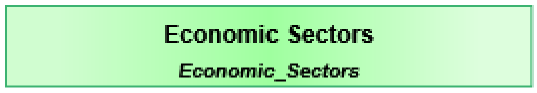
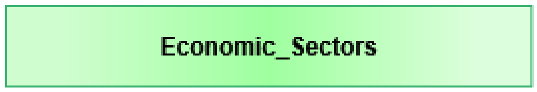
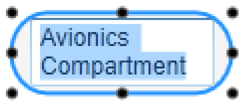
- **Font**: Type-face style of the text. Depends on the fonts installed in your Windows OS.
- **Size**: Dimensions of the text (8 to 256)
- **Style**: Options for alter the text presentation:
    - **Bold**
    - *Italic*
    - Underline
    - ~~Strike-Through~~
- **Alignment**:
    - Left
    - Center
    - Right
    - Justified (text aligned to right and left with intercalated spaces to get an homogeneous presentation)

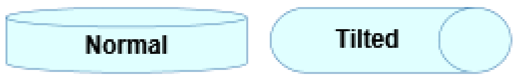
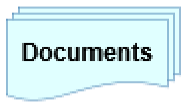
Color brush: Brush applied to the text (see the previous section about *Brushes*). This enables you to even create multicolored text, like the next sample:

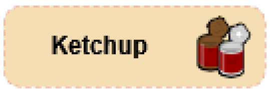
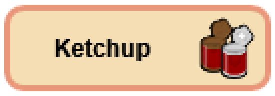
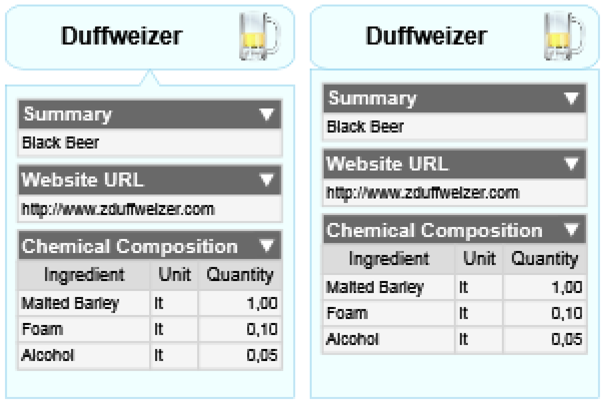
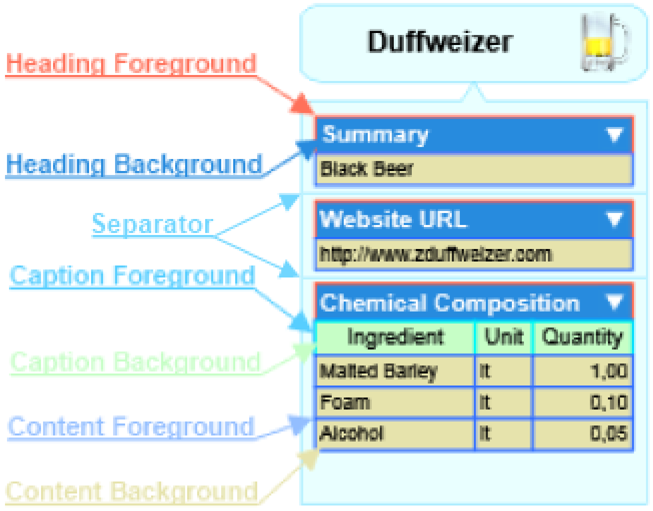
ICE CREAM AND CAKES

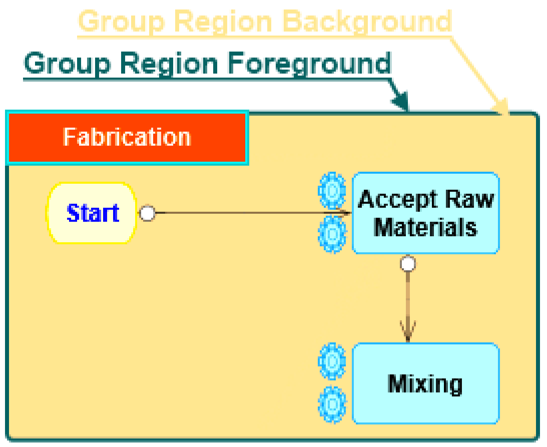
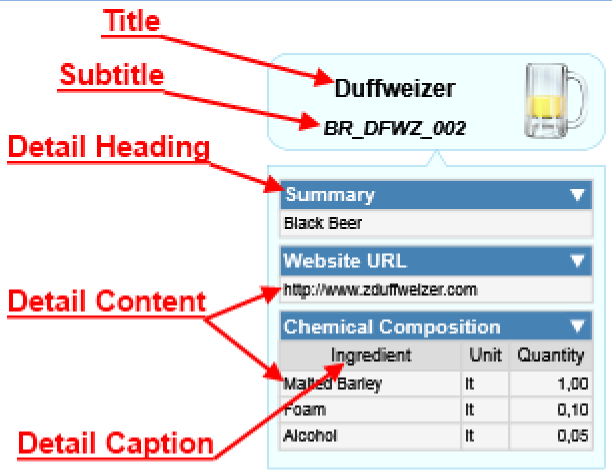
Symbol Format Definition

A Symbol Format Definition is owned by an Idea Definition and declares the graphic style properties for Symbols (and their possible Details Posters) of Ideas based on that Idea Definition. The format is applied at creation time, but later it also can be changed individually while editing (custom formatting). Its properties are:

Property	Sample
Show Global Details First: Indicates whether to show first the global/shared details (those declared on the Idea Definition), else the local/exclusive details (those of a particular Idea) are shown first.	In this case, only the last detail ("Required Accesory") is designated for the "Wollinger" beverage, the previous ones are for all the beverages.  Wollinger Summary The best Champagne Website URL http://www.wollinger.com Chemical Composition Ingredient Unit Quantity White Wine lt 1,50 Bubbles lt 0,20 Alcohol lt 0,08 Photo Sample Global/Shared Details Required Accesory Local/Exclusive Detail
Subtitle Visual Disposition: Indicates the Vertical positioning of the Subtitle (which can be the Name or Tech-Name) respect the Title. Options are: - Before - After - Hidden	The subtitle "Economic_Sectors" (Tech-Name) is shown after the title. 
Use Name as main Title: Indicates whether to use the Name as main title hence the Tech-Name as subtitle, else the Tech-Name as main title and the Name as subtitle.	Title is "Economic_Sectors" (Tech-Name) and the subtitle (hidden) is the Name. This is the usual case when diagramming Tables, Classes or other Computing related stuff. 
In-Place Editing is Multiline: Indicates whether the in-place editing of the symbol main title is multiline, so it accepts [Enter] inside.	

Property	Sample
Pictogram Visual Disposition: Indicates the Horizontal or Vertical positioning of the Pictogram respect the Title/Subtitle in the visual symbol. Options are: - Right (default) - Left - Top - Bottom - Hidden ***Note***: Pictogram is shown only when present.	Pictogram is shown at the Top of the symbol. "Frica" bread
Use Pictogram as Symbol: Show the Pictogram instead of the Symbol. The title (and subtitle) will be shown over it. This is useful to place your own *Representative Shape,* instead of the predefined available ones.	"Frica" bread
Use Definitor Pictogram as Empty-Default: Indicates to use the Pictogram of the Definition when that of the Idea is empty.	<no sample>
Initial Width: Initial width of the symbol, which may be adjusted if "*Snap to Grid*" is on and "*Fixed Width*" is off.	<no sample>
Initial Height: Initial height of the symbol, which may be adjusted if "*Snap to Grid*" is on and "*Fixed Height*" is off.	<no sample>
Fixed Width: Indicates that the symbol has a fixed (initial) width, hence the user cannot resize it.	<no sample>
Fixed Height: Indicates that the symbol has a fixed (initial) height, hence the user cannot resize it.	<no sample>
Initially Flipped Horizontally: Indicates that the symbol is created flipped on its horizontal axis.	<no sample>
Initially Flipped Vertically: Indicates that the symbol is created flipped on its vertical axis.	<no sample>
Tilted: Indicates that the symbol is created tilted 90° clockwise (flip it, to change orientation).	
As Multiple: Indicates that the symbol will be initially shown as multiple ones stacked.	
Opacity: Opacity factor for the element. (0=Transparent). ***Note***: Transparency is the inverse equivalent of Opacity.	Opacity is set to 0.50 (half transparent) for the format of the "Ketchup" Idea. Ketchup Stirred Eggs
Main Background: Brush for the main background of the object.	Main Background "Frica" bread
Line Brush: Foreground brush of the contour lines.	Line Brush "Frica" bread

Property	Sample
Line Dash-Style: Dash style of the lines. Options are: - Dash - Dash-Dot - Dash-Dot-Dot - Dot - Solid (default) - Segmented	Segmented Line style. 
Line Thickness: Thickness of the lines.	Line of Thickness=3. 
Details properties… - **Separate with Lines**: Indicates whether to insert a separator line between shown Details. - **Poster is Hanging**: Indicates whether the Details Poster appears separated and hanging from the head with a triangular hook, else appears joined. - **Heading Foreground**: Brush for the foreground of the Detail heading (and its expander), where the Designated Name/Title of that Detail is shown. - **Heading Background**: Brush for the background of the Detail heading. - **Caption Foreground**: Brush for the foreground of the Captions (only applies for Field names). - **Caption Background**: Brush for the background of the Captions (only applies for Field names). - **Content Foreground**: Brush for the foreground of the Content (only applies for text and Field values) - **Content Background**: Brush for the background of the Content (only applies for text and Field values). ***Note***: These brushes define colorization for background objects (rectangular frames) and foreground lines (for Tables). The brushes for text are defined in the respective Text-Formats, not here.	Here, the first Symbol is shown with its *Details Poster* hanging, and the second has it joined.  Duffweizer Summary Black Beer Website URL http://www.zduffweizer.com Chemical Composition Ingredient Unit Quantity Malted Barley lt 1,00 Foam lt 0,10 Alcohol lt 0,05 Duffweizer Summary Black Beer Website URL http://www.zduffweizer.com Chemical Composition Ingredient Unit Quantity Malted Barley lt 1,00 Foam lt 0,10 Alcohol lt 0,05 The next exaggerated illustration is to show, with precision, which are the described brushes.  Heading Foreground Heading Background Separator Caption Foreground Caption Background Content Foreground Content Background Duffweizer Summary Black Beer Website URL http://www.zduffweizer.com Chemical Composition Ingredient Unit Quantity Malted Barley lt 1,00 Foam lt 0,10 Alcohol lt 0,05

Property	Sample
Group Region/Line properties… - **Background**: Brush for a Group Region Complement fill background. - **Foreground**: Brush for a Group Region Complement line foreground. - **Dash**: Dash style of the Group Region/Line. - **Thickness**: Thickness of the Group Region/Line. - **Top-Border Region Place**: Initial placement for the top-border of the Group Region, respect its owning Symbol.	 Group Region Background Group Region Foreground Fabrication Start Accept Raw Materials Mixing Notice that the Region was placed as *Right Inward* respect the "Fabrication" symbol.
Text Formats of the Symbol: Exposes the Text Format properties for the next items: - **Title**: Word/Phrase text for the main name or title. - **Subtitle**: Word/Phrase text for secondary naming (such as an alias, tech-name or other relevant data). - **Detail Heading**: Detail level Heading, such as the name of a table. - **Detail Caption**: Detail level Caption, such as the name of a table field. - **Detail Content**: Detail level Content, such as data values of a table. - **Extra**: Word text for use as decorator, such as classification data (not currently in use). - **Paragraph**: Abundant text for description or summary (not currently in use).	 Title Subtitle Detail Heading Detail Content Detail Caption Duffweizer BR_DFWZ_002 Summary Black Beer Website URL http://www.zduffweizer.com Chemical Composition Ingredient Unit Quantity Malted Barley lt 1,00 Foam lt 0,10 Alcohol lt 0,05

Details Definitions

A Detail Definition is the same as a *Detail Designation*, that assigns a properly described *Detail* (Attachment, Link or Table), with the difference that is used to pre-define, pre-declare or anticipate a detail that all the Ideas, based on the owner Idea Definition, will (or should) have.

Output-Templates

Useful for *file/code generation*, an Output-Template is a specification for generate files, in an External Language, from Compositions content. The templates themselves are declared using a Template language, referencing properties from the Composition Information Model.

By default, ThinkComposer will create some initial Templates, for generating Text and XML output files, in all Domains. Plus, some Domains have extra templates for their specialized output needs (like the Data-Model and Class Diagram, having SQL and C# output, respectively).

Depending on the generalization or precision intended, the Output-Templates can be defined at two levels:

Domain's templates: Declarations of text templates, at global scope, considering…

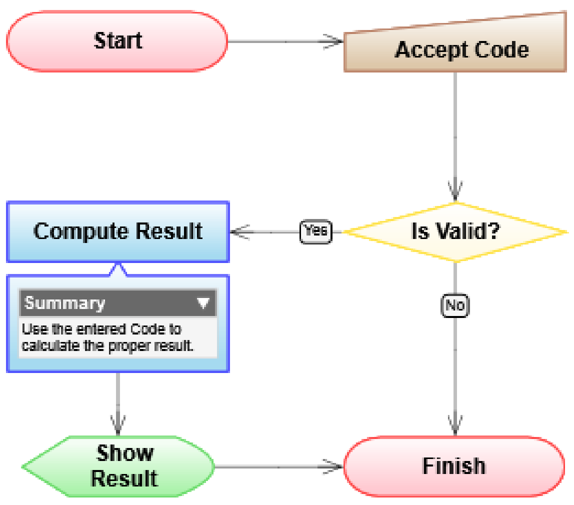
- **Composition templates**: Declares templates for files to be generated for the root of the Composition. This is useful to generate either files that contain the whole Composition generated content or unique central files (e.g.: "readme.txt", "app.config", "create-db.sql", "main.cs", etc.).

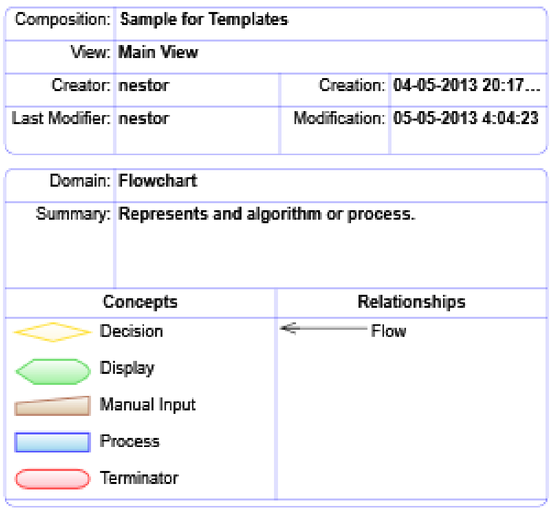
- **Concepts base templates**: Declares templates for files to be generated from Composition Concepts, all producing the same format, regarding their particular Concept Definition. They can be extended by individual Idea Definition's (of Concept Definition kind) templates.

