Rechnerpraktikum zu Kapitel 4

Softwareschnittstellen

Einleitung, Component Object Model (COM)

Zugriff auf Microsoft Excel

Zugriff auf MATLAB

Zugriff auf CATIA

*Zugriff auf Solid Edge (***NEU***)*

P4.2. Einleitung

Viele Programmsysteme haben **Softwareschnittstellen**, über die es möglich ist, Daten auszulesen oder zu schreiben. Oft ist es möglich, über eine Softwareschnittstelle Funktionen des Programmsystems aufzurufen oder zusätzliche Funktionalitäten zum Programmsystem hinzuzufügen.

Es existieren herstellerübergreifende Standards, um Softwareschnittstellen zu realisieren. Zum Beispiel **RPC** (Remote Procedure Call), **CORBA** (Common Object Request Broker Architecture) und das netzwerkbasierte **SOAP** (Simple Object Access Protocol).

Unter Microsoft Windows sind **COM-Schnittstellen** (Component Object Model) sehr verbreitet. Als Beispiele seien Microsoft Office, MATLAB und Konstruktionsprogramme wie CATIA genannt, die über solche COM-Schnittstellen verfügen.

Die C++-Programme aus diesem Kapitel basieren auf COM. Sie können nur unter Microsoft Windows ausgeführt werden!

What is COM?

Microsoft COM (Component Object Model) technology in the Microsoft Windows-family of Operating Systems enables software components to communicate. COM is used by developers to create re-usable software components, link components together to build applications, and take advantage of Windows services. COM objects can be created with a variety of programming languages. Object-oriented languages, such as C++, provide programming mechanisms that simplify the implementation of COM objects. (…)

COM is used in applications such as the Microsoft Office Family of products. For example **COM OLE** technology allows Word documents to dynamically link to data in Excel spreadsheets and **COM Automation** allows users to build scripts in their applications to perform repetitive tasks or control one application from another.

(https://www.microsoft.com/com/default.mspx)

COM Automation

Automation enables software packages to expose their unique features to scripting tools and other applications. Using Automation, you can:

- Create applications and programming tools that expose objects.
- **Create and manipulate objects exposed in one application from another application.**
- Create tools that access and manipulate objects. These tools can include embedded macro languages, external programming tools, object browsers, and compilers.

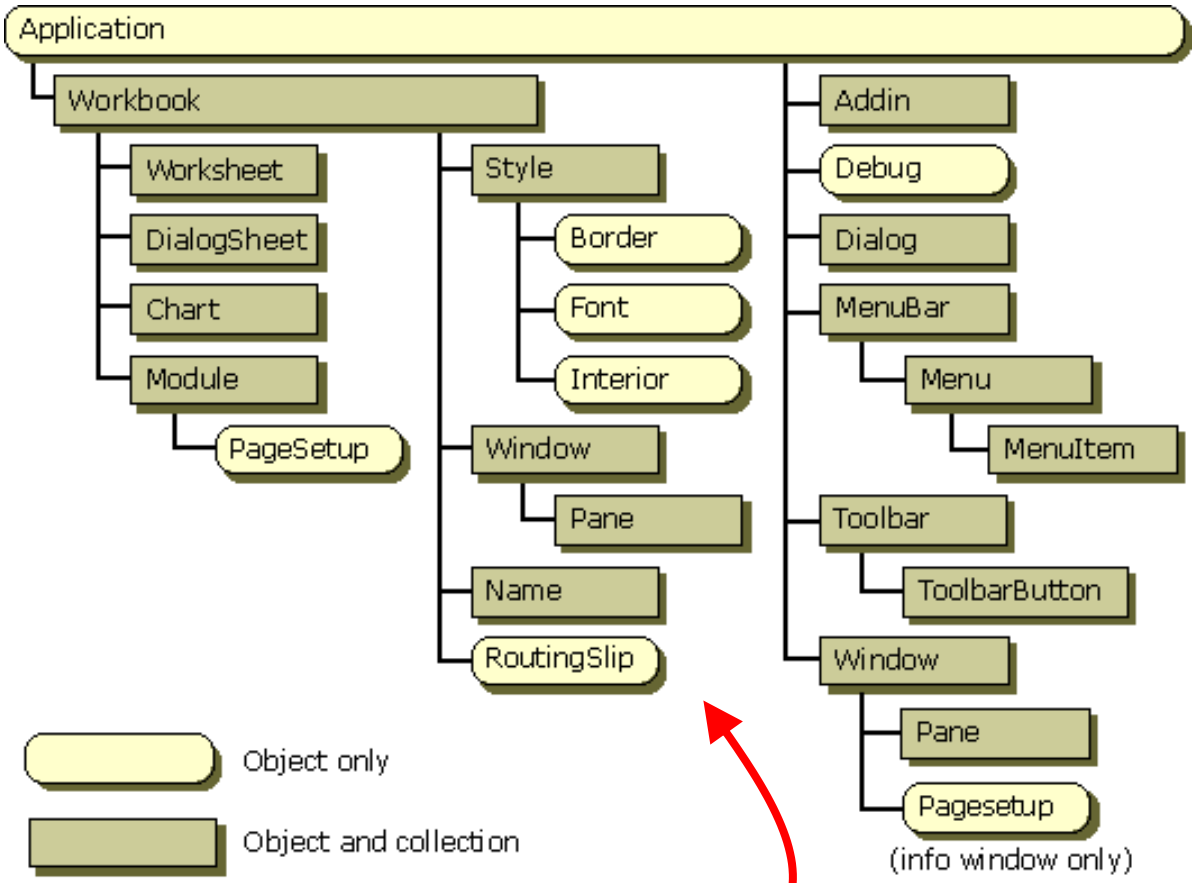
The objects an application or programming tool exposes are called **ActiveX objects**. Applications and programming tools that access those objects are called **ActiveX clients**.

(https://msdn.microsoft.com/en-us/library/windows/desktop/ms221251(v=vs.85).aspx)

ActiveX Objects

An ActiveX object is an instance of a class that exposes **properties, methods, and events** to ActiveX clients. (…)

For example, Microsoft Excel exposes many objects that you can use to create new applications and programming tools.



Within Microsoft Excel, objects are organized hierarchically.

