APPic 2.0

Framework

Manual

APP Instruments
Let's connect to NI LabVIEW™!
Web: www.app-instruments.com
Mail: info@app-instruments.com

Inhaltsverzeichnis

1. Einleitung

APPic ist ein mit LabVIEW realisiertes Applikations-Framework der Firma APP Systems, welches folgende Aufgaben erfüllt:

Programmarchitektur

- Standardisierung des Funktionsaufbaus
- Flexibilität für spätere Änderungen oder Erweiterungen
- Modularisierung und Kapselung
- Wiederverwendbarkeit

Funktionen

- Dynamisches Laden von Komponenten
- Entkoppelung von Programmabarbeitung und Funktionsausführung
- Datenübergabe zwischen asynchronen Prozessen
- Parallelisierung
- Kommunikation zwischen verteilten Systemen

1.1. Was ist ein Framework?

Ein Framework (engl. für „Entwicklungsrahmen“) ist ein Programmiergerüst, das in der Softwaretechnik, insbesondere im Rahmen der objektorientierten Softwareentwicklung sowie bei komponentenbasierten Entwicklungsansätzen, verwendet wird.

Ein Framework ist selbst noch kein fertiges Programm, sondern stellt den Rahmen, innerhalb dessen der Programmierer eine Anwendung erstellt, zur Verfügung, wobei u. a. durch die in dem Framework verwendeten Entwurfsmuster auch die Struktur der individuellen Anwendung beeinflusst wird. Ein Framework gibt somit in der Regel die Anwendungsarchitektur vor.

Ein Framework definiert insbesondere den Kontrollfluss der Anwendung und die Schnittstellen für die konkreten Klassen, die vom Programmierer erstellt und registriert werden müssen. Frameworks werden also i. A. mit dem Ziel einer Wiederverwendung „architektonischer Muster“ entwickelt und genutzt.

2. Aufbau

Ein LabVIEW-Programm, das auf dem APPic Framework aufbaut, ist üblicherweise in mehrere Bereiche gegliedert:

- (Benutzer-)Oberfläche
- Programmlogik
  - Ablaufsteuerung
  - Eventsteuerung
- Verschiedene Module

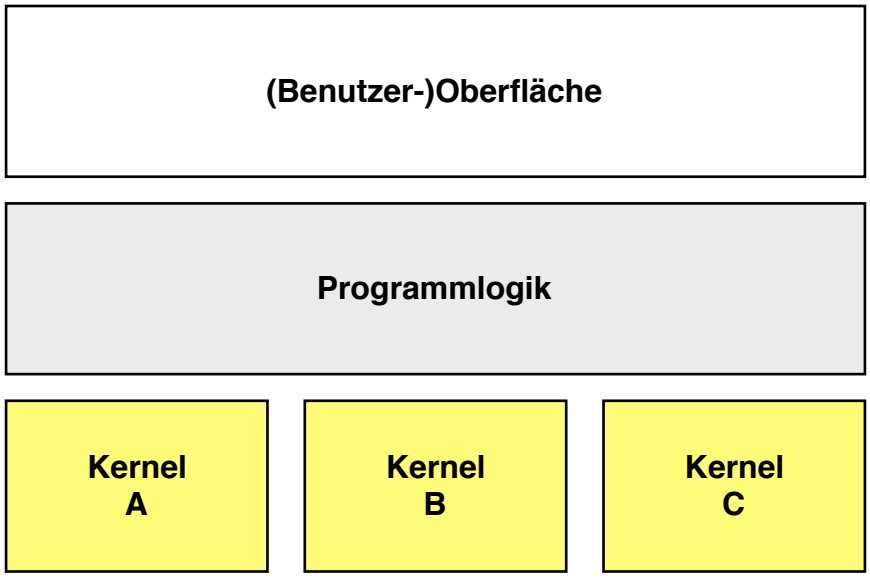

Abb. 1: Struktur

Das APPic Framework selbst besteht dabei aus den diversen Modulen (im Weiteren „Kernels“) und den notwendigen Hilfsfunktionen, um diese Kernels zu laden und zu verwenden.

Die angewandte Softwarearchitektur entspricht dem „Erzeuger-Verbraucher-Muster“, einem von vielen speziellen Software-Entwurfsmustern („*design patterns*“). Das Erzeuger-Verbraucher-Muster basiert auf dem Master/Slave-Muster und zielt darauf ab, Prozesse zu entkoppeln, die Daten mit unterschiedlicher Geschwindigkeit produzieren oder konsumieren. Umsetzungen des Erzeuger-Verbraucher-Musters in LabVIEW werden oft „*queued state machine*“ oder „*queued message handler*“ genannt.

3. Kernels

Die Kernels sind der Kern des APPic Frameworks und dienen als Funktionsbibliotheken, auf die aus dem Programmablauf heraus zugegriffen werden kann. Im Sinne des oben beschriebenen Erzeuger-Verbraucher-Musters stellen die Kernels die Verbraucher dar.

Ein einzelner Kernel beinhaltet alle notwendigen Funktionen für einen Anwendungszweck, zum Beispiel ein zu steuerndes Gerät (Netzgeräte, AIO- oder DIO-Karten etc.) oder eine Funktion (Datenbankschnittstelle, Logger etc.).

