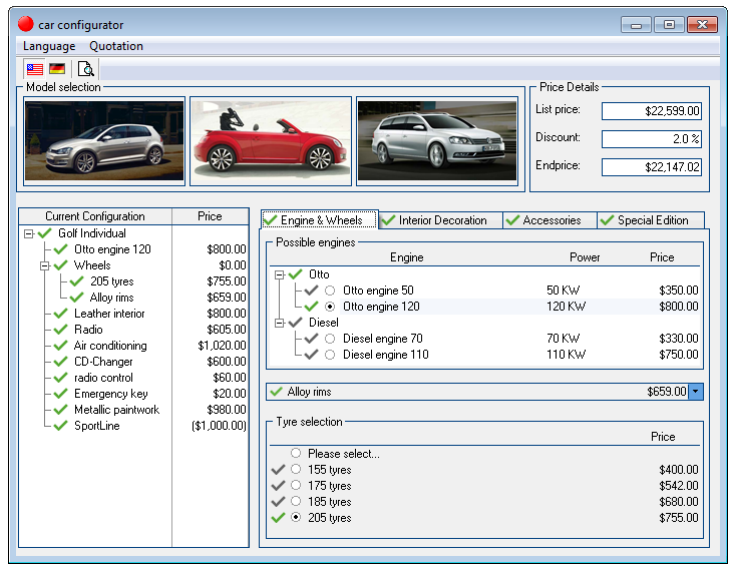
camos Develop 13.0
Modeler Training

Aim of the training



Create Project

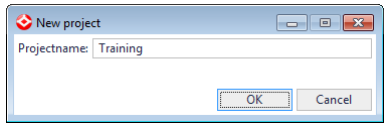
- **Project bundels the knowledge bases (KNB) that should be edited**
  - Definition of knb and version
    - -> all users using this project are editing the same KNBs
  - KNBs not included in the project can not be edited

- **Definition of a Ticket system**
  - For each action a Ticket is created
  - Integrated Ticket system

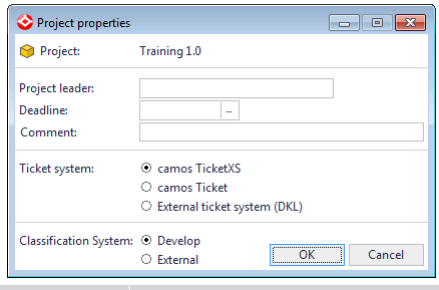
Create Project

- **Open Project Administration**

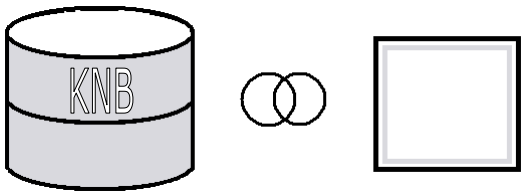
- **Create new project**



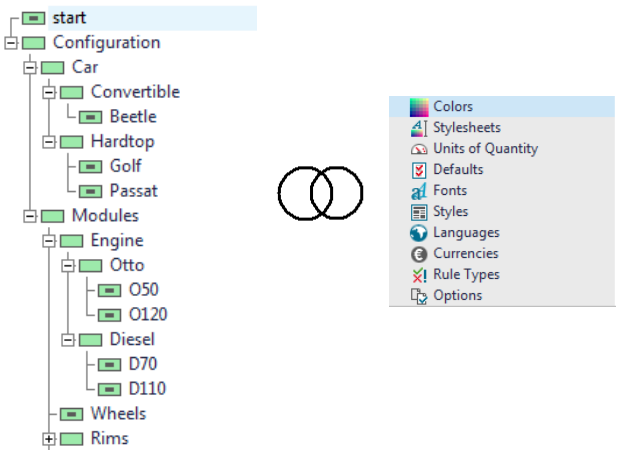
- **Connection to Ticket interface**
  - camos TicketXS
  - Identification via
    project name and
    project version



Knowledge base and frame

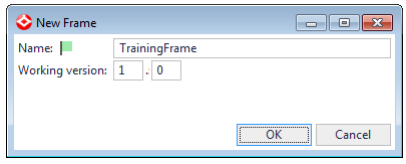


Knowledge base and frame

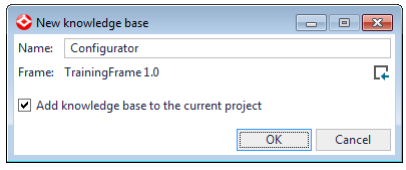


Practice: Create Frame and Knowledge base

- **Create new frame**



- **Create new knowledge base**

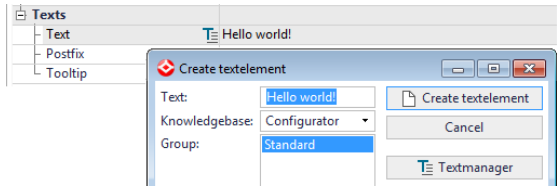


Create ticket / activate it

- **In order to create / edit classes an active ticket is necessary**
- **Open tab page TICKET**
- **Create new ticket via icon**
  - e.g. „setup knowledge base"
- **Ticket is automatically set as actual ticket**
  - alternatively set the actual ticket by

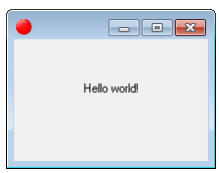
First application "Hello world!"

- **Create class „start"**
- **Create knowledge base element Form with the name „MainForm"**
- **Create form element „Label static"**
- **Define text**



First application "Hello world!"

- **Create new knowledge base element method "new"**
  - Call the command to open the MainForm
- **Start the interpreter**
- **Result at runtime:**



camos.

Starting and stopping the application

Start / Continue after a stop <F5>

- Debugger runs
- Debugger stops because of breakpoint / pause
- Interpreter error
- Debugger runs in another knowledge base

Restart <Ctrl+F5>

- The already active interpreter run is restarted

Stop

- The interpreter run is aborted at the current position

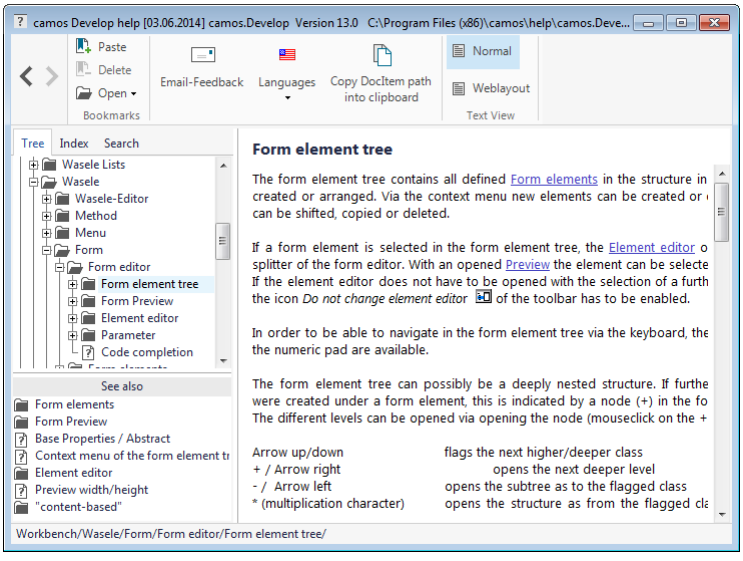
camos.

