ni.com

Advanced LabVIEW Programming

LabVIEW User Group Meeting 2003

Christian Hamp

Projekt Consultant ATE

LabVIEW Advanced Programming

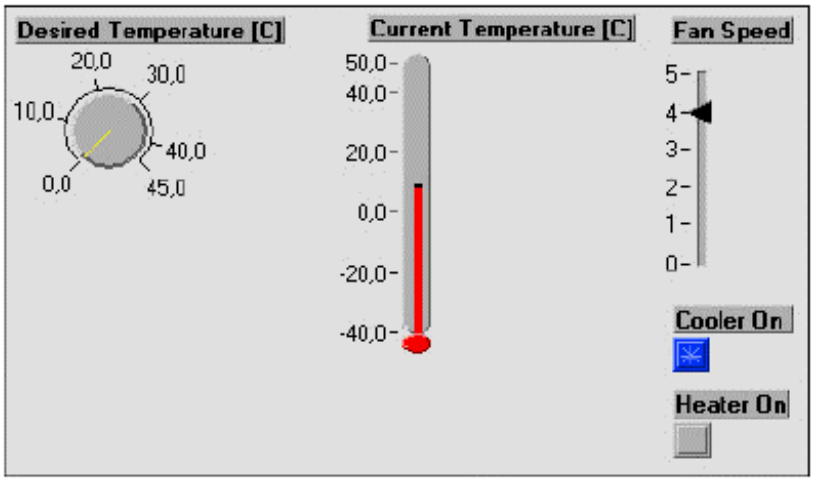
- Designkonzepte
- Kommunikationsmechanismen
- OOP
- Design Patterns
- Informationsquellen und Ressourcen

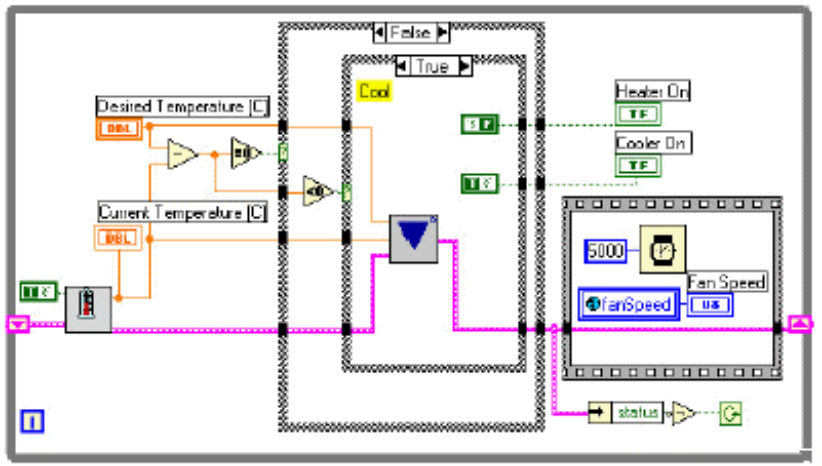
Designkonzepte

- monolithische Programme
- modulare Programme
- Zustandsautomaten

Monolithische Programme

- Bedienoberfläche und Funktionalität sind „vermischt“
- GUI-Routinen werden erstellt und mit Funktions-Code gefüllt
- gemeinsame Daten z.B. globale Variablen oder Shift-Register
- geeignet für kleine Programme, Beispiele etc.



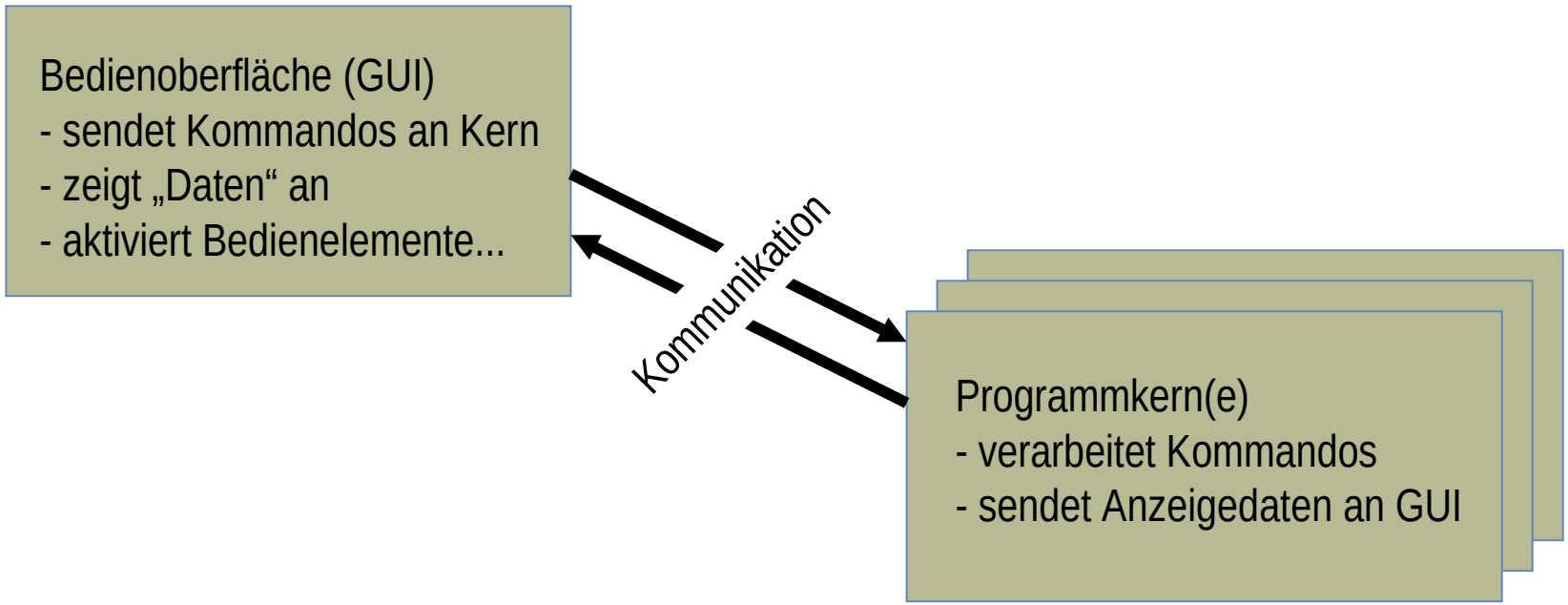


Nachteile monolithischer Programme

- **schlechte Skalierbarkeit**
  - GUI und Funktionen nutzen gleiche Daten (lokale Variablen, etc.)
  - Funktionen greifen auf GUI zu (Property-Nodes)
- **schlechte Wartbarkeit**
  - Code wird bei Erweiterungen schnell unübersichtlich
  - Änderung des GUIs verursacht auch immer Änderungen an der Funktionalität
- **kaum Wiederverwendung möglich**
  - GUI und Funktionen sind stark aneinander gebunden (s.o.)

Modulare Programme

- Bedienoberfläche und Programmkern sind getrennt
- Programmkern kann in mehrere Komponenten aufgeteilt werden
- Kommunikation der Komponenten über definierte Schnittstellen

