WP9 Validation on Platform

Deliverable D9pc.3
Process Control Application

WP9 Validation on Platform : Deliverable 9pc.3 Process Control Application
By Stanislav Benes, Petr Cvachoucek, and Ales Hajny

Published November 2003

Table of Contens

Document Presentation

Project Coordinator

Organisation:UPVLC
Responsible person:Alfons Crespo
Address:Camino Vera, 14, 46022 Valencia, Spain
Phone:+34 963877576
Fax:+34 963877576
Email:alfons@disca.upv.es

Participant List

Role	*Id.*	*Participant Name*	*Acronym*	*Country*
CO	1	Universidad Politecnica de Valencia	UPVLC	E
CR	2	Scuola Superiore Santa Anna	SSSA	I
CR	3	Czech Technical University in Prague	CTU	CZ
CR	4	CEA/DRT/LIST/DTSI	CEA	FR
CR	5	Unicontrols	UC	CZ
CR	6	MNIS	MNIS	FR
CR	7	Visual Tools S.A.	VT	E

Document version

Release	*Date*	*Reason of change*
1_0	10/11/2003	First release

Chapter 1. Introduction

Process control application is modular software solution running in embedded system supervised from attached development host running on standard PC workstation. The whole software system consist of base modules including control algorithm interpreter, ethernet communication subsystem and CANopen remote IO subsystem. Software modularity is based on open interfaces allowing easy connection of any new subsystem to algorithm interpreter.

The set of interfaces was developed in OCERA project and it allows connection of basic subsystems to interpreter according IEC 61131 norm.

Figure 1 shows block diagram of process control node. Particular application can consist of number of nodes.

List of developed components

- Ucnd library - class library for node shared data access
- Execom node communication subsystem
- CanIO CANopen remote IO subsystem manager