Context oriented Help System

- **Open the Online-Help via F1 / Icon ? (title line on the right)**
  - Tab page tree
    - Structure of topics
  - Search
    - Lookup (Ctrl+L)
    - full-text search
    - full-text search for selected chapters (Advanced search)
  - Feedback to camos

camos.

Context oriented Help System



camos.

What have you learned so far?

- Create a project
- Create and activate a ticket
- Create frame and knowledge base and link both
- Create classes
- Create form and form element
- Start the debugger
- Use the Online-Help

camos.

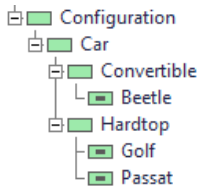
Targets of the next chapter

- Build structure of product
- Know the difference between classes and objects
- Understand advantages of inheritance
- Elaborate the form
- Define actions

camos.

Create products

- **3 different models should be available**
  - New Beetle Cabriolet
  - Golf Individual
  - Passat Variant
- **Pooling of similar products**
  → base classes
- **The following class structure results**



camos.

User Guidance

- **All elements are administrated via**
  - Context menu New, Open, Delete, Copy, etc.
  - Toolbar
  - Hotkeys and shortcuts
- **Open always via double-click!**
- **Navigation in the workbench (menu View -> Options)**
  - Tab pages
    - Elements are opened in tab pages
    - Number of editors -> definition in user options
  - List of opened classes / editors
    - Only one element is visible
    - Selection of the opened elements from a list

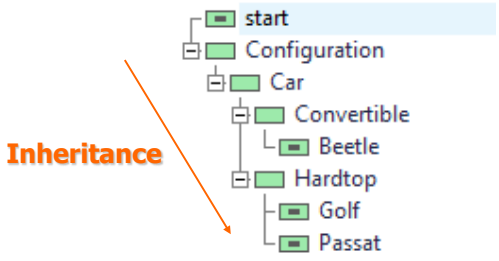
camos.

[Object Orientation] Classes & Objects

- **Classes**
  - „Model" for objects

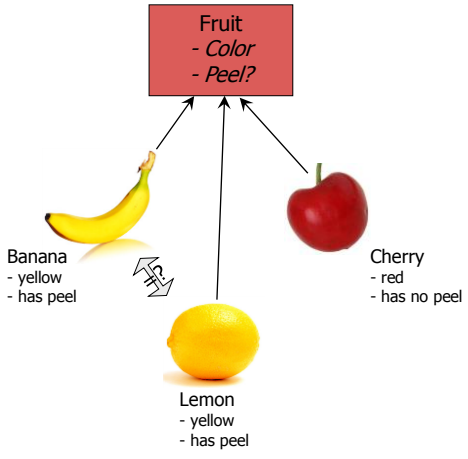
- **Objects**
  - Correspond „real" objects
  - Exist during program run
  - Act corresponding to class definition
  - Several objects of one class possible

[Object Orientation] Inheritance



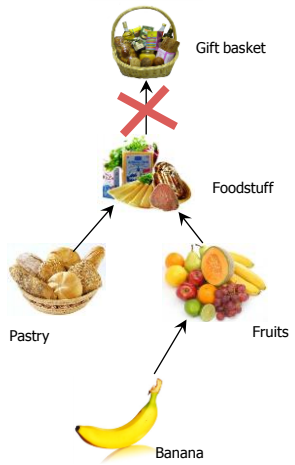
[Object Orientation] Example

- **Select abstraction/inheritance suitably for the application**



[object orientation] Example

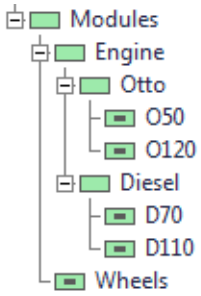
Inheritance ("is a")



Aggregation ("consists of")

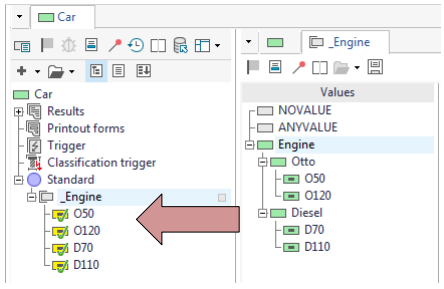
Attach construction elements – Class structure

- **Configure the following modules**
  - Engine (Otto or Diesel engine)
  - Wheels (mandatory)
- **Class structure**



Attach construction elements – Object structure

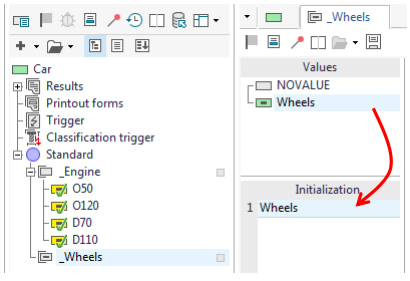
- **How to attach the engine to the car?**
  - The modules are assigned to the class "Car" as components
  - The possible engine versions are applied as potential values



Components – Initialization

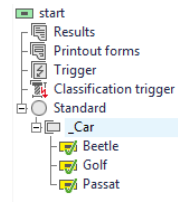
- **Initializations**
  - The wheels can not be configured
  - To mount the tyres and rims, the wheels have to exist

  → Create component _Wheels and initialize it

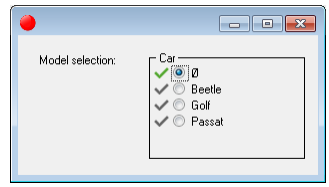


Selection of the car models on the form

- **Connection between the car and the class start**
  - Create component "_Car" in class "start"
  - Apply the tree models as potential values

