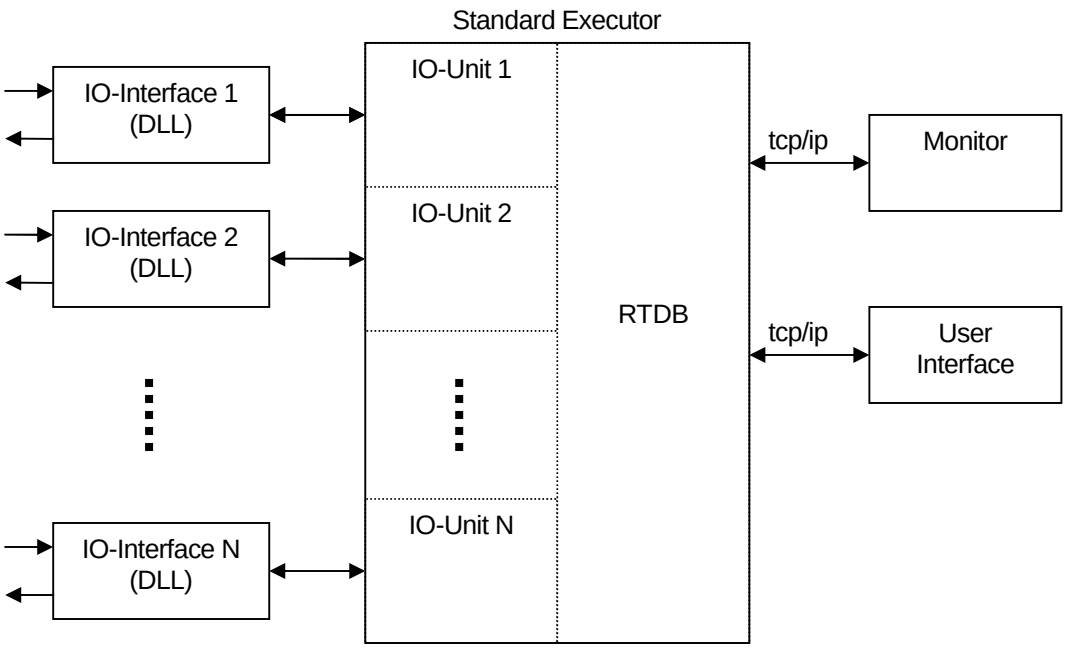
Standard Interface for StateWORKS Standard Executor

Introduction

The technical note describes the concept of Standard IO-Unit which is used to implement IO-interface with DLLs.

The StateWORKS execution environment has been originally meant as a framework for application development. The RTDB library is used to build an application. The application requires two additional components: the IO-Interface and the User-Interface. The User-Interface can be, theoretically, built as a part of RTDB but in practise it is developed as a separate client-program using tcp/ip link to the RTDB-Server (the old implementation used also the DDE link). The IO-Interface is developed using RTDB class methods which are documented in the "References for the StateWORKKS Class Library".

Introducing DLL as the IO-Interface we separate also this part from RTDB. A StateWORKS based control system is then composed from separate components shown in the Figure below.



The heart of the system is the Standard Executor which contains the RTDB and IO-Units. Each IO-Unit is the same. The number of IO-Unit incarnations is defined by specifying the control system in the StateWORKS Studio. Though the code of IO-Units is the same the number of inputs and outputs may differ and is configured by system specification.

DLL

The DLL should contain functions which supply input values to the control system and pass output values to the controlled device. As a control system operates on numerical values (numbers) an extreme simple solution would be to have two functions: Read() and Write correspondingly for input and output values. Using this solution it would be difficult to exploit fully the power of RTDB – its feature to generate control signals from the input values and to produce different output types. Therefore, we have defined a set of functions which assures a proper usage of RTDB objects. The choice of functions is based on our experience but it is not

closed. If we find a reason to expand the set of DLL functions in the future or if a user has a special requirement we can always add new functionality. The expansion is a rather simpler task and could be done rapidly.

The StdDll.h file contains declarations of C- functions that have to be implemented in the DLL. There are 3 groups of them.

The first group contains:

```
bool OpenDevice( long lAddress);
bool CloseDevice();
```

These functions are used by the IO-Unit to open and close the device driver. The device driver is opened by a start of the application. If the device driver cannot be opened the application exits. The device driver is closed when the application exits.

The second group contains “read” functions:

```
bool ReadBool( long* lValue, long lChannel);        // DIs
bool ReadShort( short int* iValue, long lChannel ); // XDAs
bool ReadLong( long* lValue, long lChannel );       // NIs
bool ReadString( char** stValue, long lChannel );   // DATs
```

These functions are used by the IO-Unit to poll input values. Each function must be written taking into account its usage in the IO-Unit:

- The `ReadBool()` is used to deliver values of DI objects which is clear defined.

- The `ReadShort()` is used to deliver values of XDA objects. The meaning of the XDA values is application dependent.

- The `ReadLong()` is used to deliver values of NI objects which meaning is clear defined. The values are normally read from D-A converters. We have increased the resolution: instead of the `short int` type which has corresponded to 16-bit converters we use now the `long` type (no limitations on resolution).

- The `ReadString()` is used to deliver strings which are written into DAT objects. The meaning of the DAT values is application dependent.