3.1. Blockschaltbild

Das Blockschaltbild eines Kernels besteht aus einer Case-Struktur, die alle implementierten Funktionen des Kernels enthält:

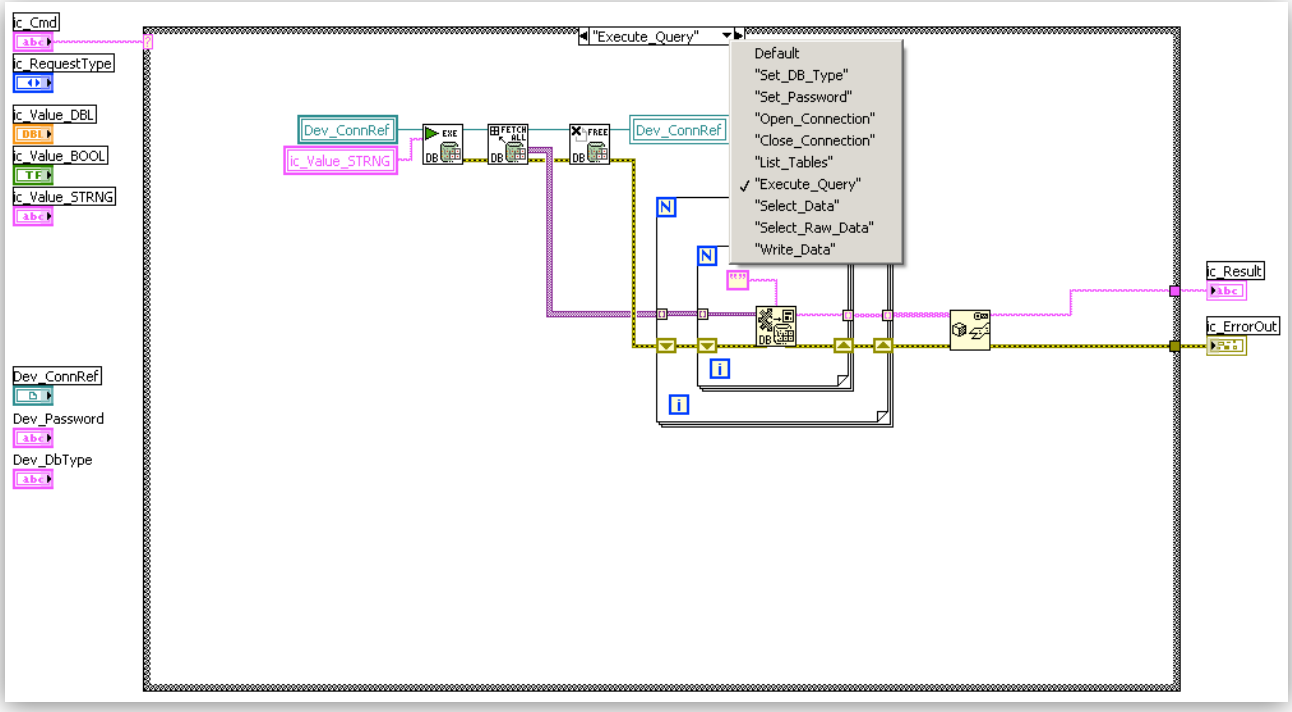

Abb. 2: Kernel Blockschaltbild

3.2. Parameter

Ein Kernel kann nur einen Übergabeparameter (formal vom Typ `String`, `Numerical` oder `Boolean`) sowie einen Ergebnisparameter (formal vom Typ `String`) haben.

Über die Funktionen „*Flatten to String*“ und „*Unflatten from String*“ können aber beliebige Datentypen und -strukturen als String übergeben werden.

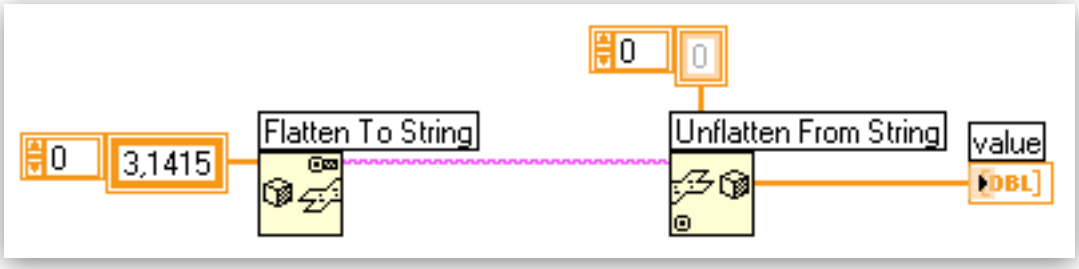

Abb. 3: Flatten und Unflatten

3.3. Initialisierung

Kernels werden bei Programmstart über eine Registrierungsdatei durch `APPic_FrameworkLoader.vi` dynamisch geladen und parametriert. Die Ausführung erfolgt parallel zum Hauptprogramm und beginnt, sobald `APPic_FrameworkSyncer.vi` Version und Status des Ladevorganges zurückgemeldet hat.

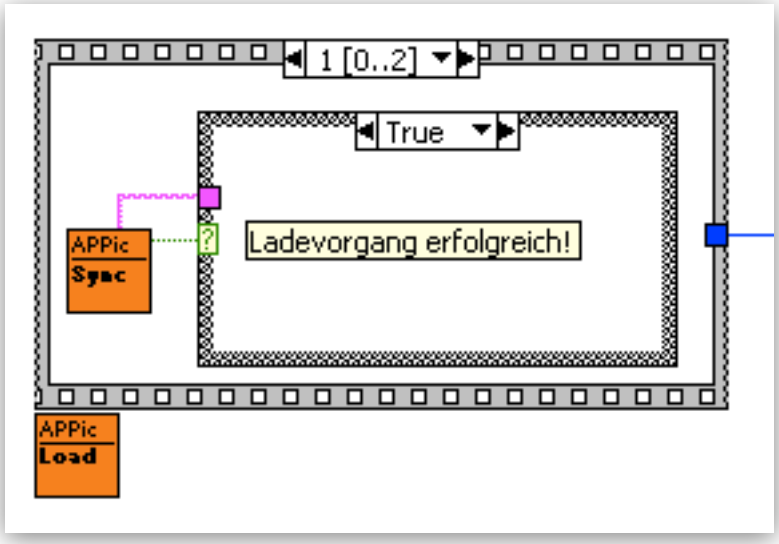

Abb. 4: Initialisierung

3.4. Deinitialisierung

Bei Programmende ist es notwendig, die geladenen Kernels durch die Funktion `APPic_Unloader.vi` zu beenden und aus dem Speicher zu entfernen.