- **Relationship base templates**: Declares templates for files to be generated from Composition Relationships, all producing the same format, regarding their particular Relationship Definition. They can be extended by individual Idea Definition's (of Relationship Definition kind) templates.

- **Idea Definitions' templates**: You can define text templates associated to individual Idea Definitions, therefore allowing the specialization of file generation per Idea Definition types. Also, these templates can extend those of their base kinds (Concepts or Relationships templates) of global scope.

Please consider the next sample Composition…





Composition:	Sample for Templates		
View:	Main View		
Creator:	nestor	Creation:	04-05-2013 20:17…
Last Modifier:	nestor	Modification:	05-05-2013 4:04:23

Domain:	Flowchart
Summary:	Represents and algorithm or process.

Concepts	Relationships
Decision	Flow
Display	
Manual Input	
Process	
Terminator	

Based on that sample, the next table shows the resulting output from the declared Templates.

Declaration Level	Sample Template	Output Result
Domain templates		

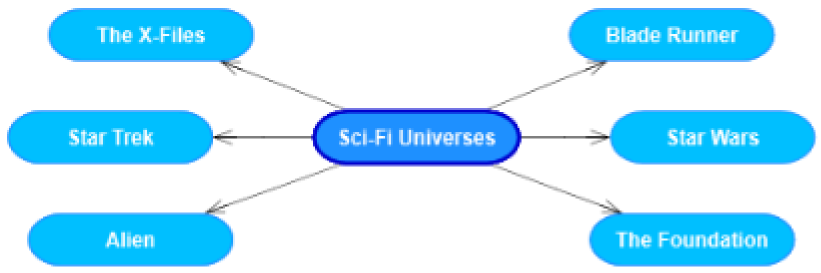
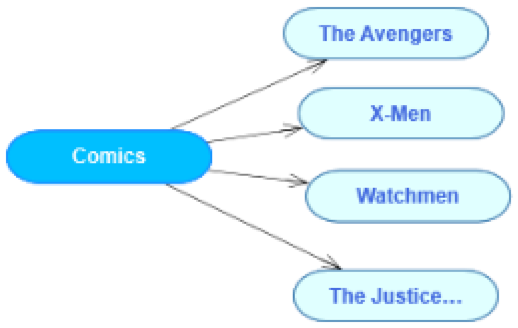
Composition templates	[Flowchart]
	Name: {{ Name }}
	Author: {{ Version.Creator }}
	[End-Flowchart]
		Generated Composition file:
		[Flowchart]
		Name: Sample for Templates
		Author: nestor
		[End-Flowchart]

Concepts templates	[Node] Name: {{ Name }} [End-Node]	**Generated Concept files**: [Node] Name: Start [End-Node] … [Node] Name: Accept Code [End-Node] … [Node] Name: Is Valid? [End-Node] … [Node] Name: Show Result [End-Node] … [Node] Name: Finish [End-Node]
Relationship templates	<None>	<None>

Idea Definition
(of the 'Process' Concept Definition)
	[Process]
	Name: {{ Name }}
	Summary: {{ Summary }}
	[End-Process]
		Generated Concept file:
		[Process]
		Name: Compute Result
		Summary: Use the entered Code to calculate the proper result.
		[End-Process]

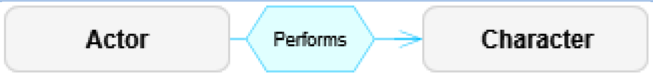
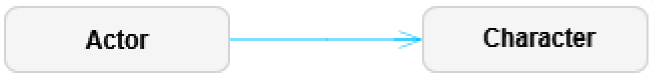
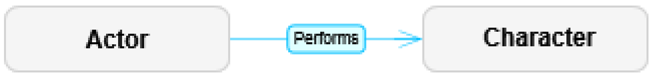
Concept Definitions

A Concept Definition is an *Idea* Definition subtype, which describes types of Concepts to be created based on it. Therefore, they have all the properties an Idea Definition has, plus the next ones about **Automatic Creation Parameters**:

Property	Sample
Definition of Concept to Create: Definition of the Concept to be automatically created.	<no sample>
Associating Relationship Definition: Definition of the Relationship to associate Concepts with the automatically created ones.	<no sample>
Positioning Mode: Indicates the ways to accommodate the symbols of automatically created Concepts. The available options for tree positioning are: - **Horizontal Alternated**: Horizontal tree, with nodes alternating at two rows on top and down sides. - **Vertical Alternated**: Vertical tree, with nodes alternating at two columns on left and right sides. - **To Bottom**: Tree with nodes to the Bottom direction. - **To Right**: Tree with nodes to the Right direction. - **To Up**: Tree with nodes to the Up direction. - **To Left**: Tree with nodes to the Left direction.	Concept "Sci-Fi Universes" with its automatically created children in a vertically alternated tree (also radial):  The X-Files Blade Runner Star Trek Sci-Fi Universes Star Wars Alien The Foundation
Positioning is Radial: Indicates to position automatically created Concepts around in a radial (semi elliptical) style.	Concept "Comics" with its automatically created children in a radial way (to the right):  The Avengers X-Men Comics Watchmen The Justice...

Relationship Definitions

A Relationship Definition is an *Idea Definition* subtype, which describes types of *Relationships* to be created based on it. Therefore, they have all the properties an Idea Definition has, plus the next ones:

Property	Sample
Is Simple: Indicates that only one target and one source Links can be established.	
Hide Central/Main-Symbol When Simple: Hides the Central/Main-Symbol when the Relationship is Simple.	
Show Name if Hiding Central/Main-Symbol: Indicates to show Relationship name, in a simple label, when hiding the Central/Main-Symbol.	

Link-Role Definitions

A Link-Role Definition declares the purpose and constraints of all *Links* implementing it. By the *Directionality* nature of the linking, a Link-Role can be of one of two types:

- **Origin/Participant**: When a derived link is the Origin in a *Directional* Relationship, or when is a Participant in a *Non-Directional* Relationship (where the linked Ideas are associated in equality).
- **Target**: When a derived link is the Target in a *Directional* Relationship.

The properties that describes and constraint how the Links, derived from it, are:

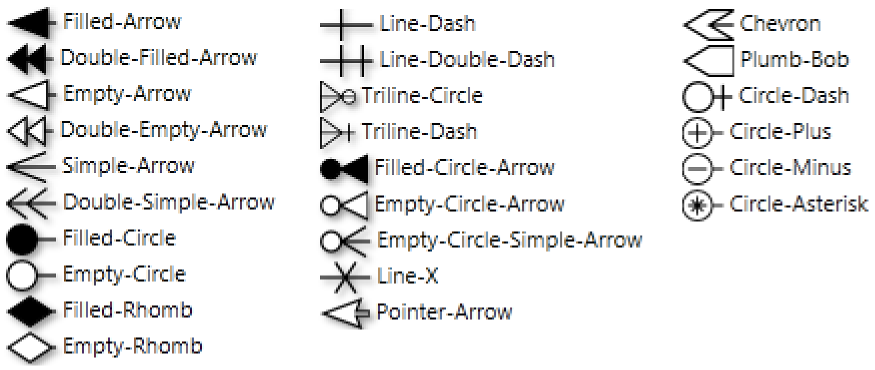
Relationship is Directional (only for the Target Link-Role Definition): Indicates that the derived Relationships will be *Directional* (with both, Origin and Target roles), else will be *Non-Directional* (only with the Participant role).

Name/Title: Identification for the Link-Role Definition.

Summary: Description of the Link-Role Definition.

Allowed Variants: Set of Variants (declared by *Variant Definitions*), available for multi-selection, to indicate which of these can be used for linking. Also, for each selected Variant, a *Connector* Plug can be assigned to be the visual representation of it. Go to the Link-Roles section to learn more.

Currently, the available plugs are:



Note: A “<None>” plug also exists for use when the line itself connects to/from the symbol.

Linkable Idea Definitions: Set of Idea Definitions, available for multi-selection, to indicate which of these can be linked to/from/with. Also, exists the alternative option "All Idea-Definitions are linkable" (selected by default).

This is useful when you want to link certain types of ideas to other that make sense. For example, you may want to associate "ships have passengers" and "passengers live-in houses", but no "ships live-in houses".

Connectors Format Definition

A Connectors Format Definition is owned by a Relationship Definition and declares the graphic style properties for Connectors, of any Role, of Relationships based on that Relationship Definition. The format is applied at creation time, but later it also can be changed individually while editing (custom formatting).

Its properties are *Opacity, Main Background* (for the Connector's Label)*, Line Brush, Line Dash-Style and Line Thickness*, which are the same as those of the *Symbol Format Definition* but applied to the Connector's lines. Plus, the next specific properties are also available:

- **Label Link Variant**: Indicates to label the Link (role) with the Variant name over the Connector.
- **Label Link Definitor**: Indicates to label the Link with the Definition name over the Connector.
- **Label Link Descriptor**: Indicates to label the Link with the Descriptor name over the Connector (if exists).

Variant Definitions

A Variant Definition declares a Variant (see the *Link-Roles* section to learn more), available to be assigned in a *Link-Role Definition*. It is simply described by the standard *Common Object Properties*.

The initial set of provided Variants are:

- "Link": General-purpose linking, which is the selected by default.
- "1..1": Multiplicity of one-only occurrence (e.g.: "a child has only one mother").
- "0..1": Multiplicity of zero-or-one occurrence (e.g.: "a ship can have none or one captain, no more").
- "1..N": Multiplicity of one-or-unlimited occurrences (e.g.: "a mother can have one or many children").
- "0..1": Multiplicity of zero-or-unlimited occurrences (e.g.: "a city can have zero or undetermined semaphores").

Marker Definitions

A Marker Definition declares a *Marker* available to be appended to an Idea. It is simply described by the standard *Common Object Properties*.

By default, there are a lot of predefined Markers in a Domain. However you can change them or create new ones (will be appended in the "User Defined" category).

Table-Structure Definitions

A Table-Structure Definition declares how Tables derived from it are constituted. It is described by the standard Common Object Properties, a rich-text *Description* and the *Structure*, in turn composed of:

- **Field Definitions**: Declares the fields that all derived Table Records must have. Each declared by the standard Common Object Properties, a rich-text *Description* and the following properties:
    - **Hide in Diagram**: Indicates that the field values must be hidden in the diagram View.
    - **Field Type**: Data type of the values to be stored in the derived fields. It constraints what information can be entered in fields. Currently, the available data types are:

Data Type	Description	Examples
Text	Accepts simple plain Text values (i.e. for names or codes), with a limit of 255 characters.	Product Name
Text-Long	Accepts long plain Text values (i.e. for summaries), with a limit of 50,000 characters.	Description of the many features of the product
Number	Accepts Numeric values (up to 10 integer digits, with 5 decimals, positive or negative).	-247.51, 301.29, -500.3
Positive	Accepts Positive values (up to 10 integer digits, with 5 decimals, positive only).	247.51, 301.29, 500.3
Integer	Accepts Integer values (up to 15 integer digits, with 0 decimals, positive or negative).	-247, 301, -500
Positive-Integer	Accepts Positive Integer values (up to 15 integer digits, with 0 decimals, positive only).	247, 301, 500
Date	Accepts Date values	Dec-21-2011
Time	Accepts Time values	14:35:27
Date-Time	Accepts Date-Time values	Dec-21-2011 14:35:27
Switch	Accepts only one of two values: Yes or No (which can be interpreted as true/false, on/off, etc.).	Yes, No
Idea Reference	References a Composition Idea.	Field "Previous Position" references the Idea "R&D Manager", for the "CEO" Position, in an Organizational Chart.
Table-Record Reference	References a Record of a Base Table.	Field "Unit of Measure" references the Record "Oz - –unce", in the Table "Units".
Table	Contains a nested Table.	Field "Studies" contains a Table having "University", "Degree" and "Year" fields.
Picture	Stores a photograph or graphic representation.	A product image.

- **Source Base Table**: Table of predefined available Records, from which…
    - Select the one to be referenced. This is valid only when the Field Type is "Table-Record Reference". See the *Base Tables* section to learn more. Or, …
    - Select a Record from which a value can be obtained into the target field. This is valid only when the Field Type is "Text" or any Numeric. By default, it selects the defined *Label* field or the first compatible.

- **Idea Referencing By:** If set, indicates the property used to Reference Ideas from this field. Available only when the data-type of the field is "Idea Reference" or "Text" (in this case, the Idea is referenced by its containment route, e.g.: "\acme company\sales area\mail delivery").

- **Table-Structure**: Assigns the Table-Structure Definition, either referencing an already existent one or locally defining it, which specifies the fields to be stored in the contained (nested) Tables of this field.

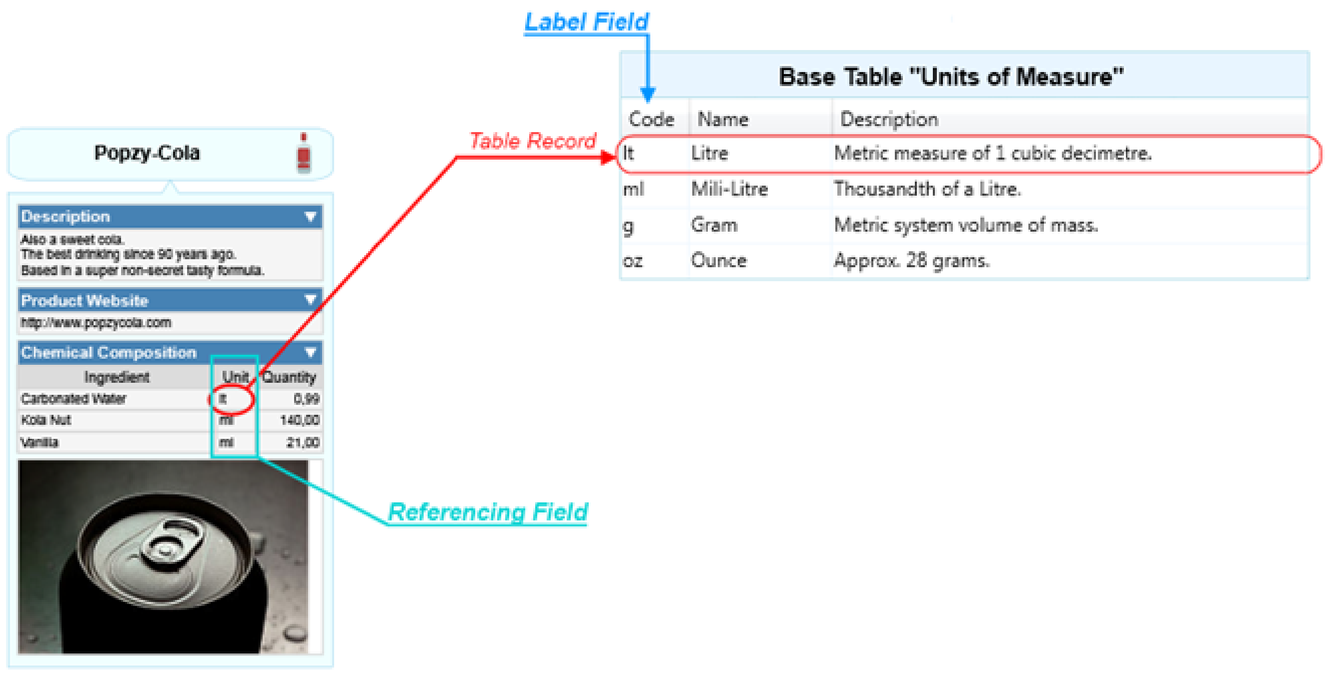
- **Label**: Set of Field Definitions, which will be concatenated as *Label* for the derived Table Records. For example, if a Table "Customers" is defined with Fields "Id", "Name", "Address", "State", etc… the Label could be "Id" + "Name". It is useful when showing a Table Record, to avoid expose all the fields.