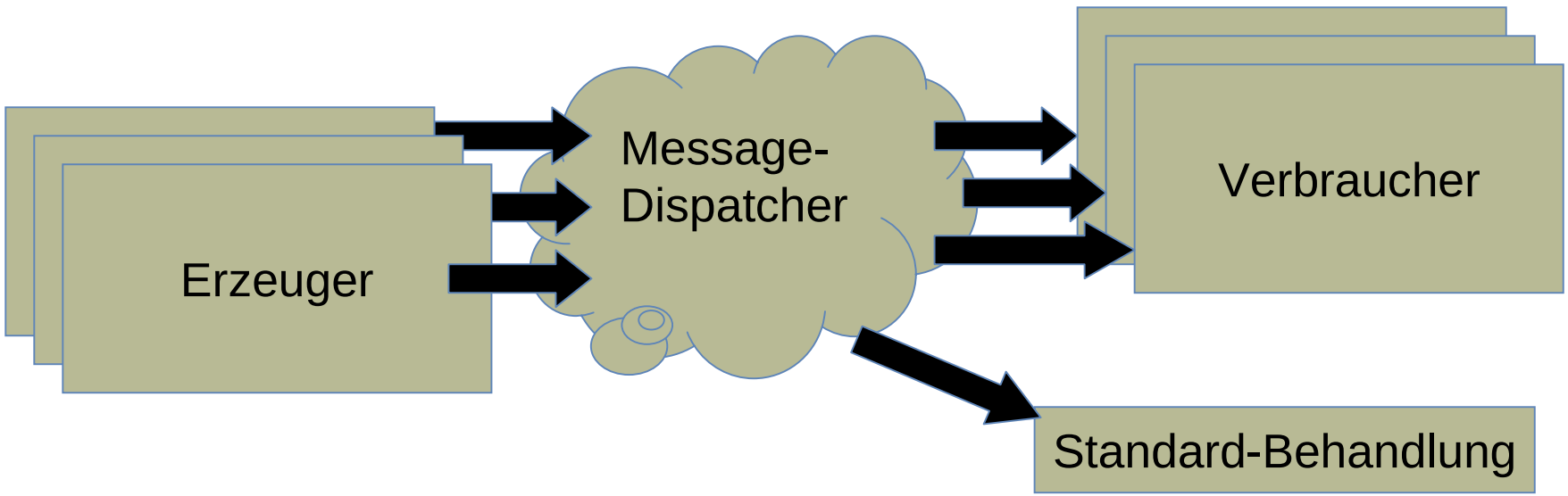


Vorteile modularer Programme

- **gute Skalierbarkeit**
  - GUI und Funktionen sind unabhänging von einander erweiterbar
  - Teilfunktionen können leicht ausgelagert werden (z.B. LabVIEW-RT)
- **Wartbarkeit**
  - Einzelne Komponenten können isoliert betrachtet werden
  - Implementation unabhängig von der Schnittstelle (Nachrichten)
- **Wiederverwendung möglich**
  - Zustandsautomaten für Teilfunktionen

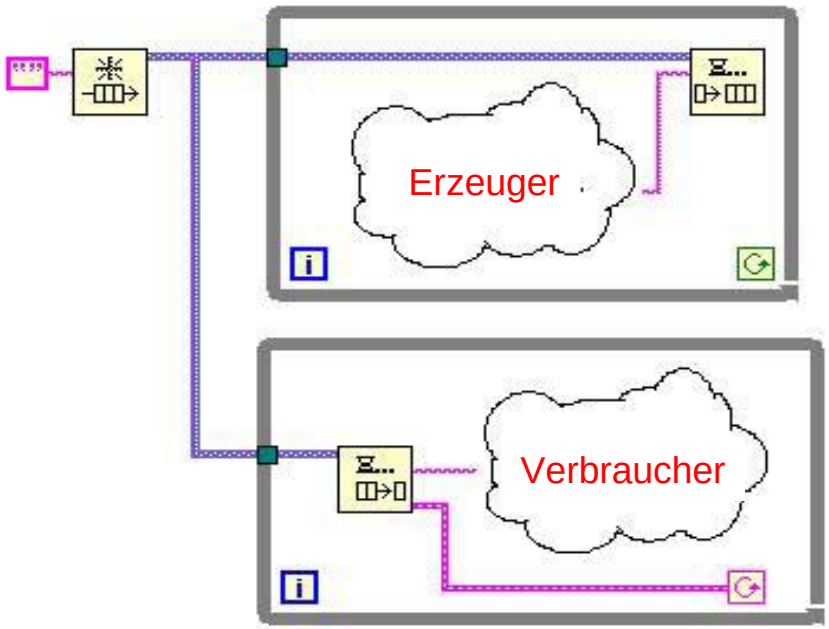
Modulare Programme

- **Weitere Ideen**
  - Broadcast Messages
  - unbekannte oder variable Zahl an Erzeugern und Verbrauchern



Erzeuger-Verbraucher-Konzept (Daten)

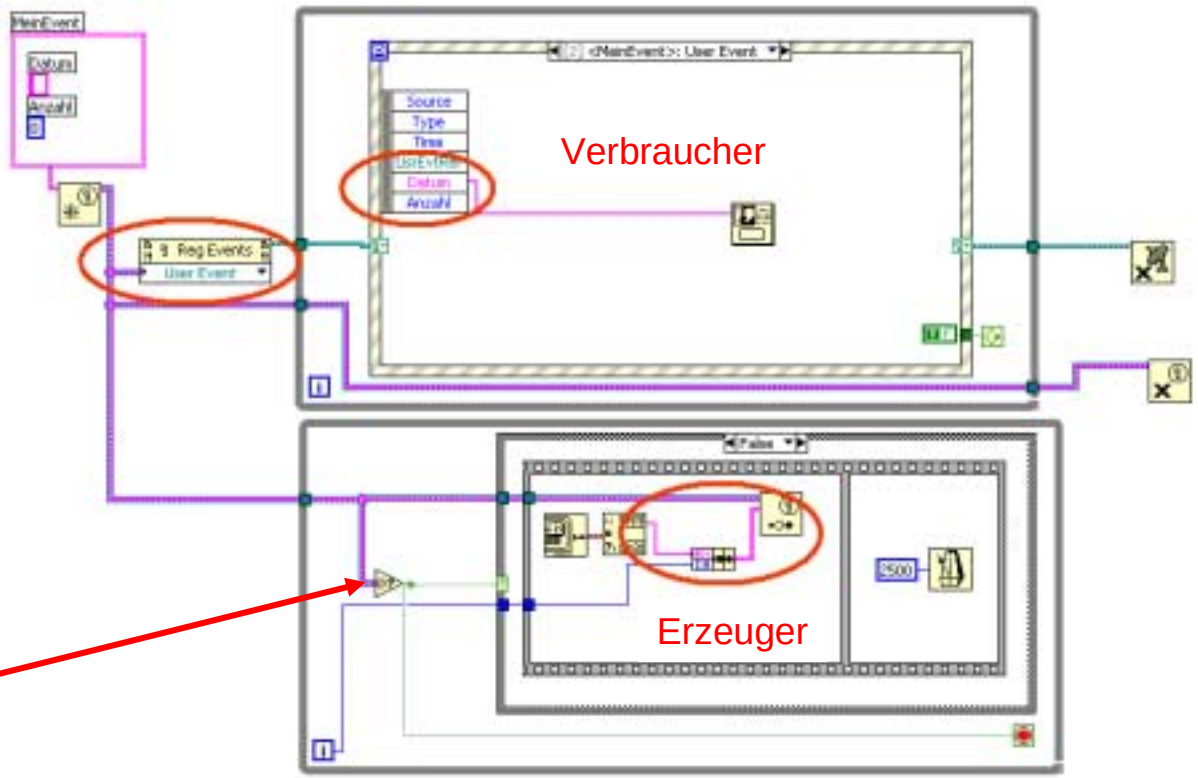
- Trennung von Programmkern und Benutzerschnittstelle
- Verhindert bei GUI das Blockieren beim Abarbeiten von Kommandos
- Kopplung über Queues



Erzeuger-Verbraucher-Konzept (Events)