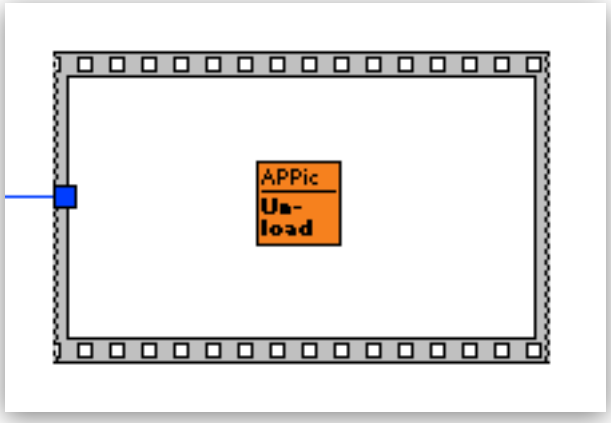

Abb. 5: Deinitialisierung

4. Verwendung

4.1. Programmaufbau

Der Rumpf eines Projektes, das auf dem APPic Framework aufbaut, sieht eine Einteilung in drei Programmabschnitte vor, die dem tatsächlichen Lebenszyklus eines Programmes nachempfunden sind:

Initialisierung (Sequenz, links)

Bei Programmstart werden zuerst das Framework und alle registrierten Kernel geladen. Danach können nach Bedarf projektspezifische Initialisierungsroutinen (Laden von Treibern, Starten von Geräten etc.) platziert werden.

Programmablauf (Schleife, mitte)

Im mittleren Abschnitt wird das eigentliche Programm umgesetzt.

Deinitialisierung (Sequenz, rechts)

Nach Beendigung des eigentlichen Programms werden im Deinit-Abschnitt sämtliche Verbindungen geschlossen, Speicherbereiche freigegeben und schlussendlich das Framework selbst kontrolliert beendet.

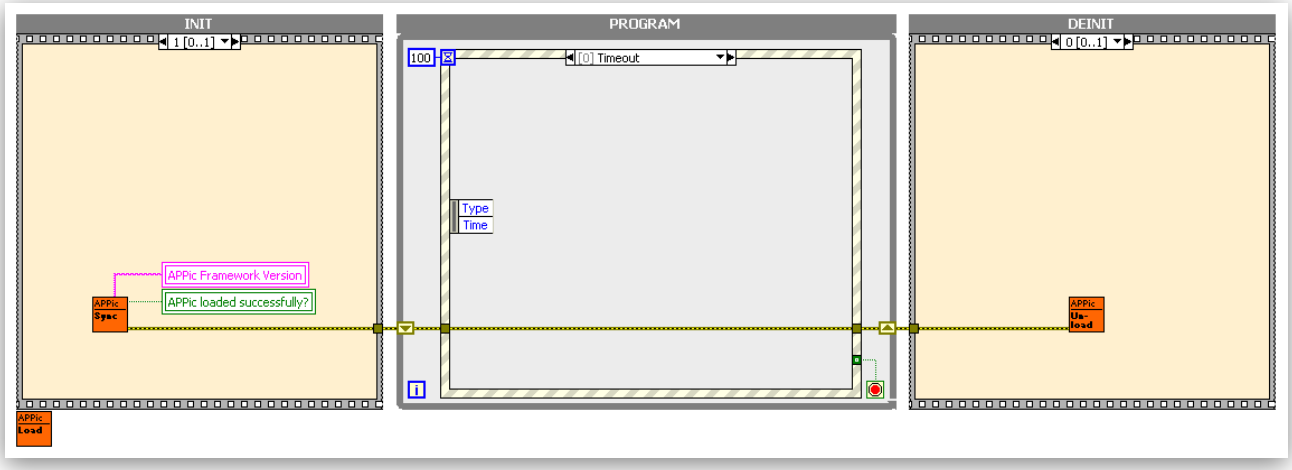

Abb. 6: APPic Projektrumpf

4.2. Hilfsfunktionen

Die APPic Hilfsfunktions-VIs übernehmen die Aufgabe, Eingangsparameter an eine bestimmte Funktion eines Kernels zu senden, und Ergebnisse der aufgerufenen Funktion in den Ausgangsparametern zur Verfügung zu stellen. Im Sinne des oben beschriebenen Erzeuger-Verbraucher-Musters stellen die Hilfsfunktionen die Erzeuger dar.

Abhängig von der Art der Funktion wird im APPic Framework nur zwischen ändernden (*Setter*) und abfragenden (*Getter*) Aufrufen unterschieden.

4.3. Änderungsfunktion: APPic Set

Setter-Aufrufe dienen grundsätzlich dem Zweck, Daten zu schreiben oder Einstellungen zu setzen.

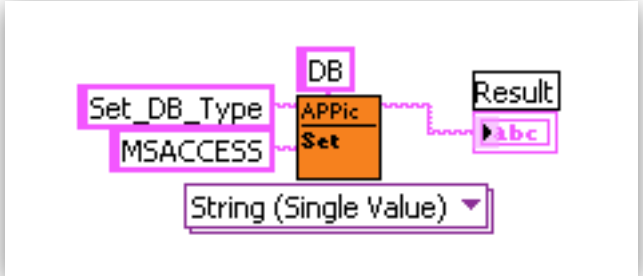

Abb. 7: APPic Set

Übergabeparameter

- `string Kernel:` Kernel-Name
- `string Command:` Funktionsname
- `string Parameter:` Übergabeparameter
- `string Error In:` Fehlereingang

Ergebnisparameter

- `string Result:` Rückgabeparameter
- `cluster Error Out:` Fehlerausgang

4.4. Abfragefunktion: APPic Get

Von *Getter*-Aufrufen spricht man, wenn der grundlegende Zweck des Aufrufs das Rücklesen von Daten ist.