By default, each Domain is created with a "Standard" Table-Structure for General purpose, having ID, Name and Description properties.

Base Tables

A Base Table is just a detail *[Table](#)*, at the Domain level, which stores a set of predefined Records to be referenced from Tables created in the derived Compositions. As each Table, they have a designated name and a declaring Table-Structure Definition upon which is based.

Consider the next illustration:



Base Table "Units of Measure"

Code	Name	Description
lt	Litre	Metric measure of 1 cubic decimetre.
ml	Mili-Litre	Thousandth of a Litre.
g	Gram	Metric system volume of mass.
oz	Ounce	Approx. 28 grams.

Popzy-Cola

Description

Also a sweet cola.
The best drinking since 90 years ago.
Based in a super non-secret tasty formula.

Product Website

http://www.popzycola.com

Chemical Composition

Ingredient	Unit	Quantity
Carbonated Water	lt	0,99
Kola Nut	ml	140,00
Vanilla	ml	21,00

As you can see in the “Chemical Composition” Detail Table, each “Unit” Field references a Record in the “Units of Measure” Base Table and its Label is just the “Code” Field.

External Languages

An External Language represents a type of structured text files, usually implementing an industry-standard or custom format, intended to be consumed outside ThinkComposer. Later, the content of Compositions can be merged into generated files, based on Output-Templates of these languages.

Application Guide

Setup

Requirements

Modern *Windows* based PC computers are capable of executing ThinkComposer, as long as they meet the next requirements:

Component	Requirement
Processor	x86-compatible at 2.0 GHz or more
Memory	2 GB of RAM or more
Disk Space	500MB free
Operating System	Windows 7, Vista, XP (SP 3) or new versions compatible with Windows Presentation Foundation (WPF)
Video Card	VGA compatible with minimum resolution of 1024x768 pixels, 1600x1200 or more is recommended

Plus, the Microsoft .NET Framework 4.0 must be preinstalled. If not already present in your PC, you can download it from...

http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=17851

Install and Uninstall

You can download the latest versions of the product and documentation from…

http://www.thinkcomposer.com/Home/Download

The installer guides you thru a few steps to complete the setup and has no relevant options, except you can choose to install the product only for the current user or for all the users of the target machine.

For uninstall the application, just go to the Windows Start Menu, then (All) Programs, then “Instrumind ThinkComposer” and click the “Uninstall.bat” program.

Version Update

Version is checked on each start. If not up to date, the application asks if you want to download and install the new version (if so selected, the new installer will be downloaded and started while the application quits).

It is also possible to update from the “*Version Update*” button in the “*About*” window.

License Activation

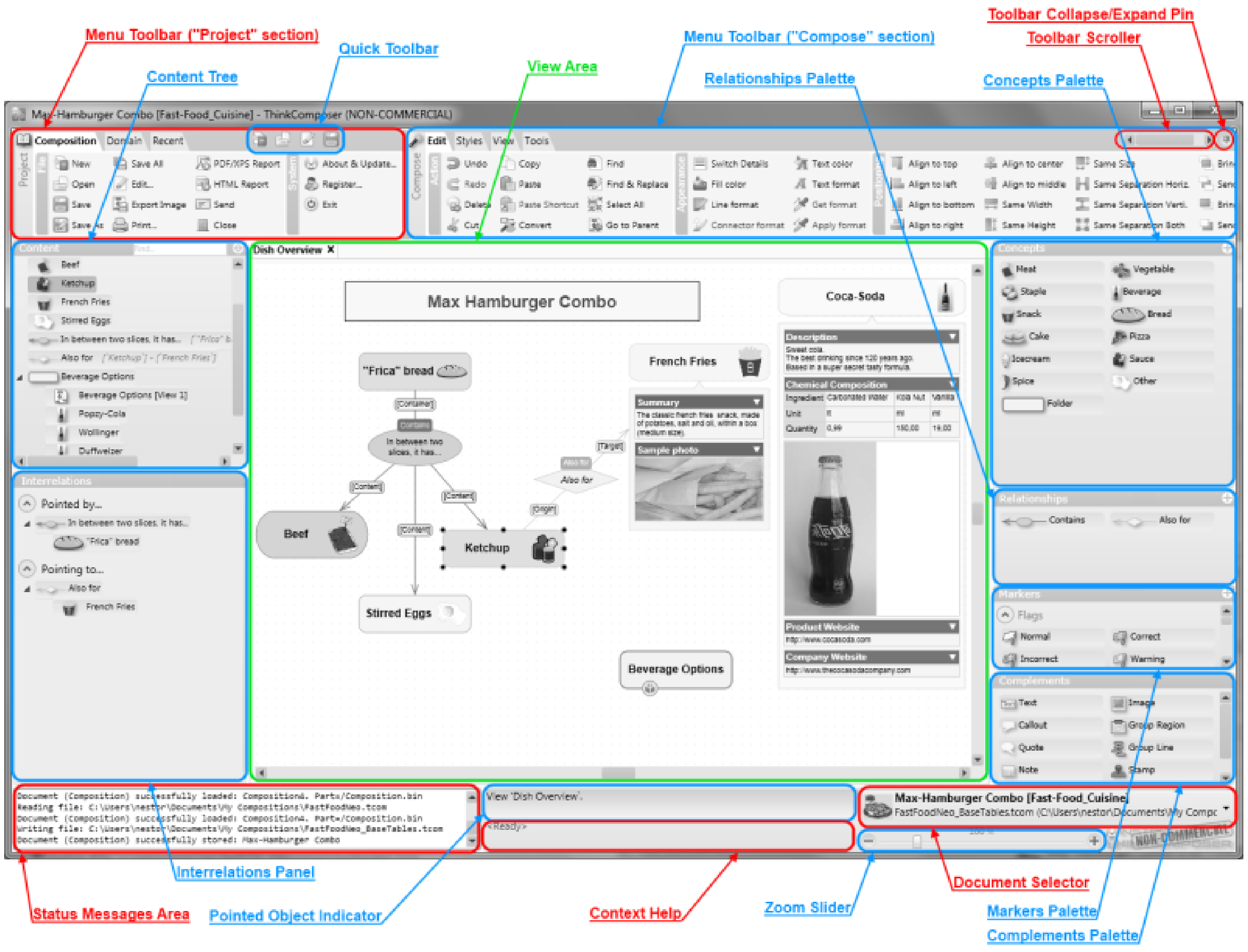
After installation, the application remains in “Trial” mode for 30 days, and then expires. For upgrade to a new License, you must register it in the “Register License” window by either loading a license file or pasting a license key from the clipboard.

User Interface

ThinkComposer is a visual tool designed for modern desktop PC workstations and advanced laptop computers, capable of display high-definition graphics in wide screens. It takes advantage of the features provided by the Windows Presentation Foundation (WPF) technology.

Main Window

The Main Window of the application is shown in the next illustration, indicating its parts:



```
Document (Composition) successfully loaded: Composition4. Parts/Composition.bin
Reading file: C:\Users\nestor\Documents\My Compositions\FastFoodNeo.tcom
Document (composition) successfully loaded: Composition4. Part=/Composition.bin
Writing file: C:\Users\nestor\Documents\My Compositions\FastFoodNeo_BaseTables.tcom
Document (Composition) successfully stored: Max-Hamburger Combo
```

View 'Dish Overview'.

<Ready>

Max-Hamburger Combo [Fast-Food_Cuisine]
FastFoodNeo_BaseTables.tcom (C:\Users\nestor\Documents\My Compo

Menu Toolbar: Area exposing buttons, lists or other controls, which execute the commands of the application. It is divided in two sections: "Project" for the global actions and "Compose" for those related to the editing of the current Composition.

Toolbar Collapse/Expand Pin: Toggles between collapsed or expanded state of the whole Menu Toolbar.

Toolbar Scroller: Displaces the content of the Menu Toolbar "Compose" section when space is insufficient.

Quick Toolbar: Set of command buttons always visible, for fast access even when the Menu Toolbar is collapsed.

Content Tree: Nested list of the *Ideas* (Concepts and Relationships) of the *Composition*. It can be used to go-to a specified Idea, by opening its containing *View*, or to drag and drop it on the current View to create Shortcuts. Click the "⊜" button to alternately sort by Name or Creation instant. Enter text in the "Find" box, and press [Tab] to find Ideas having the specified text in their name (they will be selected).

Interrelations Panel: Shows the interrelations of the View's pointed Idea, including a "*Pointed by…*" super-tree (hierarchy of Ideas pointing **to** the selected one) and a "*Pointing to…*" sub-tree (hierarchy of Ideas pointed **from** the selected one).

View Area: Shows and edits the content of the main (root) diagram *View* of the Composition, or one of its nested composed Ideas.

Concepts Palette: Shows the available *Concept Definitions*, defined in the base *Domain* of the Composition, for creating new *Concepts* in the diagram View by drag and drop. Double-click for edit the Definition. Click the "⊕" button to create a new Definition.

Relationships Palette: Shows the available *Relationship Definitions*, defined in the base *Domain* of the Composition, for creating new *Relationships* in the diagram View by drag and drop. Double-click for edit the Definition. Click the "⊕" button to create a new Definition.

Markers Palette: Shows the available *Marker Definitions*, defined in the base *Domain* of the Composition, for creating new *Markers* in the diagram View by drag and drop. Double-click for edit the Definition. Click the "⊕" button to create a new Definition.

Complements Palette: Shows the available *Complements*, for create in the diagram View by drag and drop.

Status Message Area: Shows messages about the current application parameters, plus the commands being performed and their success of failure to execute.

Pointed Object Indicator: Shows a description of the diagram View's pointed object (that under the moving mouse pointer, not necessarily the selected one).

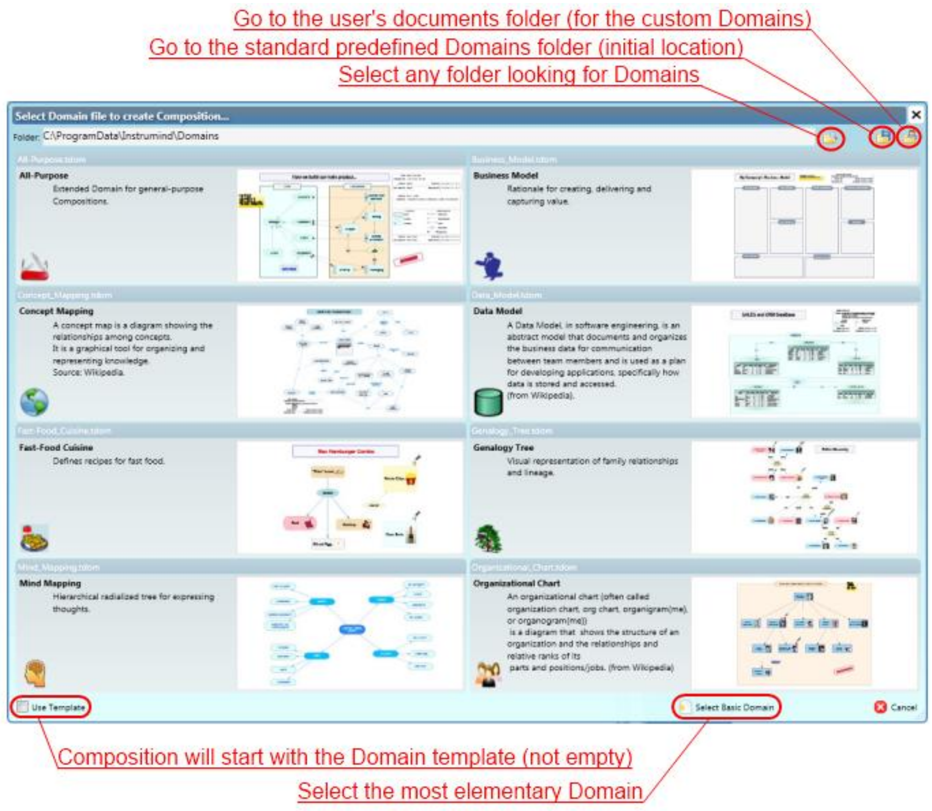
Context Help: Shows appropriate help tips about how to manipulate or use the object pointed by the mouse.

Document Selector: Shows a list of the currently opened documents (Compositions or Domains), which the user can select to change.

Zoom Slider: Shows and changes the scale/zoom on which the current diagram View is displayed.

Working with Compositions

Typically, you start your work creating a new Composition based on a predefined Domain. For that purpose, the next window is provided:


Go to the user's documents folder (for the custom Domains)
Go to the standard predefined Domains folder (initial location)
Select any folder looking for Domains
Use Template
Select Basic Domain
Cancel
Composition will start with the Domain template (not empty)
Select the most elementary Domain

Select Domain file to create Composition...

Folder: C:\ProgramData\Instrumind\Domains

All-Purpose.tdom
All-Purpose
Extended Domain for general-purpose Compositions.

Business_Model.tdom
Business Model
Rationale for creating, delivering and capturing value.

Concept_Mapping.tdom
Concept Mapping
A concept map is a diagram showing the relationships among concepts.
It is a graphical tool for organizing and representing knowledge.
Source: Wikipedia.

Data_Model.tdom
Data Model
A Data Model, in software engineering, is an abstract model that documents and organizes the business data for communication between team members and is used as a plan for developing applications, specifically how data is stored and accessed.
(from Wikipedia).

Fast-Food_Cuisine.tdom
Fast-Food Cuisine
Defines recipes for fast food.

Genalogy_Tree.tdom
Genalogy Tree
Visual representation of family relationships and lineage.

Mind_Mapping.tdom
Mind Mapping
Hierarchical radialized tree for expressing thoughts.