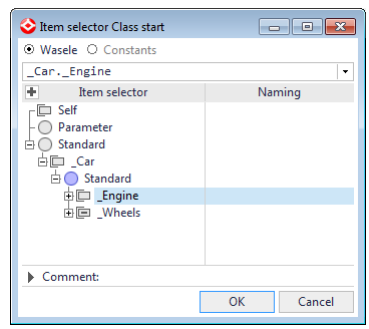


- **Display on the MainForm**



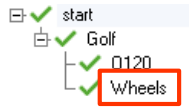
Configuration of the modules

- **The engine should be configurable on the form**
  - New Configurationbox component
  - Cause variable is the engine in the car: _Car._Engine



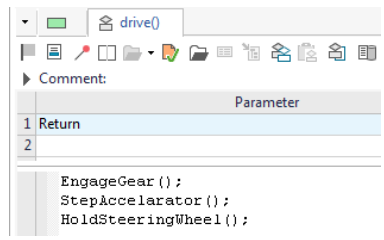
Component tree

- **Overview of configuration**
  - Create form element component tree
- **What's the use of the Component tree?**
  - Shows the current object structure
  - Includes also those objects that have not actively been selected by the user



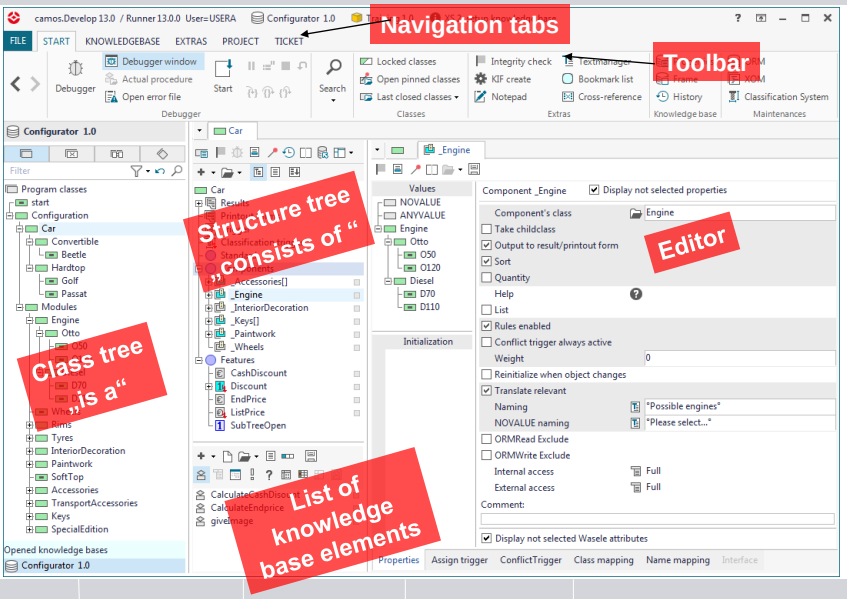
[Object Orientation] Methods

- **Methods**
  - define the behavior of an object
  - Methods contain instructions (program code)



- **Special Methods**
  - New (Constructor)
  - Delete (Destructor)

camos Develop User Interface



Difference between class tree and structure tree

- **Class tree**
  - „warehouse“

- **Structure tree**
  - „factory work room“

camos.

What have you learned so far?

- Building the product structure
- Difference between class and object
- Combine modules under base classes
- Create form elements for the configuration
  - Configurationbox component
- Create module objects without user action
  - Initialization
- camos Develop user interface

camos.

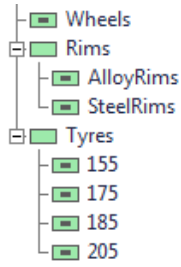
Targets of the next chapter

- Create and assign elements of the wheels
- Select car models via graphics
- Design the application multilingual
  - Selection of the dialog language
- Administrate the texts of the application

Extend class structure

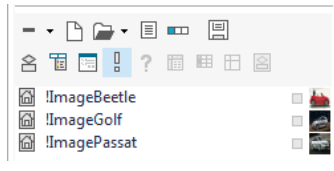
- **The wheels consist of these construction elements**
  - Tyres
  - Rims

- **Different values should be possible**

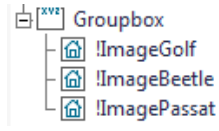


Form element graphic

- **The selection of the model is carried out via a graphic**



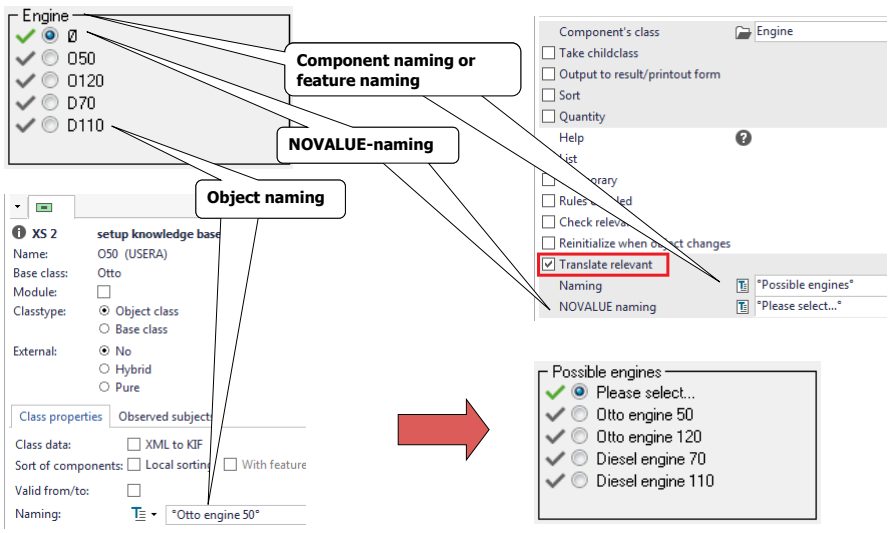
- **Delete configbox for car and create a groupbox**
  - Groupbox with three graphics
  - Graphic constants as cause variables
  - Define selection trigger



Multilingual namings

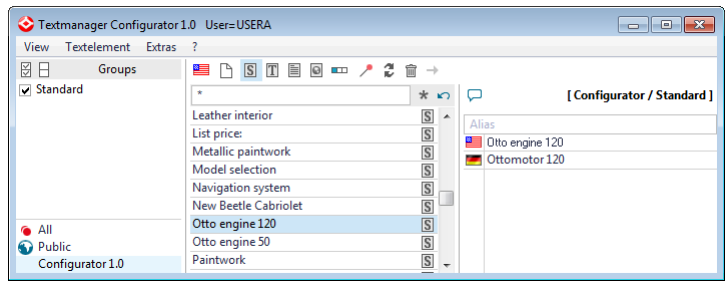
- **The configurator has to be bilingual**
  - Create new language "Germany" in the frame
- **Define individual namings**
  - Object naming: in the corresponding class editor
  - Component- and NOVALUE naming: in the component editor
    - Enable "Translate relevant"
- **Display object naming in the component tree**