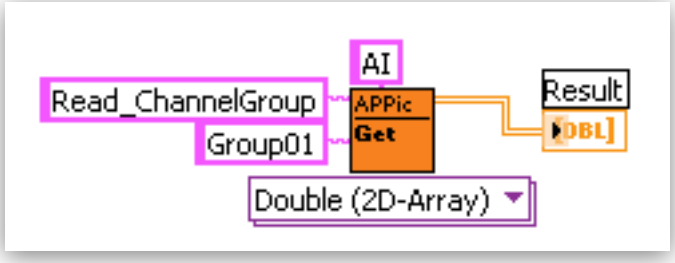

Abb. 8: APPic Get

Übergabeparameter

- `string Kernel:` Kernel-Name
- `string Command:` Funktionsname
- `string Parameter:` Übergabeparameter
- `(polymorph) Result Data Type:` Datentyp des Rückgabeparameters
- `string Error In:` Fehlereingang

Ergebnisparameter

- `(polymorph) Result:` Rückgabeparameter
- `cluster Error Out:` Fehlerausgang

4.5. Polymorphismus

Die Hilfsfunktionen sind polymorph, d.h. als Sammlung von VIs mit identischen *Connector Pane Patterns*, aber unterschiedlichen Datentypen, ausgeführt.

Derartige VIs können verschiedene Datentypen für Ein- bzw. Ausgangsparameter verarbeiten. Je nachdem, welcher Datentyp am polymorphen Eingang anliegt, wird das jeweilige VI aus der Sammlung instanziert.

Beispiel: Liegt am polymorphen Eingang eine `String`-Variable an, wird im Hintergrund das entsprechende VI für die Verarbeitung von Zeichenketten geladen. Liegt am selben Eingang des selben VIs eine `Numerical` Variable an, wird im Hintergrund das entsprechende VI für die Verarbeitung von numerischen Werten geladen. Dieser Vorgang geschieht transparent (nicht sichtbar für den Benutzer).

Alternativ kann der Datentyp (und damit die zu verwendende Instanz des polymorphen VIs) auch über das Kontextmenü des VIs gewählt werden.

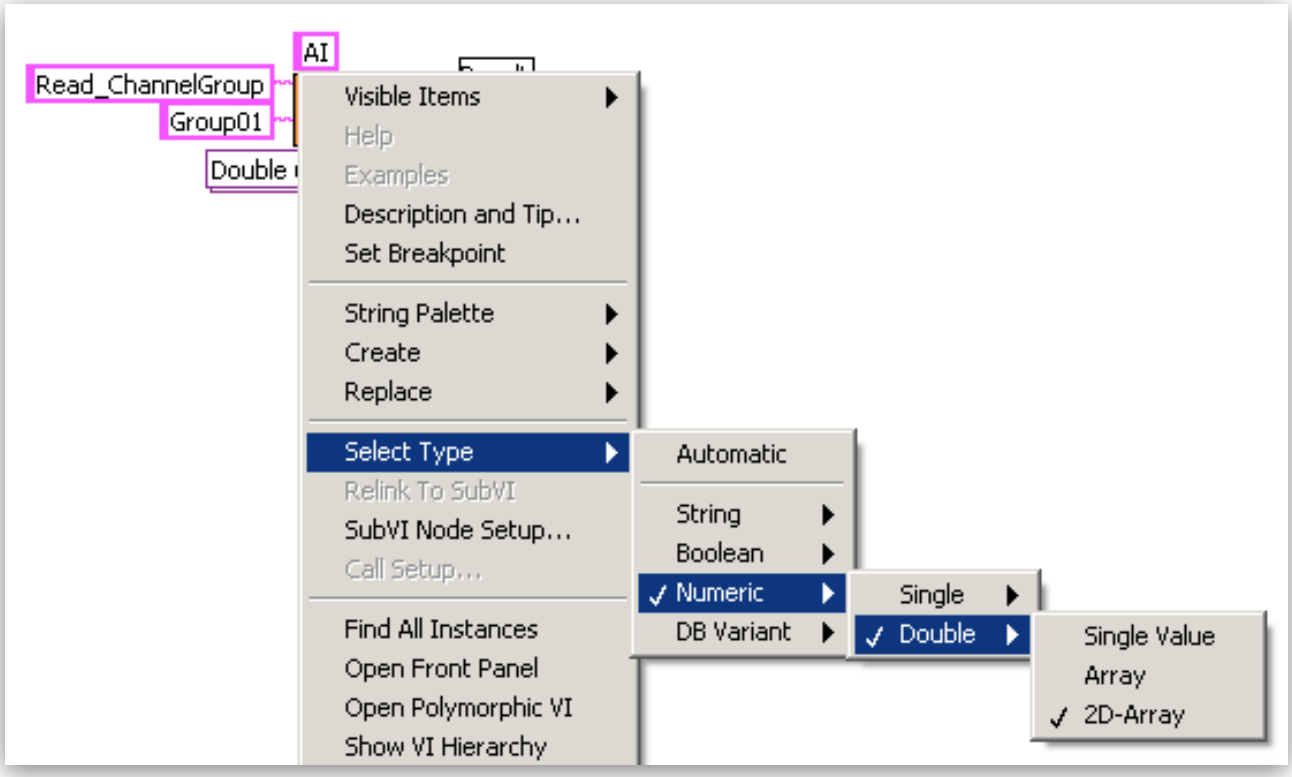

Abb. 9: Auswahl des Polymorphismus-Typs

4.6. Wrapper-Hilfsfunktionen

Da die APPic Hilfsfunktions-VIs nur einen einzelnen Übergabe- bzw. Ergebnisparameter bedienen können, bieten einige Kernels erweiterte Hilfsfunktions-VIs an, welche eine größere Anzahl an Übergabe- oder Ergebnisparametern oder zusätzliche Funktionalität zur Verfügung stellen.