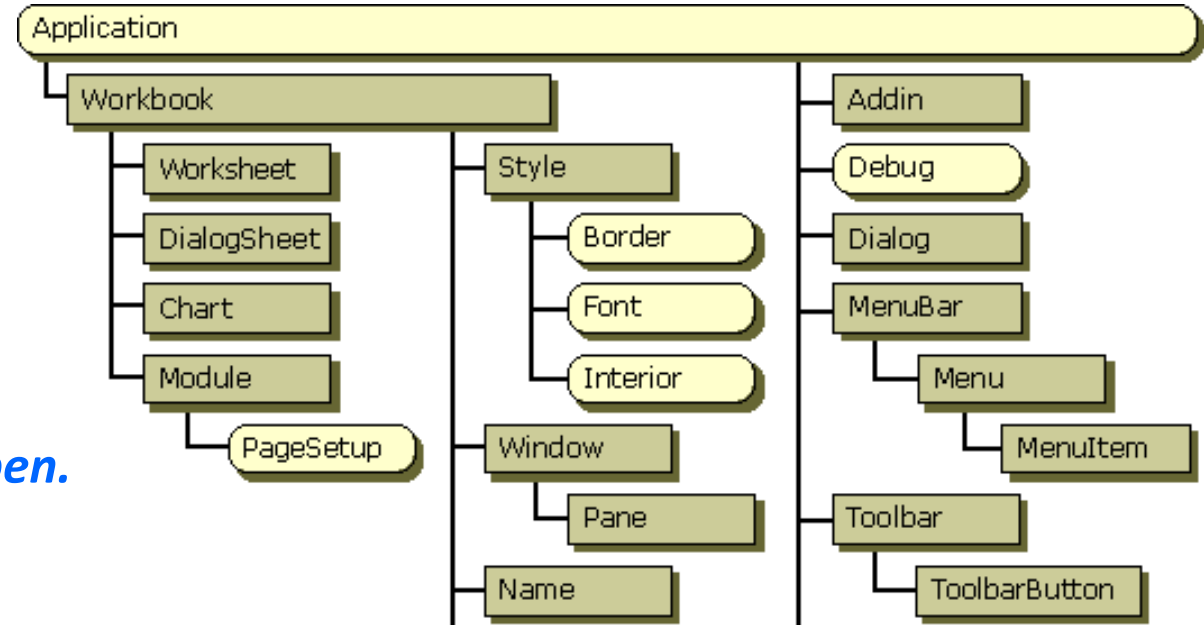
(https://msdn.microsoft.com/en-us/library/windows/desktop/ms221401(v=vs.85).aspx)

COM-Progammierung mit „ActiveQt“ *(http://doc.qt.io/qt-4.8/activeqt.html)*

- Access and use ActiveX controls and COM objects provided by any ActiveX server in their Qt applications.
- Make their Qt applications available as COM servers, with (…) Qt objects and widgets as COM objects and ActiveX controls.

Um eine externe Applikation (z. B. Excel) per COM-Schnittstelle zu steuern, müssen folgende Teilprobleme gelöst werden:

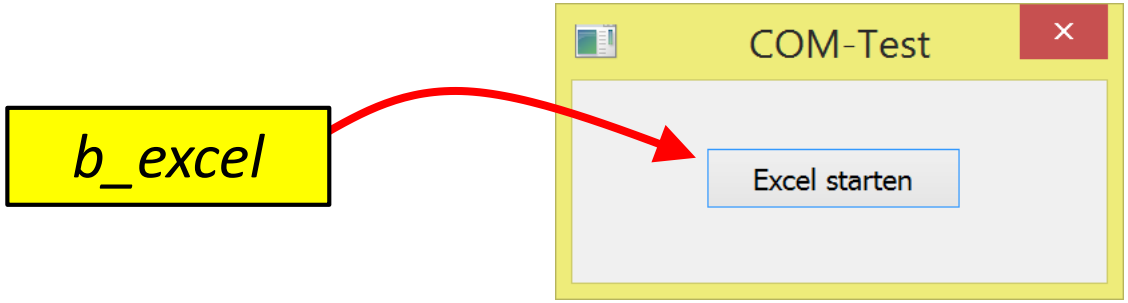
1. ***Externe Applikation starten, COM-Schnittstelle öffnen.***
2. ***Auf die Objekte der ext. Applikation zugreifen.***
3. ***Methoden aufrufen, Attribute lesen und schreiben.***



P4.7. Einleitung

(1a) Externe Applikation starten, COM-Schnittstelle öffnen

- Erstellen Sie eine neue **Dialog-Applikation** („P4_Einleitung“) mit einer einzelnen Schaltfläche („Excel starten“).



- Fügen Sie die folgenden Zeilen zur Projektdatei hinzu:

```
QT += axcontainer
```

- Fügen Sie folgende Include-Anweisung zur Datei „Dialog.h“ hinzu:

```
#include <QAxObject>
```

- Fügen Sie das folgende private Attribut zur Klasse „Dialog“ hinzu (ebenfalls in der Datei „Dialog.h“):

```
QAxObject excel;
```

(1b) Externe Applikation starten, COM-Schnittstelle öffnen

- Implementieren Sie den Slot zur Schaltfläche „Excel aufrufen“:

```
void Dialog::on_b_excel_clicked()
{
    excel.setControl("Excel.Application");
    excel.setProperty("Visible", true);
}
```

- Testen Sie Ihre Applikation:
  - ***Was passiert wenn man die Schaltfläche betätigt?***
  - ***Und wenn man die Schaltfläche mehrmals betätigt?***
  - ***Was passiert beim Beenden der Applikation?***
  - ***Öffnen Sie den Windows Task Manager und beobachten Sie dort den Microsoft-Excel-Prozess.***
  - ***Wie kommt es, dass Excel beim Schließen Ihrer Applikation automatisch beendet wird?***

(2) Auf die Objekte der externen Applikation zugreifen

```
void Dialog::on_b_excel_clicked()
{
  excel.setControl("Excel.Application");
  excel.setProperty("Visible", true);
  QAxObject *active = excel.querySubObject("ActiveSheet");
  if(!active)
  {
    QAxObject *workbooks = excel.querySubObject("Workbooks");
    workbooks->dynamicCall("Add(void)");
    active = excel.querySubObject("ActiveSheet");
  }
  QAxObject *cells = active->querySubObject("Cells(int,int)", 1, 1);
  cells->setProperty("Value", "Überschrift");
  cells = active->querySubObject("Cells(int,int)", 2, 1);
  cells->setProperty("Value", 123.456);
  // QVariant value = cells->property("Value");
  // if(!value.isNull())
  // { QMessageBox box; box.setText(value.toString()); box.exec(); }
}
```

(3) Methoden aufrufen, Attribute lesen und schreiben

- Die Methoden der COM-Objekte werden mittels **dynamicCall** aufgerufen. Aber warum steht beim ersten Aufruf ein Punkt und beim zweiten Aufruf ein Pfeil…?!

```
excel.dynamicCall("Quit(void)");
workbooks->dynamicCall("Add(void)");
```

- Attribute lesen und schreiben mittels **property** bzw. **setProperty**; dabei sind (je nach Attribut) unterschiedliche Datentypen möglich:

```
cells->setProperty("Value", "Überschrift");
cells->setProperty("Value", 123.456);

QVariant value = cells->property("Value");
if(!value.isNull()) {
  QMessageBox box;
  box.setText(value.toString());  \\ .toInt(), .toFloat(), …
  box.exec();
}
```

Rechnerpraktikum zu Kapitel 4

Softwareschnittstellen

Einleitung, Component Object Model (COM)