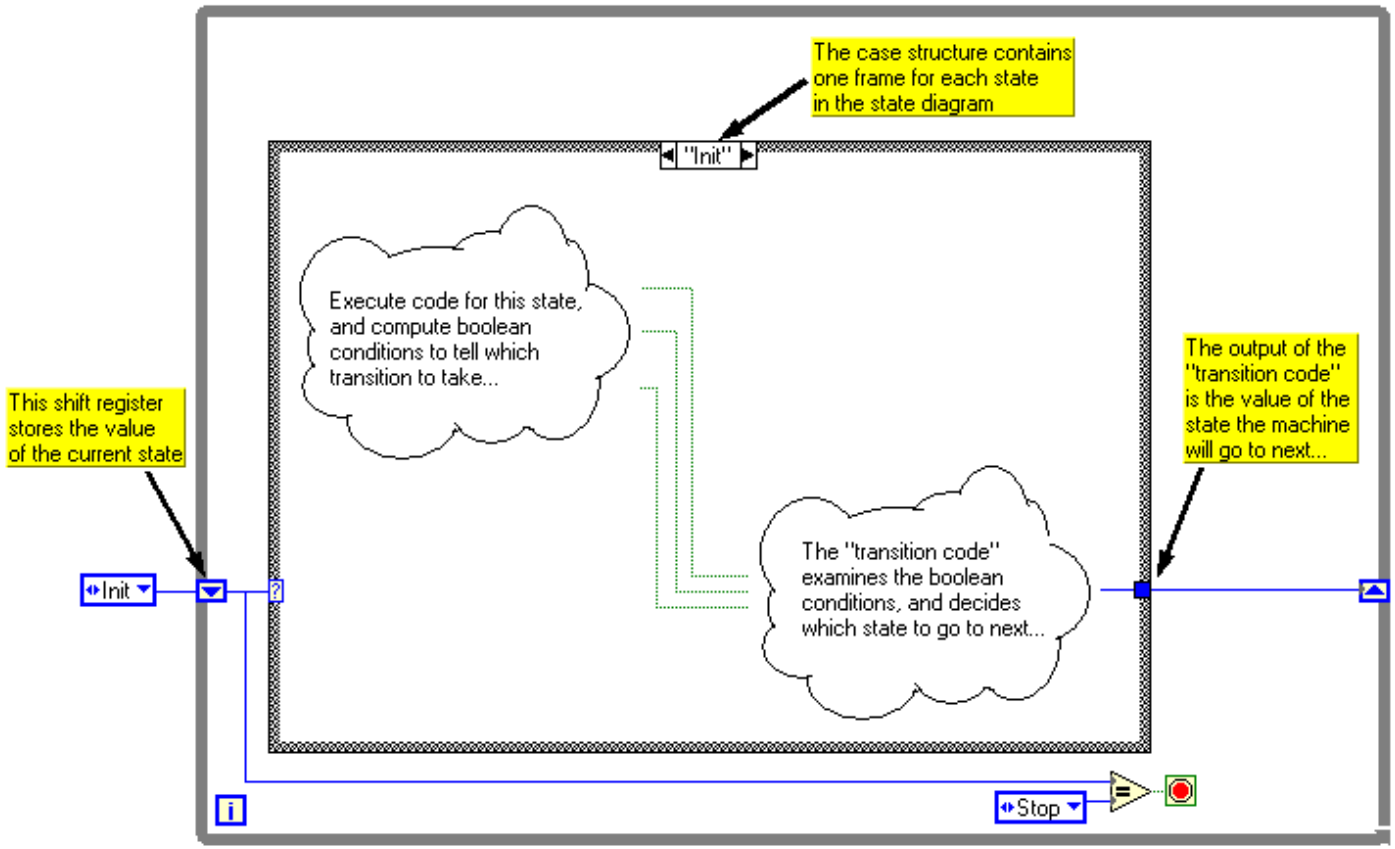
Neue Funktionen in LabVIEW 7 Express

- User Events
- Dynamische Events



Tipp zum Beenden beider Schleifen:
Prüfen auf ungültige Queue-Referenz

Zustandsautomat (State-Machine)

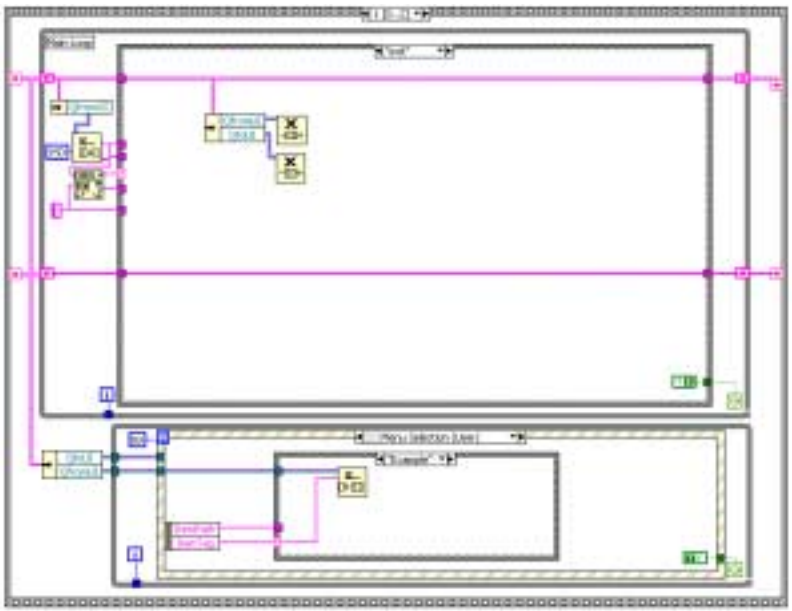


Kommunizierende Zustandsautomaten

- Zustandsautomat nimmt Kommandos entgegen und arbeitet sie ab
- Mehere Automaten für Teilfunktionen (IO, Netzwerk, Autosave,...)
- Austausch definierter Messages (Typedefinitionen)

