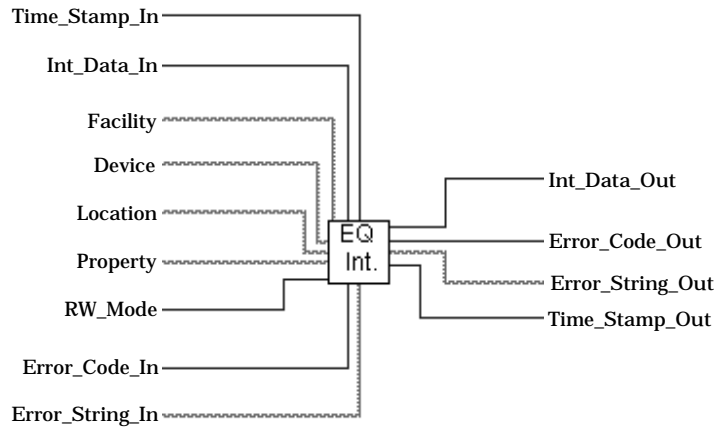


Fig.2 Front Panel and Block Diagram of a simple demonstration to read data from the fast ADC.

REFERENCES

- [1] M.Bohnert, O.Hensler, D.Hoppe, K. Rehlich, Distributed Object Oriented Control System (DOOCS), internal DESY document, Abt. MVP.
- [2] O.Hensler, K. Rehlich, P. Shevtsov, Equipment Name Server and Tesla Test Facility Control System. Proceedings of this Conference.

.Eq_Int.vi gives an example of LabVIEW Equipment Access VI:



132

Int_Data_In - input data which have to be sent to the equipment.

132

Int_Data_Out - data received from the equipment.
















Built-in LabVIEW Help Window shows information about all available LabVIEW VIs for the DOOCS communication, so the user is informed about all required data for selected VI.

Here is also an example of LabVIEW VI, where Equipment Access SubVIs is used. Example represents VIs front panel and block diagram (Fig.2).

VIs front panel represents the user interface to the system, block diagram contains graphical representation of programming code, which is executed when VI runs.

The described set of LabVIEW SubVIs with DOOCS communication protocol was used in several control applications at TTF during one year. This testing have shown the reliability and convenience of this software tools especially in control systems for the equipment, which is new, because the interface and even the layout of the system require numerous significant changes during testing.

by controls and indicators with correspondent data types. That is :

	Facility - facility name of the equipment address,
	Device - device name of the equipment address,
	Location - location name of the equipment address,
	Property - property name of the equipment address,
	RW_Mode - read from = 0 or write to = 1 the equipment,
	Error_Code_In - input error code of the call,
	Err_String_In - input error as a string,
	Error_Code_Out - returned error code of the call,
	Err_String_Out - returned error as a string,
	Time_Stamp_In - time stamp of the input data,
	Time_Stamp_Out - returned time stamp of the data,
	Length_In - length of the buffer array, provided by the caller,
	Length_Out - <i>actual length of the data array, returned to the caller,</i>
	T_Start - start time of request,
	T_Stop - end time of request.

Before each request, which will read data from or write data to the equipment, user must define the next parameters of Equipment Access VIs:

Facility
Device
Location
Property
RW_Mode

When SubVI finishes its execution, the **Error_Code_Out** parameter will contain the error code as an integer value and **Err_String_Out** will contain ASCII string with an error description. For this parameters, it's possible to wire the **Error_Code_Out** parameter of previous VI to the **Error_Code_In** parameter of the next VI. The same is true for the **Err_String_Out** and **Err_String_In** parameters. This allows to wire the error parameters of all Equipment Access SubVIs being used in the user's VI into chain and put the indicators for the error code and error string at the end of this chain.

```

device = 'PENNING' // char(0)
location = 'V_1' // char(0)
property = 'P' // char(0)
rw_mode = 0
c/*Read a single float from the equipment. */
call eq_float (facility, device, location, property, & %VAL(rw_mode),
float_data,
error_code, & err_string, time_stamp)
print *,"!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!"
write (*,1) facility, device, location, property
write (*,2) float_data
write (*,3) error_code, err_string
format ("Facility = ", A, "Device = ", A, "Location = ", A, "Property = ", A)
2format ( "DATA = ", F13.10)
3format ("ERROR_CODE = ", I6, " , ERR_STRING = ", A)
c/***** Checking EPICS call. *****/
facility = 'TTF.EPICS' // char(0)
device = 'TEMPERATURE' // char(0)
location = 'COLDB.4K.1.OUT' // char(0)
property = 'TEMP' // char(0)
err_string = 'NO_ERROR_AT_ALL' // char(0)
c /*Read a single float from the equipment. */
eq_float (facility, device, location, property, & %VAL(rw_mode), float_data,
error_code, & err_string, time_stamp)
write (*,1) facility, device, location, property
write (*,2) float_data
write (*,3) error_code, err_string
END

```

This example shows a single float read request from the equipment at TTF.VAC/PENNING/V_1/P and also reads a temperature from TTF.EPICS/TEMPERATURE/COLDB.4K.1.OUT/TEMP, which is accessible through EPICS communication.

LabVIEW VIs for DOOCS communications

LabVIEW applications are called VIs and they can call shared library functions directly or call SubVIs, which in turn call shared library function. For LabVIEW users a set of SubVIs, which realize access to the DOOCS sever properties was created. Users can use that SubVIs in their LabVIEW programm (VIs) in order to access device data and thus realise Equipment Data Access from LabVIEW VIs.

The following VIs are available:

Eq_Int.vi, Eq_Float.vi, Eq_String.vi, Eq_Int_Array.vi, Eq_Float_Array.vi,
String_Array_Read.vi, String_Array_Write.vi, Eq_Hist_Read.vi, Eq_Hist_Write.vi,
Eq_Spectrum_Read.vi, Eq_Spectrum_Write.vi

All this SubVIs do their job by the interfacing shared library call. There are a number of parameters, which are common for all Equipment Access VIs. This is almost the same set, as for shared library functions, but parameters in LabVIEW VI are defined

length.

```
void eq_spectrum (char* facility, char* device, char* location, char* property,
int rw_mode, float* float_data_array, int* length, char* comment, float* start,
float* inc, int* status, int t_start, int t_stop, int* error_code, char* err_string,
int* time_stamp)
```

Read or write a spectrum from/to the device. A spectrum consists of a floating point array with length data samples. The samples are starting at start and are ending at start+inc*length. The calling program must provide a buffer for the comment field with a length of STRING_LENGTH (80) characters.

With the aid of that library functions users can access data and perform permitted actions, recognized by different DOOCS servers in the network.

The following simple program codes give examples of this library function calls from C and FORTRAN applications.

EXAMPLES

Example of read function call from C application programm.

```
#include <lv_eq.h>
main() {
float f1;
char err_str[16];
int err, time;
/* read a single float from a device: */
eq_float ("TTF.VAC", "ION_PUMP", "V_1", "P", 0, &f1, &err, err_str,
&time);
if (err) printf("Error in eq_float : %s\n", err_str);
else printf("Result is: %g \n", f1);
}
```

File lv_eq.h includes library function prototypes as shown before. In this example C client program reads single floating point value from DOOCS address "TTF.VAC/ION_PUMP/V_1/P" and prints on the screen the result value, obtained from the server or displays an error message.

See extended information about DOOCS "addresses" and DOOCS data types in [1].

