Module <UserProtocol> of subsystem “Protocols”

Module:	UserProtocol
Name:	User protocol
Type:	Protocol
Source:	prot_UserProtocol.so
Version:	0.6.1
Author:	Roman Savochenko
Translated:	Maxim Lysenko
Description:	Allows you to create your own user protocols on any OpenSCADA's language.
License:	GPL

Contents table

Module <UserProtocol> of subsystem “Protocols” 1
Introduction 2
1. Part of the protocol for incoming requests 3
2. Part of the protocol for outgoing requests 5

Introduction

Module UserProtocol of the transport protocol is made to provide the user with the possibility of creation the implementations of different protocols by himself at one of the internal languages of OpenSCADA, usually JavaLikeCalc, without necessity of low-level programming of OpenSCADA.

The main purpose of the module is to simplify the task of connecting to the OpenSCADA system devices of data sources, that have limited distribution and/or provide access to their own data on a specific protocol that is usually fairly simple to implement in the internal language of OpenSCADA. For implementation of this the mechanism for the formation of the outgoing request protocol is provided.

In addition to the mechanism of the outgoing request protocol the mechanism for incoming request protocol is provided, which allows OpenSCADA to process the requests for data get on specific protocols, which simply can be implemented in the internal language of OpenSCADA.

The module provides the ability to create multiple implementations of different protocols in the object "User protocol" (Fig. 1).