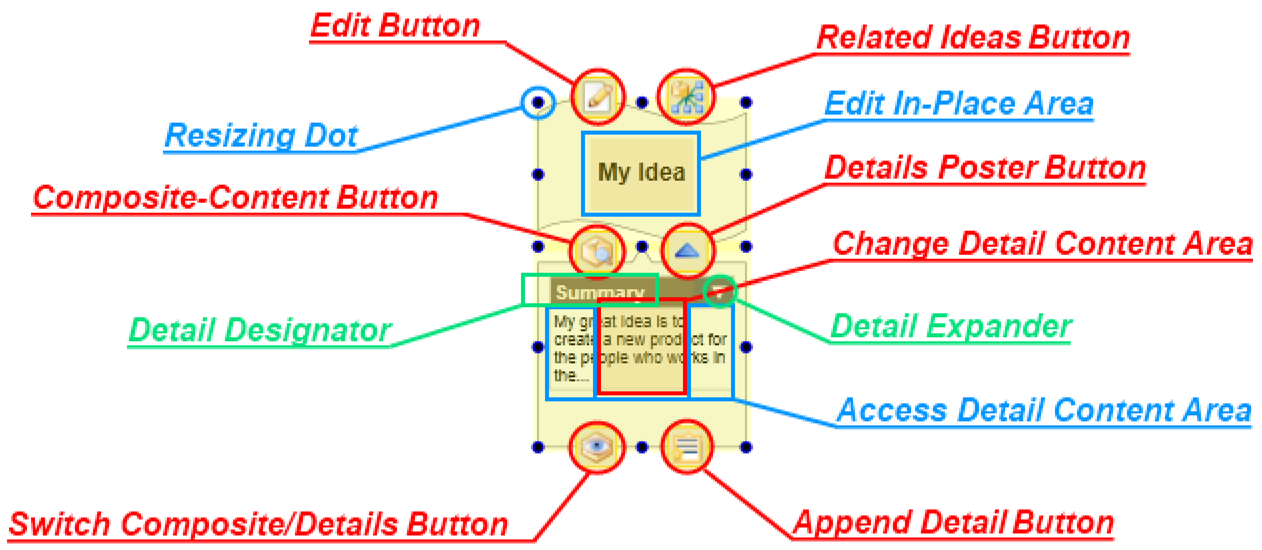
Organizational_Chart.tdom
Organizational Chart
An organizational chart (often called organization chart, org chart, organigram(me), or organogram(me))
is a diagram that shows the structure of an organization and the relationships and relative ranks of its
parts and positions/jobs. (from Wikipedia)

You can choose one of the Domains available or, if none is appropriate for what you want, then select the "Basic Domain", which will start the Composition based on the most simple and elementary Domain (with just one Concept and one Relationship defined), so you can extend and customize it for your specific purpose later.

Also, you can choose to start with the Composition template included in the selected Domain (if any), else the Composition will start empty.

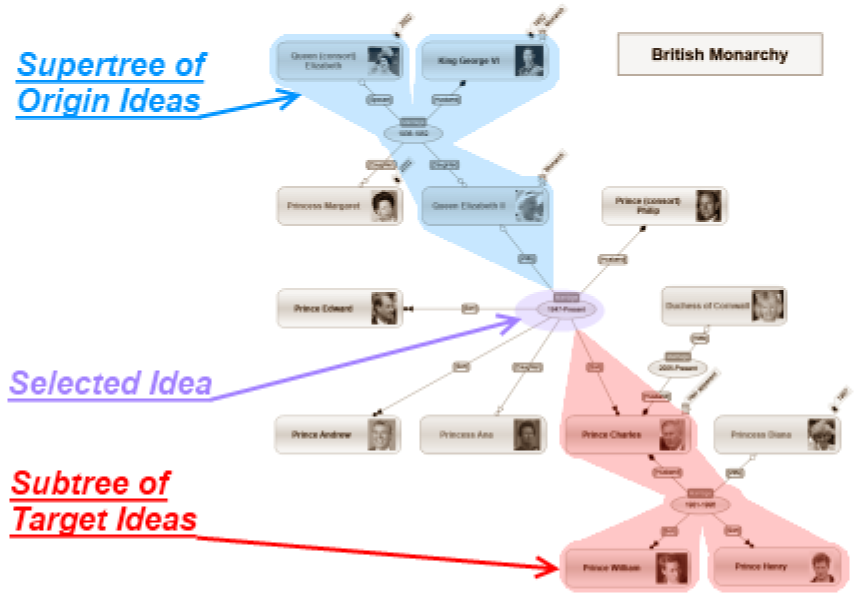
Editing Symbols

When creating Ideas (Concepts or Relationships) they have a Symbol for showing its Name/Title (*), which can be edited directly by the next set of controls:



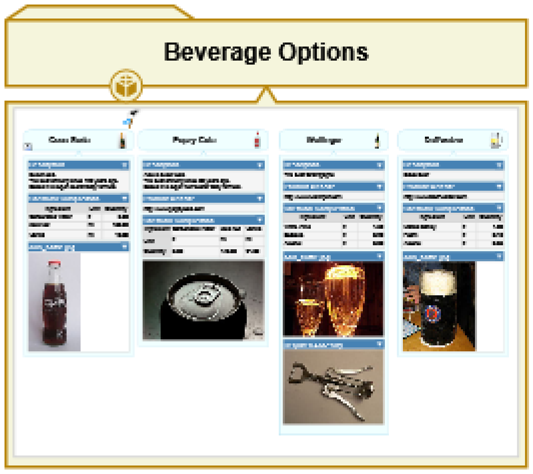
The specific functions of each of these controls and areas are:

- **Edit Button**: Opens the standard Edit Properties window.
- **Related Ideas Button**: Expands/collapses the Symbol related targeted Ideas sub-tree. Press [Alt-Left] to do the same over the origin Ideas super-tree.
    - Example:



- **Composite-Content Button**: Opens, in a new tab on the View Area, the contained Composite-Content View of the underlying Idea of the Symbol.

- **Details Poster Button**: Expands/collapses the Details Poster.
- **Switch Composite/Details Button**: Toggles between showing Details (default) or the Composite-Content View in the Details Poster.
    - Example:



- **Append Detail Button**: Shows a menu for selecting the detail kind to create: Link, Attachment or Table.
- **Resizing Dot**: Resizes the Symbol while dragging with the [Mouse-Left/Normal-Button] pressed.
- **Edit In-Place Area**: Click it to directly edit the underlying Idea Name/Title. Consider the next keyboard based behavior:
    - If the symbol's Name/Title is not configured as multi-line, then press [Enter] to accept the text. To insert a new-line within (therefore making a multi-line text) press [Alt]+[Enter].
    - If the symbol's Name/Title is already configured as multi-line, pressing [Enter] will insert a new-line within the text. Press [Ctrl]+[Enter] to finish the edit.
    - Press [Tab] to accept the entered text, or press [Shift]+[Tab] to insert a tab character.
    - Press [Esc] to discard the entered text.
- **Detail Designator**: Shows/edits the name given to the associated detail. For Tables, also allows to change the base Table-Structure.
- **Detail Expander**: Expands/Collapses its associated detail content.
- **Change Detail Content Area**: Reassigns what object is stored or referenced as detail content. For Links enables the change of the type and address of the referenced object, for Attachments allows the change of the stored file (not its content), for Tables presents the editing grid.
- **Access Detail Content Area**: Presents/edits the detail content depending on its type. For Links goes to the stored address, for Attachments invokes the associated application in the Operating System, for Tables presents the editing grid.
- **Any other area**: Moves** the Symbol while dragging with pressing the [Mouse-Left/Normal-Button]. Also, press [Ctrl] to include the related Target Ideas sub-tree and [Shift] to include the related Origin Ideas super-tree. Plus, on Relationships, pressing [Right-Alt] will lock the position of the Symbol (therefore no automatic repositioning will be performed as result of moving the connected Symbols of that Relationship).

*****: The symbol implements the visual body (for Concepts) or the central/main symbol (for Relationships), except for “simple” Relationships hiding its central/main symbol (in that case no symbol is shown).

******: Also, the selected objects (symbols or complements) can be moved by pressing the arrow keys (up, down, left and right), using steps of View's Grid size (press [Ctrl] to force steps of 1 pixel, and [Shift] to multiply those steps by 4). This ignores the View's "Snap to grid" property.

Creating Concepts

Drag the selected Concept Definition, from the respective Palette, and drop it over the diagram View Area. Then, just after the Symbol (geometric shape) of the new Concept is created, an edit-box will appear for “edit in-place” the Name/Title of the Concept. It can be single-line or multi-line (accepting new lines to be created pressing the [Enter] key), as declared in the “In-Place Editing is Multiline” property of the Symbol Format in the Concept Definition. Also, you can always press [Alt]+[Enter] to create a new-line in the text, and always press [Ctrl]+[Enter] to finish the input.

If the Concept Definition has active the “Has Group Region”, then a new Group Region (boundary) will be appended under the just created Symbol.

Click the [Mouse-Right/Alternate-Button] to cancel the Concept creation and let the pointer ready to select objects or to drop/create other type of objects (Relationships, Markers or Complements).

Automatic Creation: This feature allows the fast automatic creation of Concepts on the diagram View, by…

- Dropping a Concept Definition (selected from the respective Palette) over an existing Concept (*), or…
- Pressing [Enter] while a Concept (*) is selected to create a new child. Subsequent pressing of the [Enter] key will continue adding siblings of the currently targeted Concept.
- Pressing [Tab] while a Concept (*) is selected to create a new child.

*****: The behavior of the automatic creation is determined by the Concept Definition of the targeted Concept. Go to the *Concept Definitions* section, and see the “Automatic Creation parameters” to learn more.

Creating Relationships

Drag the selected Relationship Definition, from the respective Palette, and drop it over an existing Idea(*), then move the mouse and you will see an arrow starting from the Symbol center of that pointed idea indicating that a new Relationship is being created. Now you can either point (click over) the View, which will create the Relationship central/main Symbol (**) starting the creation of a new link, or point to an already existent Idea (*), which will create the Relationship central/main Symbol between the two Ideas.

If the target Idea's Definitor has unmarked the “*Precise Connect by default*” property, then the connector will point to the target Idea's symbol center, else it will point exactly where you aimed. Anyway, you can always do a precise pointing by pressing [Ctrl] while creating a Relationship connector or re-linking one.

After the initial Connector has been created, and if the base Relationship Definition is not “Simple”, a new Connector will start its creation from the Relationship central Symbol. You can click the [Mouse-Right/Alternate-Button] to cancel that new Concept creation and let the pointer ready to create a new Relationship.

Like Concepts, Relationships have the “edit in-place” feature for easy editing of Name/Title upon creation. Also, they can have a Group Region appended, although this is very rare to be required for Relationships.

*****: Because Concepts and Relationships are both Ideas, you can relate/link Concepts with other Concepts or Relationships, also Relationships with Relationships. See the *Relationships* section to learn more.

******: The central/main Symbol of a Relationship can be hidden if the base Relationship Definition is declared with the “Simple” and “Hide Central/Main Symbol when Simple” properties activated. See the *Relationship Definitions* section to learn more.

Extending or Modifying Relationships

Extending Relationships: You can create a new Connector from the Symbol of an existing Relationship by:

- Dragging the connecting arrow, emerging from the central Symbol, while keeping pressed the mouse button. Do not depress the mouse button or the "Edit In-Place" feature will be triggered.
- Drag the same Relationship Definition, from the respective Palette, plus dropping on the Relationship central Symbol and dragging the emerging connecting arrow, while keeping pressed the mouse button.

Modifying Relationship Linking Connectors: An existing Connector can be modified in the next ways:

- **Edit the Link Descriptor**: Do a mouse double-click over the Connector.
- **Remove the Connector**: Select the Connector while pressing [Shift] (a "✖" icon will be shown) or press the [Del] key. If the Relationship connects only one Symbol to another, then the whole Relationship will be deleted, else just the selected Connector.
- **Re-Link or reassign the Connector to/from other Symbol**: Drag the Connector from its connecting point (a "⤢" icon will be shown), and point to the new selected target or origin Symbol.

  Note: When only precise positioning is allowed, then the "⤢" icon is shown.
- **Bend the Connector**: Drag the Connector from its center (a "⤧" icon is shown over it). Also you can move the center of a "Simple" Relationship hiding its central/main Symbol (a "⤧" icon is shown).
- **Straighten (unbend) the Connector**: While pressing [Ctrl], click over the center of the bent Connector (a "⊖" icon will be shown).
- **Cycle through Link-Role Variants**: While pressing [Alt], click over the Connector to change to the next available Link-Role variant, if any (a "↻" icon will be shown).

Adding new Origin Connectors: Drag the connecting arrow, emerging from the central Symbol, while pressing [Left-Alt] into the desired new origin Idea; or drag and drop the same Relationship Definition type (from the Palette) over the new origin Idea (this will start the connecting pointer arrow) and then point to the desired "multi origin" Relationship symbol. Of course, this works only when the Relationship is not "simple" and has its Symbol visible.

Note: In case you want to create a new Relationship related from another Relationship (which is not "simple" nor hiding its central symbol), instead of extending the first one, you can press [Right-Alt] while dragging the connecting pointer arrow and then select the desired target.

Converting Ideas

Sometimes you may realize that an Idea, or a bunch of them, should not be based on their original Definition. Then you can reassign that base Definition with the "Convert" command, which allows you to select a new Definition, plus automatically reassigning the symbol and linking connectors (for Relationships).

Note: Assigned details which were designated in the original Idea Definition remains referencing it, so if they are changed (e.g. redesignated with a name change), that modification will be reflected on the details of the converted Ideas.

Assigning Markers to Ideas

Drag the selected Marker Definition, from the respective Palette, and drop it over an existing Symbol for append a new Marker. Optionally, press [Ctrl] while dropping, to add a descriptor.

Creating Complements

Drag the selected Complement, from the Complements Palette, and drop it on the diagram View. Depending on the kind of Complement, the behavior at creation or editing varies:

- **Text**: It allows you to edit in-place the text to be contained.
- **Image**: Shows the file selection dialog for setting the image. This Complement is designed to work as background for others; therefore it only can be manipulated from its borders.
- **Callout and Quote**: You must drop it over an existing Symbol, and then you can edit its text.
- **Note**: It allows you to edit in-place the text to be contained.
- **Stamp**: It allows you to edit in-place the text to be contained.
- **Info-Card**: From this object you can access the Composition properties editing window.
- **Legend**: From this object you can access the Domain properties editing window.
- **Group Region**: This object is appended to an already existent Symbol. It can be resized independently, but its movements are attached to the assigned Symbol. It works like a tray: when moved, all objects over it are moved along too. Only its background and line brush colors can be changed.
- **Group Line**: Similar to a Group Region, but with only one resizable dimension depending on the current axis (vertical, initially). Its axis can be changed with the “Change axis” option of the context-menu.

