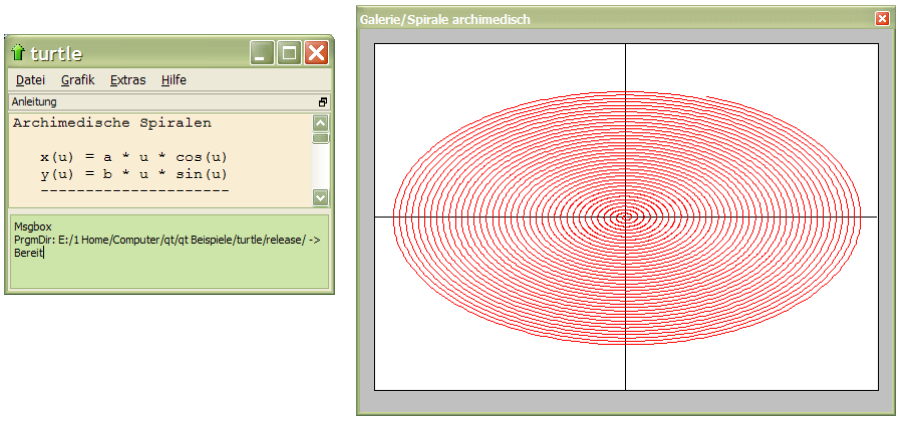
C++ Programmdokumentation turtle

Eine Beschreibung der Leistungen Programms findet man in: turtle_hilfe.pdf. Die Hauptdokumentationen befinden sich in den Headerdateien des Programms!

Eine neue Grafik in C++ erstellen:

Beispiel: "Demo Spirale"

`Siehe Module: grafiken/demo.h und grafiken/demo.cpp.`



1.Schritt: Neue Grafiken werden immer von der Basisklasse `tGrafik` abgeleitet. Es wird eine ähnliche Klasse aus dem Verzeichnis `grafiken/...` auf `grafiken/demo.h` kopiert. Der Klassenname wird auf **`tDemo`** geändert und neue Variablen und Funktionen werden definiert.

```
#ifndef DEMO_H
#define DEMO_H
#include "_grafik.h"

// C++ Grakik Demo >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
class tDemo : public tGrafik
{
public:
  tDemo(QWidget *parent=0) {}
  QString Anleitung();              // Informationen zur Grafik, Script-Beispiel
  void draw();                      // Zeichenbefehle fuer die Grafik

private:
  qreal Xu(qreal u);                // spezielle private Hilfsfunktion
  qreal Yu(qreal u);                // spezielle private Hilfsfunktion
  qreal a,b;
};
```

Ein Modul `grafiken/demo.cpp` wird angelegt und Funktion ***`Anleitung()`*** implementiert. Die Anleitungen werden im Text-Dock angezeigt.

```
#include "demo.h"

QString tDemo::Anleitung()
{
   QString s;
   s = "Archimedische Spiralen\n\n";
   s+= "    x(u) = a * u * cos(u)\n";
   s+= "    y(u) = b * u * sin(u)\n";
   s+= "    ---------------------\n\n";
  return s;
}
```

Die Implementierung der eigentlichen Zeichenfunktion ***`draw()`***:

```
qreal tDemo::Xu(qreal u) { return a*u*cos(u);}
qreal tDemo::Yu(qreal u) { return b*u*sin(u);}

void tDemo::draw()
{
  a = 0.4;
```

```
  b = 0.4;

  init();
  if (!begin()) return; // Transformationsmatrix, Klipping,
                        // Pinsel, Stift und Hintergrund usw. sind gesetzt
                        // Hintergrund und Rahmen wurden gezeichnet
  drawKoordSystem();
  setPen("red");
  const qreal umin = 0.0, umax = 240.0, du = 0.02;  a = 0.4;  b = 0.4;
  qreal u=umin;
  while (u<umax)
  { moveTo( Xu(u), Yu(u)); // Parameterlinie zeichnen
    u=u+du;
  }
}
```

