camos.

camos Develop
Developer training

ActiveX

ActiveX

- **Prerequisites**
  - Knowledge base „Carconfigurator“ at the end of the 3rd day of the modeler training
  - Installation of required ActiveX type libraries

- **Contents**
  - Use of a FileDialog for saving/loading a configuration in the car example
  - Export of the result of an car configuration to
    - Word
    - Excel + Creation of a chart
  - Opening a PDF-file by means of an ActiveX control

camos.

ActiveX

- **ActiveX**
  - Is a uniquely defined interface so that applications of a different origin can communicate with each other

- **Components**
  - Components that support ActiveX are like camos Develop object-oriented applications

ActiveX

- **ActiveX – Object types**
  - ActiveX-, OLE-object
    - Object is allowed to generate an external application
    - Events provide information e.g. object created
  - ActiveX interface object
    - Objects to which can be referenced from outside
  - ActiveX controls
    - ActiveX objects with graphical surface
    - Mostly used as special form elements

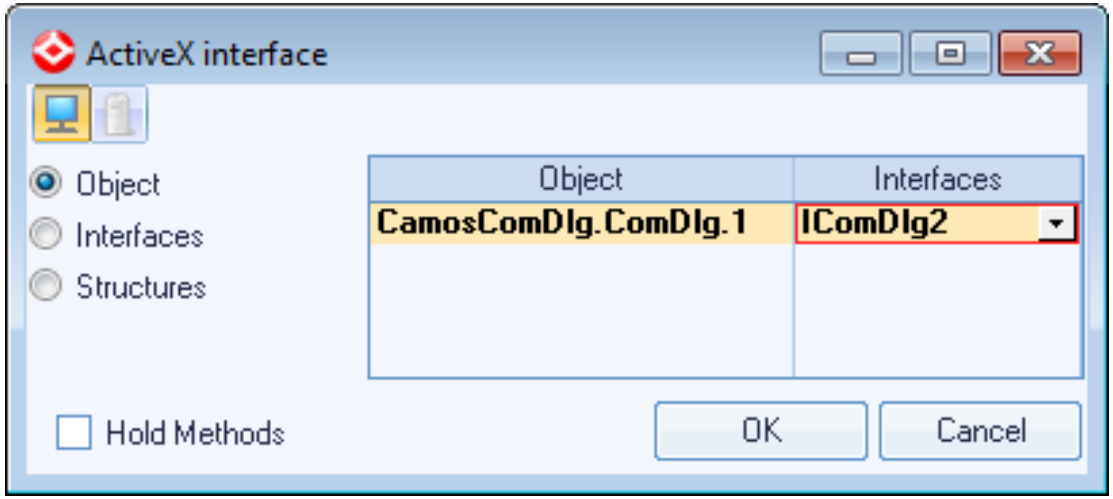
camos.

ActiveX in Develop

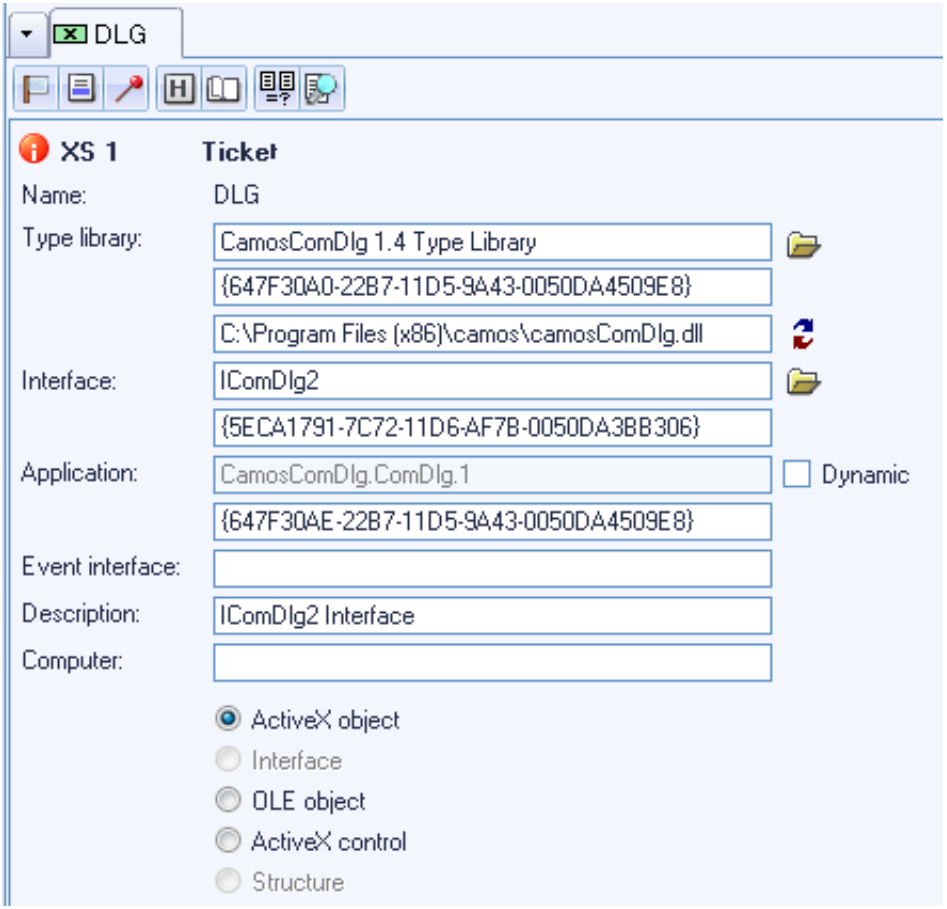
- **Exceptional features**
  - ActiveX classes need a unique name that can deviate from the name of the object
  - Every ActiveX class has to refer to an object of a type library

Exercise I: FileDialog via ActiveX

- **Create new ActiveX class „DLG“**
  - Type library: CamosComDlg 1.4 Type Library
  - Object: CamosComDlg.ComDlg.1
  - Interface: IComDlg2



Exercise I: FileDialog via ActiveX



- **Create ActiveX component _DLG in class start**

camos.

Exercise I: FileDialog via ActiveX

- **Create a method AXSave() in class start**

  Used functions:
  - ChooseFile() -> ActiveX method for file dialog
  - OT2Bin() -> converts object tree to blob
  - FileOpen() -> opens a file
  - FileWriteBin() -> writes binary data to a file
  - FileClose() -> closes a file

- **Variables:**
  - filelist[], String
  - filterlist[], String
  - filehandle, numeric
  - otbin, binary

- **Return value:**
  - numeric

Exercise I: Method AXSave()

```
# Initialize ActiveX component
_DLG := 1;
#
# Set filter: just show *.sot files
filterlist[] := {'Configurator files', '*.sot'};
# Call file selector
IF _DLG.ChooseFile(0, filelist[], filterlist[], 'C:\Temp\', 'Save', 0)
   = 1 THEN
  # Convert configuration (object tree from _Car on) to Blob
  otbin := OT2Bin(_Car);
  #
  # Open the selected file and save Blob
  filehandle := FileOpen('[Client]' & filelist[1], 'WB');
  IF filehandle <> 0 THEN
    FileWriteBin(filehandle, otbin);
    FileClose(filehandle);
  ENDIF;
ENDIF;
```

camos.