Multilingual namings



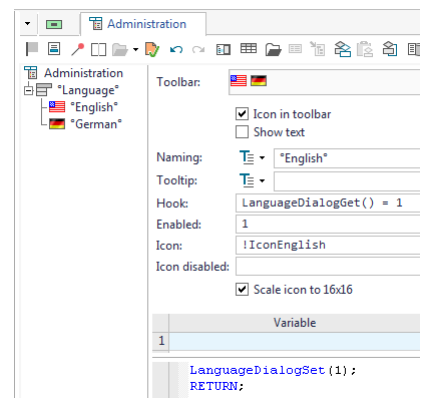
Administrate the translation in the textmanager

- Textelements are changed in the textmanager
- Define a filter to view the textelements
- Enter the translation in the right table



Menus

- **Switch language via menu on MainForm**
  - Menu "Administration" in start
    - Menu title "Language"
      - Menu trigger "English"
      - Menu trigger "German"
        - Naming
        - Icon
        - Icon in Toolbar
        - Action
        - Hook



  - Allocate menu to the MainForm

camos.

What have you learned so far?

- Using the form element graphic (product pictures)
- Define Selection trigger
- Create language in the frame
- Knowledge base element menu (language switch)
  - Create
  - Define
  - Allocate at MainForm
- Administrate texts in the Textmanager

camos.

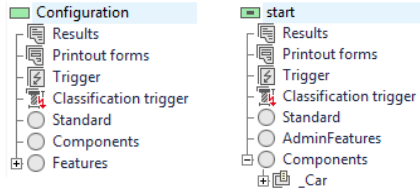
Targets of the next chapter

- Make knowledge base clearer with groups
- Understand initialization
- Create price calculation
- Maintain the engine power
- Specialize by defining overloads

Groups

- **Knowledge base elements can be structured in groups**
  - Default: group "Standard"
  - Further groups are optional

- **Create the following groups in the structure tree:**
  - "AdminFeatures" and "Components" in start
  - "Components" and "Features" in Configuration
  - Shift existing components from group "Standard" to "Components"



Create features

- **All construction elements have a price**
  - Create main currency USD in the frame

- **Define the feature "Price" in "Modules"**
  - Data type currency
  - Init value 0

- **Define prices for all modules**
  - Overload

- **Display individual prices on form**
  - Extend configboxes by a configbox column

camos.

Reinitialize when object changes

- **Check the display of the prices**
- **-> When you swap between different engines, the currently selected engine always gets the price of the first selected engine!**
  - Reason: The init value of the engine price is not newly set with the object change (selection of a different engine)
  - Solution: Enable option "Reinitialize when object changes" on feature Price

camos.

Price calculation

- **The price of the car consists of:**
- **a base price plus the sum of its construction elements**
  - Procedure: With the selection of a construction element its price is added to the list price of the car, with a deselection it is subtracted
    - Method new() -> Price addition
    - Method delete() -> Price subtraction
  - Create the currency feature "ListPrice" in car
  - Initialize it in any model, e.g. Beetle 15999,-
  - Each construction element must be able to access the ListPrice of the selected car model
    - Create a predecessor component @Car in modules

Price calculation II

- **Method PriceAdd() in Car**
  - Parameter (currency): Price_
  - Code: `ListPrice := ListPrice + Price_;`
  - Alternative shorter code: `ListPrice += Price_;`
- **Method PriceSubtract() in Car**
  - Parameter (currency): Price_
  - Code: `ListPrice := ListPrice + Price_;`
  - Alternative shorter code: `ListPrice += Price_;`
- **Check method syntax**
  - The syntax check is called via or Ctrl+S

Price calculation II

- **Method New() in Modules**

```
@Auto.PriceAdd(Price);
```

- **Method Delete() in Modules**

```
@Auto.PriceSubtract(Price);
```

- **Display the list price on the form**
  - Groupbox with a label and a form element Currency
- **Start the configurator and check the price calculation!**

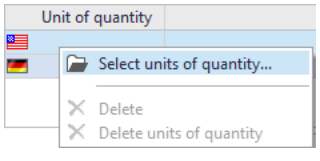
Display engine power

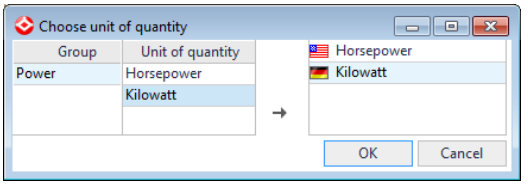
- **The power of the engine should be additionally displayed in the configbox**

- **Power is a attribute of the engine**
  - Create feature "Power" of the type numeric
  - Enable the option "Reinitialize when object changes"
  - Overload the power in each engine,
    e.g. class D110 -> Power = 110
  - Display power in configbox engine (configbox column)

Units of quantity (UOQ)

- **The power should be displayed in the corresponding unit**
  - Create unit of quantity (=UOQ) group "Power" in the frame
  - Master UOQ: "Kilowatt", unit "kW"
- **In Germany the power is specified in horsepower**
  - Second UOQ "Horsepower", unit "HP" (USA) and "PS" (Germany) with the conversion factor: 1,358
- **Allocation of the UOQ in the feature editor**
  -> Context menu in the table column "Unit of quantity"





camos.

[Object Orientation] Inheritance & Overloading

- **Repeat: child classes receive automatically the characteristics and the behavior of the father class**

- **Specialization of the child class through**
  - Adding new methods and features
  - Change the existing attributes and behavior
    → Overloading methods and initializations

- **In our example:**
  - Each engine class receives its own value for power

camos.

Further construction elements

- **The car is extended by the following classes**
  - Interior decoration: fabric, leather
  - Paintwork: normal or metallic, each in the colors blue, black, red or green
  - Soft top
  - Accessories: e.g. radio, air condition, sunroof, CD-changer, roof baggage carrier, navigation system

- **Define prices and namings**

camos.

Further construction elements

- **Add components, apply potential values**
  - Initialize Soft top (only for convertibles)
  - Several accessory parts can be selected -> list
- **Color**
  - Create textelements
  - Assign textelements in string constants
  - Use the constants as possible values
  - Important: feature has to be multilingual

camos.