2.Schritt: Für die Einbindung der Grafik in die Galerie wird auch die Klasse `tDemo_Info` in `grafiken/demo.h` benötigt.

```
class tDemo_Info : public tGrafik_Info  //------------------------------
{
public:
  QString  Titel() {return "Demo Spirale";}
  // Der Grafik-Titel muss in der Galerie eindeutig sein!

  tGrafik *New(QWidget *parent=0)  {return new tDemo(parent);}
  // Die Galerie verwendet New() um die spezielle Grafik zu instantieren
};
```

3.Schritt: Damit die neue Grafik in der Galerie aufscheint, muss `tDemo_Info` noch im Konstruktor von `tGalerie` in der Datei `galerie.cpp` eingetragen werden. Die Grafik kann danach auch über Scripte aufgerufen werden.

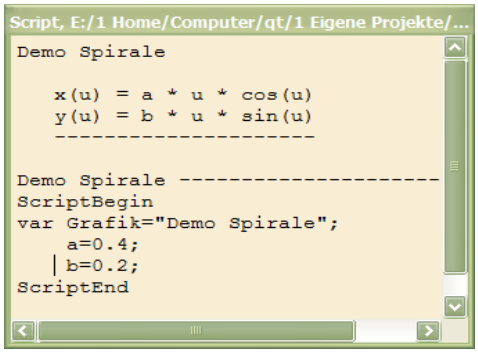
```
#include "grafiken/demo.h"

//tGalerie >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
// >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
//
tGalerie::tGalerie(): QObject(0)
{
  setObjectName("Galerie");
  // C++ Scripvorlagen -----------------
  infos.append(new tScript_Info);
  ...
  // C++ Grafiken ----------------------
  infos.append(new tDemo_Info);
  ...
```

Mit diesen drei Schritten erscheint die Grafik in der Galerie. Sie kann angezeigt oder als PDF-Datei ausgedruckt werden. Soll die Grafik auch über Scripte gesteuert werden, so sind noch folgende Schritte durchzuführen:

Ein Script zur Grafik: Scripte siehe auch: **turtle_scripte.pdf**.

Script zum Beispiel: "Demo Spirale"



Im Script `'Demo Spirale'` können die Parameter a und b gewählt werden. Mit einem Doppelklick ins Script startet man die Ausführung. In einer Textdatei können beliebig viele Scripte angelegt werden. Gestartet wird immer das Script mit dem Textcursor. Für die Implementierung müssen ***Anleitung()*** und ***draw()*** mit folgendem Code ergänzt werden. Das Script kann dann von beliebigen Texten aus gestartet werden.

draw() für das Script erweitern:

```
void tDemo::draw()
{
  ...
  a = getScriptVar("a", 0.4);  // Variable a im Scipt abfragen, Defaultwert: 0.4
  b = getScriptVar("b", 0.2);  // Variable b im Scipt abfragen, Defaultwert: 0.2

  if (!begin()) return;   // Painter.begin()

  // Code wie oben
}
```

Wichtige Objekte und Module

Objekt-Übersicht:

siehe Modul: mainwin.h.

Objekt GDock: tGDock Grafik-Dock-Fenster zum Anzeigen und von Grafik-Widgets der Basisklasse tGrafik.
Objekt Galerie: tGalerie verwaltet die Namen aller Grafiken und erzeugt/zeigt die zugehörigen GDock-Fenster. Kann Grafiken auch in eine PDF-Datei ausgeben.

Siehe Module: galerie.h, galerie.cpp.

Objekt Grafik: tGrafik ist die Basisklasse aller Grafik-Widgets. Die abgeleiteten Klassen dienen zum Zeichnen der speziellen Grafiken in einem GDock-Fenster und zum Ausgeben in eine PDF-Datei. Die eigentlichen Zeichenbefehle werden von der Klasse tTurtle geerbt.

