F. Wagner July 2008

Using tcp/ip channels as an I/O interface

Tcp/ip channels in the RTDB

RTDB is a tcp/ip server. All attributes of each object in RTDB can be accessed by tcp/ip clients. A client connects to the RTDB server using two sockets: Request and Event. The Request (R) socket is used for queering of object attributes. The Event (E) socket delivers object attributes that are registered as “advise” in RTDB. A client requires the following functionality for establishing and controlling the communication with the server; it has to be able to:

- Connect to the server
- Disconnect from the server
- Register for events (Advise)
- Terminate events (Unadvise)
- Read data
- Write data
- Receive advised data (events).

The Read/Write data operation is accomplished by using the Request socket (see Fig. 1). A client Request message is answered with a Reply message containing the value of the required attribute. A client writes an attribute value into the RTDB by sending the Poke message.

In addition we have also a Security message for sending a password if applicable. The Security message is just a variant of the Write (Poke) message and does not play any role in I/O considerations.

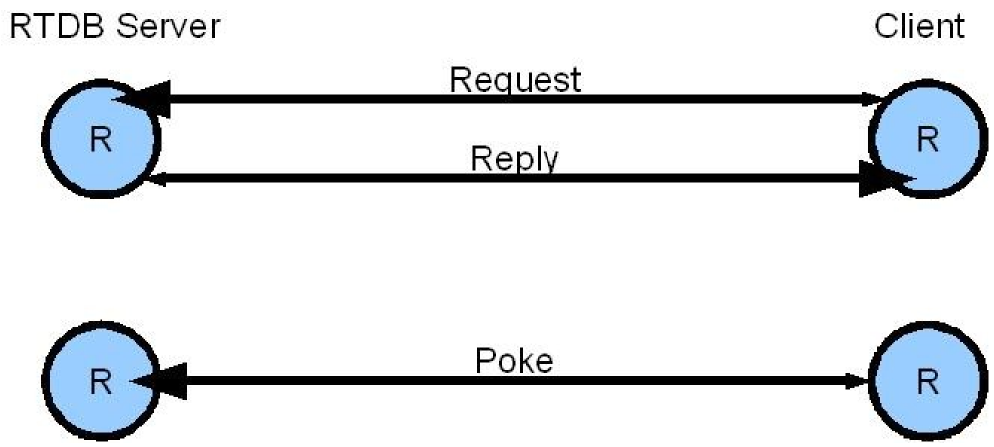

Fig. 1. Read and write data

If a client wants to have values continuously updated it has to register (advise) for events. It gets notified of any change of the advised attribute value via the event socket as an advised data (AdvData). Of course all advised clients get the change. A client registers for events by sending the AdviseStart message (see Fig. 2) that defines the object attribute to be advised. Any number of clients may be advised for an attribute. A client terminates the advise registration by sending the AdviseStop message that defines the object attribute to be unadvised.