Exercise I: Method AXLoad()

- **Create a method AXLoad() in class start**

  Used functions:
  - ChooseFile() -> ActiveX method for file dialog
  - OT2Bin() -> converts object tree to blob
  - FileOpen() -> opens a file
  - FileReadBin() -> reads binary data
  - FileClose() -> closes a file

- **Variables:**
  - filelist[], String
  - filterlist[], String
  - filehandle, numeric
  - otbin, binary

- **Return value:**
  - numeric

Exercise I: Method AXLoad()

```
# Initialize ActiveX component
_DLG := 1;
# Set filter: just show *.sot files
filterlist[] := {'Configurator files', '*.sot'};
# Call file selector
IF _DLG.ChooseFile(1, filelist[], filterlist[], 'C:\Temp\', ,Load', 0)
   = 1 THEN
  # Open the selected file
  filehandle := FileOpen('[Client]' & filelist[1], 'RB');
  IF filehandle <> 0 THEN
    otbin := FileReadBin(filehandle, otbin);
    FileClose(filehandle);
    # Load Blob
    IF Bin2OT(otbin, _Car) <> 1 THEN
      WinMessage('ERROR', GetLastError());
    ENDIF;
  ENDIF;
ENDIF;
```

camos.

Exercise I: Extend menu / Call functions

- **Adapt menu Administration in class start**
  - New title „configuration“
  - New menu trigger „Load“ and „Save“
  - Call functions in the respective menu trigger

- **Load Load/Save icons as constants in class start and assign to above created menu items**

- **Saving may only be executed if a Car was created**

camos.

Exercise II: Export from data to Word

- **Task: Export to Word**
  - The result of the Carconfigurator has to be transferred in a new document to Word
  - See MSDN documentation „Microsoft Office Development“
    - http://msdn.microsoft.com

- **Necessary steps**
  - Create ActiveX classes
  - Extend the menu for the export to Word
  - Open Word
  - Create new document
  - Insert the data in the Word document

Exercise II: Export from data to Word

- **Add ActiveX classes for Word**
  - They all point to Microsoft Word 11.0 Object Library or a newer version

Class name	Interface	Type
WordApplication	_Application	ActiveX object
WordDocument	_Document	OLE object
WordDocuments	Documents	Interface
WordSelection	Selection	Interface

  - The properties are set via the dialog, which is opened by the icon in the line Interface of the current AX class

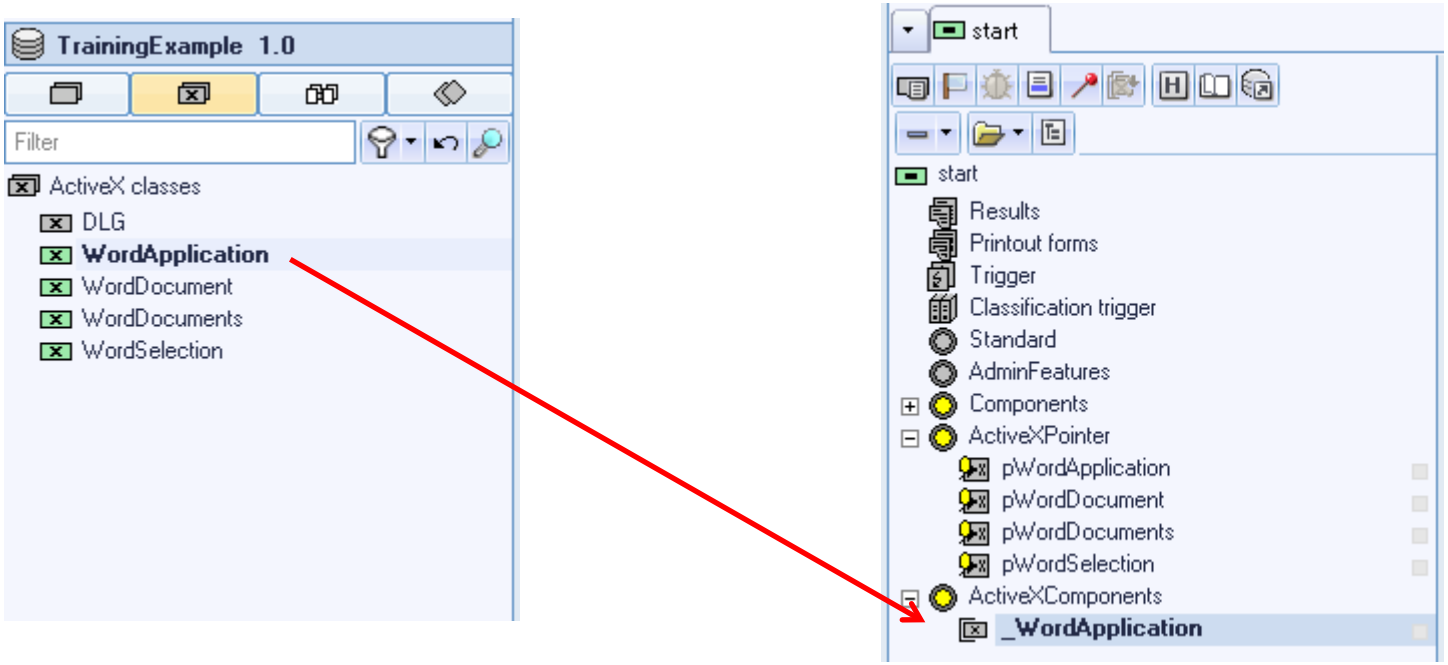
camos.

Exercise II: Export from data to Word

- **Add ActiveX pointer (class start)**
  - Create new group ActiveXPointer
  - Create the following AX pointers

ActiveX pointer	Class
pWordApplication	WordApplikation
pWordDocument	WordDocument
pWordDocuments	WordDocuments
pWordSelection	WordSelection

Exercise II: Export from data to Word



- **ActiveX component**
  - Drag the ActiveX class into the class start to create a component

camos.