Siehe Module: _turtle.h und _turtle.cpp.
_grafik.h und _grafik.cpp,
grafiken/xxx.h und grafiken/xxx.cpp,

Objekt Steuerung: tSteuerung verwaltet die Aufrufe von C++ Grafiken und Script-Grafiken.

Siehe Module: steuerung.h, steuerung.cpp.

Objekt BefehleLst: tBefehlLst verwaltet die eigenen Grafikebefehle.

Siehe Module: drawbefehle.h, drawbefehle.cpp.

Objekt Rundgang: tRundgang .Rundgang durch die Galerie (Diashow).

Siehe Module: rundgang.h, rundgang.cpp.

Objekt Setup: tSetup liefert alle Einstellungen für das Programm.

Siehe Module: setup.h, setup.cpp.

Objekt Setup: tToolScriptDlg liefert ein Toolfenster mit den Befehlen aus scriptbefehle.h. Die Definitionen aus den public slots können direkt in die Datei scriptbefehle.h kopert werden. Damit werden sie automatisch im Toolfenster angezeigt. scriptbefehle.h ist eine Textdatei, keine Headerdatei. Mit dem Suffix h kann sie übersichtlich mit einem C++ Editor angezeigt werden.

Siehe Module: toolscript.h, toolscript.cpp,
Siehe Befehlsdatei: scriptbefehle.h.

Objekt-Übersicht:

siehe auch Modul: mainwin.h.

Objekt Turtle

tTurtle stellt die grundlegenden Turtlebefehle zur Verfügung. Alle public slots können auch über Scripte aufgerufen werden.