Example of read functions calls from FORTRAN program.

```
CHARACTER*80 facility, device, location, property
INTEGER rw_mode, error_code, time_stamp
REAL float_data
CHARACTER*16 err_string
external eq_float !$pragma C (eq_float)

c /*****Checking the eq_float(...) function*****/
err_string = 'NO_ERROR_AT_ALL' // char(0)
facility = 'TTF.VAC' // char(0)
```

modified. The *rw_mode* determines the direction of the call. A *rw_mode=0* selects a read from the device. The caller has to provide all buffers. The *length* argument specifies the length of the buffers the caller has provided. On return it contains the actual number of elements filled by device server. The *t_start* and *t_stop* argument selects a time range for the request. If it's set to zero all available data is returned. With *rw_mode=1* data is sent to the device. All calls return an *error_code*. If the *error_code* is not zero an additional error string is copied into the *err_string* buffer. The caller must provide the buffer for error string with the length of 16 characters. Here is list of functions from the interfacing libraries :

```
void eq_int (char* facility, char* device, char* location, char* property, int
rw_mode, int* int_data, int* error_code, char* err_string, int* time_stamp)
Read or write a single integer from/to the device.
```

```
void eq_float (char* facility, char* device, char* location, char* property, int
rw_mode, float* float_data, int* error_code, char* err_string, int* time_stamp)
Read or write a single float from/to the device.
```

```
void eq_string (char* facility, char* device, char* location, char* property, int
rw_mode, char* string_data, int* error_code, char* err_string, int* time_stamp)
Read or write a single string from/to the device.
```

```
void eq_int_array (char* facility, char* device, char* location, char* property,
int rw_mode, int* int_data_array, int* length, int* error_code,
char*err_string, int* time_stamp)
Read or write a int array from/to the device.
```

```
void eq_float_array (char* facility, char* device, char* location, char* property,
int rw_mode, int* float_data_array, int* length, int* error_code, char*
err_string, int* time_stamp)
Read or write a float array from/to the device.
```

```
void eq_string_array (char* facility, char* device, char* location, char* prop-
erty, int rw_mode, char* string_array, int* int_array, float* f1_array, float*
f2_array, int* time_array, int* length, int t_start, int t_stop, int* error_code,
char* err_string, int* time_stamp)
Read or write a string array from/to the device. A string array is an array of a
record of a char string, an integer, two floats and a time stamp. All five parts of
the record are stored in five separate arrays with the length length. Each char
string need a buffer of STRING_LENGTH (80) characters.
```

```
void eq_hist (char* facility, char* device, char* location, char* property, int
rw_mode, int* time_array, float* float_data_array, int* status_array, int*
length, int t_start, int t_stop, int* error_code, char* err_string, int* time_stamp)
```

Read or write a historical array from/to the device. The history record consists of three elements: an array with time stamps, an array with the floating point data values and an array with status informations. All three arrays are of the same

set of LabVIEW VIs, which calls these C-style functions from its interfacing library, is provided.

This configuration allows C, Fortran and LabVIEW applications to communicate with DOOCS Servers, EPICS IOCs and systems using other protocols.