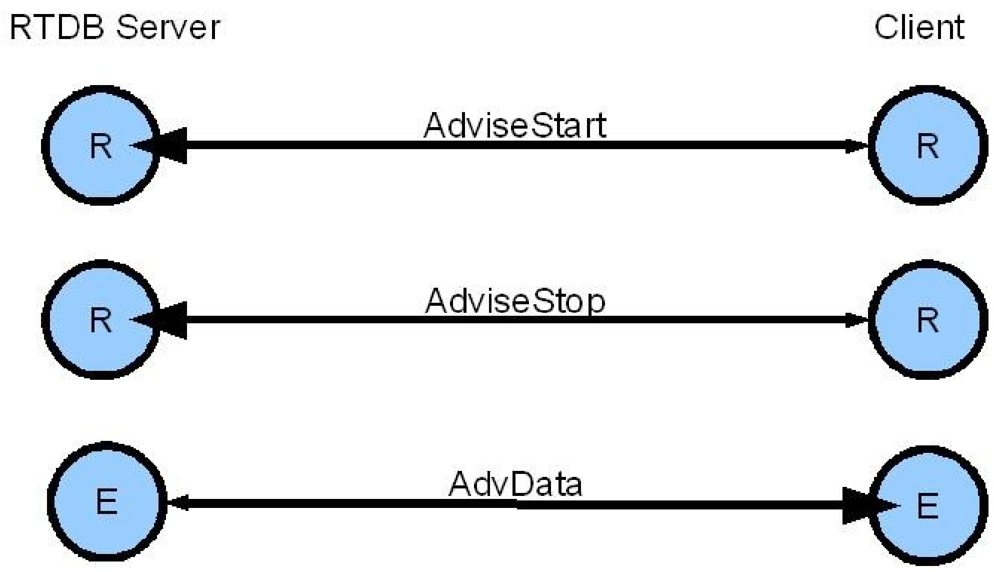

Fig. 2. Event data

Hence, if any client writes a new attribute value all advised clients get the corresponding change via their event sockets. Fig. 3 Shows a Poke message that changes an attribute value. In consequence all clients that are registered for events receive AdvData messages on its event sockets containing the changed value.

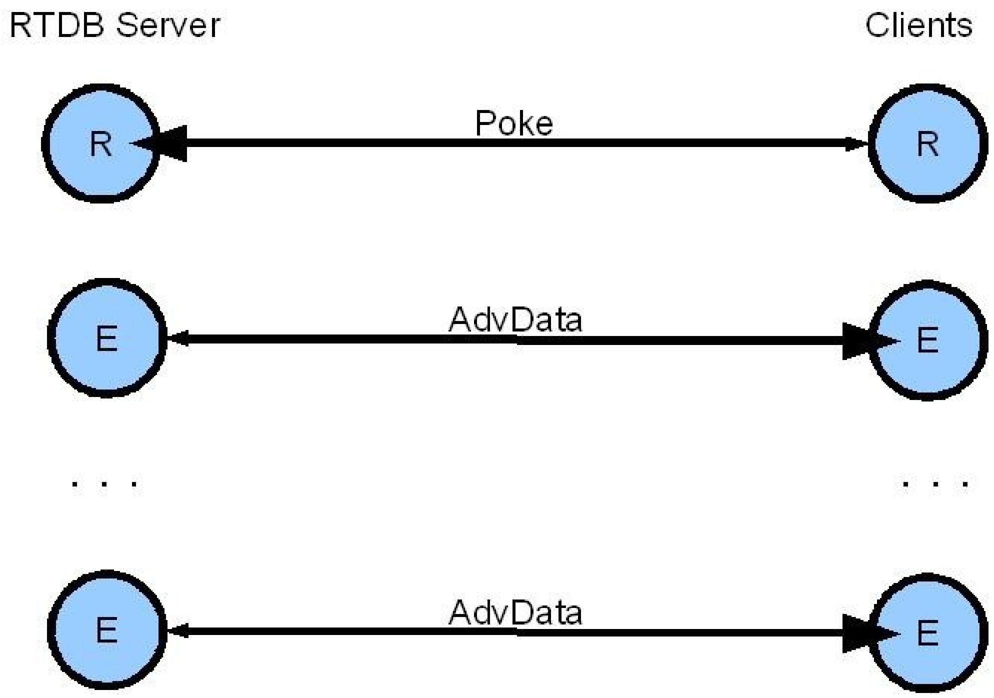

Fig. 3. Events generated after Poke

The concept of Raw Data

For the discussed topic only the data attributes (.Dat), of the many attributes normally existing, are relevant (I/O Handler transfers input/output data). A User Interface is in principle interested in RTDB data that are transformed into control values determining the (state machine) behavior and output values generated by (state machine) actions. To use the tcp/ip communication scheme for I/O purposes we need also true input values that are coming from the controlled devices as well as true output values delivered to the controlled devices. We call them Raw data and they are represented by an attribute Iatt_RawData (.Raw). In some cases the Raw data is not even stored in RTDB and must be generated at the moment if it is needed. For instance, the Data value of the object NI is stored as a value converted according to scaling and offset properties. Hence, the value supplied by a client that works as an I/O Handler must be provided in the raw form exactly as in the I/O Handler. Similarly, the Data value of a NO object does not exist in RTDB: it is a Data value of the object defined in its property Out Data. For a transfer to the client serving as an I/O Handler it must be generated as a Raw data.