Siehe Module: _turtle.h und _turtle.cpp.

```
public slots:
   /*! Begin class tTurtle Javascript ************************************/
   /*! |t.Turtle|***********************************************************/
    //Basisbefehle für die Turtle
  /*! |Einstellungen|*****************************************************/
       //Einstellungen sind nur in init() wirksam.
       //Alle setPage() Befehle setzen Pen und Brush.
   void setPage(qreal x0, qreal y0, int dinA=255);
       //Papierformat und Koordinatenursprung einstellen.
       //Linken obere Ecke: (x0,y0) in mm.
       //dinA: 0-9 Querformat. 10-19 Hochformat. 255 Setup.
  void setPage(int din=255);
       //Koordinatenursprung in Blattmitte
       //dinA: 0-9 Querformat. 10-19 Hochformat. 255 Setup.
  void setMargin(qreal RandX=-1, qreal RandY=-1);
       //RandX=-1 oder RandY=-1: Rand aus Setup.
       //RandX=-2 oder RandY=-2: Zeichenrechteck quadratisch.
  void setAutoPaint(bool on=true);
       //Automatisches Neuzeichnen abschalten.
       //true:  Grafik immer automatisch neu zeichen.
       //false: Grafik erst nach Mausklick neu zeichnen.
  void setAntiAliasing(bool on=true) ;
       //Antialaising. On: feinere Linien, langsamer.
  /*! |Pen und Brush|*****************************************************/
       //Zeichenstift und Malfarbe
  void setNoStyle();
       //Turtle zeichnet unsichtbar.
       //Aktuellen Pen und Brush merken.
       //Pen=0 und Brush=0 setzen.
  void setPen(const QString &Farbe="", qreal Breite=-1, int Style=-1);
       //Farbe siehe Hilfe Farben oder Tool: Farbmischung.
       //Breite: in mm, -1 für ein Pixel.
```

```
      //Style: 0-6, siehe Tool: Penstyle.
      //Defaults: Letzten Stift verwenden.
void setPenWidth(qreal Breite, int CapStyle=-1);
      //Breite -1: nur CapStyle setzen.
      //CapStyle: Linenenden Flat=0, Square=16 oder Rounded=32 setzen.
void setBrush(const QString &Farbe="", int Style=-1, int alpha=-1);
      //Farbe siehe Hilfe Farben oder Tool: Farbmischung.
      //Style 0-24, siehe Tools: Brushstyle.
      //Transparenz 0-255.
      //Defaults: Letzten Brush verwenden.
void setGradient(qreal x1, qreal y1, qreal x2, qreal y2, const QString &Rgb1, const QString &Rgb2);
      //Linearen Farbverlauf von Farbe1 in P1(x1,y1) nach Farbe2 in P2(x2,y2) festlegen.
      //Verwendung nur mit setBrush("",15);
void setGradient(qreal t, const QString &Rgb);
      //0<t<1: Farbe im Zwischenpunkt P1 + t*(P2-P1) festlegen.
      //Verwendung mit setBrush("",15).
/*! |Drehen|*************************************************************/
      //Winkel in Grad. Uhrzeigersinn ist negativ.
void turn(qreal w);
      //Richtung der Turtle realtiv ändern.
void turnTo(qreal w);
      //Richtung absolut einstellen, x-Achse hat 0°.
void turnTo(qreal x, qreal y);
      //Richtung (lastX(),lastY()) nach P(x,y)
void turnTo(const QString Name);
      //Richtung (lastX(),lastY()) nach Punkt "Name"
/*! |Bewegen|************************************************************/
    //Bewegungen mit aktuellem Pen. Einheit mm.
void move(qreal mm);
      //Turtle um mm in aktueller Richtung bewegen.
void moveTo(qreal x, qreal y);
      //Turtle auf Punkt (x,y) bewegen
void moveTo(const QString Name);
      //Turtle auf Punkt "Name" bewegen
void goTo(qreal x, qreal y);
      //Turtle ohne Pen auf (x,y)
void goTo(const QString Name, int Index=0);
      //Turtle ohne Pen auf Punkt "Name"[Index].
      //Index= 0: erster Polygonpunkt, usw.
      //Index=-1: letzter Polygonpunkt.
      //Index=-2: vorletzter Polygonpunkt, usw.