What have you learned so far?

- Making the knowledge base clearer with groups
- Defining currency and unit of quantity in the frame
- „Reinitialize when object changes" so the initialization is updated when the selection is changed
- Access to predecessor objects (@Car)
- Multiselection via property list ([])

camos.

Targets of the next chapter

- Administrate access rights on knowledge bases and classes
- Different class modes
- Ruling product combinations
  - Get to know the requirements to work with rules
  - Different rule types
  - Create and maintain rules

camos.

Class administration

Each class can be protected against unauthorized changes

- The security is defined with reference to user/user groups
- No access, read, edit and security …
- Default security can be defined in the knowledge base properties

camos.

Class administration

- **In the multiuser mode**
  - To edit a class -> Reserve
  - Update changes in the ticket -> Refresh
  - Confirm changes in the ticket -> Release
  - Confirm changes globally
    - -> Release ticket
    - -> Release and unlink from ticket
  - Transfer class to another user of the ticket -> Transfer
  - Cancel changes -> Discard
  - Cancel changes and unlink class from the ticket -> Discard and unlink from ticket

camos.

The rule work

- **Use**
  - Forbid, hide, assign, … potential values based on a condition
    - E.g. a sunroof cannot be selected for any convertible
- **Requirements**
  - Demand license for rule processing, via

    ```
    LicenseDemand('camos.Configurator');
    ```
  - Activate option "Rules enabled" on feature/component
    - Icon of ruled knobele in structure tree are turquoise, e.g.
- **Display of the validity via validity symbols**
  - Value is allowed
  - Value is forbidden
  - Value has to be selected

camos.

Rule types I

May-rule
- Allows a value

May not-rule
- Forbids a value. The user can still select the value.

Must-rule
- The value has to be selected, other values are automatically forbidden. The user can still make a different selection.

Assignment-rule
- Assigns the value automatically. The user can no longer make a different selection or delete the selection.

camos.

Rule types II

Assign/Delete-rule
- Like assignment-rule; however, if the condition (that lead to the assignment) is no longer fulfilled, the assignment is made undone; the knobele is set to NOVALUE.

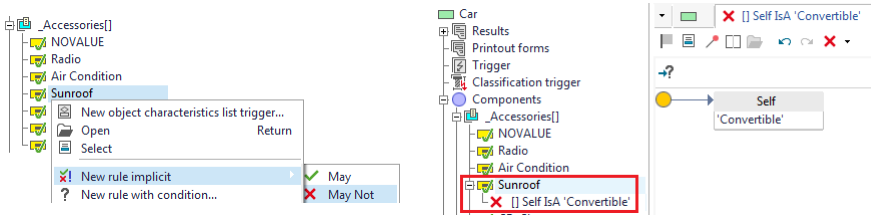
Invisible/May not-rule
- Forbids and hides the value. Contrary to a value with a may not-rule, the value is not displayed in configurationboxes.

Invisible
- Hides a value. Contrary to the Invisible/May not-rule, the value is not forbidden but only (if not currently selected) faded out.

Rule editor

- **Syntax**
  - The rule is defined on the value that has to be allowed/forbidden/etc. following the syntax: **value** - **rule type** - **condition**

-> Accessory sunroof may not be selected if car is a convertible



  - "Self" is used to access "to itself", so to the "own object"
  - The comparing operator "IsA" is used to expect, if a object is from a special class type

Rule editor

- **Condition elements**
  - "List" -> concrete feature/component
  - "Expression" -> simple expression (comparison, Get-function, method without side effects)
  - "Multibox" -> for many with "AND" linked part conditions
  - Several part conditions can be linked with AND/OR/XOR

- **The condition is true, if the result of all part conditions is logical true**

- **The written condition is shown on the tab page "Expression"**

Rule processing

Always allowed

- Values that are applied in the structure tree are by default "Always allowed"

- **Weight**
  - Rule declarations at concrete object classes are previous to rule declarations at abstract base classes
  - May not-rules are previous to may-rules
  - Rules are previous to "(not) always allowed"

- **When are the rules worked off?**
  - The rule check is activated as soon as a value of a check relevant or ruled knobele is changed in the object tree
  - More information: see chapture „Inference machine" in online help

Exercise: Creating rules

- **The diesel engines can only be selected for the Beetle and the Golf**

- **The 205-tyres are forbidden for the O50 and D70 engine**

- **The CD-changer is only available in combination with a radio**

- **A roof baggage carrier and a sunroof cannot be selected for convertible models**

- **If you choose for a Passat the engine O120 or O50, alloy rims, 185 tyres and paintwork metallic, then the paintwork color has to be black**

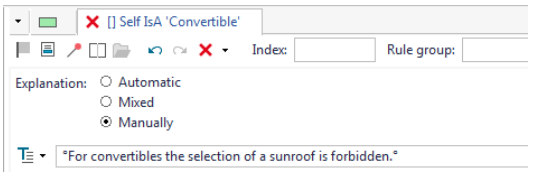
Property "Check relevant"

- **The property "Check relevant" defines, if a change in value of the feature/component activates a recalculation of the rule work**
  - Target: performance improvement via avoiding needless rule check
  - By default the property is deactivated!
  - Ruled features and components are automatically check relevant

- **Features and components, which are used in rules, has to be check relevant!**
  - Execute the integrity check and activate the property "check relevant" at the listed knowledge base elements

Rule explanation

- **Explanation mode**
  - Automatic
    - Explanation is generated automatic from condition part and rule type
  - Mixed
    - Explanation is a partly automatic generated and a partly manually entered statement
      → Engine is forbidden if {manually statement}
  - Manually
    - Explanation can completely defined manually

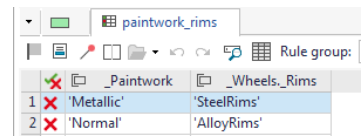


Conditions

- **Swap out repeated conditions**
  - Frequently occurring identical rule conditions can be swapped in one condition (knowledge base element ?)
- **Pros**
  - Reduces the maintenance effort
  - Changes in the condition concern immediately all rules
- **Proceeding**
  - Add knowledge base element condition
  - With context menu item "New rule with condition …" at the particular value, the condition can be used in rules
- **Practice: Convert convertible-rule into a condition:**
  - "With convertible models no roof baggage carrier and no sunroof can be selected"

Constraints

- **Use**
  - Mapping of two-way dependencies between knowledge base elements
    - Allow / forbid / hide value combinations
- **Requirements**
  - The flag "Rules enabled" has to be activated for the knowledge base elements that are used in the constraint
- **The following combinations exclude each other**
  - Paintwork metallic and steel rims
  - Paintwork normal and alloy rims
  - Air condition and sunroof
  - Sunroof and roof baggage carrier



- **To do so, create two constraints in Car**

camos.

