# APPic 2.0

## Framework

Manual

APP INSTRUMENTS

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

# 5. Communication

The APPic Framework provides all necessary means for accessing all kernels that are loaded at program start over a standardized interface.

The APPic helper functions use the method of sending a request to the kernel, which then processes this request and returns the results to the calling helper function. The helper function stops execution after sending request until the result is available or the defined timeout has been reached.

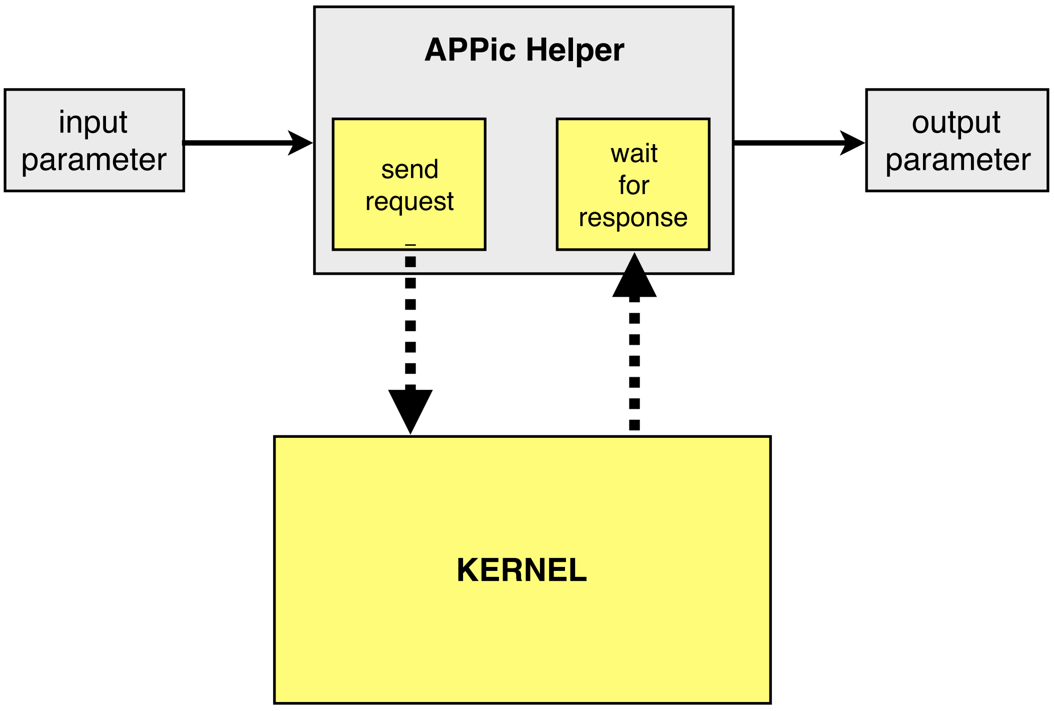


Abb. 15: Communication

Application and kernel communicate via dedicated queues. These queues effect a decoupling of a request and its processing and therefor enable:

- parallel execution of requests to different kernels
- serial execution of requests to a single kernel
- control of execution duration

## 5.1. Networking

Since all the communication with the kernels is bundled by queues, it is easily possible to “redirect” the communication to kernels running on other devices. In this way one program running on one device (”host”) can access the kernel functions of another program running on a different device (”target”) completely transparently with a conventional TCP/IP network connection.