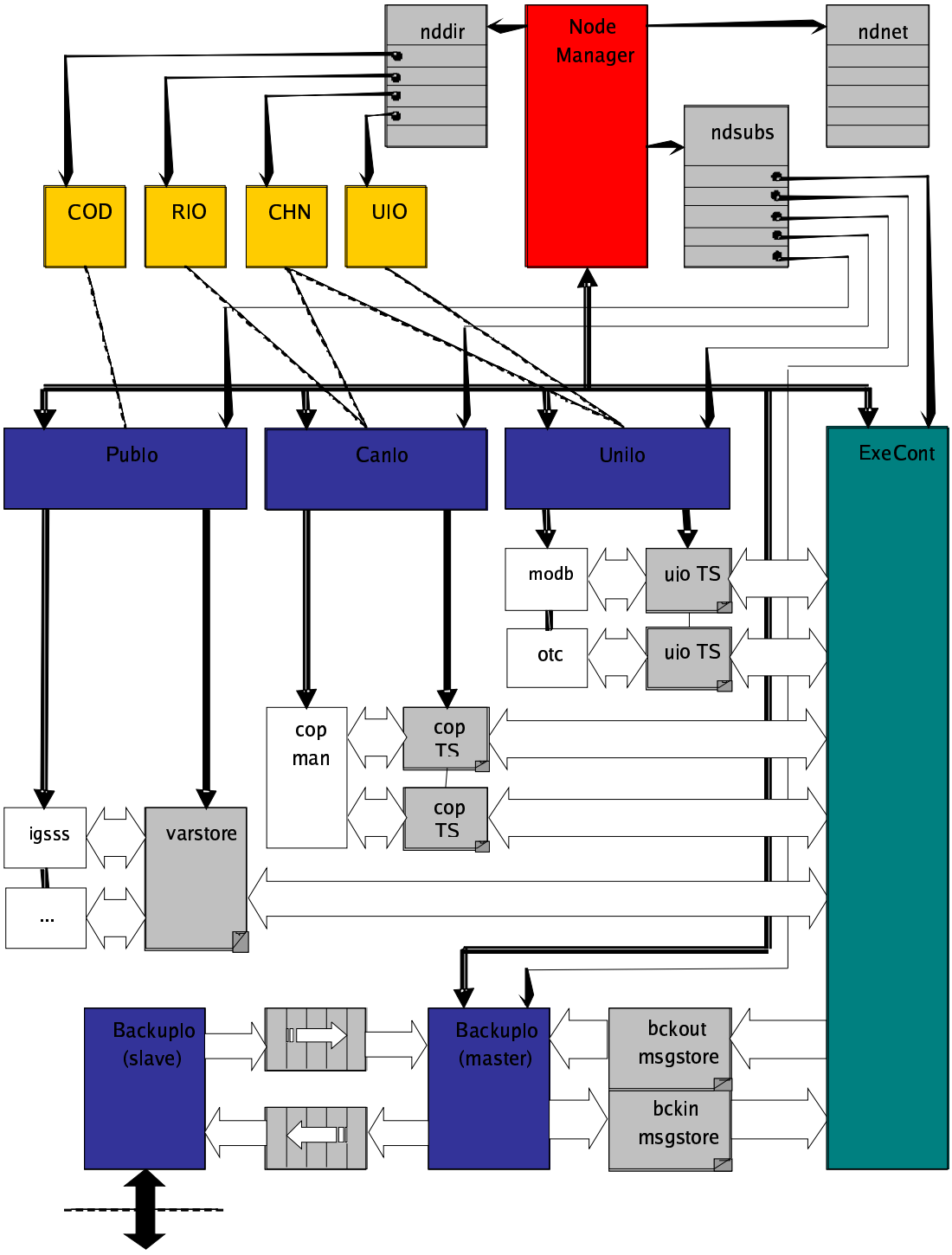

Figure 1 - node block diagram

Chapter 2. Ucnd IO interface library component

2.1.Summary

- Name
  Ucnd library
- Description
  class library for node shared data access
- Author
  Petr Cvachoucek
  cvachoucek@unicontrols.cz
- Reviewer
  Zdenek Kolisek
- Layer
  Application
- Version
  1.0
- Status
  stable
- Dependencies
  gcc 3.3.2, X86 and PPC platform
- Release date (milestone)
  7.10.2003

2.2 Description

Ucnd library is class library aimed for node shared data access. It contains common enumerate types definitions, which describe states and modes of node's actions, structures definitions and message format for communication with node manager.
Library defines namespace Ucnd. All classes and data types are defined in this namespace.

2.3 API / Compatibility

Enumerate type NdMode – node's actions modes

MODE_UNKNOWN	Node's actions mode unknown. (This constant has only sense during node initialisation phase, before node configuration is red. During runtime are actions modes of every node known).
MODE_STANDALONE	Node acts as unbackuped (standalone node).
MODE_BACKUP	Node acts in backuped pair of nodes.

```
// ----------------------------------------------------------------------
// definition of node operational modes

enum NdMode
{
 MODE_UNKNOWN                    = 0,        ///< mode is unknown (yet)
 MODE_STANDALONE                 = 1,        ///< stand-alone mode
 MODE_BACKUP                     = 2,        ///< backup mode - member of
backup pair
};
```

Node's actions mode is static parameter of node configuration and it is impossible to change it during runtime.

Enumerate type NdStatus – actual status of node or subsystem

```
// ----------------------------------------------------------------------
// definition of node/subsystem operational status

enum NdStatus
{
 STATUS_UNKNOWN        = 0,                ///< node status is unknown
 STATUS_STARTING       = 1,                ///< node is starting
 STATUS_READY                  = 2,                  ///< node is started
 STATUS_ACTIVE                 = 3,                  ///< node is active
 STATUS_STANDBY        = 4,                ///< node is standby
 STATUS_SHUTDOWN       = 5,                ///< in shutdown
 STATUS_FAILURE        = 6,                ///< fatal failure
 STATUS_STOPPED        = 7,                ///< node is stopped
};
```

STATUS_UNKNOWN	Status of node is unknown, node/subsystem isn't running.
STATUS_STARTING	Node/subsysten is starting. Start of node is controled by node manager, which starts particular subsystems in given order and interconects them into the node subsystem table. After its start node propagates to the next state.
STATUS_READY	Node/subsystem is started and ready.
STATUS_ACTIVE	Node/subsystem is active (control algorithm is running). In backuped node mode there is always only one of pair of nodes in this state.
STATUS_STANDBY	Node/subsystem is stand-by (control algorithm isn't running, node follows active partner and waits for his failure).
STATUS_SHUTDOWN	Node/subsystem is switched-off.
STATUS_FAILURE	Node/subsystem failed to start.
STATUS_STOPPED	Node was stopped by operator's command.

Node status is changing during runtime. Node manager decides about node status changes, which commands subsystems to transit between particular states too.

Class NdDirectory – access to parametrisation module table

Class NdDirectory serves as access to parametrisation data module tabel of node. Node manager takes care of this table, the rest of subsystems can the table search through according to their specific needs.

Standard data module name for the table storage is defined by macro NDDIR_NAME (in file *ucnd.h*).