/*! |Polygon|************************************************************/
//Die Turtle besitzt ein Polygon. Das Turtle-Polygon wird
//mit beginPolygon() und endPolygon(...) erzeugt.
void beginPolygon();
      //Polygon löschen, Polygonmodus einschalten. Die Endpunkte
      //jeder Turtlebewegung werden ans Polygon angehängt.
void endPolygon(int FillMode=0);
      //Polygonmodus beenden.
      //FillMode 0: Polygon schliessen, Qt::OddEvenFill.
      //FillMode 1: Polygon schliessen, Qt::WindingFill.
      //FillMode 2: Polygon schliessen, nicht füllen.
      //FillMode 3: Polygon nicht schliessen, nicht füllen.
void drawPolygon();
       //Polygon auf Turtlepos mit -richtung zeichnen.
void movePolygon(qreal x, qreal y);
      //Polygonpunkte verschieben.
      //Punkt Polygon[0] auf Punkt (x,y).
void rectPolygon(qreal dx, qreal dy, int FillMode=0);
      //Rechteckiges Polygon auf Turtlepos mit -richtung.
void trianglePolygon(qreal c, qreal w_ca, qreal a,  int FillMode=0);
      //Dreieck ABC mit A(lastX(),lastY())
      //c in Richtung lastW();
      //w_ca ist Winklel ca
void isoPolygon(qreal r, int n=-1, int FillMode=0);
      //Polygon mit n-Ecken, Radius r, Mittelpunkt Last(), Richtung lastW()
      //n=-1: Kreisnäherung, n=-2 höhere Auflösung für Kreis
void splinePolygon(qreal dt=0.1);
      //Polygon durch ein Spline-Polygon ersetzen.
void cPolygon();
      //Polygon durch C-Polygon ersetzen.
      //Polygon am Endpunkt spiegeln
void clrPolygon();
      //Polygon löschen und Polygonmodus beenden
/*! |Text|**************************************************************/
void text(const QString &s);
      //Text an Turtlepos in -richtung anzeigen.
void text(qreal x, qreal y, const QString &s);
      //Text an P(x,y) in -richtung anzeigen.
void text(qreal x, qreal y, qreal x1, qreal y1, const QString &s);
      //Text in Rechteck P(x,y) und Q(x1,y1) schreiben.
void setFont(const QString &Font, qreal Size, bool Fix=true, int Weight=50);
      //Font: "Times","Courrier","Arial" usw.
      //Size: in mm
      //Fix: Proportional
      //Weight: 0-99, Normal=50, Bold=75
void setWeight(int Weight);
      //Weight: 0-99, Normal=50, Bold=75
/*! |Rechtecke|**********************************************************/
void rahmen();
      //Blattrahmen neu zeichnen.
void rectangle(qreal dx, qreal dy);
      //Rechteck (dx,dy) in Turtlerichtung zeichnen.
```

```
/*! |Punkt|*************************************************************/
void drawPoint(qreal x, qreal y);
     //Punktmarke zu Punkt(x,y) zeichnen.
/*! |Bilder|************************************************************/
qreal bild(const QString &DateiName, qreal w_mm=-1, qreal h_mm=-1);
      //Bild an Turtleposition in -richtung zeichnen.
      //Breite w_mm, Hoehe h_mm. Ist ein Wert -1: proportional.
      //Rueckgabe: 0 oder die fehlende Breite oder Höhe
/*! |Abfragen|***********************************************************/
    //Die aktuellen Einstellungen der Turtle abfragen.
qreal  lastX();
       //Letzte x-Koordinate
qreal  lastY();
       //Letzte y-Koordinate
qreal  lastW();
       //Letzte Richtung in °
qreal  getW();
       //Richtung des Vektors (0,0)->(lastX(),lastY())
double getL();
       //Betrag des Vektors (0,0)->(lastX(), lastY())
void   msgTurtle();
       //Einstellungen der Turtle in der MsgBox ausgeben
qreal  P0x();
       //Plotrechteck: x,links,oben
qreal  P0y();
       //Plotrechteck: y,links,oben
qreal  P1x();
       //Plotrechteck: x,rechts,unten
qreal  P1y();
       //Plotrechteck: y,rechts,unten
/*! End Javascript ***************************************************/
```