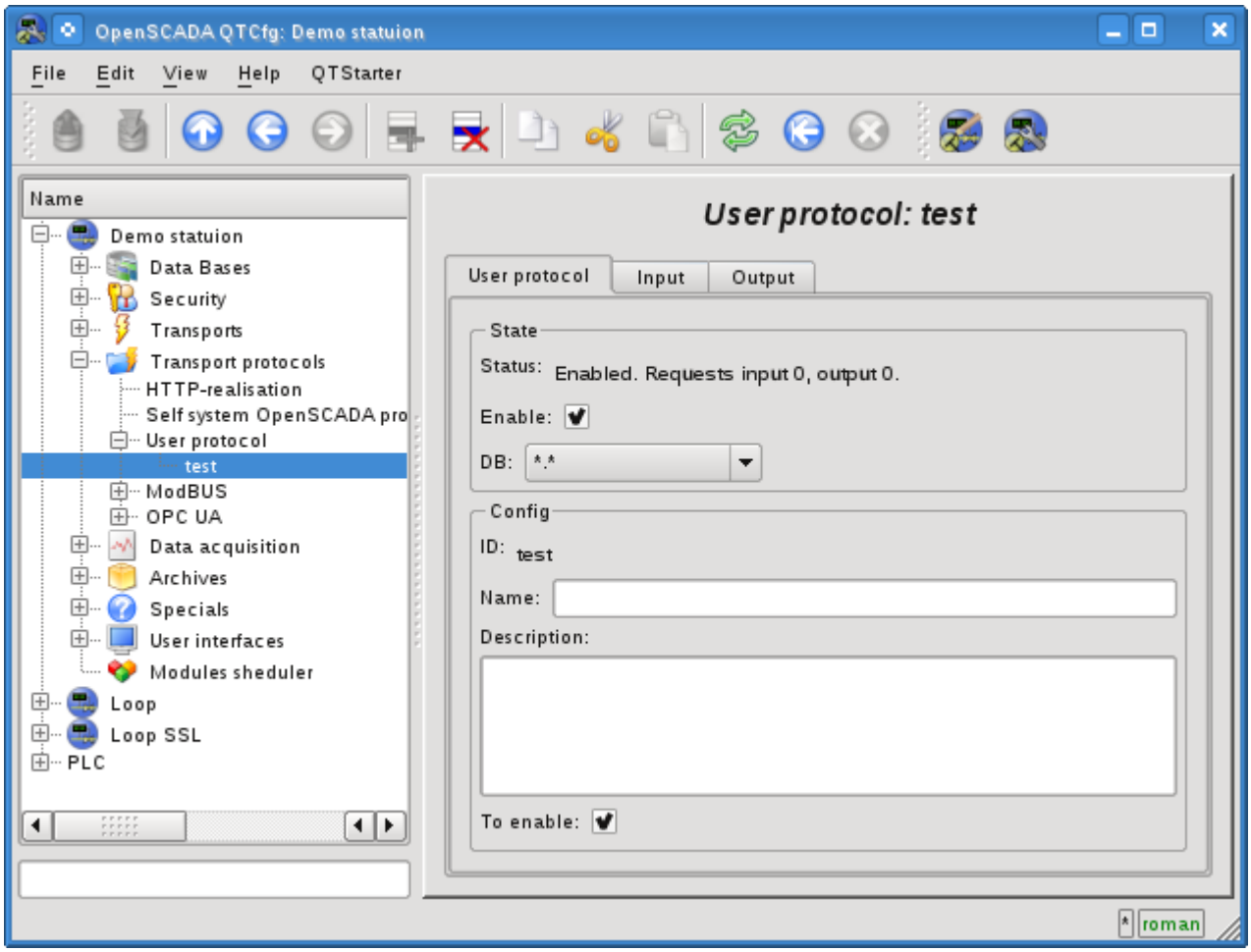

Fig.1. The main tab of the object "User protocol".

The main tab contains the basic settings of the user protocol:

- Section "Status" - contains properties that characterize the status of the protocol:
  - *Status* - current status of the protocol.
  - *Enable* - the protocol's status "Enabled".
  - *DB* - DB that stores configuration.
- Section "Config" - directly contains the configuration fields:
  - *ID* - information on the protocol's identifier.
  - *Name* - specifies the name of the protocol.
  - *Description* - brief description of the protocol and its purpose.
  - *To enable* - indicates the status "Enable", in which to transfer the protocol at startup.

1. Part of the protocol for incoming requests

Protocol of incoming requests is working in cooperation with the incoming transport and the separate object "User Protocol" is set in the configuration field of transport protocol, together with the UserProtocol module's name. In the future, all requests to the transport will be sent to the processing procedure of the protocol's request (Fig. 2).