Diese VIs „umschließen“ die eigentlichen Hilfsfunktions-VIs und werden daher auch *Wrapper*-VIs genannt. Sie übernehmen die Zusammenfassung zu einem einzelnen Übergabeparameter oder die Aufteilung in mehrere Ergebnisparameter, und werden anstelle der *Getter*- und *Setter*-Funktionen verwendet.

Beispiel 1: Der *Datenbank-Kernel* bietet ein Wrapper-VI „Select Data“, das die vielen notwendigen Parameter für eine Abfrage als Konnektoren zur Verfügung stellt und diese zusammengefasst an das APPic Get-VI weiterreicht.

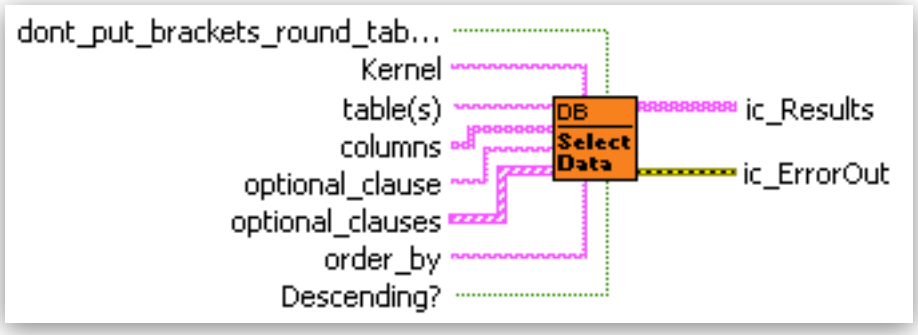

Abb. 10: DB Select Data

Beispiel 2: Der *Variablen-Kernel* bietet ein Wrapper-VI „VAR Read“, welches neben dem eigentlichen Funktionsaufruf - dem Auslesen einer Variablen - zusätzlich die Angabe eines Timeouts für den Lesevorgang erlaubt.

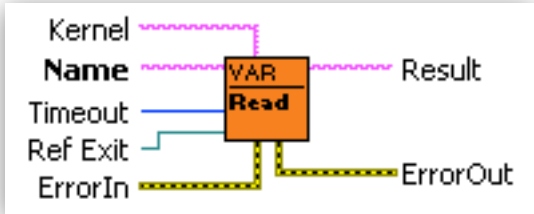

Abb. 11: VAR Read

4.7. Farbkodierung

Framework-Funktions-VIs

Nur eine kleine Auswahl der Framework-Dateien ist für die direkte Verwendung im Blockschaltbild durch den Programmierer vorgesehen. Diese VIs sind durch eine orange Einfärbung gekennzeichnet:

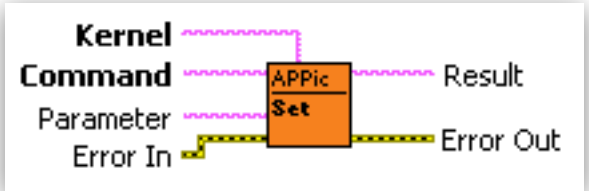

Abb. 12: Funktions-VI „APPic_Set.vi“

Framework-interne VIs

Alle Framework-internen VIs, die nur durch das Framework genutzt werden dürfen, sind durch eine rote Einfärbung und ein Hand-Symbol gekennzeichnet:

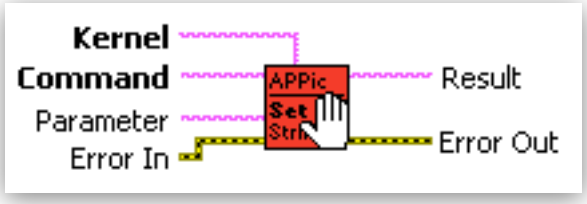

Abb. 13: Framework-internes VI „APPic_Set_STR.vi“

Sonstige VIs

Alle anderen Sub-VIs, die von APP Systems in einem Projekt angelegt werden, erhalten eine dunkelblaue Einfärbung:

Abb. 14: Sub-VI „APP_Init.vi“

5. Kommunikation

Das APPic Framework stellt alle notwendigen Mechanismen zur Verfügung, um über eine standardisierte Schnittstelle auf die bei Programmstart geladenen Kernels zuzugreifen.

Die APPic Hilfsfunktionen arbeiten dabei nach dem Prinzip, eine Anforderung an den Kernel zu senden, der diese Anforderung verarbeitet und das Ergebnis an die Hilfsfunktion zurückgibt. Die Hilfsfunktion hält in ihrer Abarbeitung so lange an, bis das Ergebnis verfügbar oder das voreingestellte Zeitlimit erreicht ist.