In addition, there is also the function:

```
bool ReadSingleBool( bool* bValue, long lChannel ); // not used
```

defined which could be used to poll single digital input channels. In this moment, it is not used. Maybe, in the future we will find a justification for it.

The third group contains “write” functions:

```
bool WriteSingleBool( bool bValue, long lChannel ); // DO
bool WriteShort( short int iValue, long lChannel ); // CMDs
bool WriteLong( long lValue, long lChannel );       // NOs
bool WriteString( char* stValue, long lChannel );   // TABs
```

These functions are used by the IO-Unit to output values. Each function must be written taking into account its usage in the IO-Unit:

- The `WriteSingleBool()` is used to write the value of a DO object to the device driver. It writes one single value at a time to a given channel.

- The `WriteShort()` is used to write the value of a CMD object to the device driver. The IO-Unit may have only one CMD object; therefore the lChannel has always the value 0. On demand it may be expanded. The meaning of the values is application dependent.

- The `WriteLong()` is used to write the value of a NO object to the device driver. The meaning of the values is clear defined. We have increased the resolution: instead of the `short int` type which has corresponded to 16-bit converters we use now the `long` type (no limitations on resolution).

- The `WriteString()` is used to write a string to the device driver. The string is an output of a TAB object. The IO-Unit can have only one TAB object; therefore the lChannel has always the value 0. On demand it may be expanded. The meaning of the string is application dependent.

A DLL must have implemented all the above functions. The function which are not required in an application just return *false*.

The actual form of the DLL function declaration file depends on the development environment. For instance, an h-file used for MS Visual Studio could have a form shown in the file StExample1.h.

IO-Unit

The IO-Unit for the DLL may access the following object types: DI, NI, XDA and DAT for read operations and DO, NO, CMD and TAB for write operation. In addition, the IO-Unit must contain 8 alarm objects; each alarm linked with a corresponding function (any failure by a read or write operations is signalled by an alarm). Hence, the minimum content of the IO-Unit is:

```
   1     Par_DllName                    11  must-not-be-changed
   2     Al_ReadDiError                  4  must-not-be-changed
   3     Al_ReadXdaError                 4  must-not-be-changed
   4     Al_ReadNiError                  4  must-not-be-changed
   5     Al_ReadDatError                 4  must-not-be-changed
   6     Al_WriteDoError                 4  must-not-be-changed
   7     Al_WriteCmdError                4  must-not-be-changed
   8     Al_WriteNoError                 4  must-not-be-changed
   9     Al_WriteTabError                4  must-not-be-changed
  10     Par_PollingTime                11  must-not-be-changed
```

By specifying the IO-Unit the following rules must be kept:

- 10 object names (2 parameters and 8 alarms) must not be changed or removed (see a remark in the Description field)..

- The first (`Par_DllName`) and the last (`Par_PollingTime`) name and its position must not be changed.

- Any number of object names may be inserted between the first (`Par_DllName`) and the last (`Par_PollingTime`) name. The order does not play any role.

The StandardIO.unt file contains a sample of the unit definition for the standard IO-Unit. The IOD-file has the following content:

```
H G:\StateWORKS\Projects\Examples\Standard\Conf\StandardUnit.iod

B  #     Name    - Object List -        Type  Description
   1     Par_DllName                      11  must-not-be-changed
   2     Al_ReadDiError                    4  must-not-be-changed
   3     Al_ReadXdaError                   4  must-not-be-changed
```

4	Al_ReadNiError	4	must-not-be-changed
5	Al_ReadDatError	4	must-not-be-changed
6	Al_WriteDoError	4	must-not-be-changed
7	Al_WriteCmdError	4	must-not-be-changed
8	Al_WriteNoError	4	must-not-be-changed
9	Al_WriteTabError	4	must-not-be-changed
10	Cmd	2	
11	Di_0	5	
12	Di_1	5	
13	Di_2	5	
14	Di_3	5	
15	Di_4	5	
16	Di_5	5	
17	Di_6	5	
18	Di_7	5	
19	Do_0	6	
20	Do_1	6	
21	Do_2	6	
22	Do_3	6	
23	Do_4	6	
24	Do_5	6	
25	Do_6	6	
26	Do_7	6	
27	Ni_0	8	
28	Ni_1	8	
29	Ni_2	8	
30	Ni_3	8	
31	No_0	7	
32	No_1	7	
33	No_2	7	
34	No_3	7	
35	Dat_1	15	
36	Dat_2	15	
37	Xda_0	10	
38	Xda_1	10	
39	Tab	19	
40	Par_PollingTime	11	must-not-be-changed

Application

Each IO-Unit has a parameter Par_DllName. It is a string which is the name of the Dll which is to be used by creating the IO-Unit and loading the DLL. The name may be written with or without the extension (.dll). An application may have several IO-Units which use several DLLs. A DLL may be used for many incarnations of IO-Units.

Each IO-Unit has also a parameter Par_PollingTime which defines the frequency of polling (in ms) for a given unit.

Summary

Using DLL as the IO-Interface is for most applications an advantage. The Standard Executor is then for the user a “black box” which just runs on the computer. With a set of DLLs the user approaches the goal of the VFSM and StateWORKS concept – to have a ready-made execution environment. The user may concentrate fully on the behaviour specification. The dream of the true executable specification becomes then a reality.