Having these considerations in mind we use for the I/O communication attributes shown in Table 1. The table contains only those objects that make sense for I/O Handler implementation.

Table 1: Raw data for tcp/ip communication

Object type	**Attribute**	**Used as**	**Attribute Property**	**Comment**
CMD	.PeV	Input	Read/Write	Peripheral value is used as Raw data
XDA	.Val	Input	Read/Write	Value is used as Raw data
XDA	.Val	Output	Read	Value is used as Raw data
PAR DAT	.Dat	Input	Read/Write	Data value is used as Raw data.
DI	.Raw	Input	Write only	
DO	.Raw	Output	Read only	
NI	.Raw	Input	Write only	
NO	.Raw	Output	Read only	

The classic I/O signals like inputs represented by DI (digital) and NI (analog) as well as outputs represented by DO (digital) and NO (analog) have a special Raw data attribute that is used while transferring the raw data. When objects of type PAR or DAT are used for representing input signals they do not need any special Raw data attribute; the Data (.Dat) attribute will do because that value is not transformed in any way and is just stored in the form as received from the I/O Handler. The same remark applies to the object of type XDA: its Value (.Val) attribute represents *the* data. For an object of type CMD we can use its Peripheral value (.PeV) as an (input) raw data.

Note that the true Raw values can be only written for Inputs (DI, NI) or read for outputs (DO, NO). On the other hand the Raw values for objects CMD, XDA, PAR and DAT can be written and read independently of their true character (Input or Output) because they do exist in RTDB. Of course for the XDA object used as an Output writing a value does not make sense and we should only read its value. Anyhow, the read feature has an auxiliary character because all outputs should be registered by interested clients to be advised, so as to be continuously updated via their event socket, thus there is no need for reading them.

Implementation details

The implementation of the sockets is a bit more complex than the principal sequences shown in the figures above. For instance, a reliable message transfer requires acknowledgment of sent messages. We do not go into these details here.

The message format used is:

```
struct r_MessagePack
{
  int                    nHeader;
  e_MessageType          MessageType;
  e_Topic                Topic;
  e_ItemAttributes       IAtt;
  e_ItemSearchResults MessageInfo;
  char                   cItemName[NAMESIZE];
  char                   cItemValue[VALUESIZE];
};
```

The present implementation limits the length of the object name (including attribute) to 64 characters (NAMESIZE) and the size of the data to 300000 (VALUESIZE) characters. VALUE SIZE covers the entire package size; thus the effective limit of the data size is about 299914 characters. Of course these limit values can be changed at any time if desired.

Sending and receiving messages is a relatively complex task (preparation of the data records, conversion between the host to TCP/IP network byte order and especially the implementation of the thread for event messages). For developer convenience we provide a library tcpipclient.lib that offers a set of functions to be used when programming a RTDB client. These functions are declared in the tcpipclient.h file.

The StateWORKS monitors (SWMon and SWQuickPro) are in principle User Interfaces. For test purposes we have extended the SWQuickPro functionality by reading/writing of Raw data as appropriate. The extended functionality is available starting from StateWORKS version 7.2.

Conclusion

The primary goal of the server/client model of the RTDB has been communication with a User Interface. When the tcp/ip communication began to be used for input/output purposes we could extend the StateWORKS server/client model very easily by completing attributes of corresponding object by a concept of RawData. The completion of Raw data are required for objects: DI, DO, NI, NO, the objects CMD, XDA, PAR and DAT possessing per se the raw data values.

The imaginable future extensions of our server/client model for I/O purpose may include a transfer of larger message packets containing data of many objects. The content of such a packet could be defined in an I/O Unit. It may make sense for an I/O Handler using a polling method for input data acquisition.