Zugriff auf Microsoft Excel

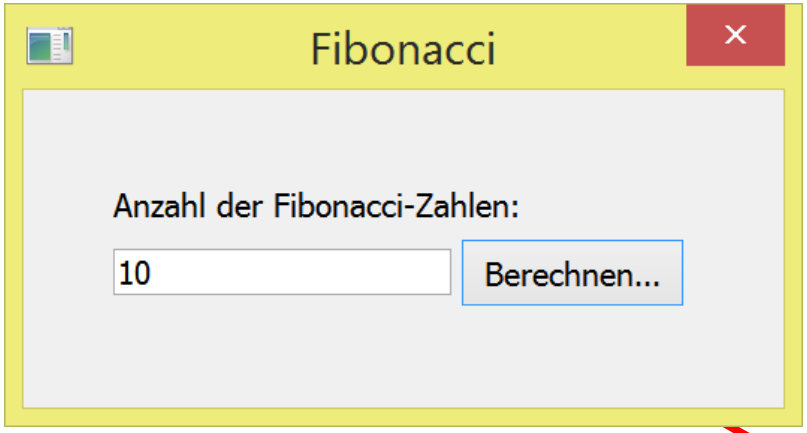
Zugriff auf MATLAB

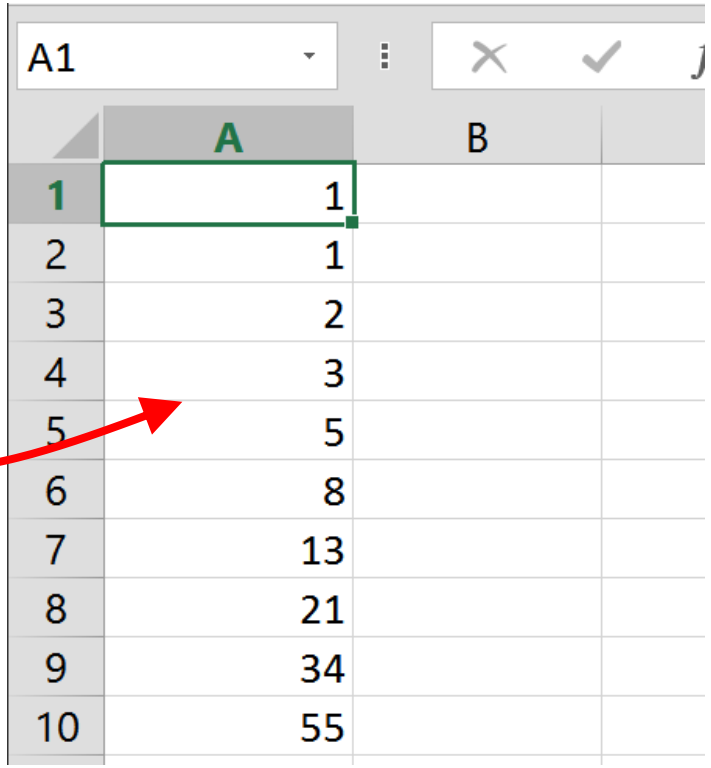
Zugriff auf CATIA

*Zugriff auf Solid Edge (***NEU***)*

1. Aufgabe, Fibonacci-Zahlen:

Erstellen Sie eine Dialog-Applikation zur **Berechnung der ersten „n“ Fibonacci-Zahlen**. Die gewünschte Anzahl wird vom Anwender im Dialogfenster eingegeben. Die berechneten Fibonacci-Zahlen werden in ein leeres Excel-Arbeitsblatt geschrieben.



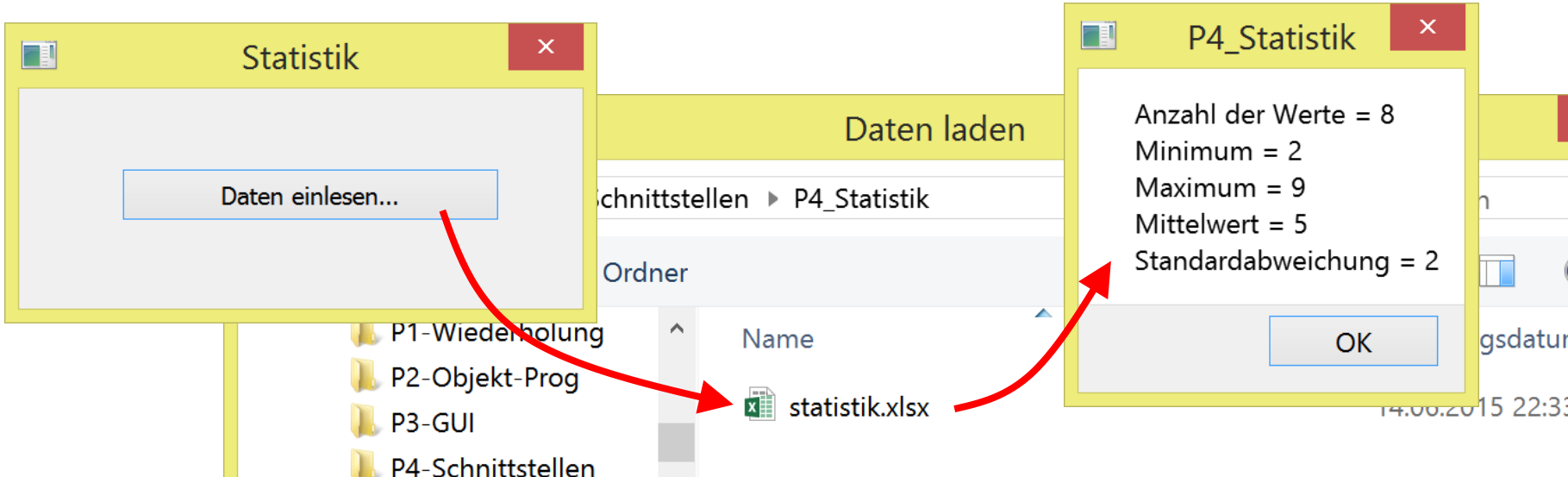


Ab welcher Anzahl werden die berechneten Fibonacci-Zahlen ungenau?

2. Aufgabe, Statistik:

Eine Dialog-Applikation soll die folgenden Aufgaben bearbeiten:

- Eine bestehende **Excel-Datei wird vom Anwender ausgewählt**, dann werden alle Werte aus der Spalte A eingelesen (aus den Zellen A1, A2, A3, A4 usw.)
- Anschließend ermittelt Ihre Applikation das Minimum, Maximum, den Mittelwert sowie die Standardabweichung dieser Werte und **gibt die Ergebnisse in einem Meldungsfenster aus**.



P4.14. Zugriff auf Microsoft Excel


```
void Dialog::on_b_einlesen_clicked()
{
  // Auswahlfenster für Excel-Dateien anzeigen...
  QString fileName = QFileDialog::getOpenFileName(0, "Laden", "", "*.xls *.xlsx");
  if(fileName.isEmpty()) return;

  // Excel starten, ausgewählte Datei öffnen...
  excel.setControl("Excel.Application");
  excel.setProperty("Visible", false);
  QAxObject *workbooks = excel.querySubObject("Workbooks");
  workbooks->dynamicCall("Open(QString)", fileName);

  // Zugriff auf Tabellenblatt...
  QAxObject *active = excel.querySubObject("ActiveSheet");
  if(!active) return;

  // Werte einlesen...
  vector<double> daten;
  int zeile = 1;
  QAxObject *cell = active->querySubObject("Cells(int,int)", zeile, 1);
  QVariant wert = cell->property("Value");

  while(!wert.isNull())
  {
    daten.push_back(wert.toDouble()); zeile++;
    cell = active->querySubObject("Cells(int,int)", zeile, 1);
    wert = cell->property("Value");
  }
  excel.dynamicCall("Quit(void)");
```