Creating Shortcuts

Drag an Idea from the *Content Tree*, and drop it into the diagram View, or copy an Idea (or group of them) and then click the *Paste as Shortcut* button.

Note: For Relationships, only their central/main Symbol is referenced by the Shortcut, not its connectors. Also, Shortcuts are not generated to reference simple Relationships with its central/main Symbol hidden.

Selection, Pan and Zoom

Selecting Objects: Drag the mouse while pressing [Mouse-Left/Normal-Button]. This will temporarily generate a segmented rectangle. All visual objects which completely are inside that rectangle will be marked as selected.

Panning: Drag the mouse while pressing [Mouse-Right/Alternate-Button]. Roll the [Mouse-Wheel] to pan vertically, and press [Shift] to do it horizontally. The context menu will be shown if no drag is performed, just after depressing that mouse button.

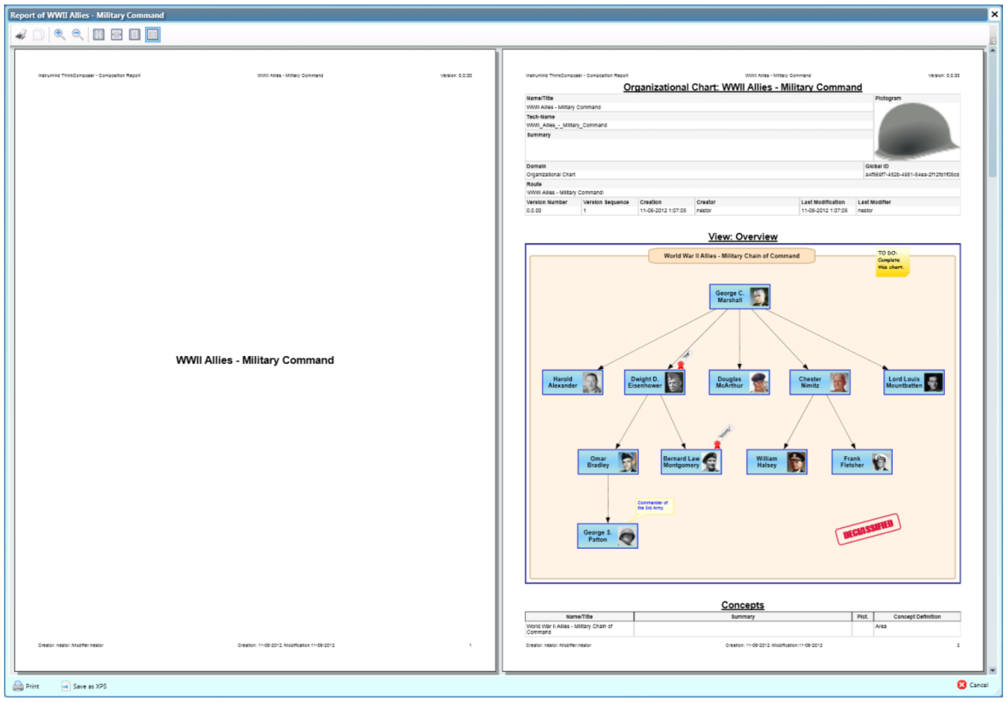
Zooming: Press [Ctrl] and roll the [Mouse-Wheel] or move the *Zoom Slider*.

Reporting

Composition's Report

For an open Composition you can generate a Report of its content, either in XPS/PDF format or as an HTML document (along with a ".content" directory with associated files and nested HTML documents).

The next screenshots shows the first 4 pages of the report for the sample Composition template "WWII Allies – Military Command", which was based on the "Organizational Chart" domain:



Instrumind ThinkComposer - Composition Report — WWII Allies - Military Command — Version: 0.0.00

WWII Allies - Military Command

Creator: nestor /Modifier:nestor — Creation: 11-06-2012 /Modification:11-06-2012 — 1

Instrumind ThinkComposer - Composition Report — WWII Allies - Military Command — Version: 0.0.00

Organizational Chart: WWII Allies - Military Command

Name/Title	**Pictogram**
WWII Allies - Military Command	
Tech-Name	
WWII_Allies_-_Military_Command	
Summary	
Domain	**Global ID**
Organizational Chart	a4f569f7-452b-4981-84ea-2f12f01f08cd
Route	
\WWII Allies - Military Command\	

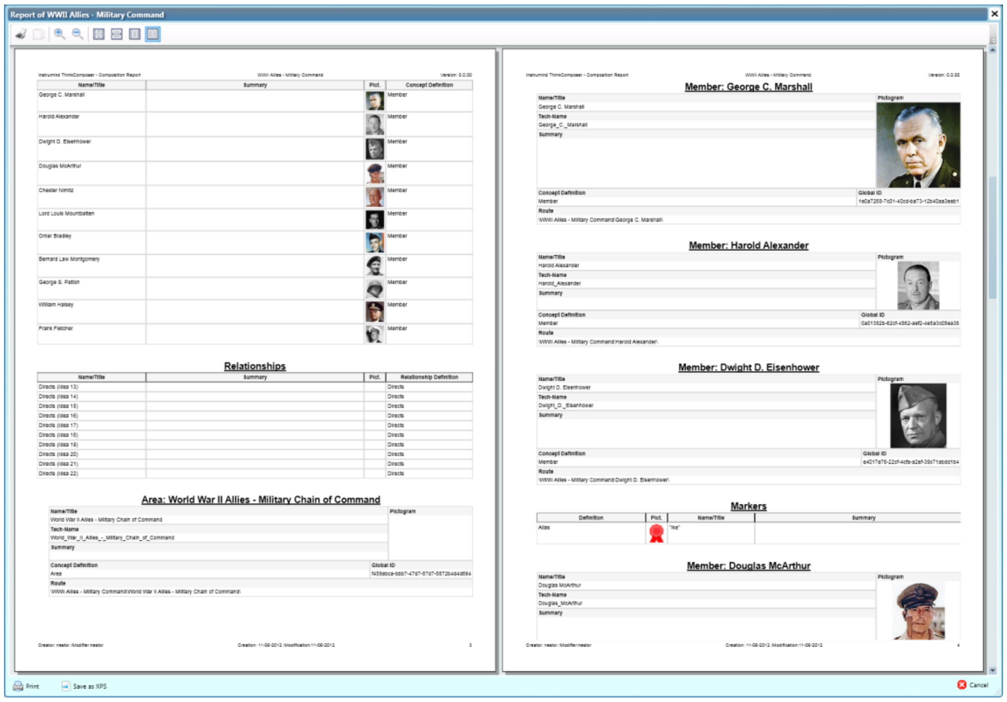
Version Number	Version Sequence	Creation	Creator	Last Modification	Last Modifier
0.0.00	1	11-06-2012 1:07:05	nestor	11-06-2012 1:07:05	nestor

View: Overview

Concepts

Name/Title	Summary	Pict.	Concept Definition
World War II Allies - Military Chain of Command			Area

Creator: nestor /Modifier:nestor — Creation: 11-06-2012 /Modification:11-06-2012 — 2



Instrumind ThinkComposer - Composition Report — WWII Allies - Military Command — Version: 0.0.00

Name/Title	Summary	Pict.	Concept Definition
George C. Marshall			Member
Harold Alexander			Member
Dwight D. Eisenhower			Member
Douglas McArthur			Member
Chester Nimitz			Member
Lord Louis Mountbatten			Member
Omar Bradley			Member
Bernard Law Montgomery			Member
George S. Patton			Member
William Halsey			Member
Frank Fletcher			Member

Relationships

Name/Title	Summary	Pict.	Relationship Definition
Directs (Idea 13)			Directs
Directs (Idea 14)			Directs
Directs (Idea 15)			Directs
Directs (Idea 16)			Directs
Directs (Idea 17)			Directs
Directs (Idea 18)			Directs
Directs (Idea 19)			Directs
Directs (Idea 20)			Directs
Directs (Idea 21)			Directs
Directs (Idea 22)			Directs

Area: World War II Allies - Military Chain of Command

Name/Title	**Pictogram**
World War II Allies - Military Chain of Command	
Tech-Name	
World_War_II_Allies_-_Military_Chain_of_Command	
Summary	
Concept Definition	**Global ID**
Area	f459abca-bbb7-47d7-87d7-8872b4d4d694
Route	
\WWII Allies - Military Command\World War II Allies - Military Chain of Command\	

Creator: nestor /Modifier:nestor — Creation: 11-06-2012 /Modification:11-06-2012 — 3

Instrumind ThinkComposer - Composition Report — WWII Allies - Military Command — Version: 0.0.00

Member: George C. Marshall

Name/Title	**Pictogram**
George C. Marshall	
Tech-Name	
George_C._Marshall	
Summary	
Concept Definition	**Global ID**
Member	1e0a7268-7c01-40cd-ba73-12b40aa3eeb1
Route	
\WWII Allies - Military Command\George C. Marshall\	

Member: Harold Alexander

Name/Title	**Pictogram**
Harold Alexander	
Tech-Name	
Harold_Alexander	
Summary	
Concept Definition	**Global ID**
Member	0a01382b-62cf-4862-aef2-4e5a3c08ea38
Route	
\WWII Allies - Military Command\Harold Alexander\	

Member: Dwight D. Eisenhower

Name/Title	**Pictogram**
Dwight D. Eisenhower	
Tech-Name	
Dwight_D._Eisenhower	
Summary	
Concept Definition	**Global ID**
Member	e4017e76-22cf-4cfa-a2ef-39c71abdd1b4
Route	
\WWII Allies - Military Command\Dwight D. Eisenhower\	

Markers

Definition	Pict.	Name/Title	Summary
Alias		"Ike"	

Member: Douglas McArthur

Name/Title	**Pictogram**
Douglas McArthur	
Tech-Name	
Douglas_McArthur	
Summary	

Creator: nestor /Modifier:nestor — Creation: 11-06-2012 /Modification:11-06-2012 — 4

Appendix A: Template language

This appendix documents the Template language used in ThinkComposer for describing Output-Templates, which is the *Liquid* markup language plus some control markups. It references object properties, from the Composition Information Model, to get the information and merge it with template text.

***Note*:** You can find more information about the original *Liquid* markup language at:
https://github.com/Shopify/liquid/wiki/Liquid-for-Designers

Control markup

This kind of markup indicates to ThinkComposer what to do with the generated text, either while applying the template text or prior to writing output files. It is prefixed with the “%%:” text.

Syntax	Description	Sample Template
FileName=<Template-Text>	Sets the Name of the generated File. - Must be at the first line, and use all of it. - If no FileName variable is declared, then the generated files will be named with the Idea's Tech-Name and a “.txt” extension.	%%:FileName=Idea-{{ TechName }}.txt [MyDocStart] My text [MyDocEnd]
[ExtensionPlace]	Sets the position (in base Templates) where extra text, from templates marked with the “Extend Base Template” property, will be included. By default, if no extension place is specified, the extra text is included at the end.	%%:FileName=Idea-{{TechName}}.txt [MyDocStart] My text %%:[ExtensionPlace] [MyDocEnd]

SubTemplate=<Identifier>	Declares a subtemplate, encompassing the next line until either another subtemplate declaration or the end of the text.
	- Useful as consumable/callable from the template or itself (therefore a recursive generation can be implemented).
	- The Identifier must be a simple text (not a markup), unique within the Composition.
	- Use the 'inject' tag to apply it where desired, passing a variable as its information context.
	- Assign the @@InjectionDepth internal variable to a local variable to get the nested levels count.
		%%:FileName=Composition-{{TechName}}.txt
		[MyDocStart]
		Root: {{ Name }}
		{%- inject 'TreeNodeReader' with This -%}
		[MyDocEnd]
		%%:SubTemplate=TreeNodeReader
		[NodeStart]
		Node-Name: {{ Name }}
		{% assign Depth = @@InjectionDepth %}
		Nested-Level: {{ Depth }}
		{%- for Idea in CompositeIdeas -%}
		   {%- inject 'TreeNodeReader' with Idea -%}
		{%- endfor -%}
		[NodeEnd]

Output markup

An Output Markup declares text which may be generated into the output file, obtained from object properties of the Information Model. They are enclosed between “{{“ and “}}”. The markup is an expression that may contain just an object property name or a combination of properties and Filters, which are functions applied to the resulting text at its left side.

This kind of markup can trim white spaces before or after it, by including a “-“ together with the curly braces at the side to be trimmed. Also, the “-“ will suppress a new-line (line-break) placed just after the markup code.

Please consider the examples of the next table:

Sample Template	Output
Title: **{{ Name }}**	Title: My Composition
{{ Name }} is a **{{ Summary | Upcase }}**.	My Composition is a VERY EXPRESSIVE AND PRECISE DOCUMENT.
“**{{ Name }}**” has **{{ Name | Size }}** characters	“My Composition” has 14 characters

Name: **{{- Name }}**
Summary:
{{ Summary -}}
;
	Name:My Compostion
	Summary:
	Very expressive and precise document;

Name/Summary: **{{ Name -}}**
/ **{{- Summary -}}**
;
	Name/Summary: My Compostion/Very expressive and precise document;

Filters

Filters are functions that take input text, plus parameters (if any), and produce output text. The input text for filters is always located at the left side of its (case-sensitive) name preceded by a '|' character, whereas the parameters (if any) are located at the right, after a semicolon.