Objekt Grafik

`tGrafik` stellt komplexe Grafikbefehle zur Verfügung. Alle public slots können auch über Scripte aufgerufen werden.

```
Siehe Module: _grafik.h und _grafik .cpp.

public slots:
   /*! Begin class tGrafik Javascript  ***********************************/
   /*! |t.Grafik|*************************************************************/
       //Erweiterte Turtlebefehle
   /*! |Koordinatensystem|*****************************************************/
       //Koordinatensystem zeichnen
   void drawKoordSystem(bool Beschriftung=true, qreal Einheit=10, qreal PSize=0.6);
        //Beschriftung true/false.
        //Einheit: Abstand der Marken in mm.
        //PSize: Grösse in mm.
   void drawRaster(qreal Einheiten_mm=10, qreal PSize=0.6);
        //Rastermarken zeichnen.
        //Einheit: Abstand in mm.
        //PSize: Grösse in mm.
   /*! |Meldungen|*************************************************************/
       //Textausgaben im Meldungsfenster
   void msg(QVariant s=" ", const QString rgb="black");
        //Meldung s im Meldungsfenster ausgeben
   void clrMsg();
        //Meldungsfenster löschen
   void msgScript(bool showWerte=true);
        //Javascript-Objekte des Scripts in der Msgbox ausgeben.
        //true : Objektnamen und Werte.
        //false: Nur Objektnamen.
   QString infoScript(bool showWerte=true);
           //String für msgScript()
   /*! |Farben und Zufall|*******************************************/
   qreal   rnd(qreal max);
           //Zufallszahlen aus [0,max]
   qreal   rndW(qreal min, qreal max);
           //Zufallszahlen [min,max] und [-min,-max]
   QString rndRGB1();
           //Zufallsfarbe 1
   QString rndRGB2();
           //Zufallsfarbe 2
   void    setRndRGB(const QString &RGB1="", const QString &RGB2="black");
           //Anfangswerte für rndRGB1() und rndRGB2()
   void    setRndHeller(int Heller=100, int rndHeller=0);
           //Änderungswertewerte für rndRGB1() und rndRGB2()
   /*! |Diashow|*************************************************************/
   qreal showDia(qreal ms, qreal Startwert, qreal Decrement=1);
         //Grafik alle ms neu zeichnen.
         //Startwert mit Decrement herunterzählen.
         //Rückgabe: DiaRestwert in ms.
         //
         //if (DiaRestwert<=0)  DiaRestwert = Startwert - Decrement;
         //else DiaRestwert = DiaRestwert - Decrement;
         //if (DiaRestwert>0)   Neuzeichnen in ms.
         //else Stop
   /*! |t.Befehle|*************************************************************/
       //Eigene Befehle für Funktionen und Polygonobjekte.
   /*! |Befehle: Funktionen|*****************************************/
```

```
    //Funktionen werden in der Form
    //function 'Name'() { Befehl; Befehl; ... }
    //gespeichert.
void defCall(const QString Name, const QString Befehle, int mode=0);
     //Funktionsdefinition 'Name' anlegen oder erweitern.
     //mode=0: Funktion "Name" neu anlegen und Befehle anhängen.
     //mode=1: Befehle an Funktion "Name" anhängen.
void recDraw(QString Name="");
     //Befehlsrecorder. Turtlebefehle als Funktion aufzeichnen.
     //Name<>"": Funktion 'Name' anlegen und Aufzeichnugsmodus ein.
     //Alle folgenden Turtlebefehle werden aufgezeichnet.
     //Name=="": Aufzeichnung beenden.
void call(const QString Name);
     //Funktion ausführen
/*! |Befehle: Polygone definieren|*************************************/
    //Mit def-Aufrufen werden verschachtelte
    //Polygonobjekte definiert.
void defPoint(const QString Name);
     //Punkt (lastY(),lastY()) als Punkt "Name" speichern
void defPoint(const QString Name, int Index);
     //Punkt aus Turtlepolygon kopieren.
     //Polygon[Index] unter "Name" speichern.
     //Index=-1 letzter Punkt.
void defPoint(const QString Name, const QString P1, const QString P2, qreal t);
     //Teilungspunkt P = P1 + t*(P2-P1) als Punkt "Name" speichern
void defVector(const QString Name, const QString P1, const QString P2, qreal t=1);
     //Ortsvektor t*(P2-P1) als Punkt "Name" speichern
void defLine(const QString Name,  qreal l);
     //P1 (lastY(),lastY()) und P2 move(l) mit LastW()
     //P1 und P2 als Befehls-Polygon "Name" speichern
void defLine(const QString Name, const QString P1, const QString P2);
     //Punkte P1 und P1 als Befehls-Polygon "Name" speichern
void defDraw(const QString Name, int mode=0);
     //Speichert unter 'Name' Listen von Polygonobjekten.
     //Ein Polygonobjekt übernimmt von der Turtle die
     //Daten: Polygon, Pen, Brush, FillMode.
     //
     //mode=0: 'Name' neu anlegen und Turtledaten speichern.
     //mode=1: Turtledaten an Befehl 'Name' anhängen.
     //mode=2: Polygon zuerst am ersten Polygonobjekt
     //klippen und  Turtledaten anhängen.
     //mode=3: Einpassen:
     //Das erste Polygon [N0, .. Nm] von 'Name'
     //durch das Turtlepolygon [P0, .. Pn] ersetzen.
     //N0!=Nm: Turtlepolygon in Polygon 'Name'