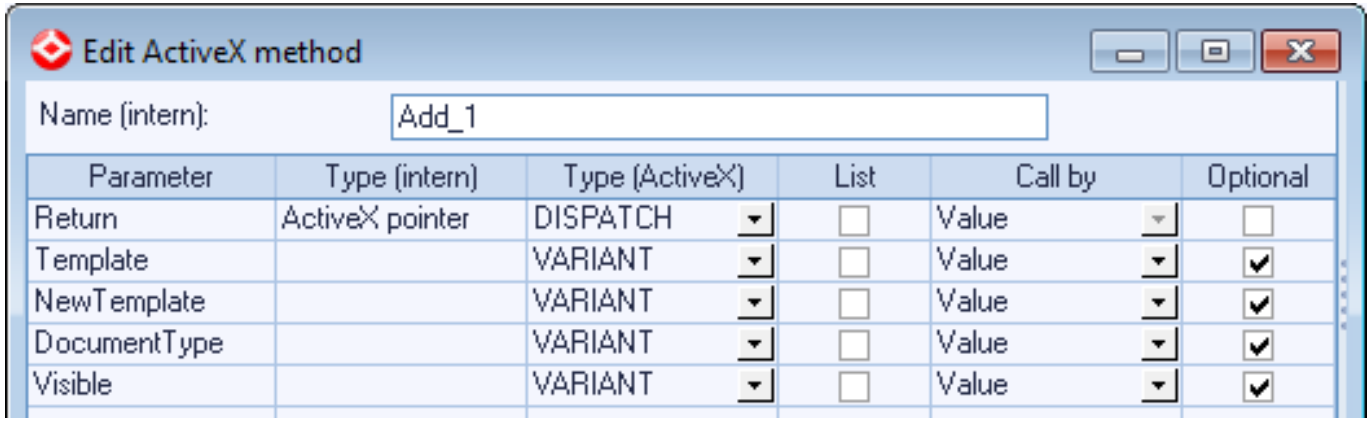
Exercise II: Export from data to Word

- **Required methods**
  - pWordApplication.SetVisible(1)
    - Word application is opened
  - pWordApplication.GetDocuments()
  - pWordApplication.Add_1()
    - A new Word document is created
  - GetDoc()
    - Generates the RTF-result
  - ClipboardWrite()
    - Graphics, ASCII-texts, HTML-texts, and RTF-texts are written to the clipboard (temporarily stored)
  - pWordApplication.GetSelection()
    - Current position in the Word document
  - pWordSelection.Paste()
    - Inserts the contents from the clipboard in the Word document

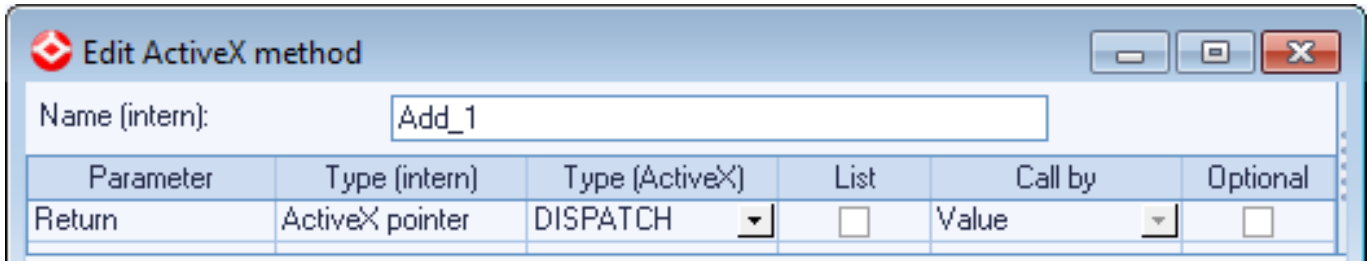
camos.

Exercise II: Export from data to Word

- **Create a new Wrapper of ActiveX method Add() in AX class WordDocuments (context menue)**



- **Delete all parameters except Return**
- **Begin from the bottom, delete via context menu**



camos.