#include <QFileDialog>

#include <vector>
using namespace std;

P4.15. Zugriff auf Microsoft Excel

- **Vorherige Folie:** Öffnen und Einlesen der Excel-Datei
- **Auf dieser Folie:** Ausgabe der Ergebnisse im Meldungsfenster

#include <sstream>

```
stringstream tmp;
tmp << "Anzahl der Werte = "    << n    << endl;
tmp << "Minimum = "             << min  << endl;
tmp << "Maximum = "             << max  << endl;
tmp << "Mittelwert = "          << mean << endl;
tmp << "Standardabweichung = " << stdaw;
QMessageBox box;
box.setText(tmp.str().c_str());
box.exec();
```

#include <QMessageBox>

- **Ausführliche Dokumentation der Excel-Programmierschnittstelle:**

  https://msdn.microsoft.com/en-us/library/office/ff194068.aspx

Rechnerpraktikum zu Kapitel 4

Softwareschnittstellen

Einleitung, Component Object Model (COM)

Zugriff auf Microsoft Excel

Zugriff auf MATLAB

Zugriff auf CATIA

*Zugriff auf Solid Edge (***NEU***)*

Auch die Windows-Version von MATLAB hat eine COM-Schnittstelle:

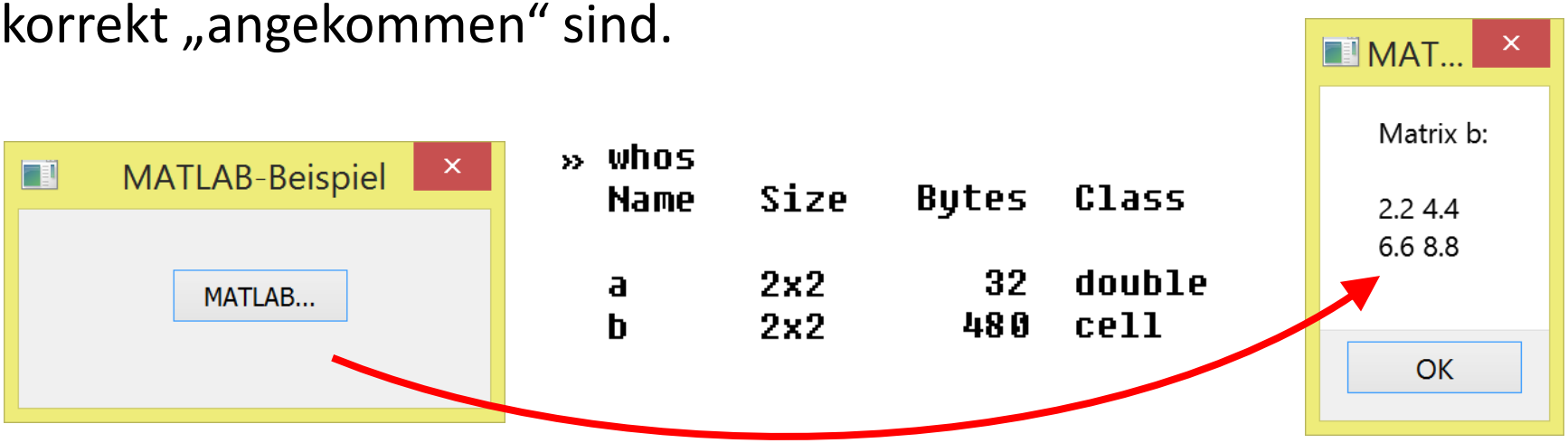
(http://de.mathworks.com/help/matlab/call-matlab-com-automation-server.html)

Functions

Execute	Execute MATLAB command in Automation server
Feval	Evaluate MATLAB function in Automation server
GetCharArray	Character array from Automation server
PutCharArray	Store character array in Automation server
GetFullMatrix	Matrix from Automation server workspace
PutFullMatrix	Matrix in Automation server workspace
GetVariable	Data from variable in Automation server workspace
GetWorkspaceData	Data from Automation server workspace
PutWorkspaceData	Data in Automation server workspace
MaximizeCommandWindow	Open Automation server window
MinimizeCommandWindow	Minimize size of Automation server window
Quit	Terminate MATLAB Automation server
enableservice	Enable, disable, or report status of MATLAB Automation server

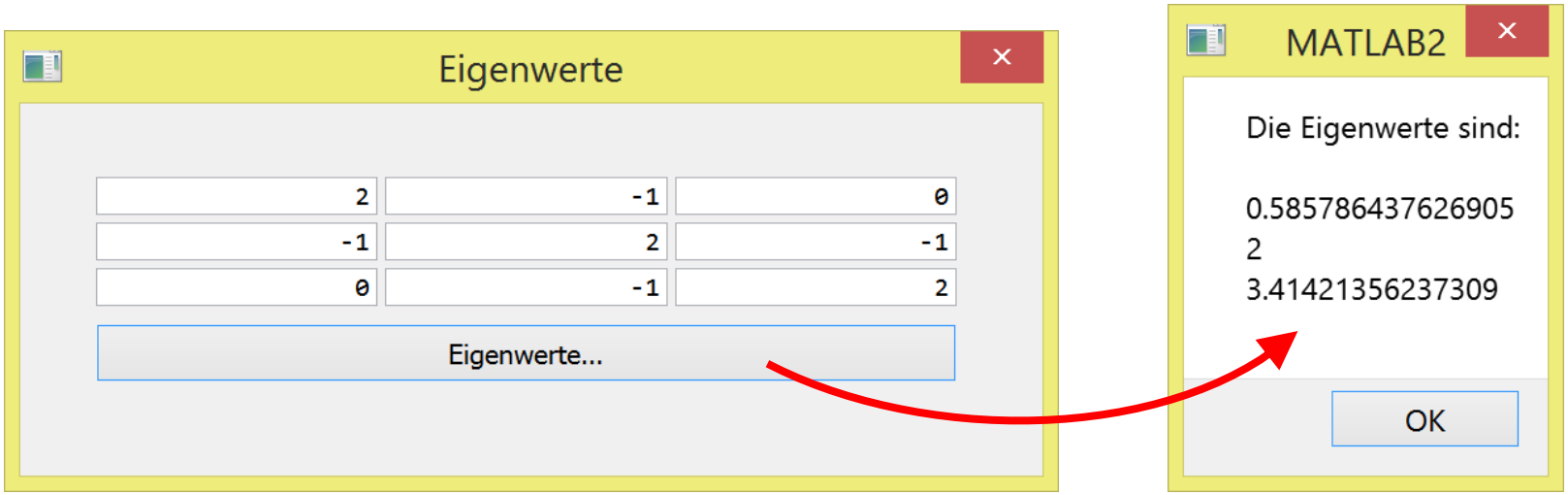
1. Aufgabe: Öffnen Sie die vorbereitete Applikation „MATLAB1“

- Mit den Methoden **PutWorkspaceData** und **GetVariable** können auf einfache Weise Daten zwischen C++ und MATLAB übertragen werden. **Matrizen werden dabei als Cell Arrays übertragen**. Mit der Methode **Execute** werden MATLAB-Befehle ausgeführt.
- Versuchen Sie, den C++-Quelltext zu verstehen und führen Sie das Programm auf Ihrem Rechner aus. Prüfen Sie auch im MATLAB-Kommandofenster, dass dort die Variablen aus dem C++-Programm korrekt „angekommen“ sind.