     // einpassen. P0 -> N0 und Pn -> Nm
     //N0==Nm: Turtlepolygon verschieben mit P0 -> N0
bool defScript(const QString Flag, const QString Grafikname);
     //Befehle eines anderen Scripts in die
     //Befehlsliste eintragen und ein Funktions-Flag setzen.
     //
     //Funktion Flag() ist in Befehlsliste:
     //  * nichts tun
     //  * Rückgabe true
     //Flag ist nicht in Befehlsliste:
     //  * Script 'Grafikname' ausführen
     //  * Funktion Flag() eintragen
     //  * Aktuelles Script neu starten
     //  * Rückgabe false
/*! |Befehle: Polygone zeichnen|*************************************/
    //Alle Polygonobjekte werden an der aktuellen Turtleposition,
    //mit der aktuellen Turtlerichtung und unter
    //Verwendung der Abbildung 'drawMatrix' ausgeführt.
void draw(const QString Name, bool UseStyle=true);
     //'Name' ist Polygonobjekte:
     //    Befehl an der aktuellen Turtleposition
     //    und mit Turtlerichtung und unter
     //    Verwendung der drawMatrix ausführen.
     //'Name' ist Funktion: Funktion ausführen.
     //UseStyle==true : Definierten Style verwenden
     //UseStyle==false: Aktuellen Turtle-Style verwenden
void drawName(const QString Name, qreal dx=0, qreal dy=0);
     //Beschriftung 'Name' an Objektposition anzeigen.
     //Position um (dy,dy) veschieben.
void shift(const QString Name,  qreal t=1);
     //Objekt 'Name' verschieben und drehen.
     //Schiebung: (lastX(),lastY()) + t*(Schiebvektor)
     //Drehung  : um lastW();
/*! |Befehle: Abbildungen|*****************************************/
    //Befehle zum Erzeugen der Abbildungsmatrix 'drawMatrix'
    //Achtung: Die Einstellungen verändern die bestehende
    //Matrix kumulativ und sind nicht kommutativ!
    //reset=true : Abbildung auf die Einheitsmatrix anwenden.
    //reset=false : Abbildung auf drawMatrix anwenden.
void setRotate(qreal Winkel, bool reset=true);
     //Drehung um Punkt(0,0), Winkel in Grad.
void setRotate(const QString Punkt, qreal Winkel, bool reset=true);
     //Drehung um Punkt, Winkel in Grad.
void setMirror(const QString Punkt1, const QString Punkt2, bool reset=true);
void setMirror(qreal x0, qreal y0, qreal x1, qreal y1, bool reset=true);
     //Spiegelung an der Geraden (Punkt1,Punkt2);
void setScale(qreal sx, qreal sy, bool reset=true);
void setShear(qreal sh, qreal sv, bool reset=true);
```

```
void setTranslate(qreal dx, qreal dy, bool reset=true);
     //Schiebung um (dy,dy)
void setIdentity();
     //Einheitsmatrix (reset)
/*! |Befehle: Turtlepolygon|*************************************************/
    //Befehle zum Abbilden des Turtlepolygons
    //mit 'drawMatrix'
    //Der erste Polygonpunkt hat den Index 0!
void setPolygon(int Index=0, int Anzahl=-1);
     //Turtlepolygons durch eine Abbildung mit drawMatrix ersetzen.
     //Der erste Polygonpunkt hat den Index 0!
     //Der letzte Polygonpunkt hat den Index -1.
     //
     //Index= 0, Anzahl=-1: Alle Punkte verwenden.
     //Index=-1, Anzahl=-1: Alle Punkte, Reihenfolge umgekehrt.
     //Index>=0, Anzahl>=0: Anzahl Punkte ab Index verwenden.
     //Index<-1, Anzahl>=0: Anzahl Punkte ab Index, Reihenfolge umgekehrt.
void addPolygon(const QString Name, int mode=0);
     //Turtlepolygon erweitern.
     //Das erste Polygon von "Name" mit drawMatrix abbilden
     //und zum Endpunkt des Turtlepolygons verschieben.
     //mode=0 : Punkte in gegebener Reihenfolge anhängen.
     //mode=-1: Punkte in umgekehrter Reihenfolge anhängen.
void setPolygon(const QString Name, int mode=0, qreal dt=0.1);
     //Turtlepolygon aus dem ersten Polygon von "Name" erzeugen.
     //"Name" wird zuerst mit drawMatrix abgebildet.
     //mode=0: Turtlepolygon durch "Name" ersetzen.
     //mode=-1: Turtlepolygon durch "Name" ersetzen. Reihenfolge umgekehrt.
     //mode=1: Turtlepolygon Duchschnitt "Name".
     //mode=2: Turtlepolygon Vereinigung "Name".
     //mode=3: Turtlepolygon geschlossen: Turtlepolygon ohne "Name".
     //Turtlepolygon offen: Turtlepolygon mit "Name" abschneiden.
     //mode = 4: "Name" liefert die Stützpunkte eines quadratischen Bezier Splines.
     //epsilon<dt<1 Bestimmt die Punktedichte pro Spline-Segment.
void  msgPolygon(bool Spline=false);
      //Polygon als Funktion im Meldungsfenster anzeigen.
      //Kann ins direkt ins Script kopiert werden
      //Spline==true: Splinepolygon erzeugen
/*! |Befehle: Sonstiges|*****************************************************/
    // Verwaltung
bool exists(const QString Name);
     //Rückgabe true: Befehl 'Name' existiert.
void msgDraw(const QString Name="");
     //'Name': Alle Befehle der Befehlsliste anzeigen
     //'Name'=="": Infos zum Befehl 'Name'
QString drawInfo(const QString Name);
     //msgDraw() als String.
void clrDraw(const QString &Name);
     //Befehl 'Name' löschen.
void clrDraw();
     //Befehlsliste löschen
/*! End Javascript *********************************************************/
```