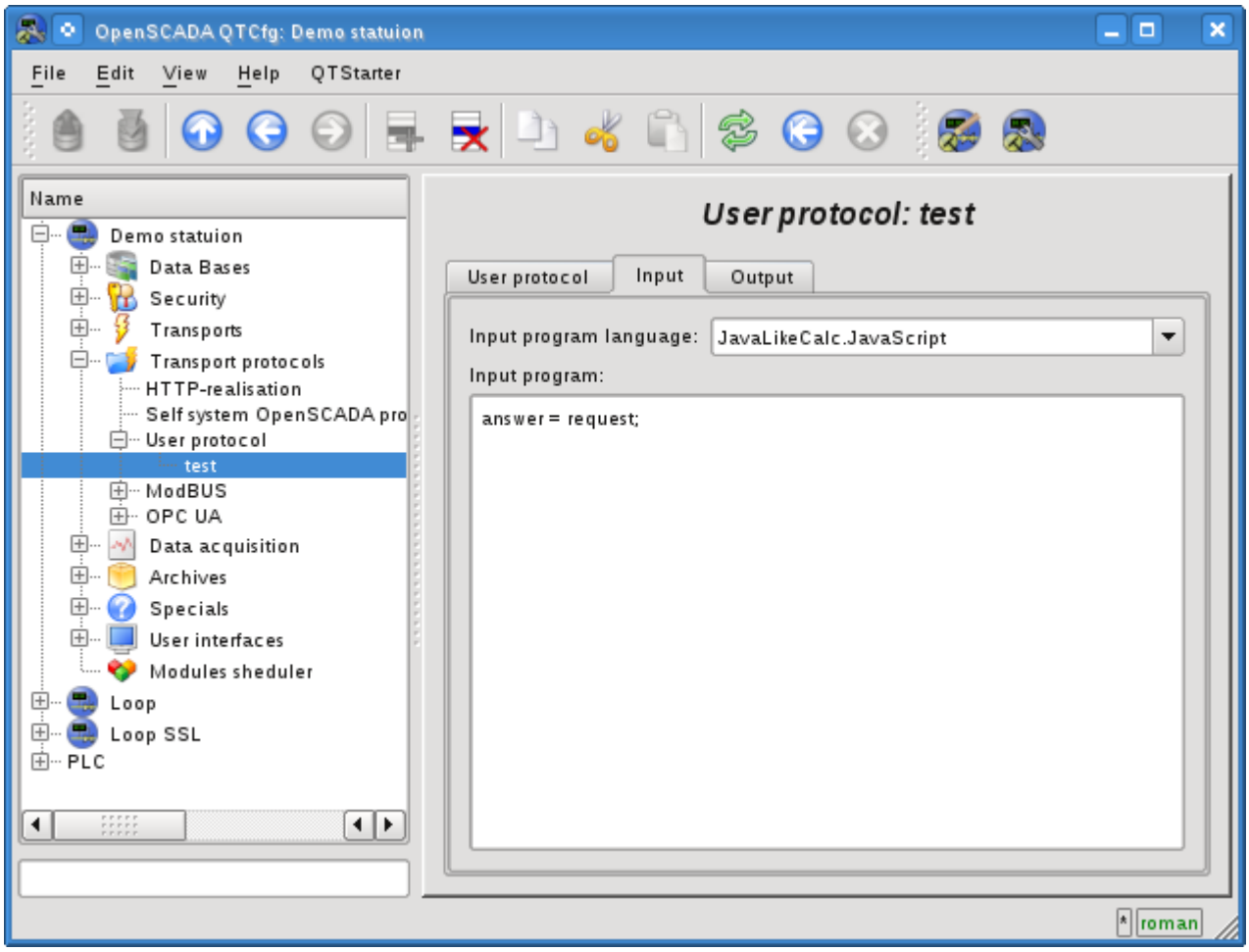

Fig.2. Tab of the processing procedures of the incoming requests.

Tab of the processing procedures of the incoming request contains the field for selecting the internal programming language of OpenSCADA and the text entry field for the typing the processing procedure.

For the processing procedure the following exchange variables with incoming traffic are predetermined:

- *rez* - processing result (false-full request;true-not full request);
- *request* - request message;
- *answer* - answer message;
- *sender* - request sender.

The overall scenario of processing of the incoming requests:

- Request is formed by the remote station and through the network it gets on the transport of OpenSCADA.
- OpenSCADA transport sends the request to the selected in the protocol's field UserProtocol module and to the objects of the user's protocol in the form of the variable's "request" values - for the block of the request and "sender" - for the sender address of the request.
- The execution of the the procedure of protocol of the incoming request is started, during which the contents of the variable "request" is analyzed and the response in the variable "answer" is formed. At the end of the procedure's execution the variable "rez" is formed, which indicates the transport to the fact of reception of full request and the formation of the correct answer (false) or to the necessity for the transport to expect for the remaining data (true).
- If the result of the processing procedure is the variable "rez" with the 'false' and the response in

the variable "answer" is not zero, then the transport sends the response and reset the accumulation of "request".

- If the result of the processing procedure is the variable "rez" with 'true' then the transport continues to expect for the data. When it receives the next portion of data they are added to the variable "request" and this procedure is repeated.

As an example, consider the implementation of query processing of protocol DCON, for some queries to a data source with the address "10":

```
//SYS.messDebug("TEST REQ: ",request);
//Test request for full
if(request.length < 4 || request[request.length-1] != "\r")
{
      if(request.length > 10) request = "";
      return true;
}
//Check for integrity of the request (CRC) and address
CRC = 0;
for(i = 0; i < (request.length-3); i++) CRC += request.charCodeAt(i);
if(CRC != request.slice(request.length-3,request.length-1).toInt(16) ||
request.slice(1,3).toInt(16) != 10) return false;
//Analysis of the request and response prepare
if(request.charCodeAt(0) == "#") answer = ">+05.123+04.153+07.234-02.356+10.000-
05.133+02.345+08.234";
else if(request.charCodeAt(0) == "@") answer = ">AB3C";
else answer = "?";
//Finish response
CRC = 0;
for(i=0; i < answer.length; i++) CRC += answer.charCodeAt(i);
answer += (CRC&0xFF).toString(16)+"\r";
//SYS.messDebug("TEST ANSV: "+answer.charCodeAt(0),answer);
return 0;
```

2. Part of the protocol for outgoing requests

The protocol of outgoing requests is working in cooperation with the outgoing transport and with the separate object of the "User Protocol". The source of the request through the protocol may be a function of the system-wide API of the user programming of the outgoing transport *int messIO(XMLNodeObj req, string prt);*, in the parameters of which it must be specified:

- *req* - request as an XML tree with the structure corresponding to the input format of the implemented protocol;
- *prt* - the name of the "UserProtocol" module.

