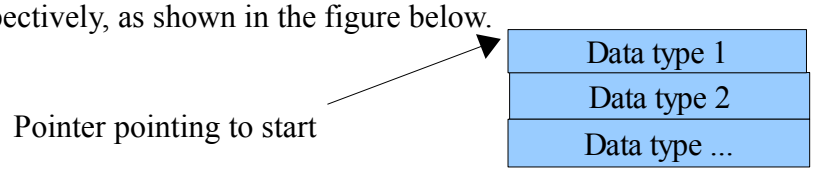
Gist of test program

1 Socket class and its client class

This class hasn't been changed except that an overloaded method *send* has been added. This send method use a very flexible way to send contents with a customised size, two arguments are passed, I.e. a pointer pointing to the start of the block of memory, and a integer typed size. Within this block of memory to be sent, users can send any types of data with a specified size. This is the way to make a socket send method more flexible, and more important thing is to make it type independent. Based on the same spirit, a *recv* method has been overloaded, with only one argument. This recv method is to receive the block of data contained in a specified memory and customised size. So the pointer will be pointing the start of the block of memory that it receives. So before sending a block of data in the memory, users should know how to package the data block. Similarly, after receiving a block of data, users should know how to decode that block of data by its types and size respectively, as shown in the figure below.



Drawing 1: package a block of data

2 ClientSocketN class

In this class, the *LogSend* and *LogRecv* methods are overloaded in order to cope with the client interface for *send* and *recv* methods described above. Again, the same type of arguments are adjusted.

3 odr.h

In this header file, the ODR packet contains many things as described as follows.

(1) Markers: there are ODRMARKER and LDAMARKER markers. When users want to send ODR commands, then the marker should be switched to ODRMARKER, in a similar way for LDA commands.
(2) odr_header_t packet: it is a structure type packet, with union contained inside. This sharing union area is used for the payload for ODR packet.
(3) Command or Parameter switch: In side of odr_header_t packet, there is a switch for the use of command or parameter, i.e., ODRPayload.ODRPacket.parameter. If it is assigned by ODRCMD, then this parameter will be used for command mode, otherwise it will be used for parameter mode when it is assigned by non ODRCMD value. Do discuss with Andrzej, he defined the ODR packet with its parameters.
(4) ODR command types: 11 commands have been defined inside.
(5) ODR_Err: this structure type is designed for ODR error types.
(6) ODR_Params: this structure is designed for ODR parameters.
(7) LDA commands: they are put in this header file just for a temporary purpose, and will be moved to LDA module.

4 ldaemu.h

This header file is used for LDA emulator test ONLY.

5 eq_odr.h

All DOOCS variables related to ODR properties are defined in this header file.

(1) port number and host name
(2) ODR device error type variable;
(3) Marker and parameters used for sending ODR commands and parameter controls.
(4) ODR and LDA machines' MAC address, and LDA packet properties: They are ONLY used for LDA emulator purpose. This will be changed in the future.
(5) DOOCS control command variable;
(6) DOOCS variables for related to spectrum and history.

6 odr_rpc_server.cc source code

The main work is done within this source code, including

(1) set control commands' value: void D_ControlCMD::set_value(int cmd)
    1. 0<cmd<80 is assigned to be used for ODR commands;
    2. 80<cmd<100 is left for LDA commands;
    3. cmd>100 is customised for parameters' value control.

(2) void* ODR::GetThreadData(): GetThreadData method is overload from ThreadN class.
    1. Spectrum and History file are filled within this thread function;
       The thread is always running to receive information which is dynamically sending from caldata (ODR control software). The information received from ODR packet are separated, including:
    2. request to get parameter information of ODR data
    3. request to get statistical information from ODR packet
    4. request to get event list from ODR packet
    5. detect the specific error type
(3) void ODR::init(): all DOOCS variables can be initialised here if necessary. Current initialisation includes: device error type, port number, host name, building a socket, marker type check, start the thread, show thread ID. Users may add more into the initialisation.
(4) void ODR::update(): this method is updating by loops over all device instances. All types of errors are checked and displayed frequently.

7 add user-defined error types:
source/serverlib/eq_fct_errors.h and source/serverlib/eq_fct_errors.cc
Please refer to my presentation for group meeting.

8 doocs scripts
Please run doocs with scripts, instead of typing in command lines. Some examples are given:

start_odr_server.csh

```
#! /bin/tcsh -f
echo ">>>   Kill all processes ..."
killall -15 odr_server

echo ">>>   Start ODR Server ..."
cd $DOOCSROOT/source/server/calice/odr
$DOOCSROOT/Linux/obj/server/calice/odr/odr_server &
```

some tips of this script:

(1) use *killall* command instead of *kill*, killall command plus launched server name. Don't use process ID since it is always changing, whereas server name is fixed.
(2) Argument of killall: use -15 instead of -9. “-9” means stop the process by force. “-15” means terminate current server properly, if there are any dependent processes or unfinished

processes, then wait to be killed until they are ended, which means it is a more safer option.

(3) Use environment variable, instead of absolute directory

Example 2: jobs_sendgui.csh

```
#! /bin/tcsh -f

setenv ENSHOST "dhcp219.pp.rhul.ac.uk"

echo ">>>   Kill all processes ..."

killall -15 ENS_auth
killall -15 ddd_run
killall -15 ddd
killall -15 rpc_util

setenv MFILE $DOOCSROOT/run-error.log
rm -vf ${MFILE}


echo ">>>   Start ODR Server ..."
#cd $DOOCSROOT/source/server/calice/odr
#$DOOCSROOT/Linux/obj/server/calice/odr/odr_server &


echo ">>>   Start RPC Util"
#cd $DOOCSROOT/Linux/obj/clients/ttf/rpc_test; ./rpc_util &;
$DOOCSROOT/Linux/obj/clients/ttf/rpc_test/rpc_util &
echo "$DOOCSROOT/Linux/obj/clients/ttf/rpc_test/rpc_util "


echo ">>>   Start ENS_AUTH ..."
cd $DOOCSROOT/source/ENS_AUTH
$DOOCSROOT/Linux/obj/ENS_AUTH/ENS_auth &

#echo ">>>   Start SIN generator ..."
#cd $DOOCSROOT/source/example/server/singenerator
#$DOOCSROOT/Linux/obj/example/server/singenerator/singenerator_server &

#echo ">>>   Start Example test ..."
#cd $DOOCSROOT/source/server/example/example
#$DOOCSROOT/Linux/obj/server/example/example/example_server &


#echo ">>>   Start rpc_test ..."
#cd $DOOCSROOT/source/server/test/test/
#$DOOCSROOT/Linux/obj/server/test/test/test_server >> & ${MFILE} &

echo ">>>   Start DDD run ..."
cd $DOOCSROOT/source/server/calice/odr
#$DOOCSROOT/Linux/obj/dddlib/ddd_run \
#     -dir $DOOCSROOT/source/example/server/singenerator &
#$DOOCSROOT/Linux/obj/dddlib/ddd \
#     -dir $DOOCSROOT/source/example/server/singenerator &
$DOOCSROOT/Linux/obj/dddlib/ddd_run -dir $DOOCSROOT/source/server/calice/odr &
#$DOOCSROOT/Linux/obj/dddlib/ddd -dir $DOOCSROOT/source/server/calice/odr &
```

All scripts can be put in a directory and are added to $PATH.