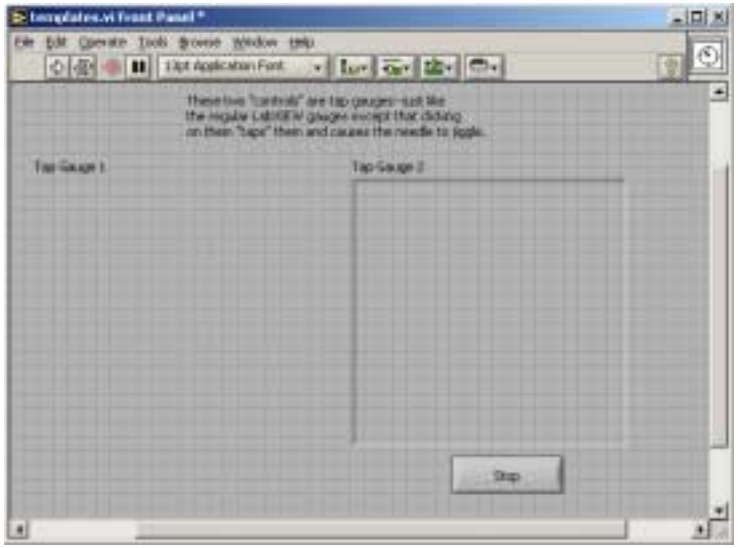
Programmkern

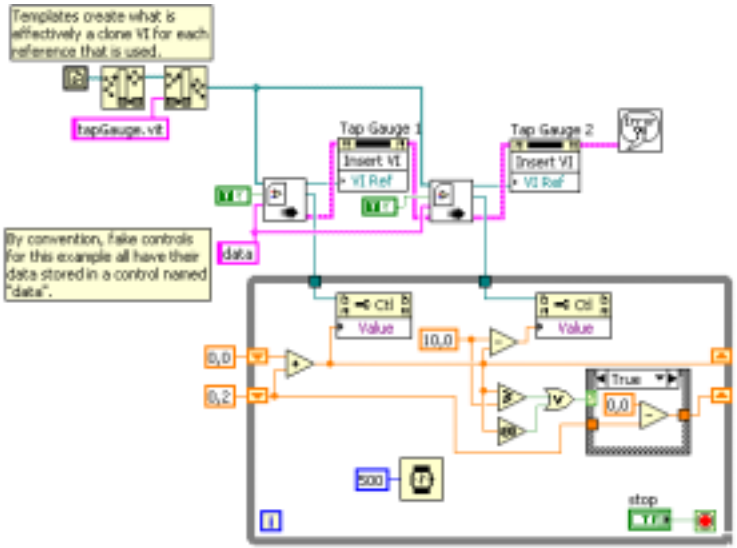
UI-Manager



Zu Modularisierung: LabVIEW 7 Subpanel

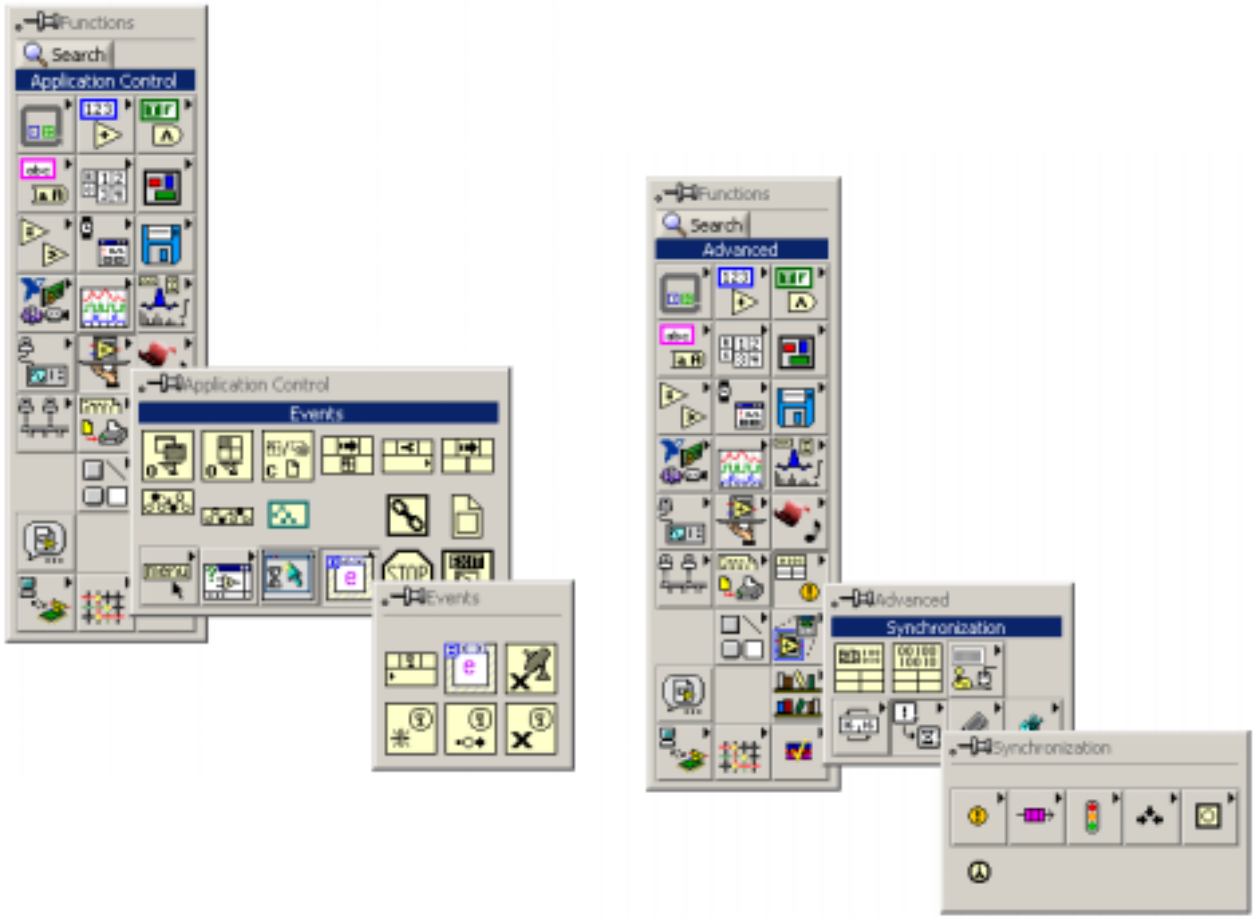
- Modularer Aufbau von Frontpaneln
- Plug-In Architektur von





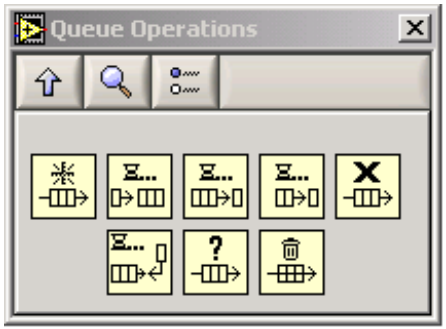
Kommunikationsmechanismen

- Queues
- Notifier
- Semaphoren
- Rendezvous
- Events

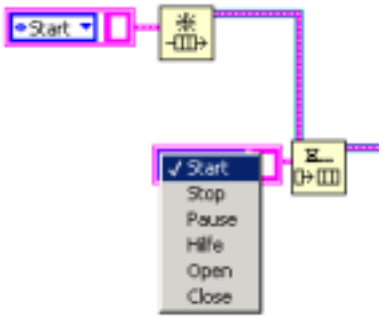


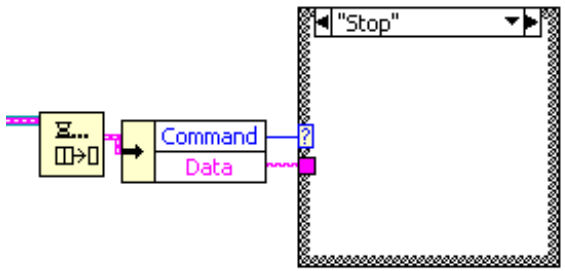
Queues

- FIFO-Prinzip
- einer oder mehere Erzeuger und ein Verbraucher
- alle Datentypen, auch Cluster, Typedefs, etc.



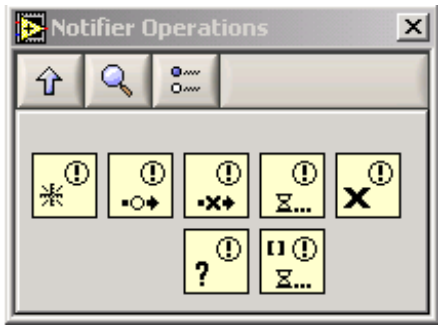
Idee für Zustandsautomaten und Kommandos mit Enums





Notifier

- nicht gepuffert
- von einem Sender an einen oder mehrere Empfänger
- von mehreren Sendern an einen oder mehrere Empfänger
- alle Datentypen, auch Cluster

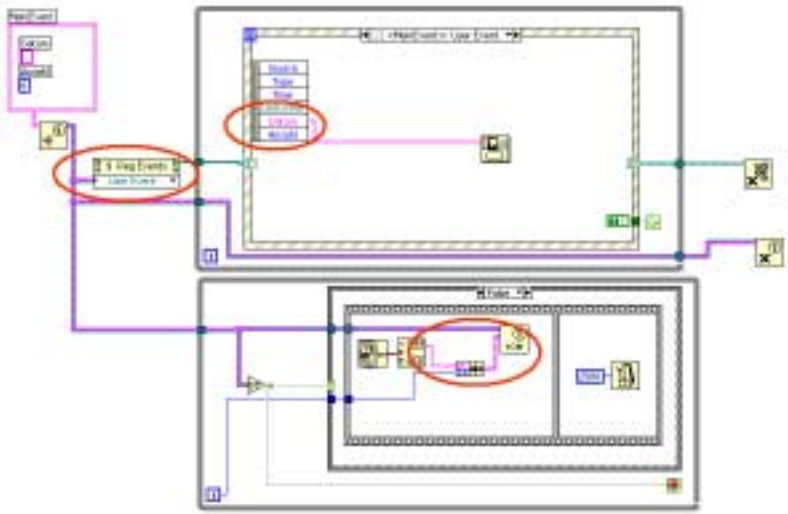


Semaphoren und Rendesvous

- Wichtig bei Multithreading-Applikationen
- Mit Semaphoren kann verhindert werden, dass bestimmte Programmteile gleichzeitig laufen
- Rendesvous können benutzt werden um parallel laufende Programmfäden zu synchronisieren

Events

- Seit LabVIEW 6.1
- Neu in LabVIEW 7.0
  - dynamische
  - selbst definierbare Events
  - Events programmgesteuert auslösen