The request which is sent with the aforesaid way is directed to the processing procedure of the protocol's request (Fig. 3) with the user protocol's ID which is specified in the attribute req.attr("ProtIt").

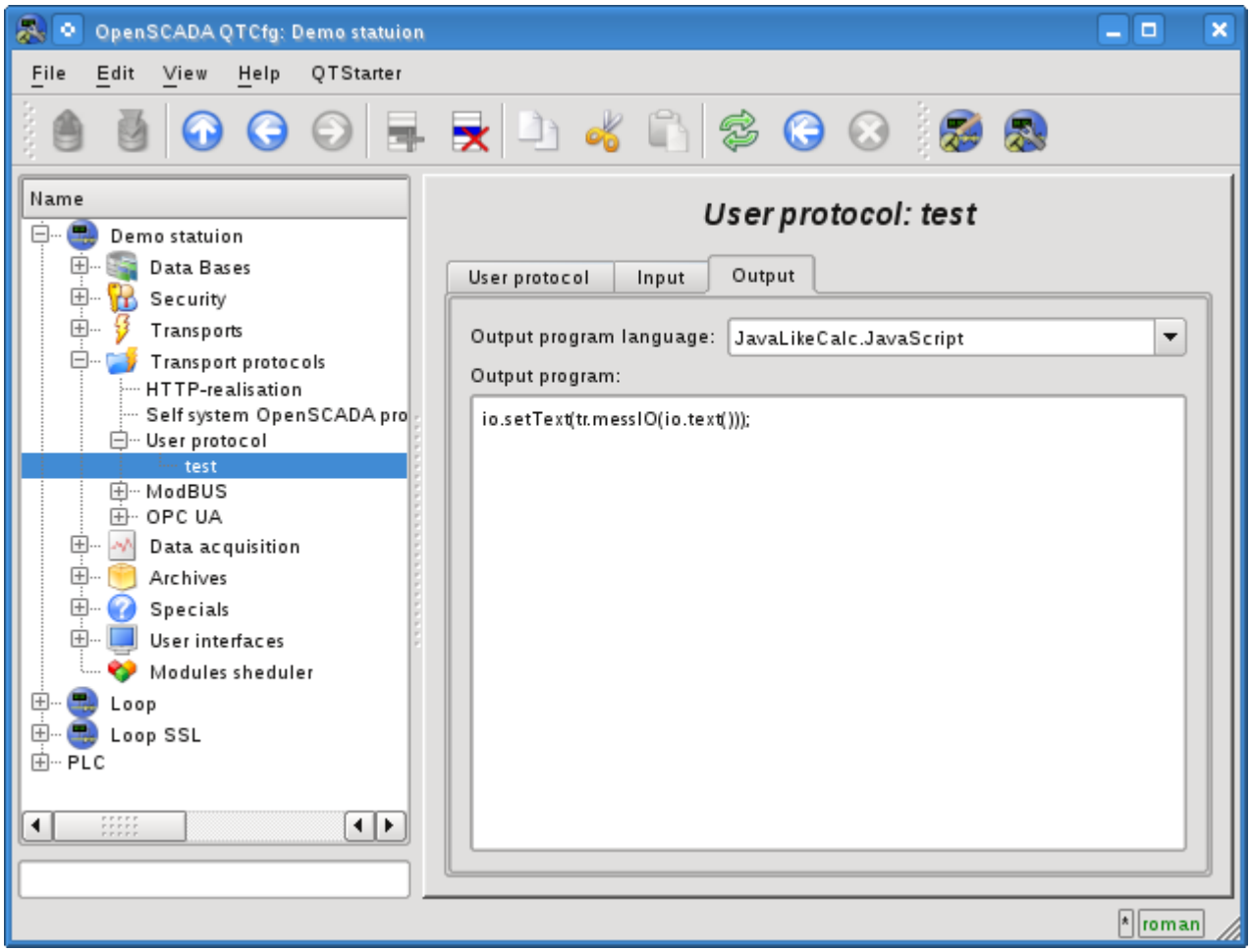

Fig.3. Tab of the processing procedures of the outgoing requests.