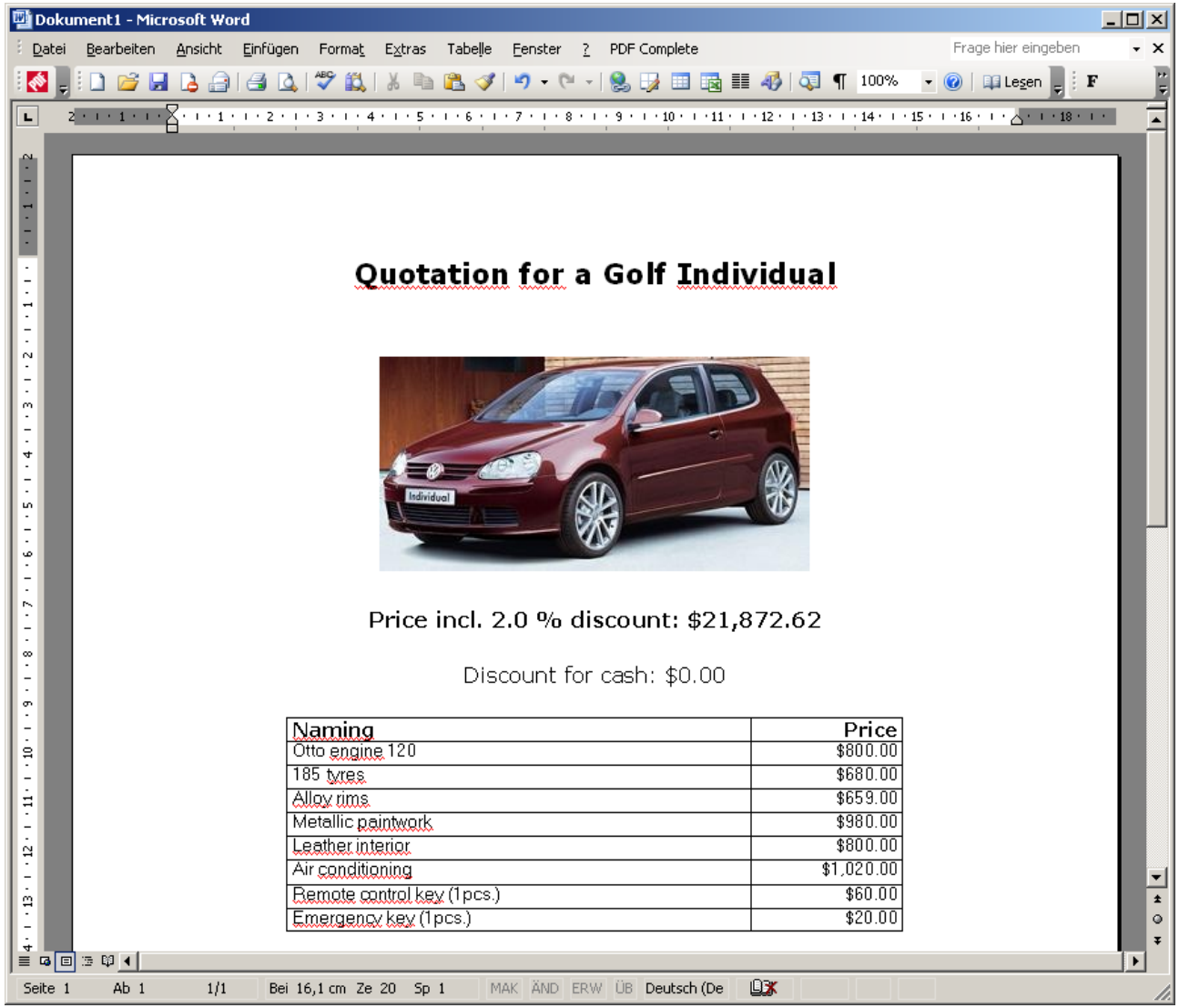
Exercise II: Export from data to Word

- **Variable ExportClipboard, RTF**

- **Source text function WordExport()**

```
# Start Word
_WordApplication := 1;
pWordApplication := _WordApplication;
pWordApplication.SetVisible(1);
# Create new document
pWordDocuments := pWordApplication.GetDocuments();
pWordDocument := pWordDocuments.Add_1();
# Write the offer to the clipboard
ExportClipboard := GetDoc('Quotation');
ClipboardWrite(ExportClipboard);
# Get the current position in the Word document and insert
pWordSelection := pWordApplication.GetSelection();
pWordSelection.Paste();
RETURN;
```

Exercise II: Export from data to Word



Naming	Price
Otto engine 120	$800.00
185 tyres	$680.00
Alloy rims	$659.00
Metallic paintwork	$980.00
Leather interior	$800.00
Air conditioning	$1,020.00
Remote control key (1pcs.)	$60.00
Emergency key (1pcs.)	$20.00

camos.

Exercise III: Export to Excel

- **Procedure**
  - Compiling the required data in list features
  - Opening the Excel application
  - Assigning the data to the Excel columns
  - Calculating the end price minus the discount
  - Creating a pie chart
  - Extending the menu

Exercise III: Export to Excel

- **Compiling the data**
  - Create two new list features in the class start
    - ObjName[] Type: String
    - ObjPrice[] Type: Currency
  - Create the method CreateComponentData() in the class Configuration
    - Parameter:
      - NameList[], type: String, call by: Write
      - PriceList[], type: Currency, call by: Write
    - Variables:
      - i, type: Numeric
      - iMax, type: Numeric
      - pModules[], type: Pointer from class Modules

Exercise III: Export to Excel

- **Create the function in the class Configuration**
- **Source code CreateComponentData()**

```
# Provides pointers to all direct successor objects of the
# current object of the type Modules
pModules[] := GetObjPtr('Modules');
iMax := MaxIndex(pComponents[]);
For i := 1 to iMax DO
   imax := MaxIndex(NameList[]) + 1;
   # Entry component name and component price in the name list
   NameList[imax] := GetNaming(pModules[i]);
   PriceList[imax] := pModules[i].Price;
   # Recursive call for subcomponents
   pModules[i].CreateComponentData(nameList[], PriceList[]);
ENDFOR;
RETURN;
```

Exercise III: Export to Excel

- **Method CreateComponentData()**
  - ... is called from the method CreateData()
  - Create method CreateData() in the class start

- **Source code of the method CreateData()**

```
ObjPrice[] := ObjName[] := NOVALUE;
# Initialization of the 1. line Name + Price Car
ObjPrice[1] := GetInitValue(_Car, 'ListPrice');
ObjName[1] := GetNaming('Car');
# Call of the recursive method for allocating the names/prices
_Car.CreateComponentData(ObjName[], ObjPrice[]);
RETURN;
```

camos.

Exercise III: Export to Excel

- **Create ActiveX classes**
  - Create the following ActiveX classes all pointing to the type library Microsoft Excel 11.0 Object Library or a more recent version

Classe name	ActiveX class	Type
ExcelApp	_Application	ActiveX object
ExcelSheet	_Worksheet	OLE object
ExcelSheets	Sheets	Interface
ExcelRange	Range	Interface

camos.

Exercise III: Export to Excel

- **Create ActiveX pointers**
  - Create the following ActiveX pointers in the class start. The pointers point to the following classes.

ActiveX pointer	Class
pActiveApplication	ExcelApp
pWorkSheet	ExcelSheet
pWorkSheets	ExcelSheets
pExcelRange	ExcelRange

- **Create ActiveX component**
  - Create an ActiveX component of the class ExcelSheet in class start

Exercise III: Export to Excel

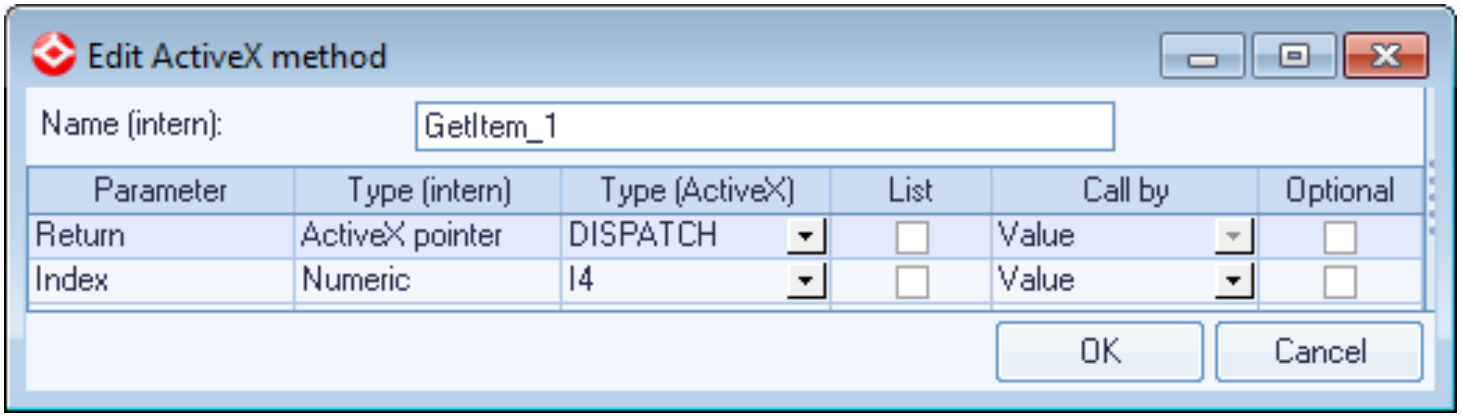
- **Create a method OpenExcel() in the class start**
  - Source text for the method

```
_ExcelSheet := 1;
# Assign pointer
pWorkSheet := _ExcelSheet;
# Return of the application object, i.e. the Excel application,
# via the GetApplication method
pActiveApplication := pWorkSheet.GetApplication();
# Get the sheets list that displays all table sheets in the
# active worksheet
pWorkSheets := pActiveApplication.GetWorksheets();
# Assign the first worksheet object from the worksheet list
pWorkSheet := pWorkSheets.GetItem_1(1);
# The register is enabled
WinSetEnable(WinGetHandle(), 'Register', 1);
RETURN;
```

Exercise III: Export to Excel

- **The syntax check does not know the method GetItem_1()**

- **This is why this method is created in the AX class ExcelSheets**
  - Create a new wrapper for the method GetItem()
  - Change the data type of the parameter Index from VARIANT to I4



- **Create a new menu title and a menu trigger in the menu Administration, which calls the method OpenExcel()**

camos.