The currently available filters are:

Filter	Description	Samples
Size	Gets the size of an array, string or collection of objects. **Note**: Usable also as property (i.e.: Collection.Size)	Source: ['a', 'b', 'c'] Template: Items = {{ Source | Size }} Output: Items = 3
Any	Indicates whether an array, string or collection has any content. **Note**: Usable also as property (i.e.: Collection.Any)	
AsChar	Gets as character the specified code ('tab' or 'newline') or number (UTF-16). **Note**: If no character can be interpreted, then the input is returned.	Template: Hello{{ 'tab' | AsChar }}Dolly Output: Hello Dolly

Filter	Description	Samples
ToBase64	Gets binary content, such as images and attached files, in its Base-64 representation	
ToUnformattedText	Gets rich-text (in XAML format), such as from the “Description” property, as simple unformatted text	
ToPlainText	Gets an arbitrary object, such as Detail's Content, in its simple text representation	
Capitalize	Capitalizes words in the input sentence	Template: {{ 'da vinci' | Capitalize }} Output: 'Da Vinci'
Downcase	Converts an input string to lowercase	Template: {{ 'da vinci' | Downcase }} Output: 'da vinci'
Upcase	Converts an input string to uppercase	Template: {{ 'da vinci' | Upcase }} Output: 'DA VINCI'
First	Gets the first element of the passed in array **Note**: Usable also as property (i.e.: Collection.First)	
Last	Gets the last element of the passed in array **Note**: Usable also as property (i.e.: Collection.Last)	
Join	Joins elements of the array with certain character between them	
Sort	Sorts elements of the array	
Map	Maps/collects an array on a given property	
Escape	Escapes a string	
EscapeOnce	Returns an escaped version of html without affecting existing escaped entities	
StripHtml	Strips html from string	
StripNewlines	Strips all newlines (\n) from string	
NewlineToBr	Replaces each newline (\n) with html break	
Replace	Replaces each occurrence	Template: {{ 'foofoo' | replace:'foo','bar' }} Output: 'barbar'
ReplaceFirst	Replaces the first occurrence	Template: {{ 'barbar' | replace_first:'bar','foo' }} Output: 'foobar'
Remove	Removes each occurrence	Template: {{ 'foobarfoobar' | remove:'foo' }} Output: 'barbar'
RemoveFirst	Removes the first occurrence	Template: {{ 'barbar' | remove_first:'bar' }} Output: 'bar'

Filter	Description	Samples
Truncate	Truncates a string down to x characters	
Truncatewords	Truncates a string down to x words	
Prepend	Prepends a string	Template: {{ 'bar' | prepend:'foo' }} Output: 'foobar'
Append	Appends a string	Template: {{ 'foo' | append:'bar' }} Output: 'foobar'
Minus	Subtraction	Template: {{ 4 | minus:2 }} Output: 2

Plus	Addition	Template: {{ '1' | plus:'1' }}
		Output: '11'
		
		Template: {{ 1 | plus:1 }}
		Output: 2

Times	Multiplication	Template: {{ 5 | times:4 }} Output: 20
DividedBy	Division	Template: {{ 10 | divided_by:2 }} Output: 5
Split	Splits a string on a matching pattern	Template: {{ "a~b" | split:~ }} Output: ['a','b']
Modulo	Remainder	Template: {{ 3 | modulo:2 }} Output: 1
Get	Gets, from a Source object, the specified Property by Tech-Name.	{% assign GrandTotal= Balance | Get:'Total' }}
GetIdeasDefinedAs	Gets, from a Source collection of Ideas, those based on Idea-Definitions having the specified Tech-Names (separated by ";").	{% assign Arachnids = Animals | GetIdeasDefinedAs:'Spider;Scorpion' %}
GetElements	Gets, from a Source collection of Elements (IIdentifiableElement), those having the specified Tech-Names (separated by ";").	{% assign Giants = Planets | GetElements:'Jupiter;Saturn;Uranus;Neptune' %}
GetLinksByVariant	Gets, from a Source collection of Role-Based Links, those having Variants with Tech-Name like those provided (separated by ';')	{% assign ZeroOrOneToMany = Links | GetLinksByVariant:'0..N';'1..N' %}
SelectMany	From a Source collection of items having a property, which is also another collection, gets the union of all the items of these collections.	{% assign AllSolarSystemMoons = Planets | SelectMany:'Moons' %}

Tag markup

A Tag Markup declares processing instructions, which are not generated in the output file, but determine how the text is generated. They names are case-sensitive and are enclosed between “{%” and “%}”.

This kind of markup also can trim white spaces before or after it, by including a “-“ together with the curly braces at the side to be trimmed. Also, the “-“ will suppress a line-break placed just after the markup code.

The currently available Tags are:

Tag	Description	Samples

assign	Assigns some value to a variable	{% assign code = '007' %}
		{% for Agent in This['Agents'].Records %}
		  {% if Agent.id == code%}
		  Jimbo!
		  {% endif %}
		{% endfor %}

capture	Block tag that captures text into a variable	{% capture MyGeneratedText %}
		{{ Name }}: {{ Summary }}
		{% endcapture %}

case	Block tag, it is the standard case...when block	{% case Name | Upcase %}
		{% when 'OK' %}
		  Correct
		{% when 'N/A' %}
		  Not-Applicable
		{% else %} //
		  Unknown
		{% endcase %}

comment	Block tag, comments out the text in the block	{% comment %}
		Template to generate custom XML
		{% endcomment %}

for	For loop. Travels a collection, giving each contained element into a named variable.
	
	Related variables (available inside the block):
	forloop.length: length of the entire for loop
	forloop.index: index of the current iteration
	forloop.index0: Like previous, but zero based.
	forloop.rindex: how many items are still left?
	forloop.rindex0: Like previous, but zero based.
	forloop.first: is this the first iteration?
	forloop.last: is this the last iteration?
		{% for Detail in Details %}
		  This is a detail of kind {{ Detail.Kind.Name }}
		  {% if foorloop.last %}
		    (this is the last element of the collection)
		  {% endif %}
		  {% if Detail.Kind.TechName == 'Table' %}
		    {% for Record in Detail.Records %}
		      Record's label: {{ Record.Label }}
		    {% endfor %}
		  {% endif %}
		{% endfor %}

Tag	Description	Samples

if	Standard if/else block.
	
	Logical operators:
	==: true when both sides are equal
	!=: true when both sides are different
	and: true when both sides are true
	or: true when at least one side is true
	Note: Must separate operators and values (I.E. "a==b" is incorrect, "a == b" is correct)
	
	Collection operators:
	contains: true when collection contains element
	empty: true when collection has no elements
		{% if Details.Size < 1 %}
		empty
		{% else %}
		There are {{ Details.Size }} details.
		{% endif %}

inject	Inserts the content of a subtemplate in the desired location, passing a variable as the information context.
	
	Useful to make recursive generation. See the 'SubTemplate' control markup for more info.
	
	Optional modifiers (after the context variable):
	keepindent: Stops deepening the indentation.
	noindent: Supress the indentation.
		{% for Child in CompositeIdeas %}
		{% inject 'ChildrenTemplate' with Child %}
		{% endfor %}
		
		…
		{% inject 'MyTemplate' with Data keepindent %}
		
		…
		{% inject 'MyTemplate' with Remarks noindent %}

raw	Temporarily disables tag processing to avoid syntax conflicts	{% raw %}
		This {{ markup }} will not be processed
		{% endraw %}

unless	Mirror of if statement	{% unless Name=='' %}
		Hello {{ Name }}
		{% endunless %}

Note: The special character back-slash ("\") must be escaped, writing it twice ("\\"), in order to be used in function parameters. For example, instead of writing "{{ TechName \| replace:'\','.' }}" you should write "{{ TechName \| replace:'\\','.' }}".

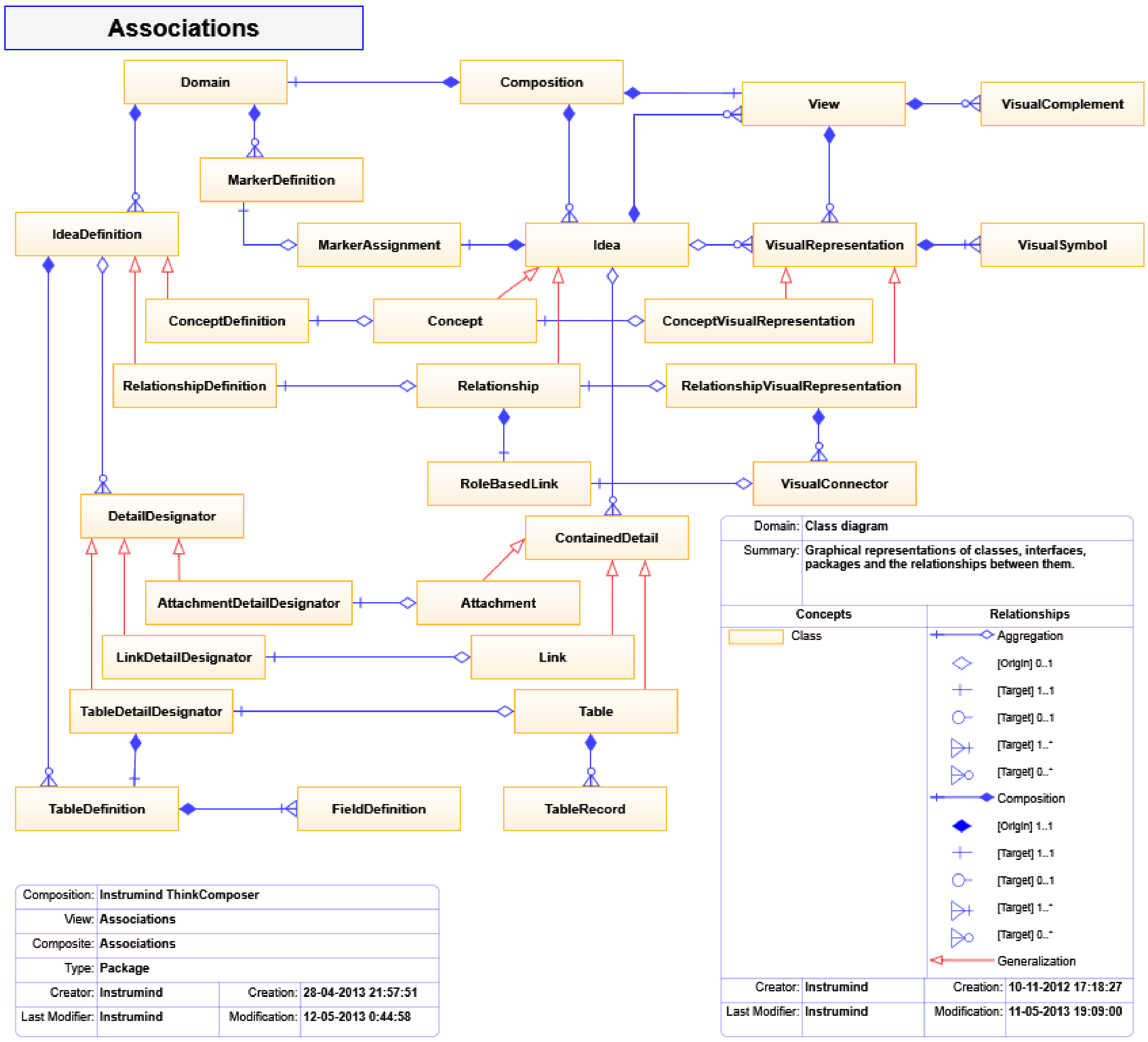
Appendix B: Composition Information Model

This appendix documents most of the information model of ThinkComposer Compositions. It is intended to be accessed for consumption via Output-Templates, declared using a Template language, so its information can be merged with the template text and be exported as generated files.

Classes Diagrams

Associations

The next diagram represents the aggregation ("has a") and composition ("is composed of") associations of exposed model classes:

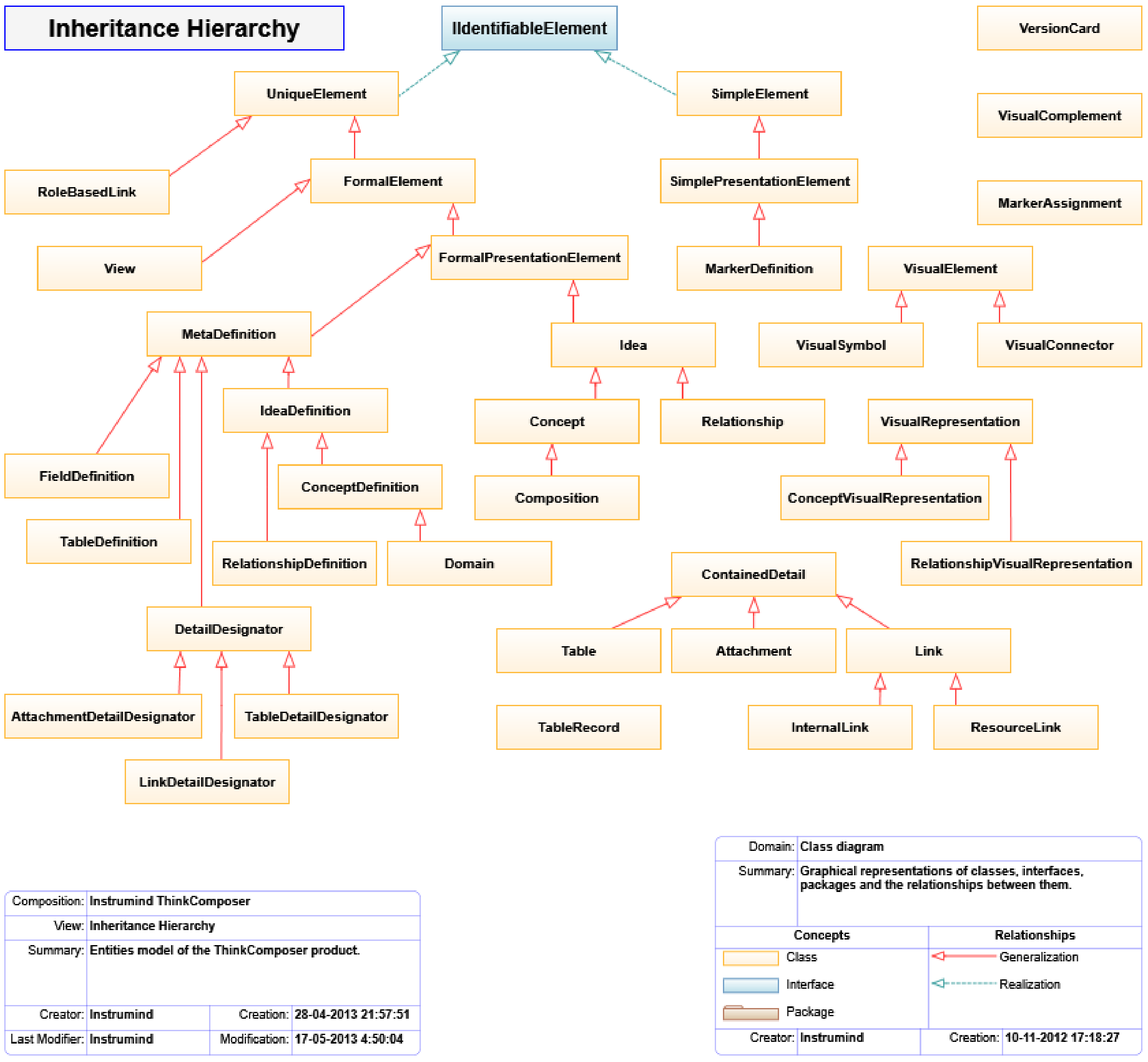


Composition:	Instrumind ThinkComposer		
View:	Associations		
Composite:	Associations		
Type:	Package		
Creator:	Instrumind	Creation:	28-04-2013 21:57:51
Last Modifier:	Instrumind	Modification:	12-05-2013 0:44:58

Domain:	Class diagram		
Summary:	Graphical representations of classes, interfaces, packages and the relationships between them.		
Creator:	Instrumind	Creation:	10-11-2012 17:18:27
Last Modifier:	Instrumind	Modification:	11-05-2013 19:09:00

Inheritance Hierarchy

The next diagram represents the inheritance (or "is a" associations) hierarchy of the exposed model classes:



Composition:	Instrumind ThinkComposer		
View:	Inheritance Hierarchy		
Summary:	Entities model of the ThinkComposer product.		
Creator:	Instrumind	Creation:	28-04-2013 21:57:51
Last Modifier:	Instrumind	Modification:	17-05-2013 4:50:04

Domain:	Class diagram		
Summary:	Graphical representations of classes, interfaces, packages and the relationships between them.		
Concepts		Relationships	
🟨	Class	←	Generalization
🟦	Interface	⇠	Realization
🟫	Package		
Creator:	Instrumind	Creation:	10-11-2012 17:18:27

Note: The *IIdentifiableElement* interface declares the Global-Id, Name, Tech-Name and Summary properties available in most ThinkComposer objects.