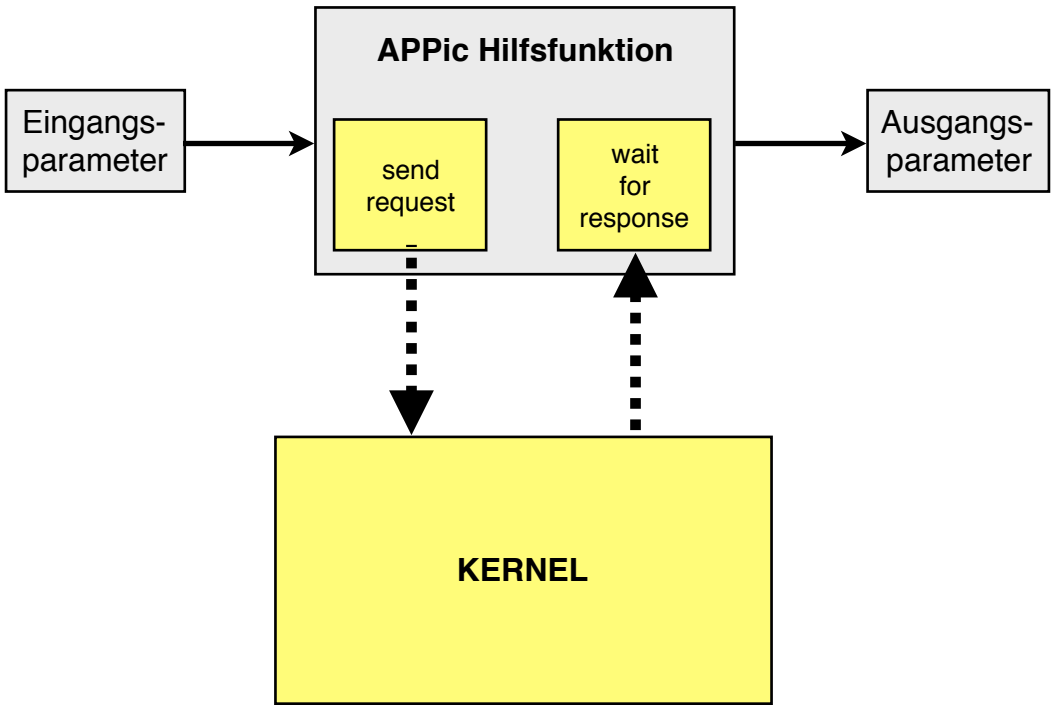

Abb. 15: Kommunikation

Die Kommunikation zwischen Programm und Kernel erfolgt über dedizierte Queues. Diese Queues bewirken eine „Entkoppelung“ und ermöglichen:

- die parallele Abarbeitung von Aufrufen verschiedener Kernels
- die serielle Abarbeitung der Aufrufe eines Kernels
- eine kontrollierte Abarbeitungsdauer

5.1. Vernetzung

Da die gesamte Kommunikation mit einem Kernel gebündelt durch die oben beschriebenen dedizierten Queues erfolgt, ist es auf einfache Art und Weise möglich, die Kommunikation auf Kernels anderer Geräte „umzuleiten“. Dadurch kann das betreffende Programm („*Host*“) über eine herkömmliche TCP/IP-Netzwerkverbindung auf Kernelfunktionen eines anderen im selben Subnetz laufenden Gerätes („*Target*“) zugreifen.

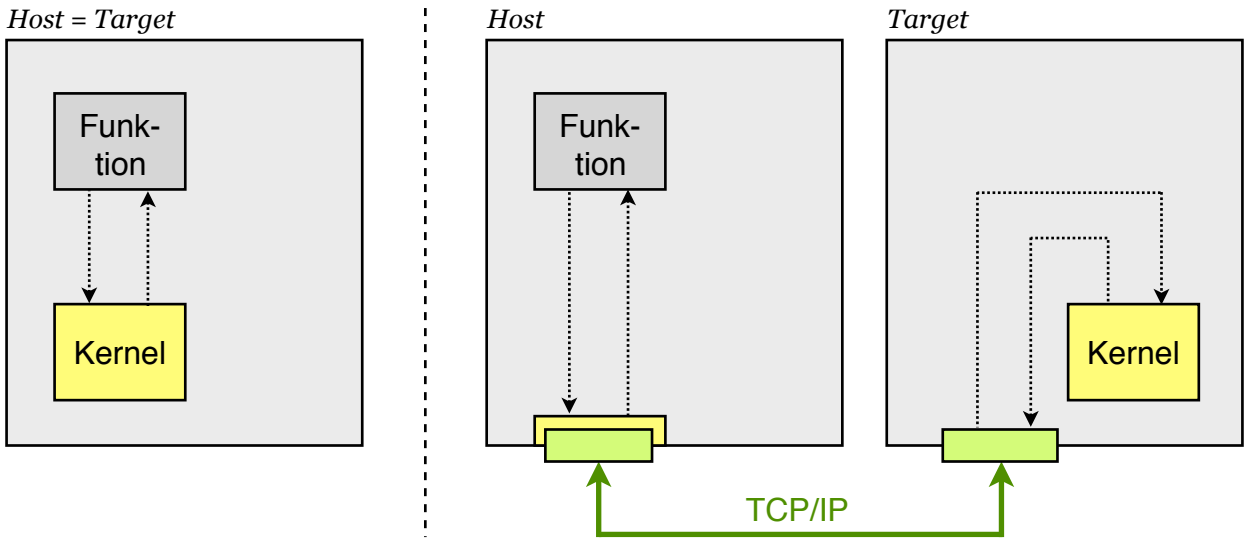

Abb. 16: Vernetzung

Transparenz für die aufrufende Funktion

Für die aufrufende Funktion ist nicht ersichtlich, ob der Kernel, an den die Anfrage gesendet wird, lokal am Host oder entfernt am Target läuft. Dadurch muss bei der Erstellung des Programmes hinsichtlich der Framework-internen Kommunikation nicht darauf acht gelegt werden, wo ein Kernel laufen wird.

Transparenz für den Kernel

Für den Kernel ist nicht ersichtlich, ob eine Anfrage von einem lokalen oder einem entfernten Funktionsaufruf stammt. Dadurch muss bei der Erstellung des Kernels hinsichtlich der Framework-internen Kommunikation nicht darauf acht gelegt werden, wo ein Programm laufen wird.

Verwendung

Um die Kernels eines Programmes im Netzwerk freizugeben, muss am Target die Funktion `APPic_NetDXLoader.vi` parallel zum restlichen Programmablauf (vgl. `APPic_FrameworkLoader.vi`) eingebunden werden, die den Serverdienst startet.

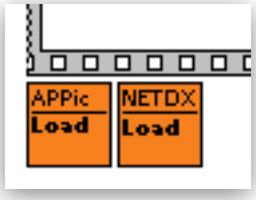

Abb. 17: NetDX Loader

Bei Programmende ist es notwendig, den zuvor gestarteten NetDX Serverdienst durch die Funktion `APPic_NetDXUnloader.vi` zu beenden.

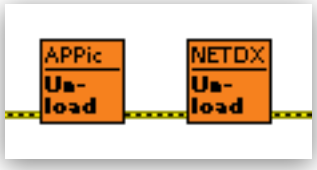

Abb. 18: NetDX Unloader

6. NetDX

Der NetDX-Kernel stellt die für eine Netzwerkkommunikation (siehe 5.1) notwendigen Mechanismen zur Verfügung.

