

DSDDataSocket

DeviceServer User's Guide

Version courante du document : 1.0

Dernière modification le 29/11/2004

Historique des modifications

Date	Revision	Description	Author
18/07/03	1.0	Initial Version	A. Buteau

1	Main fonctionnalités of DeviceServer	2
1.1	Gateway between Tango and LabView	2
1.2	Key design issue	2
1.3	Quick description of DataSocket protocol	3
1.4	Sharing a Labview variable	3
2	An unforecast usage : gateway with OPC world	3
3	Software Architecture.....	4
3.1	The three components of a DataSocket based application	5
3.2	Design consideration	5
3.2.1	Threading problems	5
3.2.2	Synchronous vs Asynchronous mode.....	5
4	Current known limitations.....	6
4.1	Start to end status of connections.....	6
4.2	Bad name for URL	6
5	Performances	6
5.1	Array exchange of 10000 doubles	6
5.2	Exchange of many scalars.....	6
6	Software Setup	6
6.1	Installation of CVI Runtime	6
6.2	Installation of National DataSocket Server	6

1 Main fonctionnalités of DeviceServer

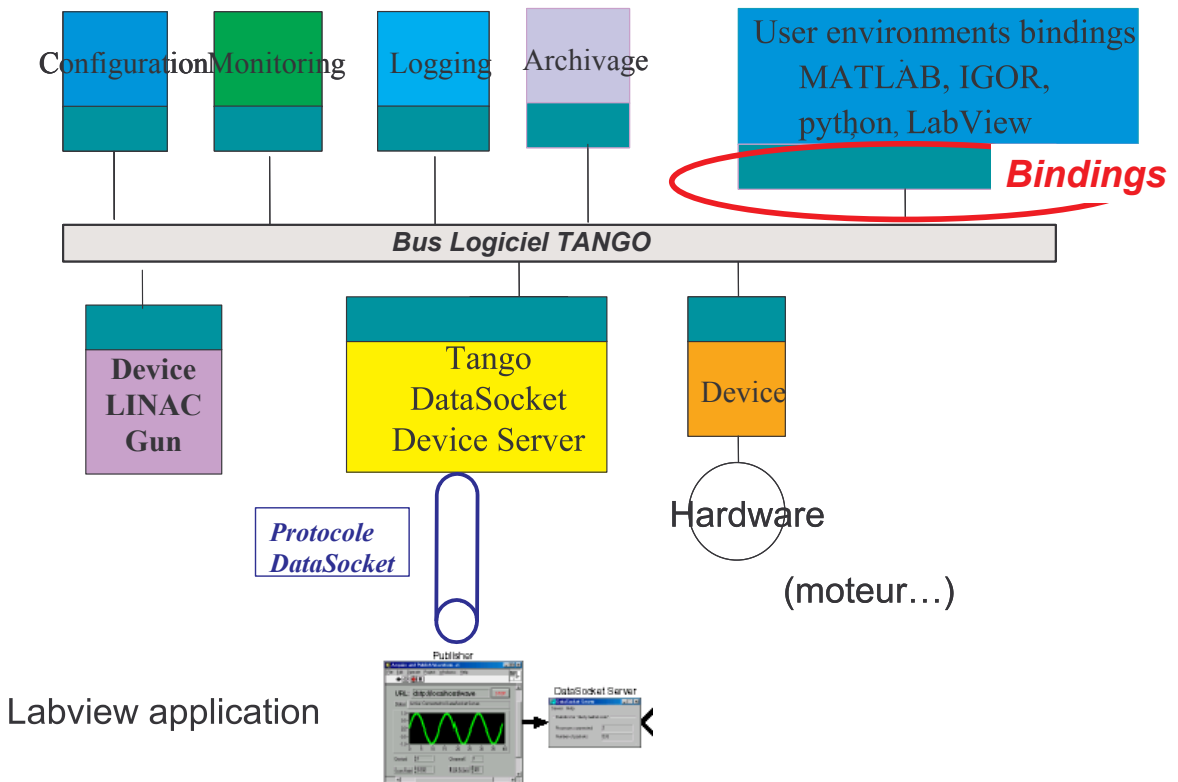
1.1 Gateway between Tango and LabView

The purpose of this “*Tango DataSocket DeviceServer*” is to exchange Data between TANGO and a LabView application. It acts as a gateway whose only purpose is to **get and set** values which are available in a Labview Application from the TANGO world. On the following “*Tango architecture diagram*”, we see that the Labview application appears at the same level that a piece of hardware which is driven by it’s Tango DeviceServer.

As this “*Tango DataSocket DeviceServer*” doesn’t have any knowledge of the Labview application it is communicating with, other Tango Devices should be written that represent the Labview application Devices in the Tango world (as it is show on the diagram with the *Linac Gun Device*)

Important note : This DeviceServer is has nothing to do with the Labview binding !! The “*Tango Labview binding*” purpose is to allow a Labview application to access data from the Tango world. So, it’s working on the client side, and not at the server side as this “*Tango DataSocket DeviceServer*”.