Exercise III: Export to Excel

- **-> Excel application was opened**

Target:

- **Collected data has to be transferred to Excel**

- **Create a method WriteExcel()**
  - The headings are defined in the cells A1 to D1
  - Column A contains all component names
  - Column B contains all prices for the components
  - Column C contains the discount in %
  - Column D contains the end price

- **All prices have to be converted from the type Currency to the type numeric so that a pie chart can be displayed later**

Exercise III: Export to Excel

- **Source text of the method WriteExcel() – Part 1**

```
# Set the name and price list in Excel
#+ Discount rate + Calculation of the end price
iMax := MaxIndex(ObjName[]);
FOR i := 1 TO iMax DO
   IF i = 1 THEN
# Set the heading for
# component in position A1/A1
pExcelRange := pWorkSheet.GetRange_1('A1', 'A1');
pExcelRange.SetValue_1('Component');
# Price in postion B1/B1
pExcelRange := pWorkSheet.GetRange_1('B1', 'B1');
pExcelRange.SetValue_1('Price');
# Discount in position C1/C1
pExcelRange := pWorkSheet.GetRange_1('C1', 'C1');
pExcelRange.SetValue_1('Discount[%]');
# End price in position D1/D1
pExcelRange := pWorkSheet.GetRange_1('D1', 'D1');
pExcelRange.SetValue_1('End price');
   ENDIF;
```

Exercise III: Export to Excel

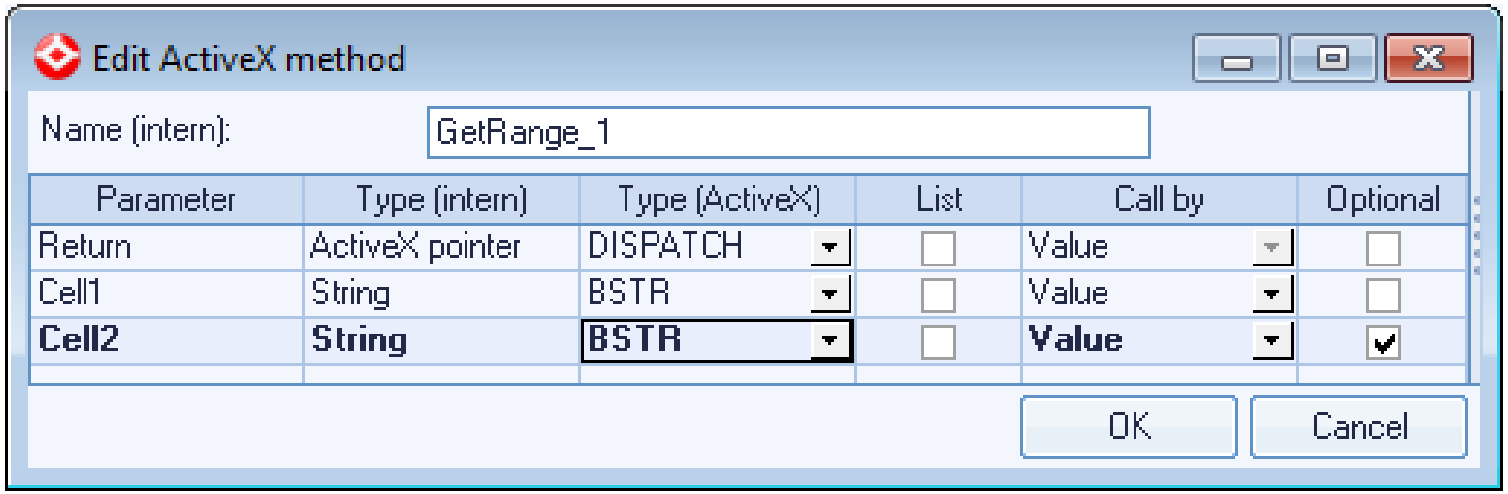
- **Source text of the method WriteExcel – Part 2**

```
    # Set values for ...
    # ...component column
    pExcelRange := pWorkSheet.GetRange_1(StrConcat('A', i + 1),
    StrConcat('A', i + 1));
    pExcelRange.SetValue_1(ObjName[i]);
    # ...price column
    pExcelRange := pWorkSheet.GetRange_1(StrConcat('B', i + 1),
    StrConcat('B', i + 1));
    # Type conversion Currency -> Numeric for displaying in pie chart
    pExcelRange.SetValue_2(Currency2Num(ObjPrice[i], GetCurrency()));
    # ...discount column
    pExcelRange := pWorkSheet.GetRange_1(StrConcat('C', i + 1),
    StrConcat('C', i + 1));
    pExcelRange.SetValue_2(_Car.Discount);
    # ...End price column
    pExcelRange := pWorkSheet.GetRange_1(StrConcat('D', i + 1),
    StrConcat('D', i + 1));
    # Calculate end price minus discount for result in end price
    column
    pExcelRange.SetFormulaR1C1_1('=RC[-2] * (1- (RC[-1]/100))');
ENDFOR;
RETURN;
```