2. Aufgabe: Öffnen Sie die vorbereitete Applikation „MATLAB2“

- Im Dialogfenster kann eine 3x3-Matrix eingegeben werden.
- Wenn der Anwender die Schaltfläche „Eigenwerte“ betätigt, sollen **die Eigenwerte der eingegebenen Matrix berechnet** und in einem Meldungsfenster ausgegeben werden.
- Die eingegebene Matrix wird nach MATLAB übertragen und dort die Eigenwertberechnung durchgeführt („**doc eig**“ eingeben…). Schließlich werden die Eigenwerte zurück nach C++ übertragen.



Rechnerpraktikum zu Kapitel 4

Softwareschnittstellen

Einleitung, Component Object Model (COM)

Zugriff auf Microsoft Excel

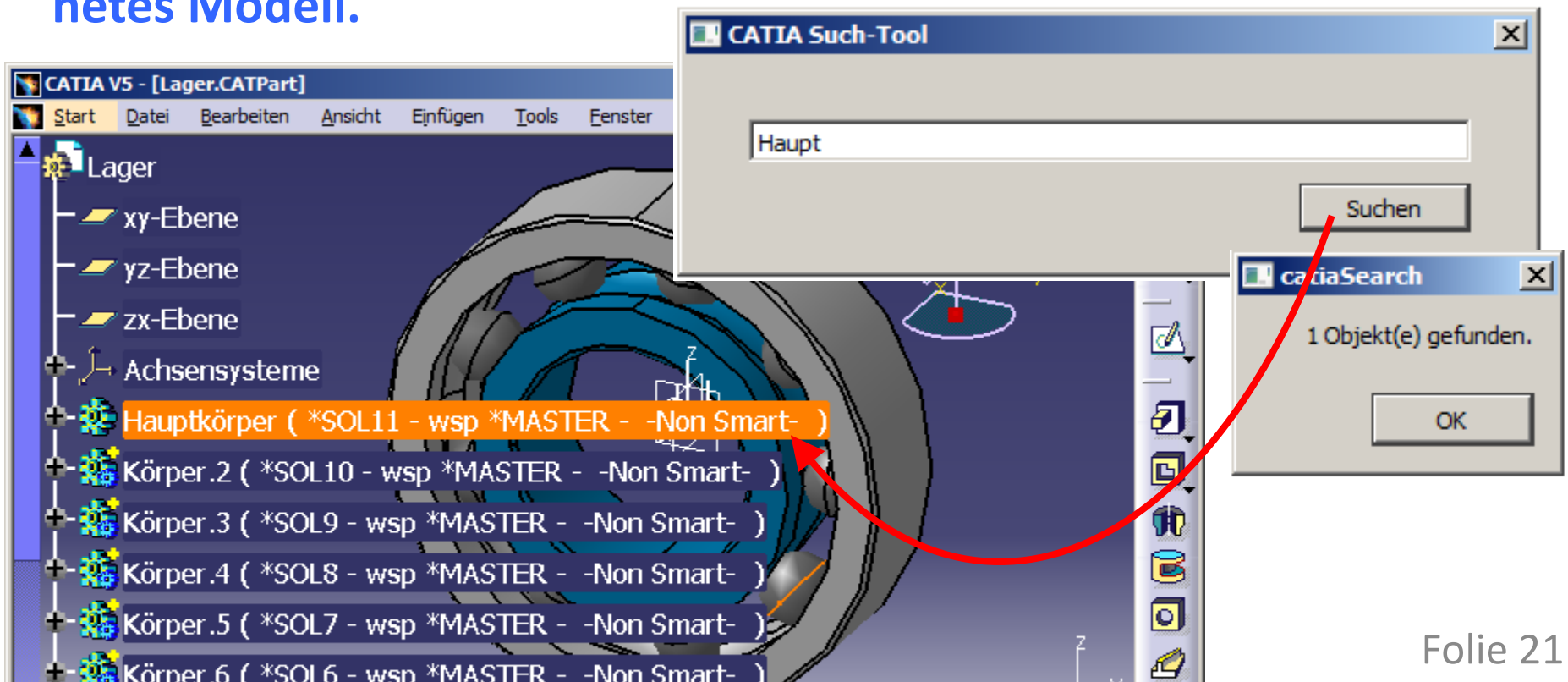
Zugriff auf MATLAB

Zugriff auf CATIA

*Zugriff auf Solid Edge (***NEU***)*

1. Aufgabe: Öffnen Sie die vorbereitete Applikation „catiaSearch“

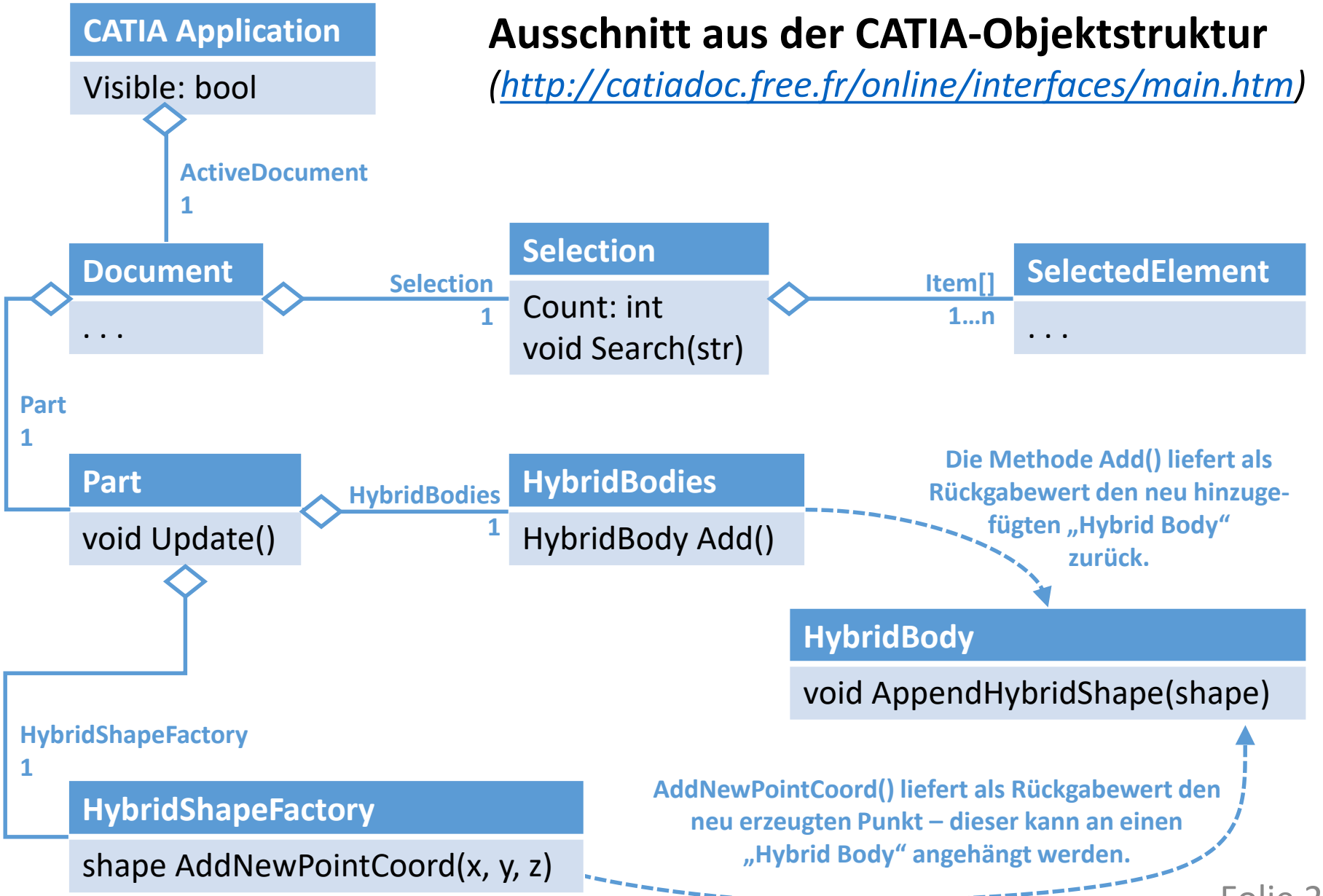
- Diese Applikation sucht und selektiert alle Elemente, deren Namen mit einer bestimmte Zeichenkette beginnen.
- Versuchen Sie, den C++-Quelltext zu verstehen und führen Sie das Programm aus. **Starten Sie zuvor CATIA und laden Sie ein geeignetes Modell.**