Tango « high level » applications



1.2 Key design issue

On key design consideration of this DeviceServer was to make this data exchange with **no or very few developments** on the LabView side. That’s what motivated the choice of the DataSocket protocol as the technical basis of this development. In SOLEIL’s case, this “*Tango DataSocket DeviceServer*” will be use to communicate with an application developed by a third party firm to control the LINAC.

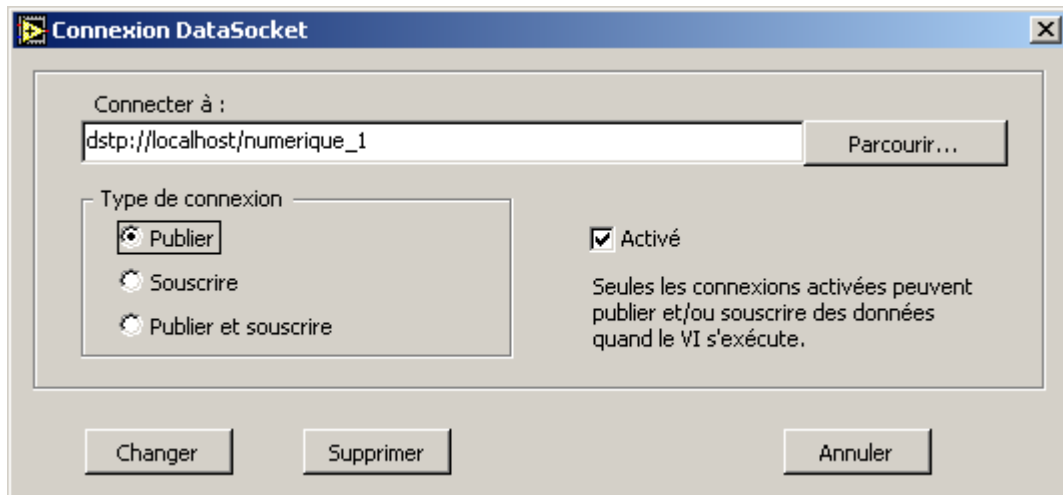
1.3 Quick description of DataSocket protocol

DataSocket is a proprietary protocol developed by National Instrument to exchange data between their software tools Labview, Labwindows, Measurement Studio, etc. A more detailed presentation is available at <http://www.ni.com/pdf/wp/wp1680.pdf>.

The exchanged data may be of different kinds : scalars, arrays and even structures.

1.4 Sharing a Labview variable

To share a variable of the Labview application in the DataSocket world, the only thing to do is to right click on it in Labview and to define it as “published” for data that must be read from the outside, and “subscribed “ for a data that must be written from the outside world



2 An unforecast usage : gateway with OPC world

2.1 Communication scheme

The CVI library is able to read/write a value available in a OPC server with the same set of functions than necessary to get values from a DataSocket Server.

Therefore the *Tango DataSocket DeviceServer* may access data in an OPC server, only by giving an URL which is OPC compatible.

For instance, we can access the following OPC item (which is produced by an OPC demo application given by National Instrument) just by calling the **ReadDouble** command on the *Tango DataSocket DeviceServer* with the following URL.

[opc://localhost/National Instruments.OPCDemo/sine:0.0..8.0:3.0](opc://localhost/National%20Instruments.OPCDemo/sine:0.0..8.0:3.0)

2.2 OPC naming issues

Basically an OPC variable is accessible with an URL done with the following fields:

URL field	Comment
Opc::	Tells the Datasocket server that variable is accessed through the OPC protocol (rather than http or DataSocket protocol)

<i>localhost</i> or <i>britten.synchrotron-soleil.fr</i>	Name of the remote computer on which the OPC server runs
<i>National Instruments.OPCDemo</i> Or <i>Fortetion.OPCSimulation</i>	Name of the OPC server
<i>Area1.Simulate.Sine Wave</i>	Item Name which must be read/write on the OPC server. Its may contain special characters as . or space.

Remarks :

- The group name (an OPC concepts) is a client side notion. It allows you to arrange items in generic OPC browsers, but it should be use in the URL name (as a server is not aware of this group name)
- Some OPC server are not browseable: therefore the list of their items cannot be seen from an OPC browser. To consult these variables, the exact items names must be known
- A DataSocket Server must be started on the computer running the TangoDeviceServer

2.3 Troubleshooting OPC Interface Problems

The OPC specification is rather loose and is often interpreted differently by different parties. Due to encapsulation of the technology and lack of troubleshooting tools, OPC issues are extremely time consuming. It is typical that a failure is reproducible only with a particular client/server combination. The same client works with other servers and the same server works with other clients.