Exercise III: Export to Excel

- **The syntax check does not know the method GetRange_1()**

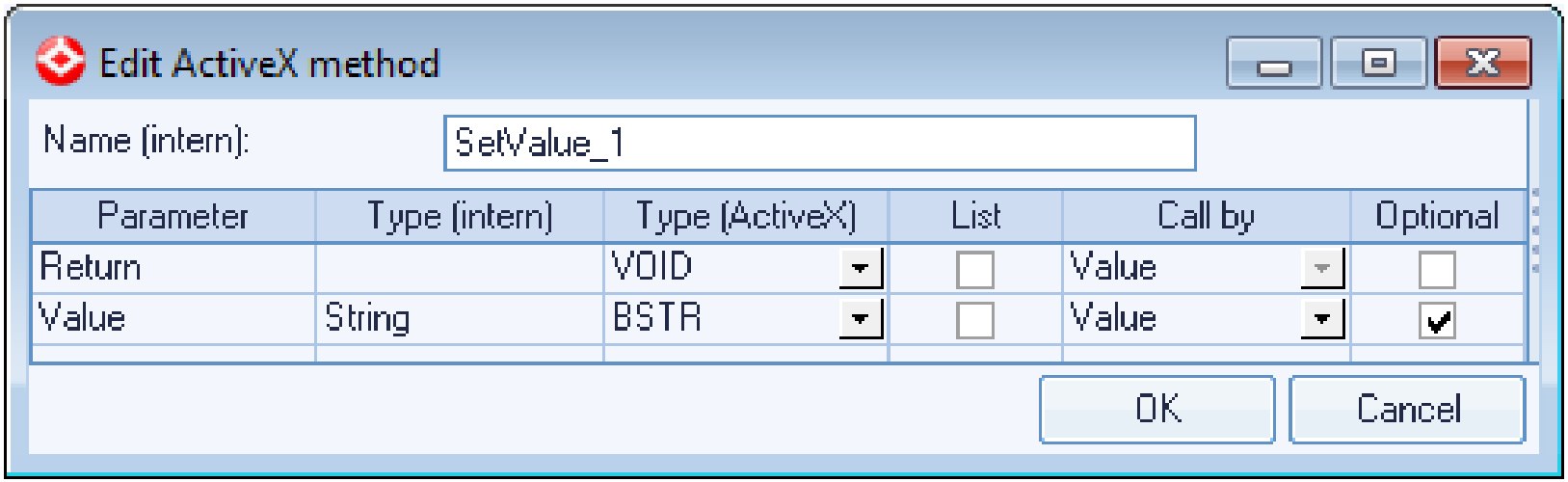
- **Thats why it is created in the AX class ExcelSheet**
  - Create a new wrapper for the method GetRange()
  - Change the data type of the parameters Cell1, Cell2 from VARIANT to BSTR



Exercise III: Export to Excel

- **The syntax check does not know the method SetValue_1()**

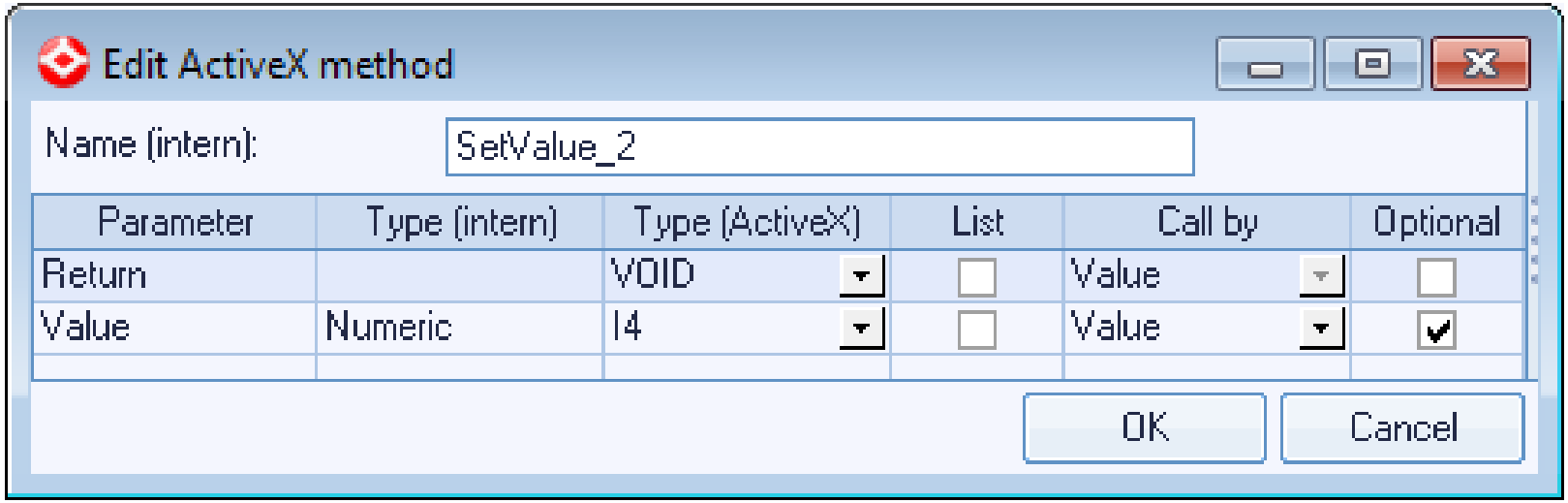
- **Thats why it is created in the AX class ExcelRange**
  - Create a new wrapper for the method SetValue()
  - Delete the Parameter RangeValueDataType
  - Change the data type of the parameter Value to BSTR



Exercise III: Export to Excel

- **The syntax check does not know the method SetValue_2()**

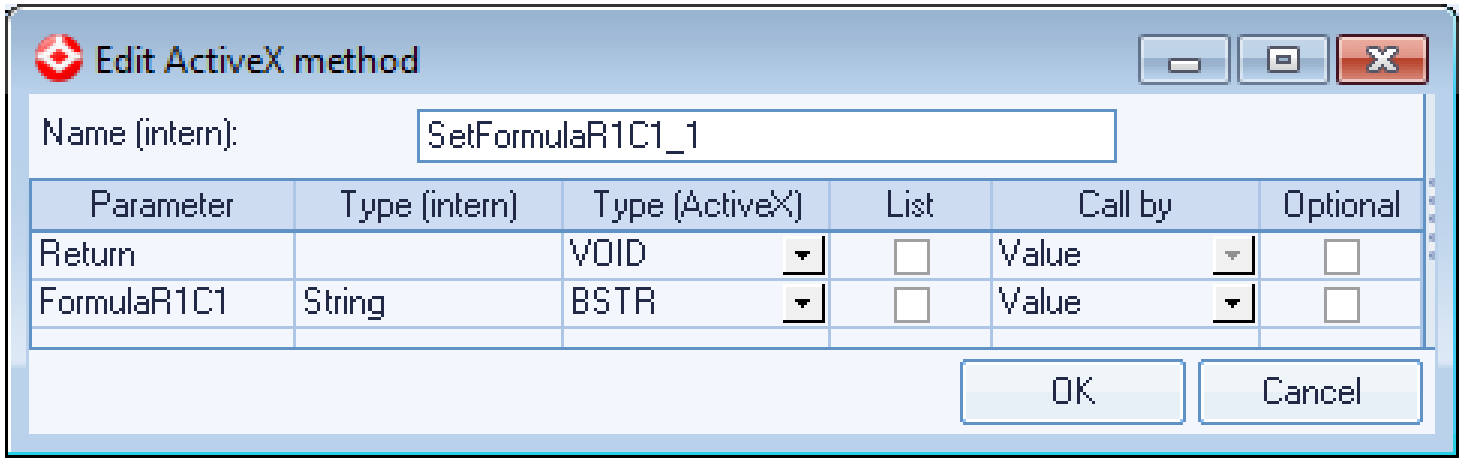
- **Thats why it is created in the AX class ExcelRange**
  - Create a new wrapper for the method SetValue()
  - Delete the Parameter RangeValueDataType
  - Change the data type of the parameter Value to I4