Special cases: Custom Fields, Details and Table data access

The Custom Fields, Details and Table Records data are easily accessible, using the special variables and syntaxes shown in the next table:

Content	Access	Samples

Custom Fields	**_['<A-Custom-Field-Tech-Name>']**
	
	The "_" property references the Custom Fields record of the current Idea. It has a Tech-Name based indexer to get individual Custom Fields.
		Standard Idea's Name:
		Name: {{ Name }};
		
		Alternative name (an alias) stored as Custom Field:
		Also-known-as: {{ _['Alias'] }}

Details	**This['<A-Detail-Tech-Name>']**
	
	The "This" property references the current Idea. It has a Tech-Name based indexer to get individual Details.
		Detail having Attached text file:
		{{ This['MyAttachment'] }}
		
		Detail having a Table:
		Count of records: {{ This['MyTable'].Records.Size }}

Table Records	**RecordVariable.<A-Field-Tech-Name>**, or **RecodVariable['<A-Field-Tech-Name>']**
	
	Having a Table-Record variable, enables you to access their fields either directly or by its Tech-Name based indexer.
		Suppose a 'People' Table having 'Name' and 'Age' Fields…
		
		The {{ This['People'].Records.Size }} persons are…
		{% for Person in This['People'].Records %}
		  Name: {{ Person.Name }}; Age: {{ Person['Age'] }}
		{% endfor %}

Domain's Base Tables	**<Domain>.BaseContentRoot['<Base-Table-Tech-Name>']**
	
	Access a Domain variable, then reference its 'BaseContentRoot' property and get the desired Base Table by its Tech-Name based indexer.
		From a Composition template…
		{% for Unit in CompositionDefinitor.BaseContentRoot['UnitsOf'] .Records %}
		  Unit #{{ forloop.index }}: {{ Unit.Name }}
		{% endfor %}
		
		From an Idea template…
		{% for State in OwnerComposition. CompositionDefinitor.BaseContentRoot['States'] .Records %}
		  {{ State.Code }}: {{ State.Name }}
		{% endfor %}

Specification of Model Classes

The next table details the exposed model classes and their properties. Notice that classes may have an ancestor, from which they inherit properties.

Name	Type/Ancestor	Summary
Attachment	**ContainedDetail**	**Represents an arbitrary embedded object, obtained from external source, such as: image, video, data file, etc.**
AssignedDesignator	Assignment<DetailDesignator>	Attachment content designator
Designator	AttachmentDetailDesignator	Attachment designator.
Kind	ModelDefinition	Returns the kind of this detail.
MimeType	String	Detected MIME-Type when the attachment was loaded.
Source	String	Location-of/route-to the resource origin.
AttachmentDetailDesignator	**DetailDesignator**	**Associates an Attachment definition to an Idea.**
Composition	**Concept**	**Semantic, informational and visual set of ideas, expressing knowledge about a subject. This document or book is conformed by the hierarchical nesting of containers (logical documents based on a Composite Concept), starting from a root.**
ActiveView	View	Active View of the composition.
CompositionDefinitor	Domain	Domain definition of this Composition.
Pictogram	ImageSource	Graphic representation of the object.
RootView	View	Initial and central View of the Composition.
UsedDomains	EditableList<Domain>	Collection of used Domains in this Composition.
ViewsPrefix	String	Prefix for naming related views of this document.
Concept	**Idea**	**Concrete object, subtype of Idea, which can be associated to others through Relationships.**
ConceptDefinitor	Assignment<ConceptDefinition>	Concept Definition on which this Concept is based.
ConceptDefinition	**IdeaDefinition**	**Represents the definition of a Concept type.**
AncestorConceptDef	ConceptDefinition	References the ancestor Concept definition of this one.
AutomaticCreationConceptDef	ConceptDefinition	Definition of the Concept to be automatically created.
AutomaticCreationPositioningIsRadialized	Boolean	Indicates to position automatically created Concepts around in a radial (semi elliptical) style.
AutomaticCreationRelationshipDef	RelationshipDefinition	Definition of the Relationship to associate Concepts with the automatically created ones.
Pictogram	ImageSource	Graphic representation of the object.
ConceptVisualRepresentation	**VisualRepresentation**	**Visually represents a Concept.**
RepresentedConcept	Concept	Represented Concept by this visual element.
ContainedDetail	**[NONE]**	**Represents an object stored as detail for an Idea (contained within its DetailsContainer).**
IsCustomDetail	Boolean	Indicates whether this detail is Custom, which means this was not designated at the Idea Definitor, but into a particular Idea.
OwnerIdea	Idea	Container owning this Contained-Detail.
DetailDesignator	**MetaDefinition**	**Base ancestor for the designators of detailed data.**
Owner	Ownership<IdeaDefinition, Idea>	Owner of this table detail designator.
SubOwnerFieldDef	FieldDefinition	Optional Field-Definition sub-owning this detail designator.
Domain	**ConceptDefinition**	**Unifies a set of metadefinitions about a business area, which rules the creation of Composite-Content. This considers the definition of: Graph schematization, visual representation and information structures.**
BaseContentRoot	Concept	Idea owning the Domain's predefined base-content (such as Base Tables).
DefaultTableDef	TableDefinition	Table-Structure Definition used as the by-default for new Tables.
IdeaClusters	EditableList<SimplePresentationElement>	Simple clusters for grouping Idea Definitions on palettes (i.e. visually).

LinkRoleVariants	EditableList<SimplePresentationElement>	Predefined variants available for Link-Role Definitions (e.g.: Used for declaring Multiplicities/Cardinalities).
MarkerClusters	EditableList<SimplePresentationElement>	Simple clusters for grouping Marker Definitions on palettes (i.e. visually).
OwnerComposition	Composition	Composition owning this Domain instance.
Pictogram	ImageSource	Graphic representation of the object.
ViewBackgroundImage	ImageSource	Image to be initially assigned to the View's background. If bigger than 500x500 then it is adjusted to fit in the View, else it is repeated/tiled.
FieldDefinition	**MetaDefinition**	**Defines a table structure field.**
ContainedTableDesignator	TableDetailDesignator	If set, stores or references the Table-Structure Definition declaring the type of the Tables contained by the implementing Fields.
ContainedTableIsSingleRecord	Boolean	Indicates whether the contained-table, if set, has only one record, else is Multi-Record.
HideInDiagram	Boolean	Indicates that the field values must be hidden in the diagram view.
IsRequired	Boolean	Indicates whether the field value must be stored or can be let empty (null).
OwnerTableDef	TableDefinition	Table-Structure Definition owning this Field Definition.
StorageIndex	Int32	Actual position, within the table record structure, where the values of this field are located.
FormalElement	**UniqueElement**	**Standard entity type, globally unique, with Name, Tech-Name, Summary, Tech-Spec, plus rich-text Description and Version information.**
Description	String	Detailed description (rich) text of the object.
Name	String	Name or Title of the object.
NameCaption	String	Gets the Name for single-line display (without new-line characters).
Summary	String	Summary of the object.
TechName	String	Technical-Name of the object. Should be unique. Intended for machine-level usage as code, identifier or name for files/tables/programs.
TechSpec	String	Technical-Specification of the object. Intended as a machine-level representation for computation (i.e. for use as script, template, formula or other kind of expression).
Version	VersionCard	Stores the versioning information of the object, such as Creator, Last-Modifier, dates and version number.
FormalPresentationElement	**FormalElement**	**Standard entity with identification, versioning and visual representation.**
Pictogram	ImageSource	Graphic representation of the object.
Idea	**FormalPresentationElement**	**The most basic Composition element of the Instrumind's Graph schema, from which Concepts and Relationships are descendants. Combines on a single -derived- instance the attributes of Graph existence, visual representation and information storage.**
_	TableRecord	Alias of the Custom-Fields record.
AssociatingLinks	EditableList<RoleBasedLink>	Collection of links which associate this Idea to a Relationship.
BaseKind	ModelDefinition	Gets the kind (Domain, ConceptDefintion or RelationshipDefinition) of the definitor of this idea final instance type.
CompositeActiveView	View	Current active view of the composite Idea.
CompositeContentDomain	Domain	Domain which rules the content of this Idea.
CompositeDepthLevel	Int32	Gets the level of compositional depth of this Idea.
CompositeIdeas	EditableList<Idea>	The collection of composing Ideas of this one.
CompositeViews	EditableList<View>	Views for contained children Ideas when this is composite.
DefinitionIsShared	Boolean	Indicates whether the assigned Idea definition is Shared (owned by its Domain), or Local (owned by this Idea).
DescriptiveCaption	String	Short text describing the Idea.
Details	EditableList<ContainedDetail>	Collection of contained details.
HasDetailedContent	Boolean	Indicates whether the Idea has detailed content (tables, attachments or non-internal links).
IncomingLinks	IEnumerable<RoleBasedLink>	Links targeting to this Idea.
IsComposite	Boolean	Indicates whether the Idea is composed of others, else is atomic.
LinkedFrom	IEnumerable<Relationship>	Incoming Relationships linking-to/whose-target-is this Idea.
LinkingTo	IEnumerable<Relationship>	Outgoing Relationships linked-from/whose-origin-is this Idea.
MainRepresentator	VisualRepresentation	Gets the primary Visual Representator of this Idea.
MainSymbol	VisualSymbol	Gets the Main Symbol of the primary Visual Representator of this Idea.
Markings	EditableList<MarkerAssignment>	Collection of assigned markers.

OppositeOriginLinks	IEnumerable<RoleBasedLink>	Links, opposite to incoming-links and of the same Relationships, pointed from origin Ideas.
OppositeTargetLinks	IEnumerable<RoleBasedLink>	Links, opposite to outgoing-links and of the same Relationships, pointing to target Ideas.
OutgoingLinks	IEnumerable<RoleBasedLink>	Links originating from this Idea.
OwnerComposition	Composition	Composition owning this Idea.
OwnerContainer	Idea	Container owning this Idea, which is composing a dominant one.
RelatedFrom	IEnumerable<Idea>	Ideas pointing to this Idea (through incoming Relationships)
RelatingTo	IEnumerable<Idea>	Ideas pointed by this Idea (through outgoing Relationships)
SelfKind	ModelDefinition	Gets the kind (Composition, Concept or Relationship) of this idea final instance type.
This	Idea	Returns this Idea (To support access to Details thru indexer).
VisualRepresentators	EditableList<VisualRepresentation>	Collection of visual representations for this Idea.
IdeaDefinition	**MetaDefinition**	**Common ancestor for Metadefinitions about Concepts and Relationships. It can have attached user-defined Information and Visualization assignments.**
AutomaticGroupedConceptDef	ConceptDefinition	Definition of the Concept to be automatically created onto an appended Group Region/Line.
CanAutomaticallyCreateGroupedConcepts	Boolean	Indicates whether the Ideas of this type will automatically create grouped Concepts when linking a Relationship into an appended Group Region/Line.
CanAutomaticallyCreateRelatedConcepts	Boolean	Indicates whether the Ideas of this type will automatically create related Concepts in editing (by pressing [Enter], [Tab] or dropping Idea Definitions over them).
CanGroupIntersectingObjects	Boolean	Indicates whether the Ideas of this type will group objects intersecting its symbol or Group Region.
CompositeContentDomain	Domain	If set, indicates the Domain which rules the content of the defined Idea.
ConceptDefinitions	EditableList<ConceptDefinition>	Collection of Concept definitions which are part of this one.
ClusterKey	FormalPresentationElement	Cluster to which this Idea-Definition is associated (used for better organization/grouping of the Definitions).
CustomFieldsTableDef	TableDefinition	Definition of Custom-Fields (based on a Table-Structure Definition).
DefKind	ModelDefinition	Gets the kind (Domain, ConceptDefinition or RelationshipDefinition) of this idea definition final instance type.
DetailDesignators	EditableList<DetailDesignator>	Collection of Detail Designators declared for this Idea definition.
HasGroupLine	Boolean	Indicates whether the defined Ideas are created with a Group Line complement (like a 'life line') appended.
HasGroupRegion	Boolean	Indicates whether the defined Ideas are created with a Group Region complement (a boundary) appended.
IsComposable	Boolean	Indicates whether the defined Ideas can be composed of others in a whole view/diagram contained inside.
IsVersionable	Boolean	Indicates whether the defined Ideas can maintain versioning information.
OwnerDefinitor	IdeaDefinition	References the composite Idea definition owning this one.
Pictogram	ImageSource	Graphic representation of the object.
PreciseConnectByDefault	Boolean	Indicates to connect from/to precise aimed positions inside the Symbol, by default, else from/to the Symbol center.
RelationshipDefinitions	EditableList<RelationshipDefinition>	Collection of Relationship definitions which are part of this one.
RepresentativeShape	String	Shape illustrating the definition, to be exposed as the visual symbol of the represented Ideas.
TableDefinitions	EditableList<TableDefinition>	Collection of declared Table-Structure Definitions.
InternalLink	**Link**	**References an internal property.**
AssignedDesignator	Assignment<DetailDesignator>	Link assigned designator
Designator	LinkDetailDesignator	Internal-Link assigned designator.
Link	**ContainedDetail**	**References an external object (such as: File, Folder, Web adress) or an internal one (such as an Idea property).**
Kind	ModelDefinition	Returns the kind of this detail.
TargetAddress	String	Address of the resource.
LinkDetailDesignator	**DetailDesignator**	**Associates an Link definition to an Idea.**
LinkRoleDefinition	**MetaDefinition**	**Represents the definition of a Role Link.**
AllowedVariants	EditableList<SimplePresentationElement>	Allowed link-role variants and related plug style for the relationship link role.
AssociableIdeaDefs	EditableList<IdeaDefinition>	List of linkable idea definitions. If none is assigned, then all can be linked.