Der Client (Host) sendet eine Anfrage an den Server (Target) und erhält als Reaktion darauf eine Antwort. Diese Nachrichten werden via TCP als Datenstrom (*byte stream*) versendet. Die Kommunikation erfolgt asynchron über ein proprietäres, zustandsbehaftetes Protokoll.

Der Server wartet auf Port **33333** auf eingehende Verbindungen. Die komplette Übermittlungsdauer einer Nachricht darf nicht mehr als 500ms betragen.

6.1. Anfrage (Request)

Eine Anfrage besteht aus:

- Länge des Datenblocks (LEN): Signed long int
- Kodierung (COD): Unsigned byte (default: 0)
- Aufruftyp (METHOD): Unsigned int (APPic Get = 0, APPic Set = 1)
- Datenblock (DATA): String

LEN	COD	METHOD	DATA
4 bytes	1 byte	2 bytes	LEN bytes

Datenblock

Innerhalb des Datenblocks werden mehrere Informationen *serialisiert* übergeben. Als Trennzeichen der einzelnen Informationsblöcke dient das Dollarzeichen (ASCII-Code `0x24`).

- Kernel-Name (KERN): String
- Funktionsname (CMD): String
- Parameter (PARAM): String

KERN		CMD		PARAM
`string`	„$“	`string`	„$“	`string`

6.2. Antwort (Response)

Eine Antwort besteht aus:

- Länge des Datenblocks (LEN): Signed long int
- Kodierung (COD): Unsigned byte (default: 0)
- Datenblock (DATA): Cluster mit variabler Länge

LEN	COD	DATA
4 bytes	1 byte	LEN bytes

Datenblock

Der Datenblock der Antwort besteht aus einem Cluster (ein Cluster ist das LabVIEW-Pendant zu *records* oder *structs*). In diesem Antwort-Cluster werden das Ergebnis (der Rückgabewert) der aufgerufenen Funktion sowie ein Fehler-Cluster übergeben.

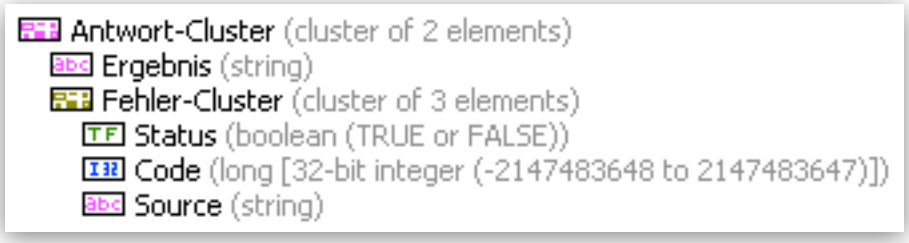

Abb. 19: Datenstruktur des Clusters im Datenblock der Antwort

- Antwort-Cluster
  - Länge von RES (LEN1): Signed long int
  - Ergebnis (RESULT): String
  - Fehler-Cluster
    - Fehlerstatus (STAT): Boolean
    - Fehlercode (COD): Signed long int
    - Länge SRC (LEN2): Signed long int
    - Fehlerquelle (SOURCE): String

LEN1	RESULT	STAT	COD	LEN2	SOURCE
4 byte	LEN1 bytes	1 byte	4 byte	4 byte	LEN2 bytes

Siehe auch Kapitel 6.4.

6.3. Beispiel

Anfrage

Ein Client sendet in Klartext an den Kernel `DIO` das Kommando `SetChannel` mit dem Parameter `Ch1`:

```
DIO$SetChannel$Ch1
```

Der so gebildete Datenblock hat inklusive der beiden Dollarzeichen eine Länge von 18 (0x12) Zeichen. Die gesamte Anfragenachricht (Länge des Datenblocks, Kodierung, Methode und Datenblock) als *byte stream* lautet daher:

```
0000 0012 0000 0144 494F 2453 6574 4368 616E 6E65 6C24 4368 31
```

Antwort

Der Server antwortet in Klartext mit dem Ergebnisstring „ACK“ (Zeichenlänge = 3) und einem leeren Fehler-Cluster. Der Datenblock (Länge des Ergebnisstrings, Ergebnis, Status, Code, Länge des Source-Strings und Source) als *byte stream* lautet daher:

```
0000 0003 4143 4B00 0000 0000 0000 0000
```

Die gesamte Antwortnachricht (Länge des Datenblocks, Kodierung und Datenblock) als *byte stream* lautet:

```
0000 0010 0000 0000 0341 434B 0000 0000 0000 0000 00
```

7. Kernels

7.1. GLOBALS

Der GLOBALS-Kernel stellt eine Alternative zu globalen Variablen oder Funktionsglobalen dar und bietet darüber hinaus den Vorteil, dass auch eine Verwendung im Netzwerk (gleiche Variablen auf verschiedenen Systemen) ohne zusätzlichen Aufwand realisiert werden kann.

Der GLOBALS-Kernel wird im Blockschaltbild über einen definierten Namen (zB ‚GLOBALS-ABC') angesprochen setzt sich aus dem eigentlichen Kernel selbst (`GLOBALS\GLOBALS.vi`) und einem projektspezifischen Container für die Variablen (zB `GLOBALS\GLOBALS_ABC.vi`) zusammen.