Exercise III: Export to Excel

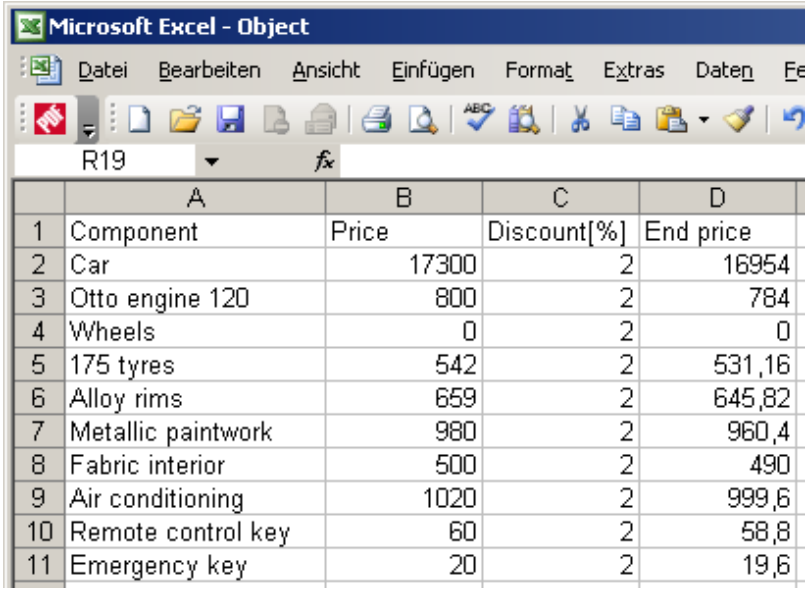
- **The syntax check does not know the method SetFormulaR1C1_1() either**

- **Create this method in the AX class ExcelRange**
  - Create a new wrapper for the method SetFormulaR1C1()
  - Change the data type of the parameter FormulaR1C1 from VARIANT to BSTR



Exercise III: Export to Excel

- **Intermediate test of the Excel export up to now**
- **Create a method ExcelExport() in the class start**
  - In this method the methods
    - CreateData()
    - OpenExcel()
    - WriteExcel()
  - are called



	A	B	C	D
1	Component	Price	Discount[%]	End price
2	Car	17300	2	16954
3	Otto engine 120	800	2	784
4	Wheels	0	2	0
5	175 tyres	542	2	531,16
6	Alloy rims	659	2	645,82
7	Metallic paintwork	980	2	960,4
8	Fabric interior	500	2	490
9	Air conditioning	1020	2	999,6
10	Remote control key	60	2	58,8
11	Emergency key	20	2	19,6

- **Extension of the menu for the export to Excel**
  - Extend the menu Administration by inserting a menu item „Export to Excel“ in the menu title „Export data“
  - Call the method ExcelExport() in the selection field

camos.

Exercise III: Export to Excel

- **Extension of the Excel export → Displaying the chart**
  - Create a pie chart
    - To do so, the cells to be included have to be selected
    - A new chart object of the type Pie chart has to be created
  - Create further ActiveX classes (Type library Microsoft Excel 11.0 Object Library or more recent)
  - Create ActiveX pointers in class start

Class name	ActiveX class	Type
ExcelCharts	Charts	Interface
ExcelChart	_Chart	OLE object

ActiveX pointer	Class
pCharts	ExcelCharts
pChart	ExcelChart

Exercise III: Export to Excel

- **Create a method CreateExcelChart()**

```
# Determine the cells that have to be used
pExcelRange := pWorkSheet.GetRange_1('A1', StrConcat('B',
MaxIndex(ObjName[]) + 1));
# Create pointer to the charts of the current folder of the application
pCharts := pActiveApplication.GetCharts();
# Add new chart
pCharts.Add_1();
# Initialize pointer to the new chart
pChart := pActiveApplication.GetActiveChart();
# Set chart type Pie chart 5 = xlpie
pChart.SetChartType_1(5);
# Assign source range, column reference
pChart.SetSourceData_1(pExcelRange, 2);
# The chart is set to a new sheet that
# gets the name Chart
pChart.Location_1(1, 'Diagramm');
# Determine chart label: Type = Specifications in per cent, legend
symbols
# on data point = none
pChart.ApplyDataLabels_1(5, 0);
RETURN;
```

camos.

Exercise III: Export to Excel

- **The syntax check does not know the method Add_1()**
- **That´s why this method is created in the class ExcelCharts**
  - Create a new wrapper for the method Add()
  - Delete all parameters except Return

- **The syntax check does not know the methods SetChartType_1(), SetSourceData_1(), Location_1() and ApplyDataLabels_1() either**
- **That´s why these methods are created in the class ExcelChart**

camos.

Exercise III: Export to Excel

- Create a new wrapper for the method SetChartType()
- New Parameter ChartType, data type = I4