What have you learned so far?

- Class administration (Security and Editing)
- 7 Rule types
- Rule syntax (value – rule type – condition)
- Necessity of the property Check relevant
- Different kinds of rule explanations
- Knowledge base element condition and constraint

camos.

Targets of the next chapter

- Calculate discounts
- Create quotations
- Protect different knowledge base stands
- Expand the class structure
- Show the model specific interface

Discount calculation

- **A discount can be given to the list price**
  - Create a numeric feature "Discount" in class Car
    - Init value 0
    - Format: 2 digits before and 1 digit after the dot (##.#)
    - Define the UOQ % in the frame
  - Create a currency feature "EndPrice"
  - Extend the form (groupbox price)

- **Method CalculateEndPrice()**

```
EndPrice := ListPrice * (1 - Discount / 100);
```

- **Method has to be called when either the discount or the list price changes -> assign trigger**

```
CalculateEndPrice();
```


Ranges

- **Discount has to be between 0 and 10 percent**
  - Define the range [0 .. 10] and apply it as a potential value
  - Enable the option "Rules enabled"

- **End price calculation only with valid discount**

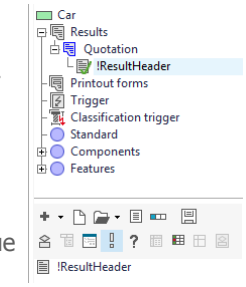
```
IF not IsValid(Discount) THEN
WinMessage('ERROR', °Discount has to be between 0 and 10 %°);
Discount := LastValue;
ENDIF;
CalculateEndprice();
```

- **LastValue contains the last value**

camos.

Results

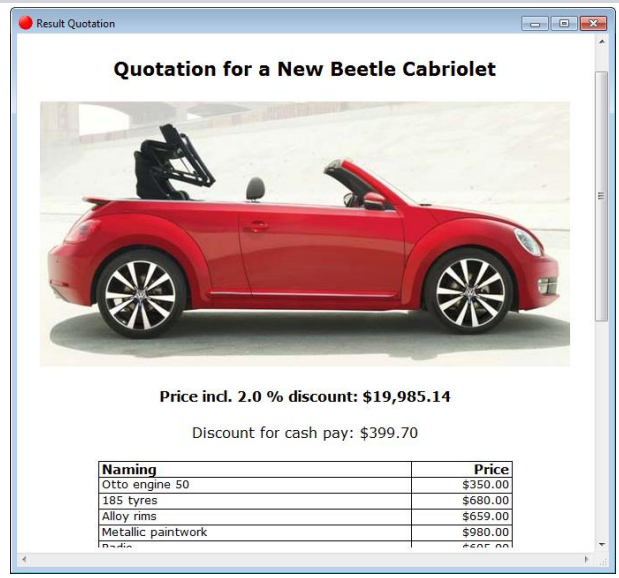
- **Basics**
  - Defined in the knowledge base properties
    - Layout, format and footer/header can be set for each language
  - A result consists of individual text modules
    - Constants of the type String, RTF or HTML
    - Cause variables are replaced by the current value during runtime
  - Assign the result in the structure tree to the class and apply the text modules
- **Warning: set the option "Output to result/printout form"**



camos.

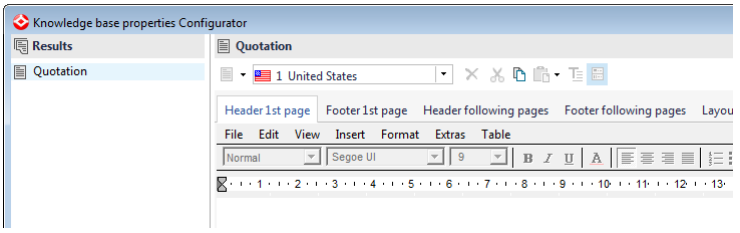
Exercise: Result I

Create a quotation like this example:



Exercise: Results II

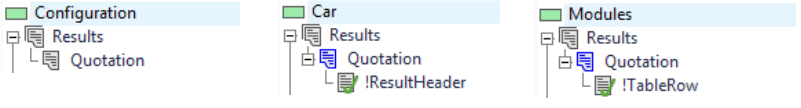
- **Create result "Quotation" of the type RTF**



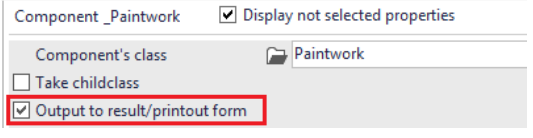
- **Create RTF constants and fill them with cause variables**
  - Quotation header with product icon and price information (in class Car)
  - Table row with description and price of construction elements (in class Modules)

Exercise: Results III

- **Define result "Quotation" in class "Configuration"**
- **Apply text modules in "Car" and "Modules"**



- **Enable option "Output to result/printout form" at all components**
  - Tip: search and replace with the search for properties



Exercise: Results IV

- **Result has to be called via menu**
  - Extend the menu Administration
    - Menu trigger "Quotation"
      - Load graphic constant *!IconQuotation* and apply to menu trigger
      - Enable option "Icon in Toolbar"
      - Procedure editor opens the modal result window

```
WinStartModal(WinOpenDoc('Quotation', 0, 0, 800, 600));
```

- **Result can only be opened if a car is selected**
  - Enable-zone controls if a menu trigger is active
    - 0 -> enable
    - <> 0 -> disable
  - Insert expression: `_Car = ANYVALUE`

Versioning

- **Aim: to fix a particular development state**
- **-> freeze the working version and create a release version**

- **Menu Knowledge base -> Administrate ...**

Create new working version
Freeze working version
Create new release version

- **Before you release**
  - all classes have to be released and unlinked from the ticket / the ticket has to be released
  - a release-frame has to be assigned

camos.

Versioning

- **Release process**
  - Step 1: Release frame
  - Step 2: Freeze knowledge base
  - Step 3: Release knowledge base
  - Step 4: Create a new working version (1.1 or 2.0) of the frame and the knowledge base
- **Frozen working versions and released versions can be opened, but they cannot be edited later anymore.**
- **The following exercises will be made in the new working versions!**

camos.