Setzen von Variablen

Variablenwerte werden durch die Hilfsfunktion `APPic_Set.vi` (siehe 4.1.1) zugewiesen:

Abb. 20: Setzen von „variable" auf „wert"

Lesen von Variablen

Variablenwerte werden durch die Hilfsfunktion `APPic_Get.vi` (siehe 4.1.2) zurückgelesen:

Abb. 21: Lesen von „variable"

Definition der Variablen

Die Variablen werden in der Datei `GLOBALS\GLOBALS_ABC.vi` (der Suffix `_ABC` entspricht dem Namen, mit dem der Kernel im Blockschaltbild angesprochen wird) definiert, indem der äußeren *case*-Struktur ein Rahmen mit dem jeweiligen Variablennamen als *case selector label* hinzugefügt wird:

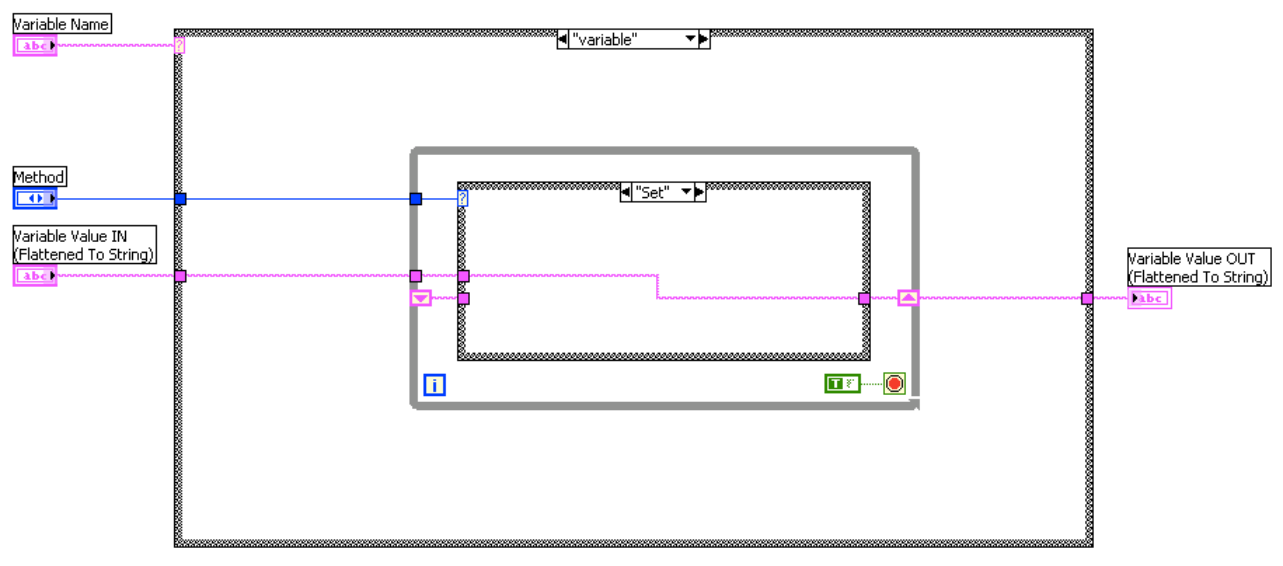

Abb. 22: Definition von „variable“ im projektspezifischen VI

Die einzelnen Cases sind identisch und beinhalten je eine funktionale globale Variable (oft auch LabVIEW-2-Variable genannt). Daher lassen sich neue Variablen durch simples Duplizieren eines vorhandenen Cases erzeugen.

ANHANG A: Dateiliste

```
/APPic
      /Framework
            /System
                  APPic_Decoder.vi
                  APPic_Events.vi
                  APPic_Get_BOOL.vi
                  APPic_Get_BOOL_ARR.vi
                  APPic_Get_BOOL_ARR2D.vi
                  APPic_Get_NUM-DBL.vi
                  APPic_Get_NUM-DBL_ARR.vi
                  APPic_Get_NUM-DBL_ARR2D.vi
                  APPic_Get_NUM-SGL.vi
                  APPic_Get_NUM-SGL_ARR.vi
                  APPic_Get_NUM-SGL_ARR2D.vi
                  APPic_Get_STR.vi
                  APPic_Get_STR_ARR.vi
                  APPic_Get_STR_ARR2D.vi
                  APPic_Get_VAR-DB_ARR2D.vi
                  APPic_GetQueueRef.vi
                  APPic_Globals.vi
                  APPic_Kernel.vi
                  APPic_Logger.vi
                  APPic_QDataType.ctl
                  APPic_RequestTypes.ctl
                  APPic_Set_BOOL.vi
                  APPic_Set_BOOL_ARR.vi
                  APPic_Set_BOOL_ARR2D.vi
                  APPic_Set_NUM-DBL.vi
                  APPic_Set_NUM-DBL_ARR.vi
                  APPic_Set_NUM-DBL_ARR2D.vi
                  APPic_Set_NUM-SGL.vi
                  APPic_Set_NUM-SGL_ARR.vi
                  APPic_Set_NUM-SGL_ARR2D.vi
                  APPic_Set_REF-ARR.vi
                  APPic_Set_REF-WAVE.vi
                  APPic_Set_STR.vi
                  APPic_Set_STR_ARR.vi
                  APPic_Set_STR_ARR2D.vi
```

Fortsetzung auf nächster Seite

Fortsetzung:

```
/APPic
      /Framework
            /System
                  APPic_Timeout_GetStatus.vi
                  APPic_Timeout_GetValue.vi
                  APPic_Timeout_SetStatus.vi
                  APPic_Timeout_SetValue.vi
                  APPic_Translate_GetValue.vi
                  APPic_TranslateControls.vi
                  APPic_TranslatePanel.vi
                  APPic_WaitForAckn.vi

/APPic
      /Framework
            /NetDX
                  APPic_NetDX.vi
                  APPic_NetDX_Globals.vi
                  APPic_NetDX_Server.vi

/APPic
      /Framework
            APPic_FrameworkLoader.vi
            APPic_FrameworkSyncer.vi
            APPic_FrameworkUnloader.vi
            APPic_Get.vi
            APPic_NetDXLoader.vi
            APPic_NetDXUnloader.vi
            APPic_Set.vi
```

APPic
Framework
Manual v2.0 (Rev: 06)

APP Instruments
Let’s connect to NI LabVIEW™!
Web: www.app-instruments.com
Mail: info@app-instruments.com