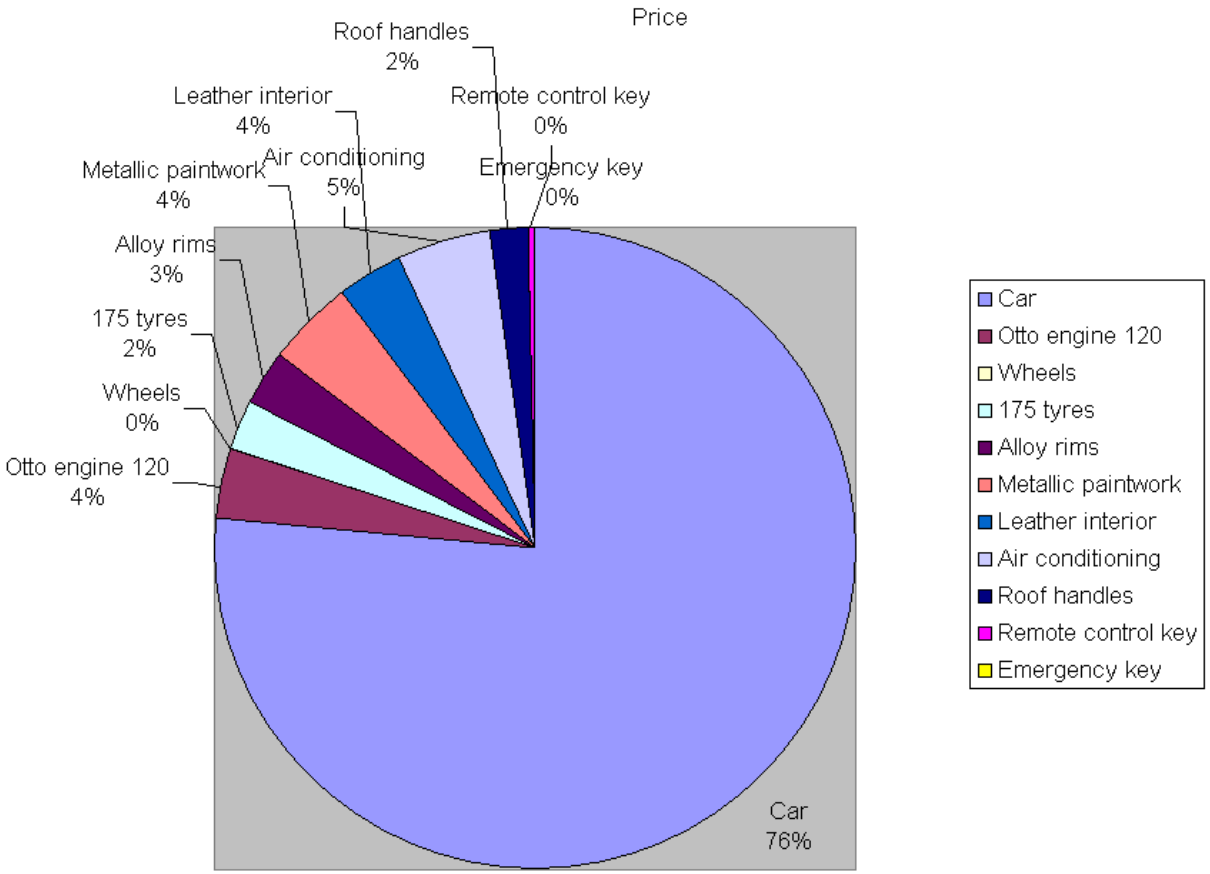
- Create a new wrapper for the method SetSourceData()
- Change the data type of the parameter PlotBy to I4

- Create a new wrapper for the method Location()
- Change the data type of the parameter Where to I4
- Change the data type of the parameter Name to BSTR

- Create a new wrapper for the method ApplyDataLabels()
- Delete all parameters except Type and LegendKey
- Change the data types of both parameters to I4

Exercise III: Export to Excel

- **Call the method CreateExcelChart() in the method ExcelExport()**



camos.

ActiveX controls

- **ActiveX controls are special ActiveX classes with graphical surface → Usage as form element**
- **Usage of a control by means of the Adobe Acrobat Reader**

- **Exercise:**
  - Create a new knowledge base
  - Create a start class
  - In this start class you create a new form
  - Place an editline (fileName), an ActiveX control and a button (Open) on the form
  - Create a String feature fileName
  - Create a new() method in the class start, open the form here
  - ActiveX pointer to the ActiveX class AcrobatReader (see next slide)

camos.

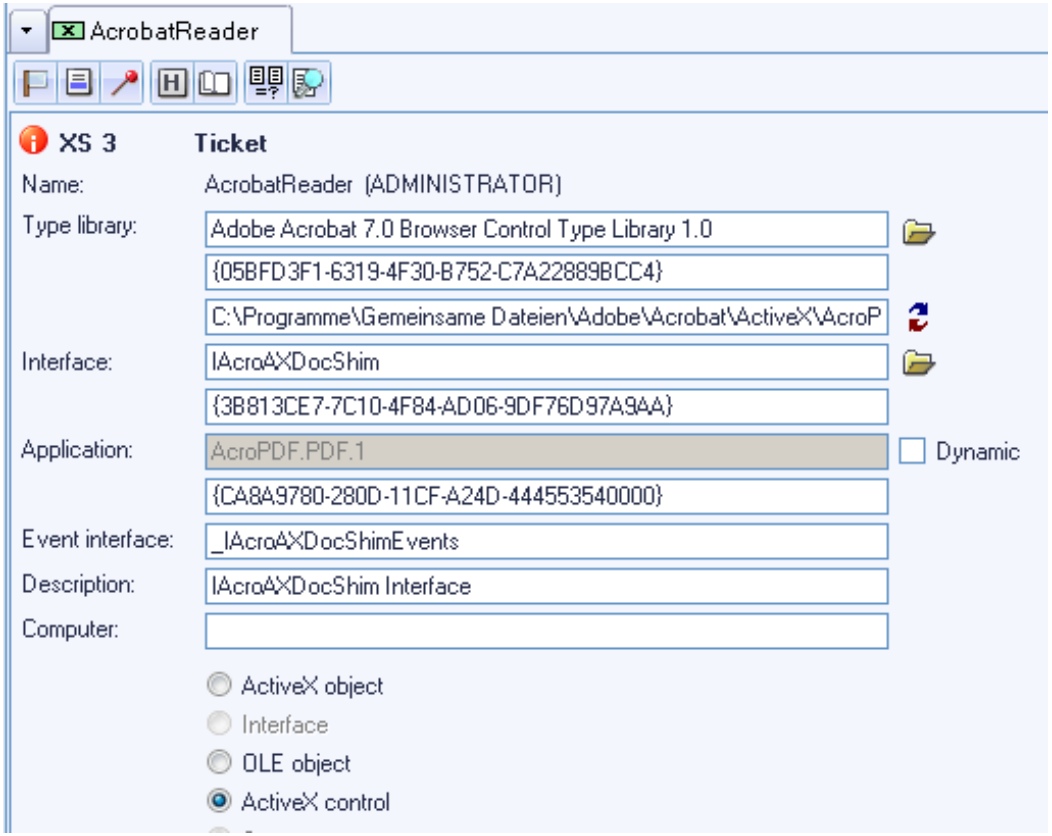
ActiveX controls

- **ActiveX controls are special ActiveX classes with graphical surface → Usage as form element**
- **Usage of a control by means of the Adobe Acrobat Reader**

- **Exercise:**
  - Create a new knowledge base
  - Create a start class
  - In this start class you create a new form
  - Place an editline (fileName), an ActiveX control and a button (Open) on the form
  - Create a String feature fileName
  - Create a new() method in the class start, open the form here
  - ActiveX pointer to the ActiveX class AcrobatReader (see next slide)

ActiveX controls

- **Exercise:**
  - Create the ActiveX class AcrobatReader from the type library Adobe Acrobat 7.0 Browser Control type library 1.0



camos.

ActiveX controls

- **Form:**
  - The editline uses the feature filename as cause variable
  - The ActiveX control uses the ActiveX pointer to Acrobat Reader as cause variable
  - Selection trigger of button „Open“
    - pAcrobatPointer.LoadFile(AcrobatFile);

-> The defined PDF file is displayed