Extend class structure

- **For the Passat additional accessories can be selected (multiple choice)**
  - New base class "TransportAccessories"
    - Roof rail
    - Transport box
    - Bicycle carrier
  - Define prices for the new construction elements
  - Create a list component of "TransportAccessories" in Passat
    - Apply potential values
    - Set option "Output to result"
    - Define a rule for transport accessories: "If a bicycle carrier or a transport box is selected, the roof rail is automatically selected."

camos.

Subforms

- **Target: The choice of the transport accessories are only displayed and possible if a Passat is selected**
- **New form, which is visible in the MainForm**
  - Create a new form "DetailForm" in class "Car"
- **"DetailForm" contains only elements for the Passat (= add specific components only in the corresponding class)**
  - Overload "DetailForm" in class "Passat" and add a configbox for the _TransportAccessories[]
- **Create a dynamic subform in the "MainForm"**
  - Subform should display the "DetailForm" of the praticular car
    - Form object: _Car
    - Form name: DetailForm

camos.

What have you learned so far?

- Calculate the discount
  - Numeric format
  - Assign trigger
- Result
  - Definition in the knowledge base properties
  - "Output to result/printout form" for components
- Save development state
  - Working version
  - Frozen version
  - Release version
- Dynamic Subforms

Targets of the next chapter

- Calculate cash discount
- Combine the special equipment of the Golf
  - Price advantage & module combination
- Ruling prices with quantity
- Import and export knowledge bases
- Tipps and tricks

2D-Tables

- **Cash discount if a customer pays cash**
  - Cash discount factor is deposit in a 2D-Table
    - Input variables: model and end price
    - Return value is a factor

X-value type: $ Y-value type: € Return type: 1 Default: 0

		X1	X2	X3	X4
		'Beetle'	'Golf'	'Passat'	
Y1	< 17000	1.2	1.5	2	
Y2	< 25000	2	2.25	2.75	
Y3	>= 25000	3	4	5	

- **Define in class Car:**
  - Currency feature CashDiscount
  - 2D-Table CashDiscountTable as shown above

Calculate cash discount

- **Method CalculateCashDiscount() in Car**
  - Create local variable factor (numeric)

```
factor := CashDiscountTable(Self, EndPrice);
CashDiscount := factor * (EndPrice / 100);
```

- **Discount is shown in result**
  - Include feature CashDiscount with cause variable
  - Call calculation of cash discount before opening the quotation
    - In the procedure editor of the menu trigger quotation

```
_Car.CalculateCashDiscount();
```


Extend class structure

- **Keys can be selected to a car**
  - New base class "Keys" with object classes
    - "Remote control key" ($ 60)
    - "Emergency key" ($ 20)
  - List component _Keys[] in Car
    - Init one key per remote control- and emergency key
    - Apply potential values
  - Add a configbox component including a price column

- **It should be possible to select several remote control- and emergency keys**
  - Define a UOQ "Amount" in the frame
  - Assign the UOQ in the class editor of "Keys"
  - Set option "Quantity" at the component _Keys[]
  - Add a quantity column to the configbox

Price calculation for quantities

- **The quantity of the keys has to be considered in the price calculation**
  - Overload the method new() in "Keys"

```
@Car.PriceAdd(GetQuantity(Self) * Price);
```

  - Overload the method delete() in "Keys"

```
@Car.PriceSubtract(GetQuantity(Self) * Price);
```

  - Create a method ChangeQuantity() in "Keys"
    - Method is called when the quantity of an object changes
    - The internal parameter NewQuantity contains the new quantity

```
@Car.PriceSubtract(GetQuantity(Self) * Price);
@Car.PriceAdd(NewQuantity * Price);
```


Ruling quantities

- **The quantity of remote control- and emergency keys is limited**
  - In general no more than 5 remote control keys are permitted
  - Permitted quantity of emergency keys varies depending on the model
    - Beetle max. 3 emergency keys
    - Golf max. 4 emergency keys
    - Passat max. 4 emergency keys
- **Activate the option "Rules enabled" for the component _Keys[]**
  - May not-rule on RemoteControlKey with quantity > 5, always forbidden
  - May not-rule on EmergencyKey
    - with quantity > 3, if Beetle is selected
    - with quantity > 4, if Golf or Passat is selected
- **Forbid deleting**
  - Assign rule per value
    - Condition: 'Remote control key' NotIn _Keys[]
    - Condition: 'Emergency key' NotIn _Keys[]

Extend class structure

- **For the Golf a special edition can be selected**
  - New base class "SpecialEdition" below the Modules
    - Object classes "TrendLine", "ComfortLine" and "SportLine"
- **A price reduction is granted when choosing a special edition**
  - Initially value the feature Price with negative values:

model	**price reduction**
TrendLine	- 400 $
ComfortLine	- 750 $
SportLine	- 1000 $

- **Create a component in Golf and overload the DetailForm**
  - Apply NOVALUE as a potential value
  - Create Configbox component for _SpecialEdition
- **If a special edition is selected, certain components have to be automatically assigned -> Use of a decision table**

Decision table

- **Comfortable way to enter/display assign- and assign/delete-rules**
- **Assignments are triggered due to conditions**
- **Processing automatically via inference machine**

Rule number

Condition number

	Cond./Act.		R1	R2	R3
B 1	_SpecialEdition		'TrendLine'	'ComfortLine'	'SportLine'
A 1	_Paintwork				'Metallic'
A 2	_InteriorDecoration		'Fabric'		'Leather'
A 3	_Wheels._Rims		'SteelRims'	'AlloyRims'	'AlloyRims'
A 4	_Wheels._Tyres			'185'	'205'
A 5	_Accessories[]		'Radio'	'Radio'	'Radio'
A 6	_Accessories[]			'Air Condition'	'Air Condition'
A 7	_Accessories[]				'CD-Changer'

Condition(s)

Action(s)

Action number

Text section

Action section

Rule type

Exercise: Decision tables

- **Decision table "SpecialEquipment" in class "Golf":**
  - For each special edition certain construction elements are assigned

	Cond./Act.		R1	R2	R3
B 1	_SpecialEdition		'TrendLine'	'ComfortLine'	'SportLine'
A 1	_Paintwork				'Metallic'
A 2	_InteriorDecoration		'Fabric'		'Leather'
A 3	_Wheels._Rims		'SteelRims'	'AlloyRims'	'AlloyRims'
A 4	_Wheels._Tyres			'185'	'205'
A 5	_Accessories[]		'Radio'	'Radio'	'Radio'
A 6	_Accessories[]			'Air Condition'	'Air Condition'
A 7	_Accessories[]				'CD-Changer'