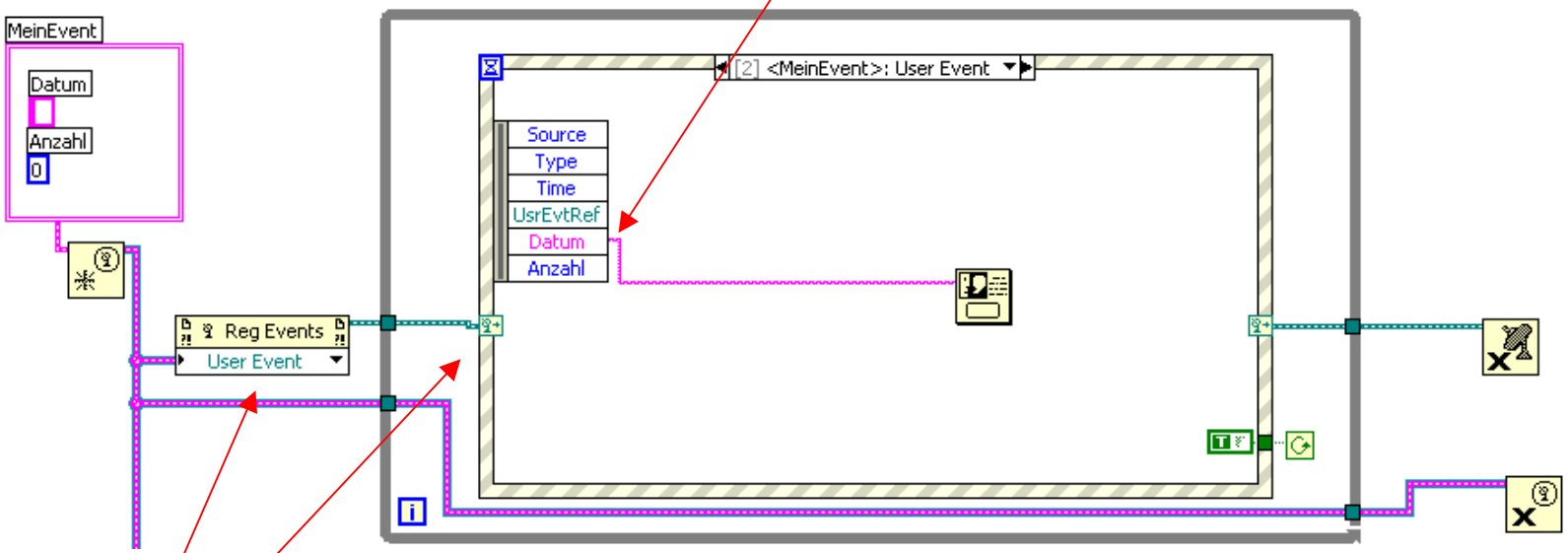


Dynamische Events

User Events

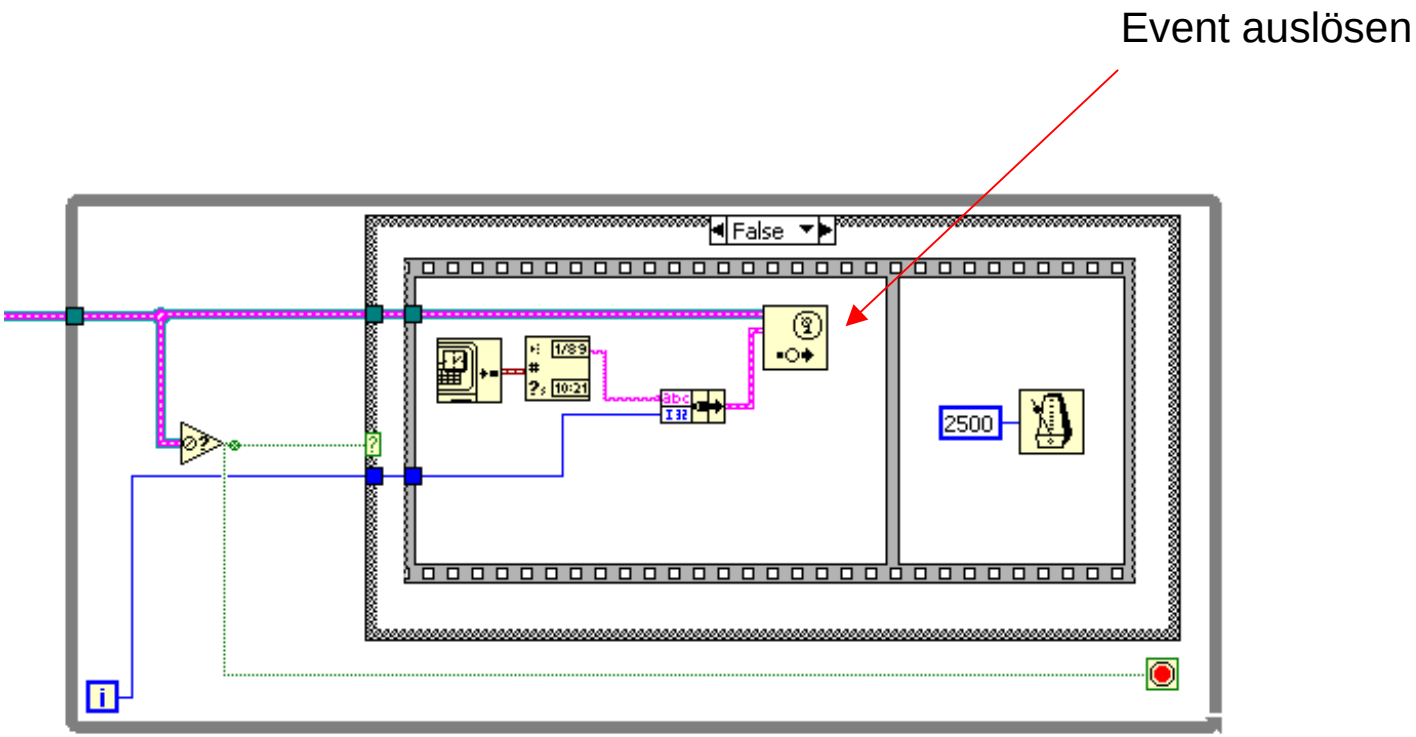
Ausgabe der Eventdaten im Event-Case

Eventdaten definieren



Event registrieren und der Struktur bekannt machen

User Events



Objektorientierung

Objekte

- **Repräsentieren „reale“ Dinge wie Textdokumente, Autos, PCI-Karten**
- **Funktionen (Methoden)**
- **Daten (Variablen, Eigenschaften)**
- **Kapseln Daten im Inneren**
- **Kein direkter Zugriff auf Daten nur über Methoden**

Objekt „Automotor“

-Methoden
- -Anlassen
- -Gas geben
- -...

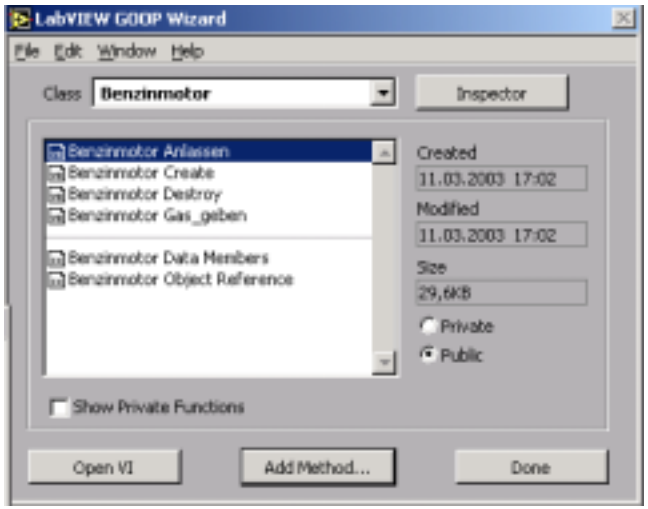
-Eigenschaften
- -Drehzahl
- -Zündwinkel
- -...

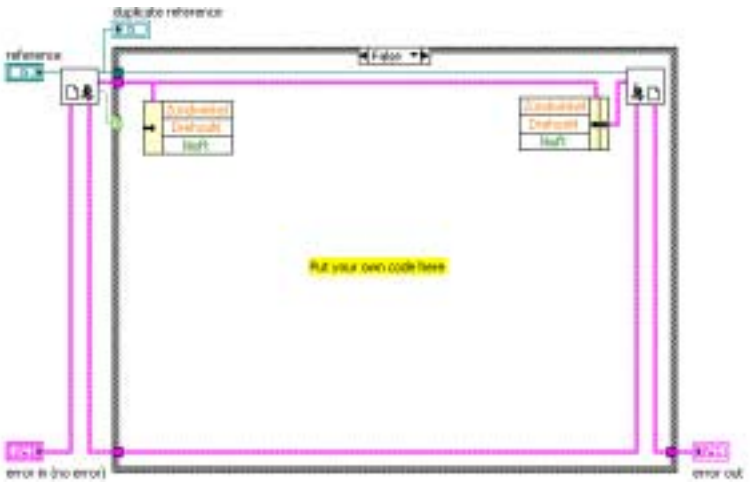
- Beim Anlassen wird von der Methode des Objekts alles notwendige durchgeführt um den Motor zu starten
- Als Ergebnis wird eine Statusmeldung zurückgegeben
- Benutzer brauche ich keine Motorparameter zu beachten
- Umstieg von „Benziner“ auf „ Diesel “ ohne Änderung beim Benutzer