P4.22. Zugriff auf CATIA

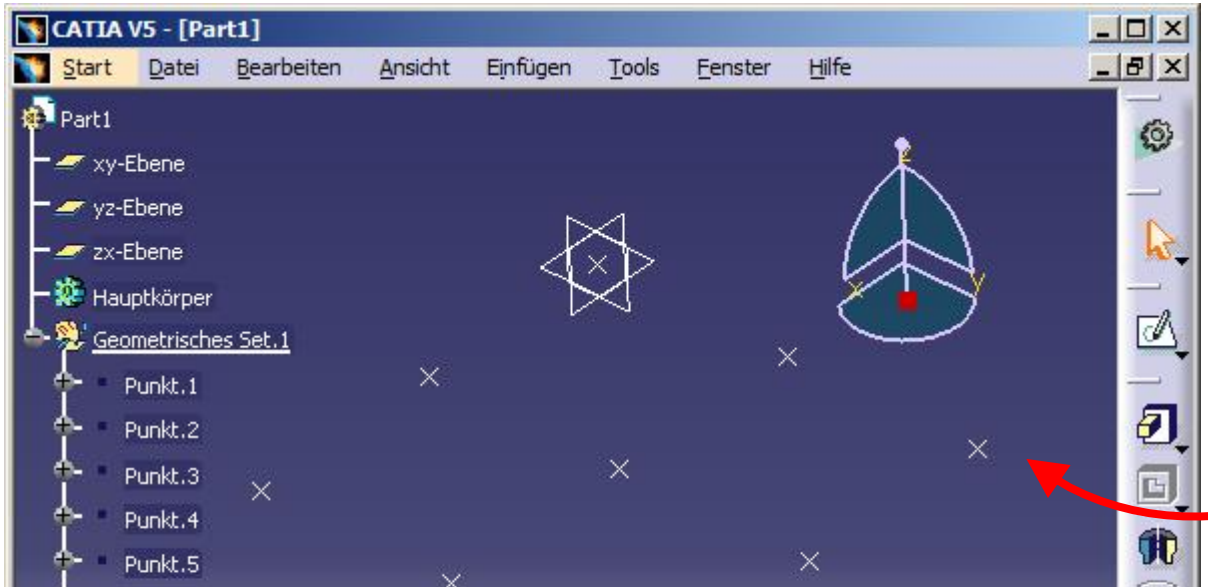
Ausschnitt aus der CATIA-Objektstruktur

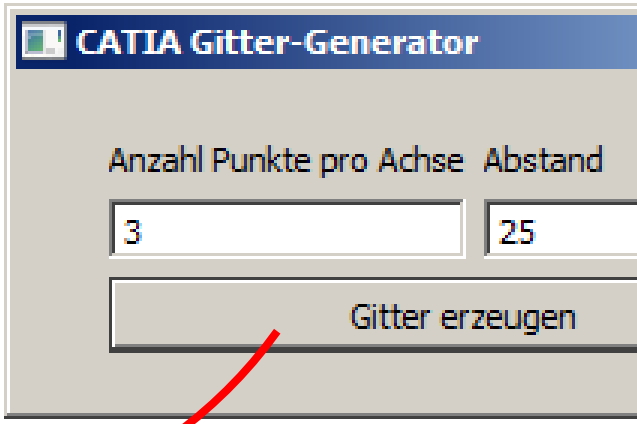
(http://catiadoc.free.fr/online/interfaces/main.htm)



2. Aufgabe: Öffnen Sie die vorbereitete Applikation „catiaGridGen“

- Diese Applikation generiert automatisch ein Raster aus n×n Punkten in der x-y-Ebene (mit z-Koordinate gleich null). Der Anwender gibt dazu die Anzahl n und den gewünschten Abstand der Punkte ein.
- Die Applikation ist schon teilweise programmiert: Es wird gezeigt, wie man einen einzelnen Punkt in CATIA hinzufügt. Ergänzen Sie das Programm, sodass das gewünschte Raster generiert wird.
  Starten Sie zuvor CATIA und öffnen Sie ein neues/leeres Part.





Weitere Informationen und Beispielprogramme zum Zugriff auf die COM-Schnittstelle von CATIA finden Sie hier:

- http://www.scripting4v5.com/
  In Visual Basic, aber als C++-Programmierer leicht zu verstehen…
- http://www.tech-ecke.de/index_quereinstieg.htm?/catscript/selection.htm
  Ebenfalls Beispiele in Visual Basic bzw. CATScript…

In Dateiform:

- Hilfedatei „V5Automation.chm“ im CATIA-Installationsverzeichnis

Oder in Buchform:

- J. Hansen: CATIA V5 automatisieren, Hanser-Verlag, 2009
  Beispiele in Visual Basic und in C#

Rechnerpraktikum zu Kapitel 4

Softwareschnittstellen

Einleitung, Component Object Model (COM)