MaxConnections	UInt32	Number of maximum Ideas that can be linked by the role. Zero for unlimited. The default is one.
OwnerRelationshipDef	RelationshipDefinition	Relationship definition owning this link role definition.
Pictogram	ImageSource	Graphic representation of the object.
RelatedIdeasAreOrdered	Boolean	Indicates whether the related Ideas for the relationship link role are free or follows an order.
MarkerAssignment	**[NONE]**	**Represents the assignment of a Marking to an Idea. Optionally, a descriptor can be also associated.**
Descriptor	SimplePresentationElement	Optional descriptor for the Marker.
MetaDefinition	**FormalPresentationElement**	**Represents, at a metalevel of abstraction, the definition of the data structure upon which create schema objects of a type.**
MetaId	Int32	Simple identifier for indirectly associate created objects with definitions.
ModelDefinition	**[NONE]**	**Base class for the definition of model classifiers and their members..**
Name	Int32	User-level name of the defined object.
TechName	String	Name of the defined object.
Summary	String	User-level description of the defines object.
Relationship	**Idea**	**Association between multiple ideas, connected using link-roles, forming a nexus.**
DescriptiveCaption	String	Short text describing Relationship links.
IsAutoReference	Boolean	Indicates whether this represents an auto-reference for the connected Idea (non-excluvise). This means that can exists links pointing from/to another Ideas.
IsAutoReferenceExclusive	Boolean	Indicates whether this represents an exclusive auto-reference for the connected Idea. This means that all links points from/to the same Idea.
Links	EditableList<RoleBasedLink>	Collection of implemented Links.
OriginIdeas	IEnumerable<Idea>	Gets the Ideas from which this Relationship is Originted (includes Participants).
OriginLinks	IEnumerable<RoleBasedLink>	Links associating the origin (or participant) Ideas
RelationshipDefinitor	Assignment<RelationshipDefinition>	Relationship Definition on which this Relationship is based.
TargetIdeas	IEnumerable<Idea>	Gets the Ideas to which this Relationship is Targeted.
TargetLinks	IEnumerable<RoleBasedLink>	Links associating the target Ideas.
RelationshipDefinition	**IdeaDefinition**	**Represents the definition of a Relationship type.**
AncestorRelationshipDef	RelationshipDefinition	References the ancestor Relationship definition of this one.
HideCentralSymbolWhenSimple	Boolean	Hides the Central/Main-Symbol when the Relationship is defined as Simple.
IsDirectional	Boolean	Indicates whether this relationship if from an origin to a target, else is between equivalent participants.
IsSimple	Boolean	Indicates that only one target and one source Links can be established.
OriginOrParticipantLinkRoleDef	LinkRoleDefinition	Definition for the Origin/Source link role. This is the participant role in a non-directional relationship.
Pictogram	ImageSource	Graphic representation of the object.
ShowNameIfHidingCentralSymbol	Boolean	Indicates to show the Relationship name when hiding the Central/Main-Symbol.
TargetLinkRoleDef	LinkRoleDefinition	Definition for the Target/Destination link role. This has no use in a non-directional relationship.
RelationshipVisualRepresentation	**VisualRepresentation**	**Visually represents a Relationship.**
RepresentedRelationship	Relationship	Represented Relationship by this visual element.
VisualConnectors	IEnumerable<VisualConnector>	Gets the visual connectors.
VisualConnectorsCount	Int32	Gets the count of visual connectors.
ResourceLink	**Link**	**References a resource, such as a file, folder or web address.**
AssignedDesignator	Assignment<DetailDesignator>	Resource Link content designator
Designator	LinkDetailDesignator	Reseource-Link designator.
TargetAddress	String	Address of the resource.
TargetLocation	String	Location-of/route-to the resource.
RoleBasedLink	**UniqueElement**	**Links a Relationship with a related Idea, following a Link Role Definition.**
AssociatedIdea	Idea	References the associated Idea of this link.
Descriptor	SimplePresentationElement	Optional description of this link.
OwnerRelationship	Relationship	References the owning Relationship.

RoleDefinitor	LinkRoleDefinition	Related Link-Role Definition.
RoleVariant	SimplePresentationElement	Indicates the Link-Role Variant for this link.
SimpleElement	**[NONE]**	**Basic entity type, globally unique, with Name, Tech-Name, Summary and Tech-Spec.**
Name	String	Name or Title of the object.
NameCaption	String	Gets the Name for single-line display (without new-line characters).
Summary	String	Summary of the object.
TechName	String	Technical-Name of the object. Should be unique. Intended for machine-level usage as code, identifier or name for files/tables/programs.
TechSpec	String	Technical-Specification of the object. Intended as a machine-level representation for computation (i.e. for use as script, template, formula or other kind of expression).
SimplePresentationElement	**SimpleElement**	**Simple entity having a visual representation.**
Pictogram	ImageSource	Graphic representation of the object.
Table	**ContainedDetail**	**Stores structured information, containing one or multiple data records of the same type.**
AssignedDesignator	Assignment<DetailDesignator>	Table designator.
Count	Int32	Returns the count of contained Table-Records.
Definition	TableDefinition	Gets the designated Table-Structure Definition.
Designator	TableDetailDesignator	Table designator.
Kind	ModelDefinition	Returns the kind of this detail.
Records	EditableList<TableRecord>	Collection of records belonging to this Table.
RecordsLabel	String	Text representation of the Table's Records data (only the first 3 records).
TableDefinition	**MetaDefinition**	**Defines a Table-Structure type.**
FieldDefinitions	EditableList<FieldDefinition>	Collection of declared Field definitions of this Table-Structure Definition.
LabelFieldDefs	EditableList<FieldDefinition>	List of ordered field definitions used as Labels (for title usage).
OwnerDomain	Domain	Domain owning this Table-Structure Definition.
TableDetailDesignator	**DetailDesignator**	**Associates Table-Structure Definitions to an Idea, Idea-Definition or Field-Definition (Contained Table field type).**
ContainedTableSubOwner	FieldDefinition	If set, references the Field-Definition sub-owner of a field contained Table. IMPORTANT: In this case, the Owner must point to the related Domain.
DeclaringTableDefinition	TableDefinition	Table-Structure Definition which declares the data structure of this Table.
TableDefIsOwned	Boolean	Indicates that the Table Definition belongs to the detail's owner (not shared).
TableRecord	**[NONE]**	**Groups variable data items that conforms a data entity, composing a Table with others of the same kind.**
Index	Int32	Index of the record in the owner Table (one based, for users).
Label	String	Returns the record's Label: Text composed of the fields (definitions) marked as being part of the Label in the Table-Structure Definition.
OwnerTable	Table	Table owning this table record
UniqueElement	**[NONE]**	**Object with unique non-repeatable identity among others of any kind.**
GlobalId	Guid	Global unique identifier of the object.
VersionCard	**[NONE]**	**Stores version control information for versionable objects.**
Annotation	String	Comment in reference to edition activities performed or pending.
Creation	DateTime	Date-time of the creation.
Creator	String	Creator user name.
LastModification	DateTime	Date-time of the last modification.
LastModifier	String	Last modifier user name.
VersionNumber	String	Optional manual/external generated version number (i.e: 'major-release.minor-release[.build[.revision]]')
VersionSequence	Int32	Sequential number, starting from one. The real version number.
View	**FormalElement**	**Visual representation of the Composite-Content of an Idea or Composition.**
BackgroundImage	ImageSource	Image to be shown at the Background (behind the diagram sheet, over the background color). If bigger than 500x500 then it is adjusted to fit in the View, else it is repeated/tiled.
GridSize	Double	Gets or sets the context Grid size (range: 2 to 20 pixels).

GridUsesLines	Boolean	Indicates that the context Grid should be based on Lines, else on Points.
IsOpen	Boolean	Indicates whether the content is presented/expanded or hidden/collapsed.
IsOutlined	Boolean	Indicates whether the content is presented surrounded by a border.
OwnerCompositeContainer	Idea	References the Idea Composite Container owning this View.
PageDisplayScale	Int32	Scaling percentage for displaying the view page.
ShowConceptDefinitionLabels	Boolean	Indicates whether to display Labels with the Concept Definition name over the Concept.
ShowContextBackground	Boolean	Indicates whether to display the assigned Background.
ShowContextGrid	Boolean	Indicates whether to display the assigned Grid.
ShowIndicators	Boolean	Indicates whether to display Indicators over the Ideas.
ShowLinkRoleDefNameLabels	Boolean	Indicates whether to display Labels with the Link-Role Definitor name over the Connectors.
ShowLinkRoleDescNameLabels	Boolean	Indicates whether to display Labels with the Link-Role Descriptor name over the Connectors.
ShowLinkRoleVariantLabels	Boolean	Indicates whether to display Labels with the Link-Role Variant over the Connectors.
ShowMarkers	Boolean	Indicates whether to display Markers over the Ideas.
ShowMarkersTitles	Boolean	Indicates whether to display the Title of the Markers over them.
ShowRelationshipDefinitionLabels	Boolean	Indicates whether to display Labels with the Relationship Definition name over the Relationship.
ShowSmoothEdges	Boolean	Indicates whether to show smoth edges for the displayed shapes, else they are displayed sharpened.
SnapToGrid	Boolean	Indicates whether the postioning of objects should be aligned to grid points.
ViewSize	Size	Size of the View.
VisualCountOfFloatings	Int32	Current count of visual floating content.
VisualLevelForBackground	Int32	Maximum z-order level currently assigned for visual background content.
VisualLevelForRegions	Int32	Maximum z-order level currently assigned for visual regions content.
VisualComplement	**VisualObject**	**Individual visual object exposing attached information, such as note, callout, legend and info-card.**
BaseArea	Rect	Area of the figure.
BaseCenter	Point	Central position where the figure is displayed around.
BaseHeight	Double	Height of the figure bounds rectangle area, containing the actual geometry which maybe is not rectangular.
BaseLeft	Double	Horizontal left position of the figure bounds rectangle area, containing the actual geometry which maybe is not rectangular.
BaseTop	Double	Vertical top position of the figure bounds rectangle area, containing the actual geometry which maybe is not rectangular.
BaseWidth	Double	Width of the figure bounds rectangle area, containing the actual geometry which maybe is not rectangular.
Content	Object	Contained text or image.
Kind	SimplePresentationElement	Type of Complement implemented.
Target	Ownership<View, VisualSymbol>	Visual object targeted by this Complement.
VisualConnector	**VisualElement**	**Makes a visual connection between two elements.**
IntermediatePosition	Point	Intermediate optional position of the connector.
OriginEdgePosition	Point	Source edge-position of the connector respect the source symbol.
OriginPlug	String	Gets the plug type code for the origin side.
OriginPosition	Point	Source position of the connector.
OriginSymbol	VisualSymbol	Symbol where this Connector originates.
OwnerRelationshipRepresentation	RelationshipVisualRepresentation	References the owning relationship visual representator.
RepresentedLink	RoleBasedLink	References the represented Role Based Link.
TargetEdgePosition	Point	Destination edge-position of the connector respect the target symbol.
TargetPlug	String	Gets the plug type code for the target side.
TargetPosition	Point	Destination position of the connector.
TargetSymbol	VisualSymbol	Symbol pointed by this Connector.
VisualElement	**VisualObject**	**Identifiable base ancestor for visual representators such as symbols and connectors.**
OwnerRepresentation	VisualRepresentation	References the owning visual representator.

VisualRepresentation	**UniqueElement**	**Groups Visual Elements to conform the exposed representation of an Idea.**
AreRelatedOriginsShown	Boolean	Indicates whether the related Origin representations are shown.
AreRelatedTargetsShown	Boolean	Indicates whether the related Target (and participant) representations are shown.
DisplayingView	View	View showing this visual representation.
IsShortcut	Boolean	Indicates that this visual object points to an Idea contained outside the current (Idea) Container.
MainSymbol	VisualSymbol	Gets the major symbol of this representation. The body symbol for Concepts, or the main-symbol for Relationships.
RepresentedIdea	Idea	Represented Idea by this visual representation.
VisualSymbol	**VisualElement**	**Base ancestor for visual symbols such as vector-based drawings, images and texts.**
AreDetailsShown	Boolean	Indicates whether the details are currently being shown on the view.
BaseArea	Rect	Gets the symbol's heading rectangle.
BaseCenter	Point	Central position where the symbol is displayed around.
BaseContentArea	Rect	Area for the content to be shown in the heading of the symbol.
BaseHeight	Double	Height of the symbol bounds rectangle area, containing the actual geometry which maybe is not rectangular. For Concepts, this refers to the body; for Relationships, this refers to the main-symbol.
BaseLeft	Double	Horizontal left position of the symbol bounds rectangle area, containing the actual geometry which maybe is not rectangular. For Concepts, this refers to the body; for Relationships, this refers to the main-symbol.
BaseTop	Double	Vertical top position of the symbol bounds rectangle area, containing the actual geometry which maybe is not rectangular. For Concepts, this refers to the body; for Relationships, this refers to the main-symbol.
BaseWidth	Double	Width of the symbol bounds rectangle area, containing the actual geometry which maybe is not rectangular. For Concepts, this refers to the body; for Relationships, this refers to the main-symbol.
Complements	EditableList<VisualComplement>	Attached visual complements, such as Callouts.
DetailsArea	Rect	Gets the symbol's detail poster area.
DetailsContentArea	Rect	Area for the content to be shown in the details poster of the symbol.
DetailsPosterHeight	Double	Current details poster height, even if not shown.
IsAutoPositionable	Boolean	Indicates whether this visual object can be positioned without explicit user interaction.
IsHorizontallyFlipped	Boolean	Indidcates that the symbol is horizontally flipped.
IsVerticallyFlipped	Boolean	Indidcates that the symbol is vertically flipped.
OriginConnections	EditableList<VisualConnector>	List of originating connectors whose destination is this symbol.
OwnerRepresentation	VisualRepresentation	References the owning visual representator.
ShowCompositeContentAsDetails	Boolean	Indicates to show composite-content as (instead of) details.
TargetConnections	EditableList<VisualConnector>	List of targeted connectors whose origin is this symbol.
TotalArea	Rect	Gets the current content area, considering the Heading and Details (if displayed).
TotalHeight	Double	Height of the symbol plus its details poster, if shown.

Note: The TableRecord class has dynamic properties/fields, based on the Field-Definitions created for its Table-Definition. For access them, see the Special cases: Custom Fields, Details and Table data access section.

Also, the next classes can contain or reference data in the way described…

- **EditableList<Item-Type>**: Contains a collection of ordered items. Relevant properties:
  - **Count**: Gets the current number of items.
- **EditableDictionary<Key-Type, Value-Type>**: Contains a collection of key-value pairs. Relevant properties:
  - **Count**: Gets the current number of items.

- **Assignment<Key-Type, Value-Type>**: References an object which may be local (owned by this instance), which implies writing access, or external, thus having read-only access to it. Relevant properties:
    - **IsLocal**: Indicates whether the object is locally owned.
    - **Value**: Object referenced.
- **Ownership<Global-Type, Local-Type>**: Indicates ownership, Global (shared) or Local (exclusive), of the parent instance and references an owner instance, which can be based on one of two possible types. Relevant properties:
    - **IsGlobal**: Indicates whether the ownership is Global/Shared, else is Local/Exclusive.
    - **Owner**:.Gets the owner as is stored.
- **StoreBox<Content-Type >**: Contains objects which requires conversion to/from a storable (i.e. to disk) format. Relevant properties:
    - **Value**: Object contained.