- **Important: The flag "Rules enabled" has to be activated for each cause variable (action) that is affected by the decision table!**
- **For the cause variables in the condition row the property "check relevant" has to be activated!**

Export of Knowledge Bases

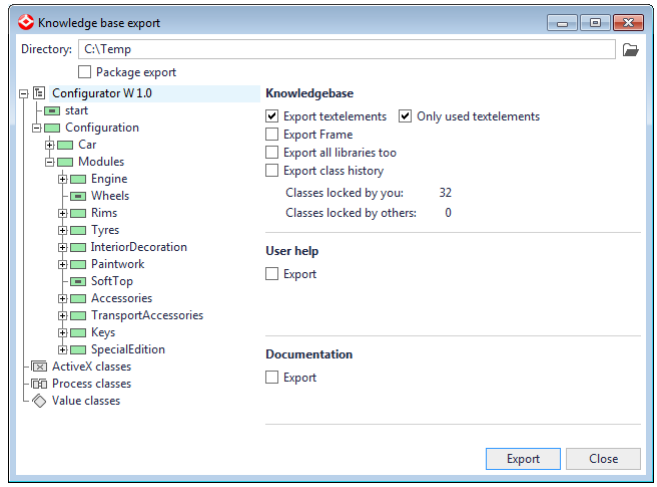
Export

- To find in the context menu of the class tree heading
- Knowledge base export *.kbx
- Complete knowledge base or individual classes
- Optional incl. textelements, frame, libraries and documentations

  - Package export (*.pgx) optional
    - Classes
    - Frame
    - Libraries
    - Textelements
    - ORM
    - ...

Export of Knowledge Bases

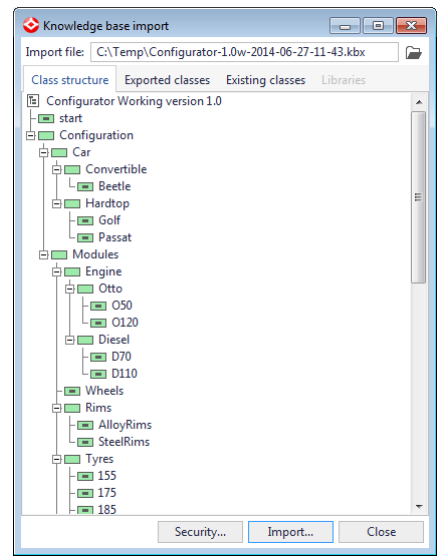
Export



Import of Knowledge Bases

Import

- Can be found in the context menu of the class tree heading
- Select the classes, that should be imported
- Determine security for imported classes
- Delete or keep existing classes



Create KIF

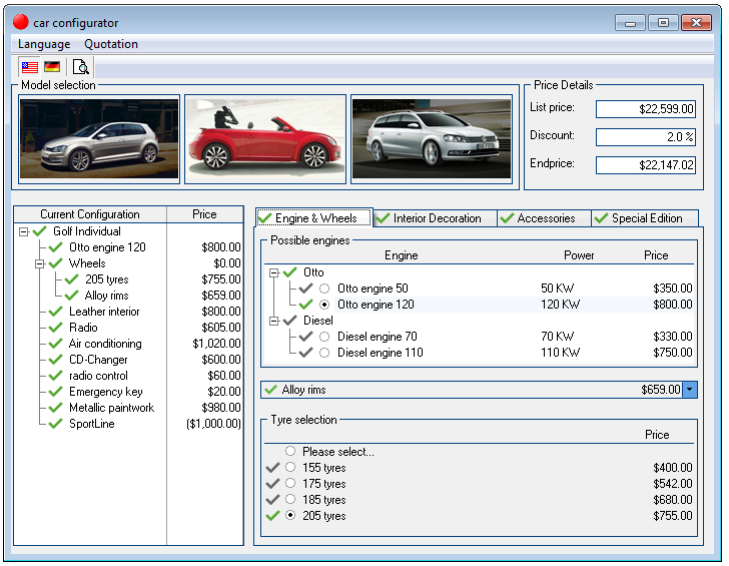
- **To start the application detached from the development environment, a KIF has to be created**
  - Release all classes (unlink from ticket or release ticket)
  - Menu Knowledge base -> KIF create
    - KIF stands for **K**nowledge **I**nformation **F**ile
    - Directory = KifDir of the camosRunner
    - Important: The option "Include textelements" has to be checked
  - Start KIF with camosWinClient
    - Create link to file *C:|Program Files|camos|camosWinClient.exe*
    - Open properties of the link and specify start parameters
      - -knb -> Name of the execute knowledge base rather whose KIF
      - -ver -> Specifies if the application is a working- or release version
      - further parameters see online help

Advanced form-layout

- **Arrange MainForm more user friendly by use of**
  - Tab frames
  - Frames, colors, lines
  - Options at configboxes
  - Optimize component tree
  - Headings, namings, tooltips
- **Delete-request when changing the model**
  - Selection trigger
- **Fit alignment**

camos.

Optimization of the user interface



camos.

Tips & Tricks: Knowledge base

- **Open the node of the component tree every time**
  - Feature SubTreeOpen in class Configuration, type numeric, init value 1
- **Start interpreter via F5**
  - Preset start class in the knowledge base properties (options)
- **Expand class tree completely**
  - Button * on numeric keypad
- **Default for NOVALUE naming**
  - Frame -> Defaults -> Features/Components
- **Dealing with the recycle bin**

camos.

Tips & Tricks: Rules

- **Forbidden values should not be seen in a configbox**
  - Enable option "Allowed values only" on the form element
- **„Ruled initializations" through assignment rule**
  - Preset color depending on the car
    - Set rule node „NOT IsUserTouched"-> Rule executes only, if the user did not select the color manually
- **Display lines in trees**
  - Menu User -> User options -> Tab page „Miscellaneous" -> „Hide lines in trees"
    - Restart of Develop necessary

camos.

Tips & Tricks: Result design

- **Change the order of the construction elements**
  - Enable in the class Car Sort -> Components
  - Resort via D&D in the dialog "Sort of components"
  - Set the option "Sorting" on all affected components
- **Fade out the construction elements "Wheels" and "SoftTop"**
  - Possibility 1: overload the RTF-constant
  - Possibility 2: may-not-rule on text module
- **Feature power should be displayed for the engine**
  - Overload the RTF-constant -> insert "Power" as cause variable
- **Color should be displayed for the paintwork**
  - Overload the RTF-constant -> Insert "Color" as cause variable

camos.

Thank you very much for your attention!

… and have a lot of fun working with camos Develop!