Aufruf eines Scripts:

Die Funktion `Dock->getSource(ScriptBegin, ScriptEnd)` der Klasse `tDock` ermittelt den Script-Text an der Textcursorposition im Text-Fensters.

Aus dem Script-Text wird zuerst die Zeile **`var Grafik="Name"`** ausgewertet, um den Namen des Grafikfensters zu bestimmen. Die Funktion show() der Klasse `tGalerie` erzeugt und startet ein Grafikfenster mit der Grafik **`"Name"`** und dem Script.

Die spezielle Grafikklasse **`"Name"`** ist Erbe der Basisklasse `tGrafik`. Die Basisklasse implementiert drei Möglichkeiten zur Kommunikation des Scripts mit dem C++ Programm.

1. Variablen aus dem Script von C++ aus abfragen:

   Text im Script: **`var dx = 0.6`**

   Auswertung in C++: `dx=getScriptVar("dx",0.2);  // 0.2 Defaultwert`

2. Vom Script aus Funktionen aus C++ Klassen aufrufen:

   Text im Script:
   ```
   function init()
   { t.setPage(-100.0, 100.0, 4, 8);
   }
   ```

   In C++: Script **`t`** wurde auf die Klassen `tGrafik` und `tTurtle` eingestellt.
   Damit stehen alle Funktionen dieser Klassen aus dem Abschnitt public slots dem Script automatisch zur Verfügung.

3. Von C++ aus eine Funktion im Script aufrufen:

   Text im Script: **`var a = 0.3;`**

```
function Xu(u) { return a * u * Math.cos(u);}
```

Auswertung in C++:

```
QScriptValueList args;
QScriptValue Xu= Script()->evaluate("Xu");

qreal u; args<<u;
qreal ergebnis = Xu.call(thisScript, args )
```

Die Klasse `tGrafik` stellt `Script()` und `thisScript` zur Verfügung.

Das ursprüngliche Entwicklungsziel waren C++/Qt-Widgets für berechnete Grafiken mit einem mathematischen Koordinatensystem und Turtle-Funktionen. Die sehr gute Javascriptschnittstelle von C++/Qt führte zum vorliegenden Programm.

Das Programm wurde mit Qt 4.7.4 mit Qt-Creator entwickelt. Das Progamm kann also auch für Apple- oder Linux-Betriebssysteme compiliert werden.