Zugriff auf Microsoft Excel

Zugriff auf MATLAB

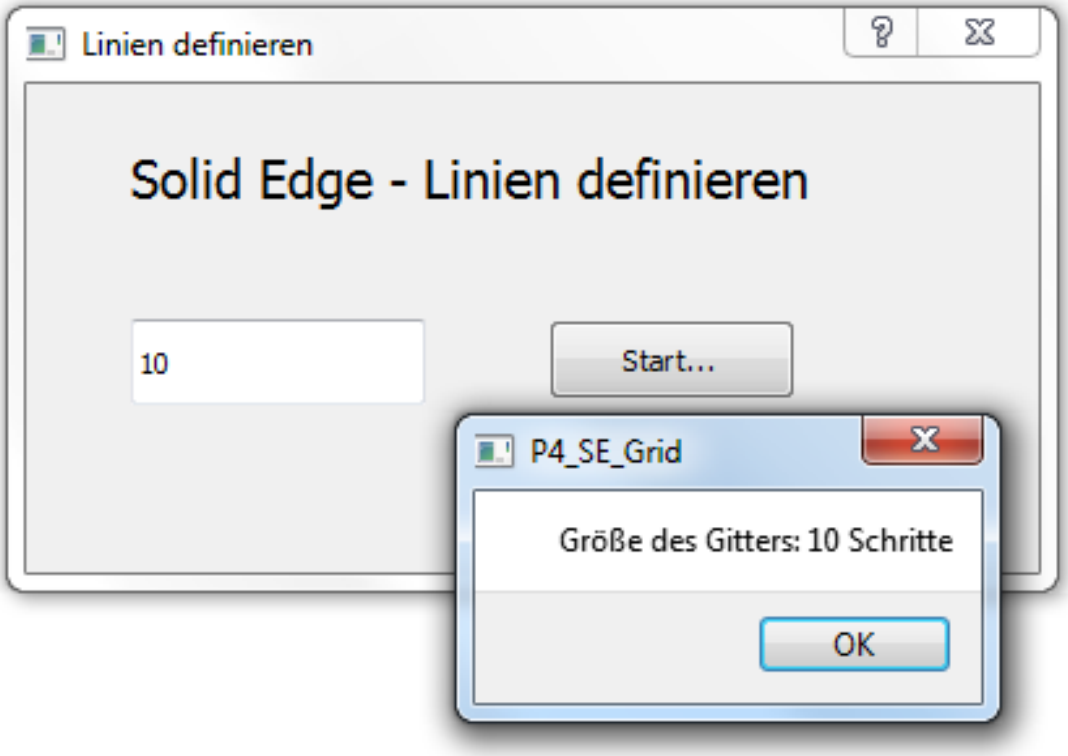
Zugriff auf CATIA

*Zugriff auf Solid Edge (***NEU***)*

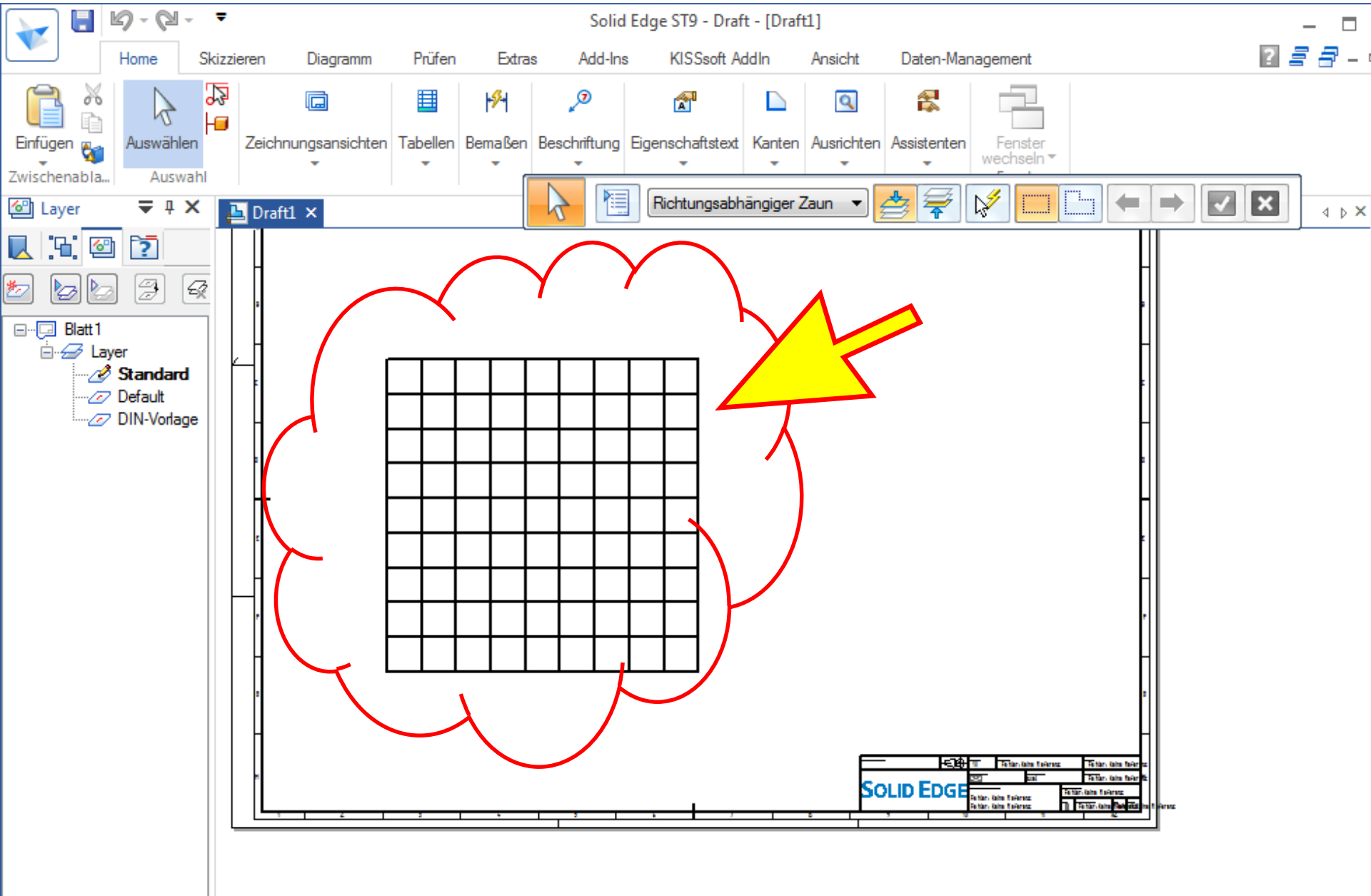
P4.26. Zugriff auf Solid Edge (***NEU***)

1. Aufgabe: Öffnen Sie das vorbereitete Projekt „P4_SE_Grid“

- Diese Applikation öffnet eine COM-Verbindung zu Solid Edge. Ziel ist es, ein Gitter aus einer vom Anwender vorgegebenen Linien-Anzahl automatisiert zu erstellen.
- Versuchen Sie, den C++-Quelltext zu verstehen.
- Vervollständigen Sie die Methode Dialog:: on_b_start_clicked() in der Datei dialog.cpp.