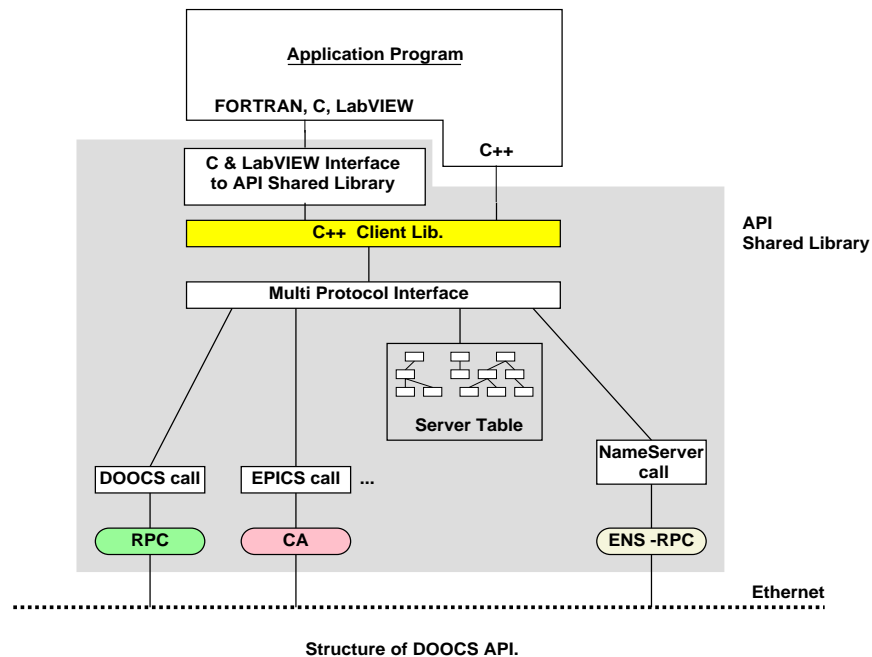


Fig.1 Structure of DOOCS API.

C & LabVIEW Interface Library Functions

An intermediate interfacing C-style libraries is available for Solaris1 and Solaris2 platforms and is into operation at the TTF at DESY site. Functions allow to send requests to DOOCS servers and cover frequently used data types, which are available in the DOOCS communication. All functions have some common parameters:

- char* facility** - facility name of the equipment address
- char* device** - device name of the equipment address
- char* location** - location name of the equipment address
- char* property** - property name of the equipment address
- int rw_mode** - read from = 0 or write to = 1 the equipment
- int* error_code** - returned error code of the call
- char* err_string** - returned error code of the call
- int* time_stamp** - returned time stamp of the data
- int* length** - length of the array
- int t_start** - start time of request
- int t_stop** - end time of request
- int t_stop** - end time of request

The equipment address is set by the calling program. It is an input to the call and will not be

Integration of LabVIEW into TTF Control System.

**S.Goloborodko (IHEP, Protvino), O. Hensler (DESY, Hamburg),
K. Rehlich (DESY, Hamburg).**

Abstract

TESLA is an international collaboration which has been established to design and build a prototype for a superconducting (SC) linac. The Tesla Test Facility (TTF) consists of the infrastructure for SC cavity processing and the prototype linac including control system to operate it and test stands for equipment tests.

At TTF linac the **Distributed Object Oriented Control System** (DOOCS) from DESY with Remote Procedure Call (RPC) based communication is used and also provides a Multi Protocol Interface to the accelerator systems. It's modular design allows to add additional control protocols in order to realize access from application programs to distributed control subsystems with different communication protocols.

The LabVIEW industrial package for graphical programming is widely used in the TTF control system as a convenient tool to evaluate the performance of the linac and its subsystems. Integration of the LabVIEW package into the TTF control system requires an interface to DOOCS. The paper describes software tools, which provide interface to the controls for LabVIEW Virtual Instruments (VI) and also for the application programs, written in C or FORTRAN programming languages.

Introduction

DOOCS is a control system tool which is used to solve a variety of control tasks in TTF [1]. The communication in DOOCS is based on RPCs with eXternal Data Representation (XDR) network format. It runs on SUN SPARC stations under SunOS, Solaris 2.4 and on PCs with UNIX operating systems. The DOOCS libraries are completely written in C++ and follow the client-server model. The DOOCS approach defines each hardware device as a separate object and this object is represented in a network by a device server, which handles all device functions. Such object oriented approach defined a choice of C++ as a programming language for the client and server parts of DOOCS libraries. Instances of devices and all properties of these devices are defined in the server program. All this definitions are transparent to the client programs and after the start of the device server all properties for that partial device are available to all clients in the network. This completely separates client programs from device servers and modification of the server program has no influence on the clients because of symbolic access to the data. With the DOOCS Multi Protocol Interface client Application Programming Interface (API) can handle not only RPC, but also EPICS calls and uses an Equipment Name Server (ENS) [2], which resolves the names (IP-addresses) of device servers in the network. The DOOCS API is a C++ library.

In the LabVIEW package VIs send or receive data to/from device via Call Library Function node. This node calls specified shared library function directly. But called functions must be written in C code.

In order to link these two different systems two level interface was designed (Fig.1). The first level is a C++ shared library with C-style functions to access C++ API. On top of it a