GOOP-Toolkit

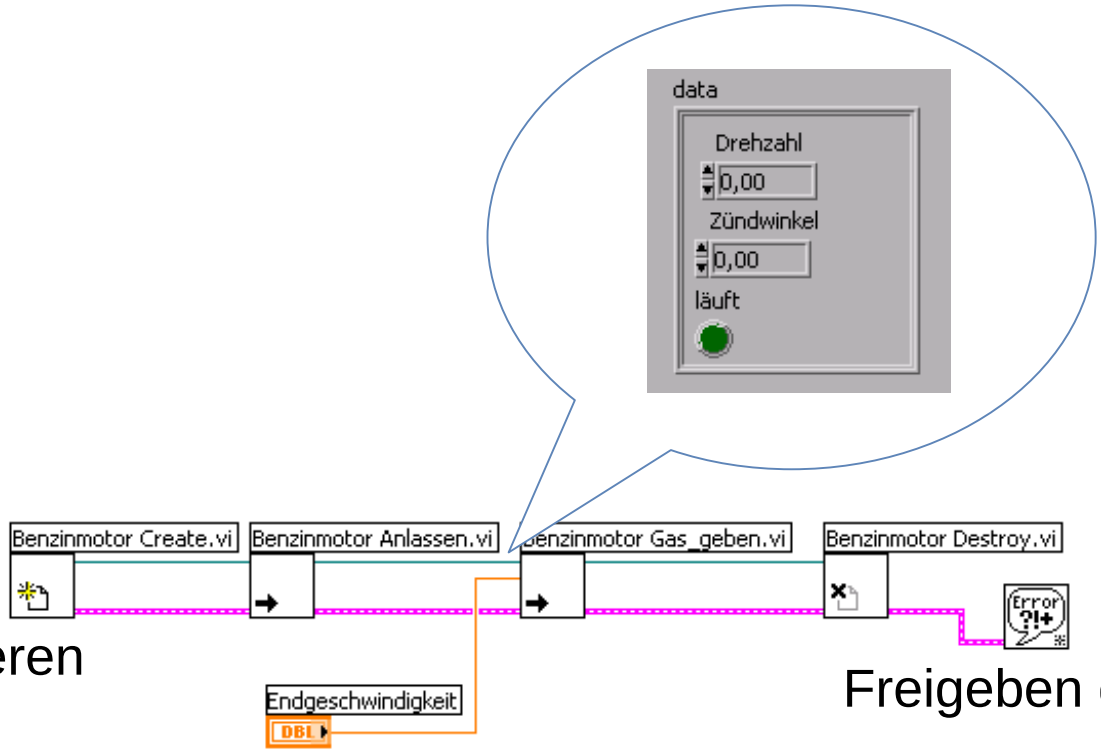
Prinzip

- Toolkit für LabVIEW (Windows, Linux, MacOS)
- Modellierung von Objekten durch Wizzard
- Erstellen von VIs als Methoden und Typdefinitionen für die Eigenschaften





Beispiel einer GOOP-Anwendung



Initialisieren

Freigeben des Objekts

Benutzen des Objekts ohne
die Daten direkt zu verwenden

Weiterführende Infos und Links

- **Kommunikationsmechanismen**
  - Moderne Betriebssysteme
    Andrew Tannenbaum
    Prentice Hall
- **OOP**
  - www.google.de
- **GOOP**
  - zone.ni.com

Design Patterns

Was sind Design-Patterns?

- Wiederkehrende Strukturen in Programmen
- Grundlagen für Programme
- Typische Lösungsansätze
- Standardisierte Lösungsansätze (z.B. Fehlerbehandlung)

"…simple and elegant solutions to specific problems in … software design. [They] capture solutions that have developed and evolved over time…"

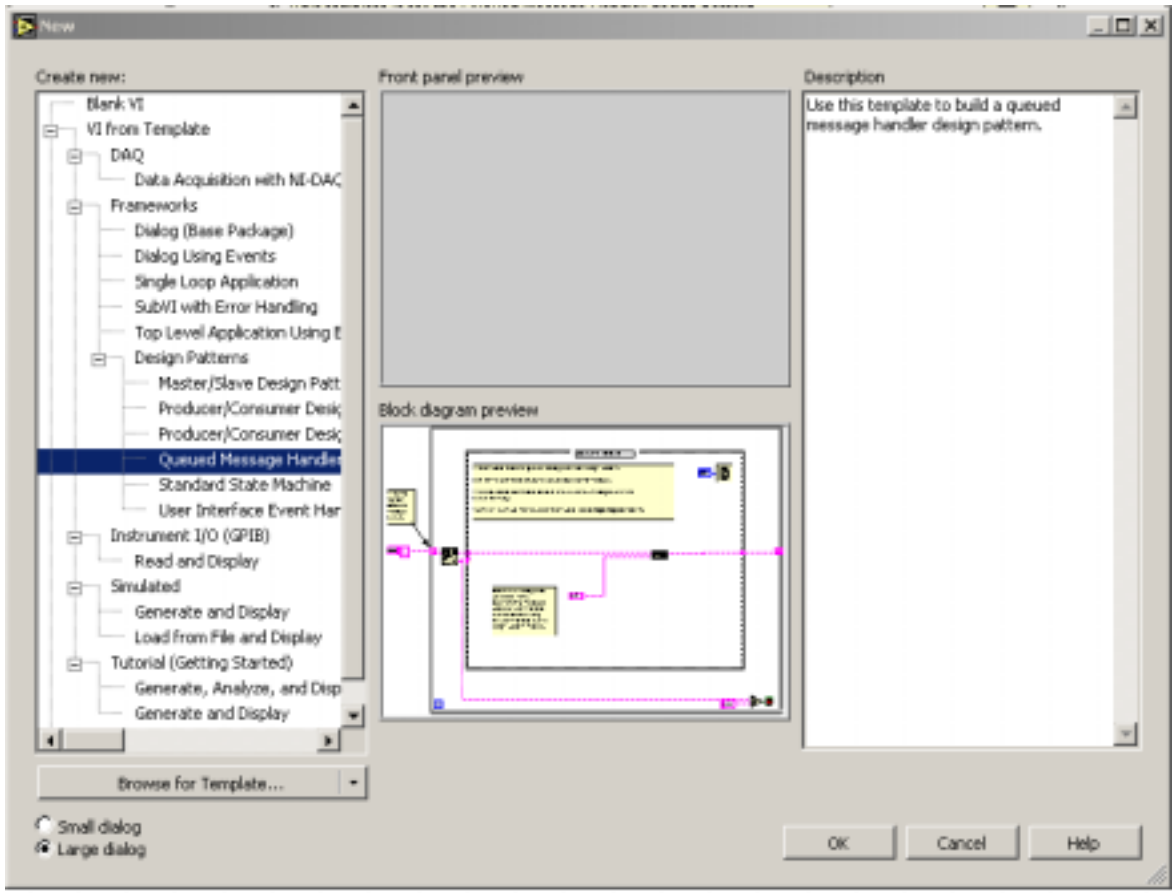
– Design Patterns, Gamma, Helm, Johnson, Vlissides

Design Patterns

Vorteile

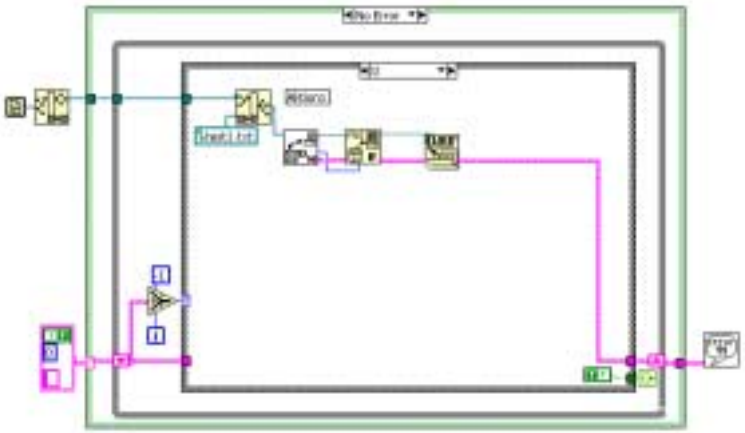
- Leichtere Einarbeitung in Code anderer Entwickler
  durch einheitliches Erscheinungsbild des Programmcodes
- Schnellere und effizientere Entwicklung durch fertige Komponenten
- Stabilere Programme durch getestete Komponenten und Designvorlagen

Templates in LabVIEW 7 Express

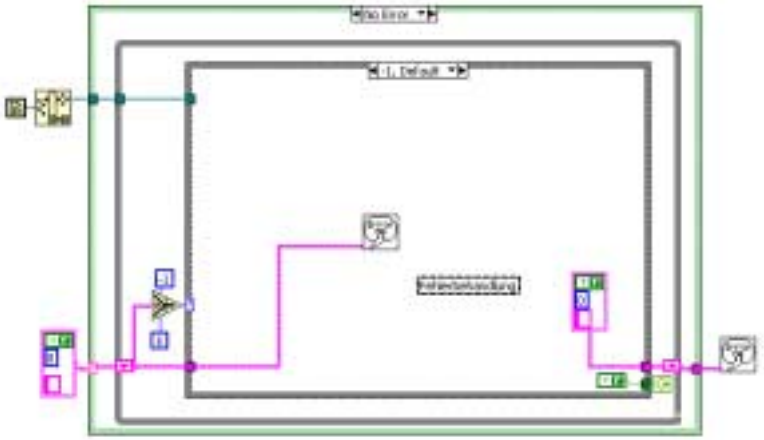


Beispiel 1: Sequenz mit Fehlerbehandlung

- LabVIEW-Sequenz kann nicht vorzeitig beendet werden
- While-Loop und Case-Struktur als Ersatz
- Abbruch durch Fehler