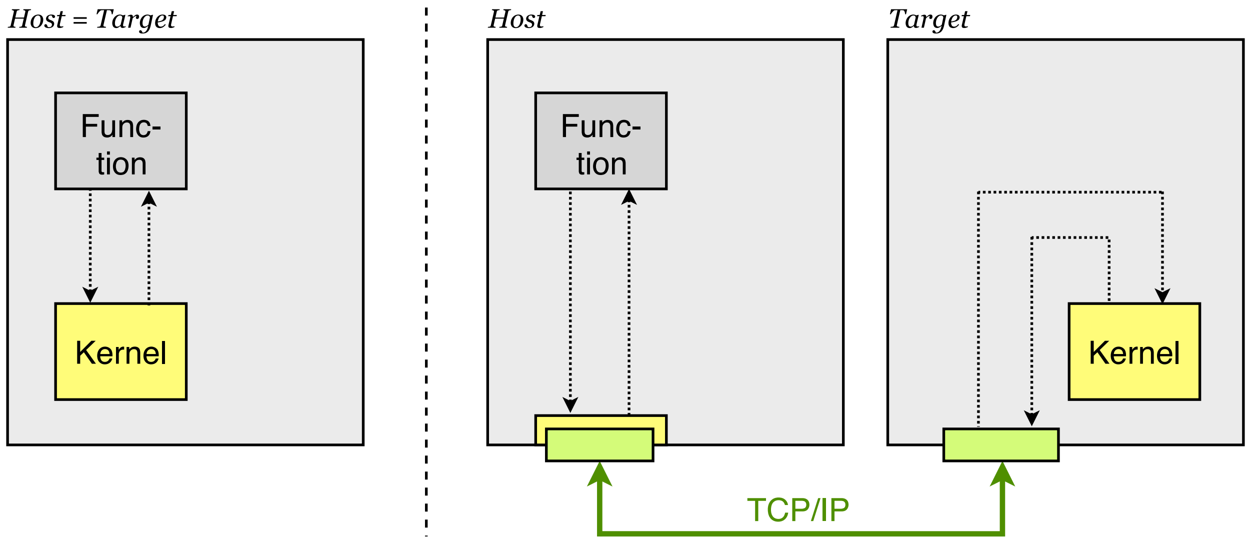


Abb. 16: Networking

### Transparency for the caller

For the calling function or helper VI it doesn‘t make any difference wether the addressed kernel is runnning locally or on a remote target. It is therefor not necessary to differentiate during development of the application in regard to network communication.

### Transparency for the kernel

The kernel does not know wether a request was issued locally or from a remote application and so again it is not necessary to differentiate during development of the application.

# 6. NetDX

The NetDX kernel provides all means for network communication.

The client (host) sends a request to the server (target) and receives a response. These messages are sent via TCP as bytestream. Communication takes place asynchroneously via a proprietary, stateful protocol.

The server listens on port **33333** for incoming connections. Transmission has to finish within 300 ms.

## 6.1. Request

A request consists of:

- Length of the data block (LEN): Signed long int
- Encoding (COD): Unsigned byte (default: 0)
- Accessor data type (typ): Unsigned byte
    - String = 0
- Request method (METHOD): Unsigned byte
    - 0 = Get
    - 1 = Set
    - 2 = Set w/o response
- Data block (DATA): String

| LEN | COD | TYP | MET | DATA |
|---|---|---|---|---|
| 4 bytes | 1 byte | 1 byte | 1 byte | LEN bytes |

**Data block**

Within the data blocks a lot of information are transmitted *serialized*. The dollar sign (ASCII-Code `0x24`) serves as a separation character for the following three elements:

- Kernel name (KERN): String
- Function name (CMD): String
- Parameter (PARAM): String

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

## 6.2. Response

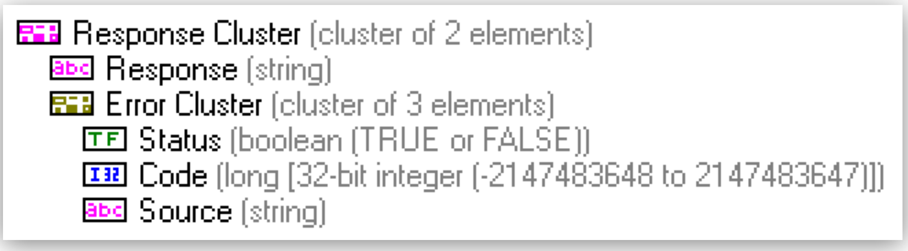
A response consists of:

- Length of data blocks (LEN): Signed long int
- Encoding (COD): Unsigned byte (default: 0)
- Data block (DATA): Cluster, variable length

| LEN | COD | DATA |
|---|---|---|
| 4 bytes | 1 byte | LEN bytes |

### Data block

The data block of the response is a cluster (a cluster is the LabVIEW pendant to *records* or *structs*). This response cluster contains the result (the return value) of the called function and an error cluster.



```
Response Cluster (cluster of 2 elements)
  Response (string)
  Error Cluster (cluster of 3 elements)
    Status (boolean (TRUE or FALSE))
    Code (long [32-bit integer (-2147483648 to 2147483647)])
    Source (string)
```

Img. 19: Data structure of the resonse data block‘s cluster

- Response Cluster
  - Length of RES (LEN1): Signed long int
  - Result (RESULT): String
  - Error Cluster
    - Error status (STAT): Boolean
    - Error code (COD): Signed long int
    - Length of source (LEN2): Signed long int
    - Error source (SOURCE): String

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

See Chapter 6.4.

## 6.3. Example

**Request**

A client sends the plain-text command `SetChannel` with parameter `Ch1` to the kernel `DIO`:

```
DIO$SetChannel$Ch1
```

The resulting data including the two dollar characters has a length of 18 (`0x12`) characters. Hence, the complete request message (Length of data block, encoding, method and data block) as *byte stream* is:

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

**Response**

The server responds in plain-text with the result string „ACK“ (string length = 3) an an empty error cluster. The data block (length of result string, result, status, code, length of source-string und source) as *byte stream* is therefore:

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

The complete response message (length of data block, encoding und data block) as *byte stream* is:

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

APPic
Framework
Manual v2.0 (Rev: 06)

APP
INSTRUMENTS

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