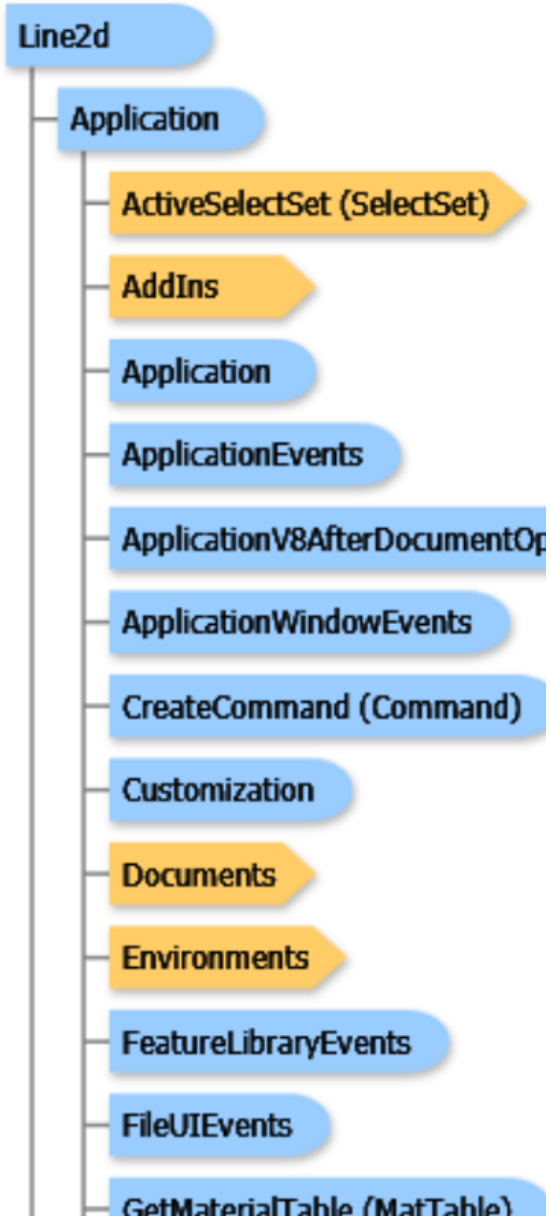


P4.27. Zugriff auf Solid Edge (***NEU***)



P4.28. Zugriff auf Solid Edge (***NEU***)

Object Model



Die COM-Schnittstelle von Solid Edge ist auf den Webseiten der Firma Siemens dokumentiert (mit Beispielen!):

- http://dl2.plm.automation.siemens.com/solidedge/api/sesdk_web/TypeLibraries.html

Dokumentation zur Klasse Line2d:

- http://dl2.plm.automation.siemens.com/solidedge/api/sesdk_web/SolidEdgeFrameworkSupport~Line2d.html

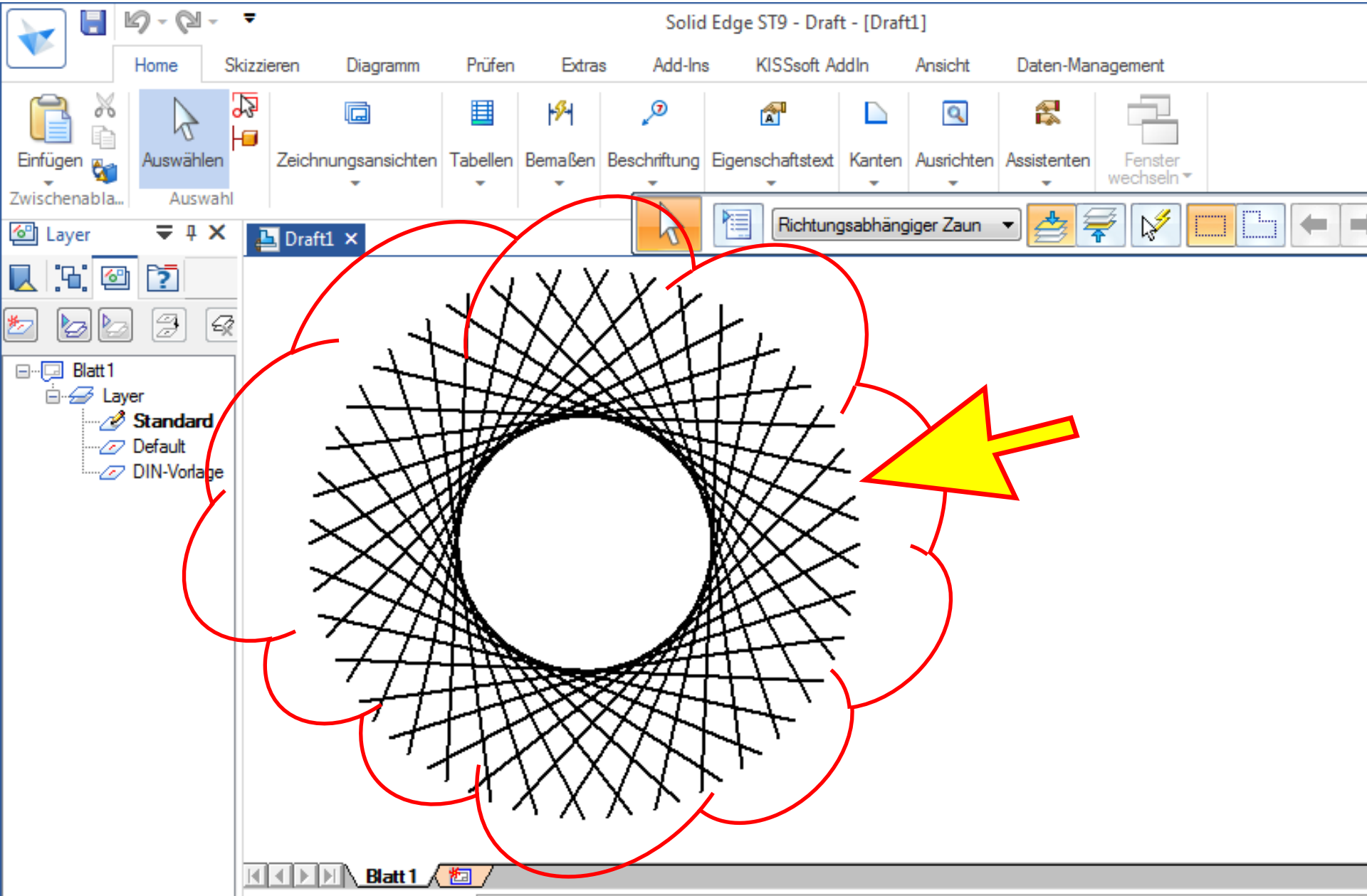
Programmer's Guide:

- https://www.plm.automation.siemens.com/zh_cn/Images/Solid_Edge_API_tcm78-125829.pdf

2. Aufgabe: Öffnen Sie das vorbereitete Projekt „P4_SE_Datei“

- Die Applikation soll eine Textdatei mit Linien-Koordinaten einlesen, die Koordinaten in einem std::vector speichern und schließlich die eingelesenen Linien in Solid Edge zeichnen.
- Im Projektverzeichnis liegt die vorbereitete Datei „data.txt“: In jeder Zeile sind die x1-, y1-, x2- und y2-Koordinaten einer einzelnen Linie abgelegt.
- Ergänzen Sie den C++-Quelltext in der Datei „dialog.cpp“.

P4.30. Zugriff auf Solid Edge (***NEU***)



Tipp: Einlesen der Textdatei mit Linien-Koordinaten

```
// Die zu öffnende Datei abfragen...
auto filename = QFileDialog::getOpenFileName(this, "", "", "");
if(filename.length() == 0)
        return;

// In jeder Zeile stehen vier Koordinaten: x1, y1, x2, y2
// Diese werden eingelesen und im Vektor "data" gespeichert.
Line my_line;
data.clear();
ifstream strm(filename.toStdString());

while(strm >> my_line.x1 >> my_line.y1 >> my_line.x2 >> my_line.y2)
{
    data.push_back(my_line);
}
```