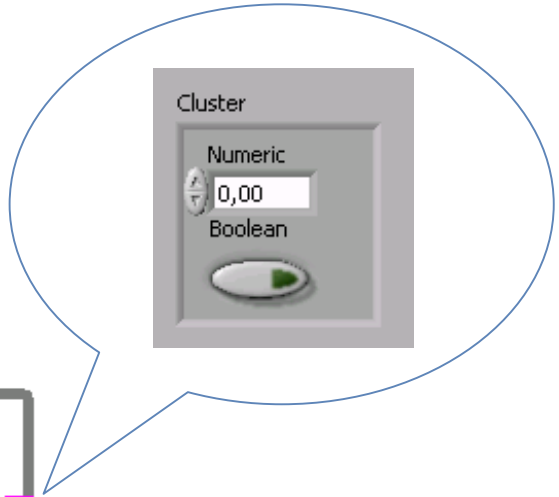
hier könnte ein Fehler auftreten

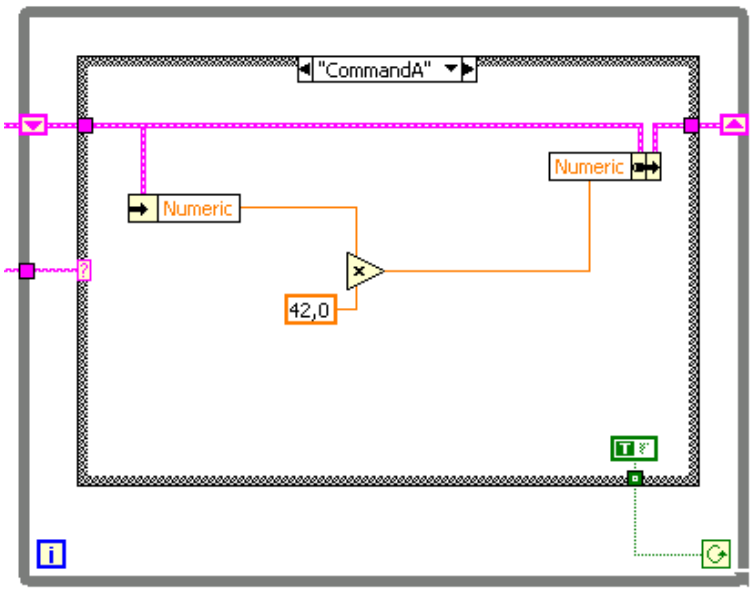


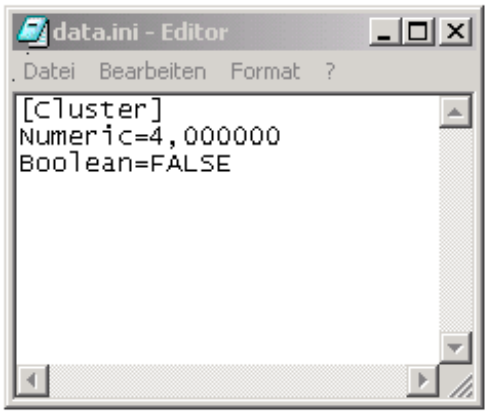
Case für die Fehlerbehandlung

Beispiel 2: Konfigurationsdaten-Verwaltung

- Laden der Programmdaten zu Beginn
- Speichern beim Ende
- Konfigurationsdaten (Sprache, Abtastrate,...)





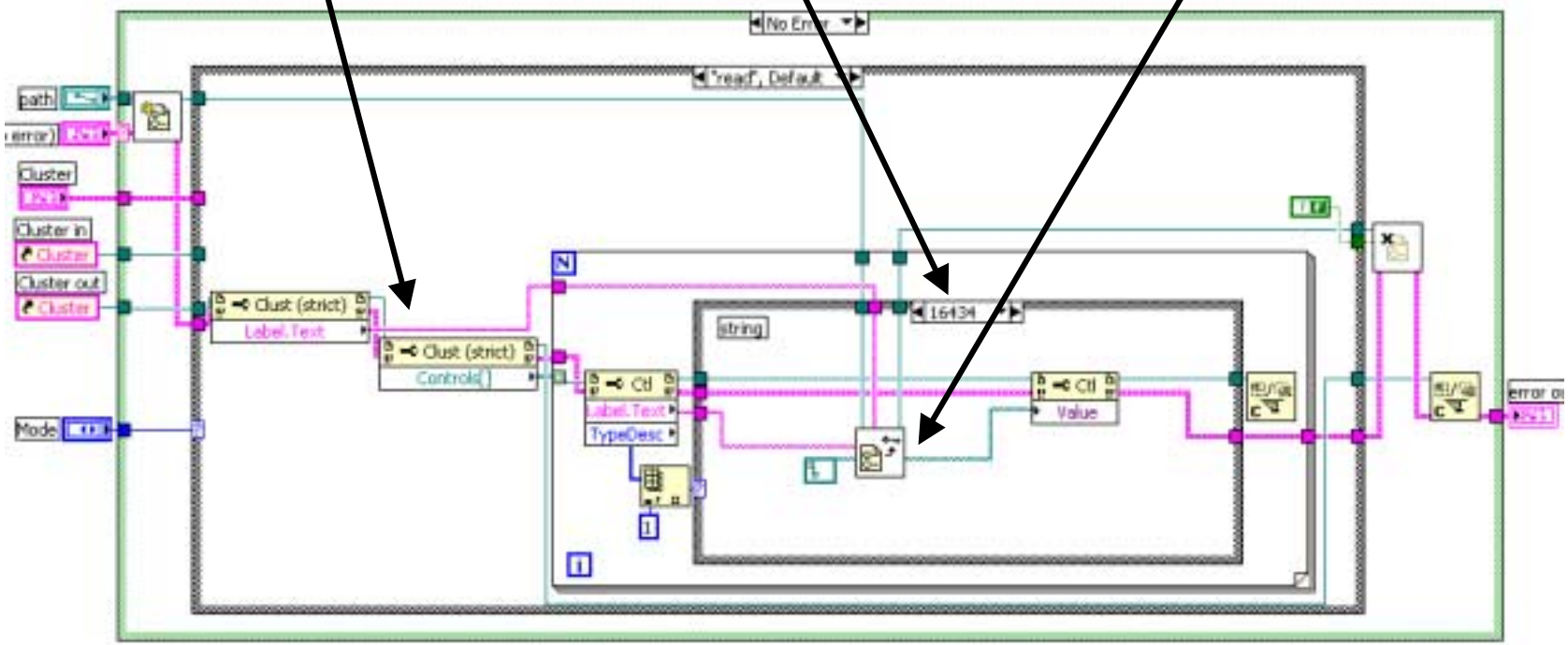


Beispiel 2: Konfigurationsdaten-Verwaltung

Zugriff auf die Cluster-Inhalte
Über Referenz (universell!)