The tab of the processing procedure for outgoing requests includes the field to select the internal programming language of OpenSCADA and text field for typing the processing procedure.

For the processing procedure the following exchange variables are predetermined:

- *io* - XML node of the exchange with the client, through which the protocol gets the requests and into which it puts the result with the format implemented in the procedure;
- *tr* - The transport object is provided for the call the transport function *string messIO(string mess, real timeOut = 1000);* "tr.messIO(req)".

The overall scenario of the formation if the outgoing request:

- Building of the XML-tree in accordance with the structure implemented by the protocol and setting of the user protocol identifier in the attribute "ProtIt".
- Sending the request to transport through the protocol *SYS.Transport["Modul"]["OutTransp"].messIO(req,"UserProtocol");*.
- Selection of the user interface in accordance with req.attr("ProtIt") and initialization of variables

of outgoing transport io - respectively to the first argument messIO() and tr - object of the "OutTransp".

- Calling the procedure for execution which after the processing the "io" structure forms the direct request to the transport *tr.messIO(req);*, result of which is processed and put back in io.

The essence of the allocation the protocol part of the code to the procedure of the user protocol is to facilitate the interface of the client exchange for multiple use and assumes the formation of the structure of XML-node of the exchange as the attributes of the addresses of remote stations, addresses of the read and write variables and the values of the variables themselves. The entire work of direct coding of the request and decoding of the response is assigned to procedure of the user protocol.

As an example, consider the implementation of the requests by protocol DCON, to the handler, implemented in the previous section. Let's start with the implementation of the protocol part:

```
//Result request prepare
request = io.name().slice(0,1)+io.attr("addr").toInt().toString(16,2)+io.text();
CRC = 0;
for(i=0; i < request.length; i++) CRC += request.charCodeAt(i);
request += (CRC&0xFF).toString(16)+"\r";
//Send request
resp = tr.messIO(request);
while(resp[resp.length-1] != "\r")
{
      tresp = tr.messIO("");
      if(!tresp.length) break;
      resp += tresp;
}
//Analysis response
if(resp.length < 4 || resp[resp.length-1] != "\r") { io.setAttr("err","10:Error or
no response."); return; }
//Check response to the integrity (CRC)
CRC = 0;
for(i = 0; i < (resp.length-3); i++) CRC += resp.charCodeAt(i);
if(CRC != resp.slice(resp.length-3,resp.length-1).toInt(16))
{ io.setAttr("err","11:CRC error."); return; }
if(resp[0] != ">") { io.setAttr("err","12:"+resp[0]+":DCON error."); return; }
//The result return
io.setAttr("err","");
io.setText(resp.slice(1,resp.length-3));
```

And the procedure is immediate dispatch DCON request, through the previous procedure protocol. This procedure should be put in the necessary task or an intermediate function OpenSCADA, such as the procedure of the controller DAQ.JavaLikeCalc:

```
//Request prepare
req = SYS.XMLNode("#").setAttr("ProtIt","DCON").setAttr("addr",10);
//Send request
SYS.Transport["Serial"]["out_TestDCON"].messIO(req,"UserProtocol");
if(!req.attr("err").length) SYS.messDebug("TEST REQ","RES: "+req.text());
//Second request prepare
req = SYS.XMLNode("@").setAttr("ProtIt","DCON").setAttr("addr",10);
//Send second request
SYS.Transport["Serial"]["out_TestDCON"].messIO(req,"UserProtocol");
if(!req.attr("err").length) SYS.messDebug("TEST REQ","RES: "+req.text());
```