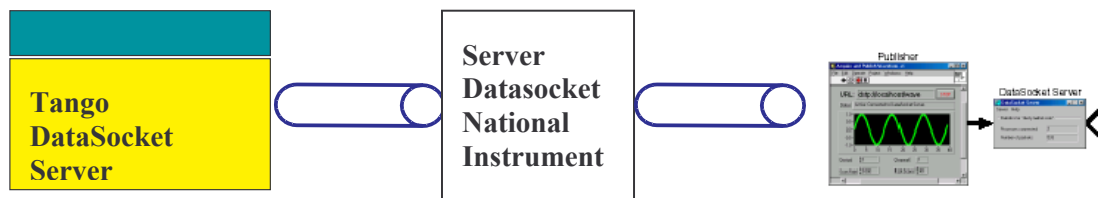
The following National Instrument document <http://zone.ni.com/devzone/conceptd.nsf/webmain/401CB3A4E216356286256BC7004B5831?opendocument#3> gives some ideas to investigate.

A good OPC client browser is the **National Instruments Server Explorer** available on <http://www.ni.com>

3 Software Architecture

3.1 The three components of a DataSocket based application

To establish a DataSocket based communication between 2 applications, a third component is mandatory : “the DataSocket Server”. It is an standard executable given by National Instrument which act as a relay between both applications. In our case, the 3 following executables are mandatory



Notes :

- The DataSocket Server executable must be started to make data exchange possible. If it crashes, communication is **broken** between TANGO and LabVIEW.

3.2 Design consideration

3.2.1 Threading problems

One major problem during the development phase was that the thread which calls the CVI function **DS_Open** *should not be killed* in any case (otherwise the threads forked by this **DS_Open** call becomes zombies).

This behaviour is incompatible with the normal thread management of OmniORB, which spawns a thread for each new client and kills it after some inactivity time.

A *dedicated* and *permanent* thread is therefore created at startup of Device, which is in charge of opening and closing DataSocket handles.

The usual TANGO threads communicate with this so called "CVI thread" through a synchronisation Object called **SharedData**. Synchronisation of activity between these 2 threads is accomplished through Condition variables

3.2.2 Synchronous vs Asynchronous mode

The CVI DataSocket library proposes to work in following modes:

- Asynchronous : when the data changes in the Labview application, the client side (in our cas the Tango side) is wake up and signaled that a new value is available.
- Synchronous: to get a fresh data from LabView, a call to the **DS_Update** function must be explicitly done on the Tango side.

A this DeviceServer acts only as the gateway, and is not the final client of the data produced by the Labview application (the final client are other Tango Devices) , the asynchronous mode does not appear very interesting, because asynchronous should be also be done till final client and nowadays no Tango mechanisms allows it .

A possible extension (if performances justify it but which is not not forecast for now) would be to make the *Tango DataSocket DeviceServer* :

- Open Datasocket when requested by other Tango Devices

- Use CVI asynchronism to “subscribe” to values changes on the LabView side
- Internally cache updated values
- Give other Tango Devices cached value

4 Current known limitations

4.1 Start to end status of connections

It is impossible with the CVI DataSocket functions to have the real status of connection if the final Labview Application is down (it sounds crazy but it's true: after a discussion with NI , it could be an interesting feature in the next version ...). It means, that as long as the National DataSocket Server runs, it is possible to obtain datasocket data even if no Labview application is producing them anymore.

To face with this problem, a heartbeat variable (which is always increasing) must be implemented in the Labview application. It's then up to the TangoDeviceServer to monitor this heartbeat variable and declare the Labview application broken if the variable is no more increasing.

4.2 Bad name for URL

It's impossible for the “*Tango DataSocket DeviceServer*” to detect that a bad URL name has been given as a argin, as the National DataSocket Server accept connection to non existing URL and even returns a value for scalars for such non existing URL!!

5 Performances

I don't have realistic LabView application to make performances measurements. Nevertheless, here are the figures I could get with my small test applications.

Hardware setup : 2 PC under WIN2000 PIII ~1GHz , on a 100 Mbit/s switched network and no CPU Activity

5.1 Array exchange of 10000 doubles

30 millisecondes for the execution of the method :

```
Tango::DevVarDoubleArray
DSDataSocket::read_double_array(Tango::DevString argin)
```

5.2 Exchange of many scalars

Not done because I lacked a Labview application producing so many variables

6 Software Setup

6.1 Installation of CVI Runtime

CVI requires a Run Time to be installed on the computer running the “*Tango DataSocket DeviceServer*”. This RunTime can be found here : A CVI Run Time is available in the *script* directory of the Device.

6.2 Installation of National DataSocket Server

The Setup is available here at SOLEIL.
[Linac\ThreadDSDataSocket\Install_Runtime_Labwindows\setup.exe](#)