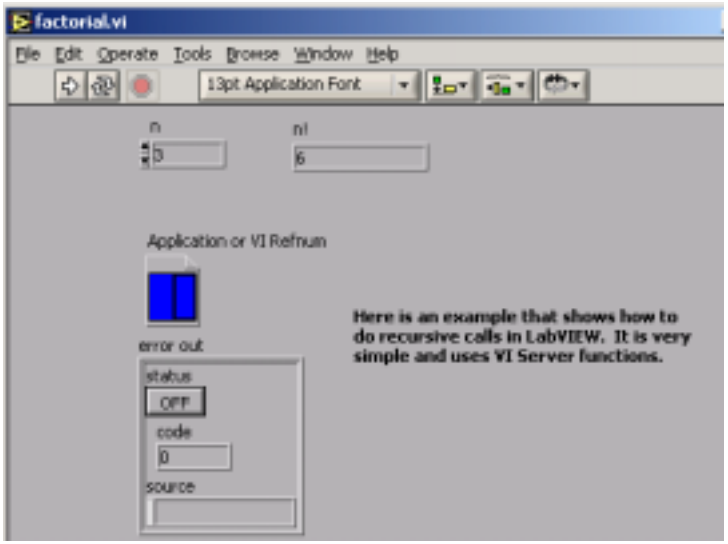
Unterscheidung
der Datentypen

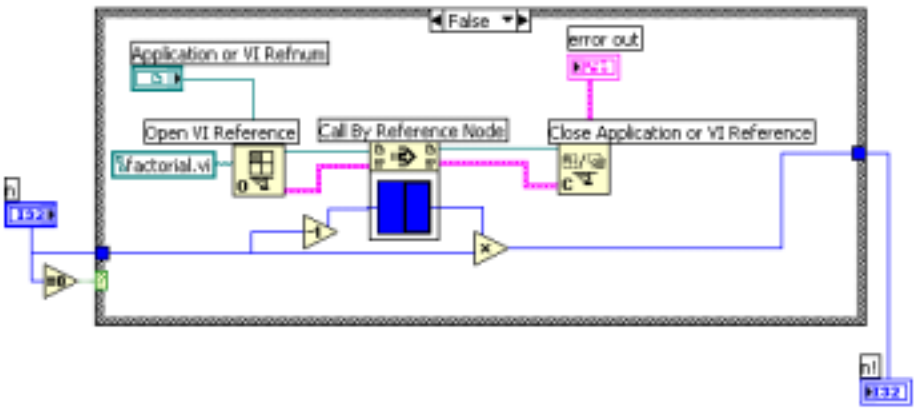
Lesen aus *.ini-Datei



Beispiel 3a: Rekursion

- Berechnen der Fakultät einer Zahl
- Nutzt VI-Server
- kein Debugging da reentrant-execution

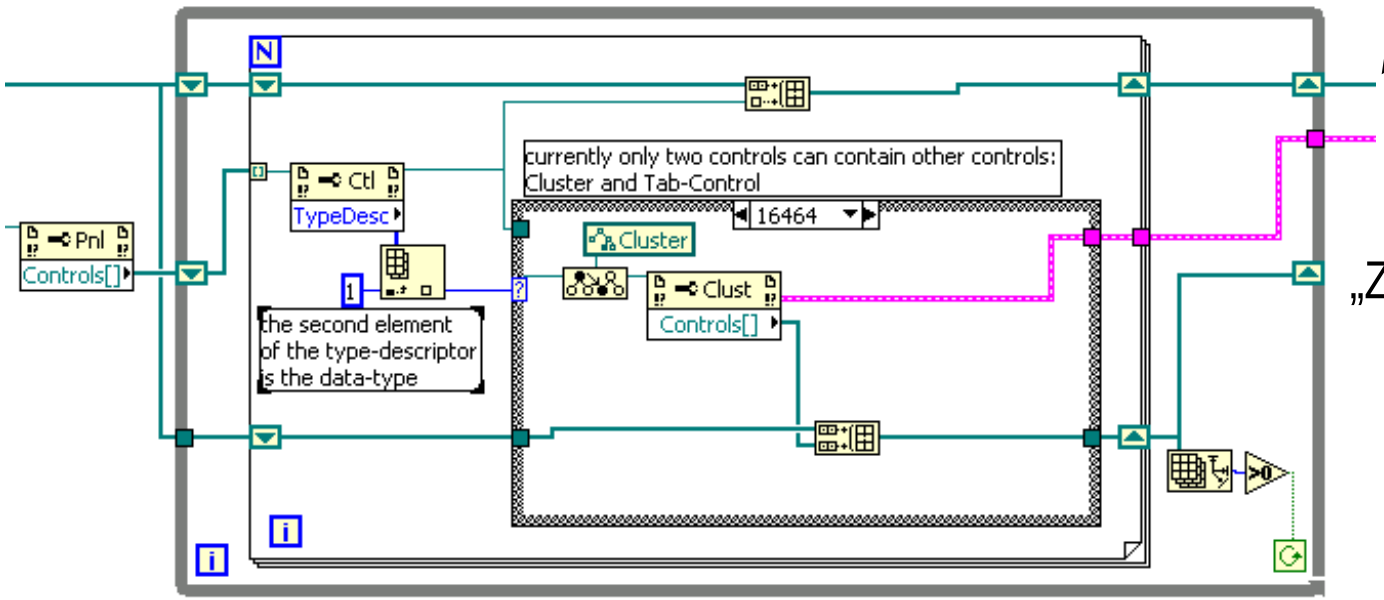




Beispiel 3b: Rekursion mit Schleifen

- Suchen in Dateibäumen
- Zugriff auf Frontpanelelemente über Referenzen
- mathematische Algorithmen
- alle rekursiven Algorithemen lassen sich mit Schleifen realisieren

Referenzen auf alle Controls des Frontpanels



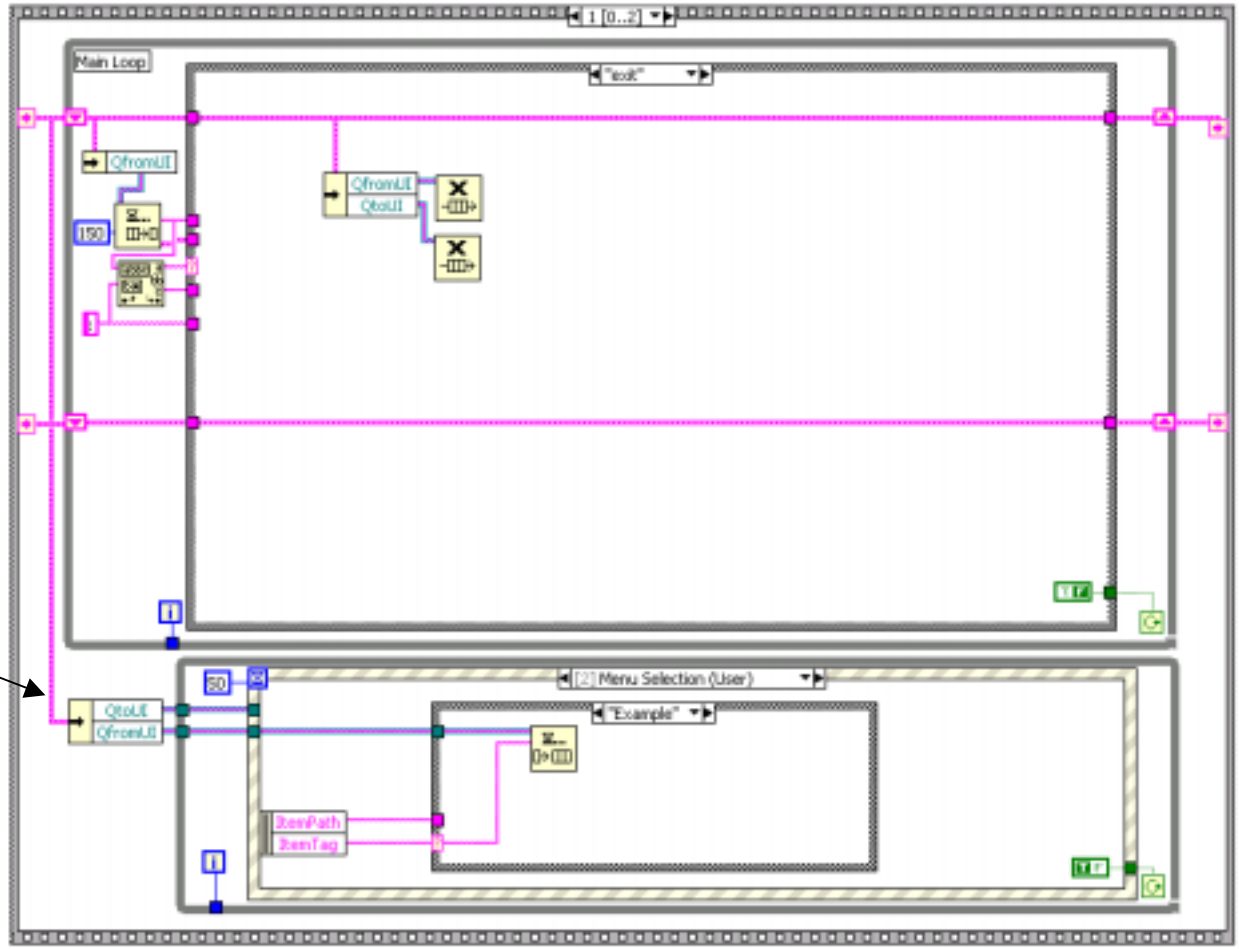
„Zwischenspeicher“

Beispiel 4: Programm-Grundgerüst

Programmkern

Verbindung nur über Queue

UI-Manager



Weiterführende Infos und Links

- LTR Publishing
- www.openg.org
- LabVIEW Hilfe: Development Guidelines
- zone.ni.com

Fragen?

Christian.hamp@ni.com