Each module is described in the table by this structure:

```
struct Module
{
 char    name[32];          ///< module name
 int     type;              ///< module type
 int     key;               ///< module key (unique id number)
 int     version;           ///< module version
 int     status;            ///< module status
};
```

name	Name of parametrisation module. The name is restricted to 31 characters + '\0' character.
type	Parametrisation module type. This parametrisation module types are predefined: TDCT_UNDEF 0 type not defined TDCT_NTD 1 NTD parametrisation module TDCT_CHN 2 CHN parametrisation module TDCT_COD 3 COD parametrisation module TDCT_LIO 4 LIO parametrisation module TDCT_RIO 5 RIO parametrisation module TDCT_UIO 6 UIO parametrisation module TDCT_TASK 32 ExeCont interpreter task
key	Jey of parametrisation module (unique id of module).
version	Parametrisation module version.
status	Parametrisation module status.

NdDirectory::Create – creation of module table

Create method creates module table in data module of given name. Module table is created by node manager in data module, which name is defined by macro NDDIR_NAME, the rest of process and subsystems connects to the table by Open method.

```
void Create(String dirName, int nMods)
```

Parameters:

dirName Data module name, where table will be created.
nMods Maximum number of modules in table.

NdDirectory::Open – connection to the module table

Open method opens module table in data module module of given name. Module table is created by node manager in data module, which name is defined by macro NDDIR_NAME, the rest of process and subsystems connects to the table by Open method.

```
void Open(String dirName)
```

Parameters:

dirName Data module name, where table will be created.

NdDirectory::Close – termination of work with module table

Close method serves for termination of work with module table, it performs unlinking from data module with table.

```
void Close()
```

NdDirectory::AddModule – adding module to table

AddModule method serves for module enlisting to the table. Modules are enlisted to the table by node manager.

```
int AddModule(const char * name,
        int type, int key, int version, int status)
```

Parameters:

name	Parametrisation module name.
type	Parametrisation module type.
key	Parametrisation module key.
version	Parametrisation module version.
status	Parametrisation module status.

Return value:
Method returns id of module, under which is module registered in the table.

NdDirectory::RemoveModule – removal of the module from table

RemoveModule method serves for removal of parametrisation module from module table. Modules are removed by node manager.

```
bool RemoveModule(int mid)
```

Parameters:

mid	Id of module to remove from the table

Return value:
Method returns true in case the module was found in the table and was removed. In case there is no module with given mid in the table, method returns false.

NdDirectory::FindModule – looking for module in the table

FindModule method serves for looking up for modules in the table according to given criteria. Calling process specifies module name, which can contain wilscards '*' and '?'. Moreover the type and key of looked up module can be specified.

FindModule method can be used iteratively to find all modules compliant to given specification in the table.

```
int FindModule(int mid, Module & mod, const char * mask = "*",
        int type = -1, int key = -1)
```

Parameters:

mid	Module id, from which the table will be looked through. -1 for search from the beginning.

mod	Reference to the structure, which will be filled with description of found module.
mask	Name of looked for module, which can contain wildcards '*' and '?'
type	Type of looked for module, value -1 for any type of module.
key	Type of looked for module, value -1 for any key.

Return value:
Method returns id of found module, which can be used for next call of FindModule method to find next module compliant to specification. In case no next compliant module is found, method returns -1.

NdDirectory::GetModule – getting of module atributes

GetModule method serves for getting of atributes of module with given id.

```
bool GetModule(int mid, Module & mod)
```

Parameters:

mid	Id of asked module.
mod	Reference t the structure, which will be filled with found module description.

Return value:
Method returns true in case module was found in the table. In case the table doesn't contain the module with given mid, method returns false.

NdDirectory::GetModuleStatus – reading of module status

GetModuleStatus method returns status of given module.

```
int GetModuleStatus(int mid)
```

Parameters:

mid	Id of asked module.

Return value:
Method returns status of given module, in case the module isn't in the table, returns -1.

NdDirectory::SetModuleStatus – setting of module status

SetModuleStatus method sets status of given module.

```
bool SetModuleStatus(int mid, int status)
```

Parameters:

mid	Id of asked module.
status	New module status.

Return value:
Method returns true, in case status was set for the module. In case the module with given mid isn't in the table, method returns false.

Class NdNetTable – access to the table of node in network status.

Class NdNetTable serves for access to the table containing basic information about own node and other nodes in Unicon stations network.

Standard data module name for the table storage is defined by macro NDDIR_NAME (in file *ucnd.h*).

Each module is described in the table by this structure:

```
struct Node
{
	int		nodeNum;		///< node number
	NdMode	mode;			///< oper. mode (stand-alone, backup)
	NdStatus	status;			///< node status (active, standby ...)
	int		arch;			///< architecture (68k, x86, ...)
	int		bckNodeNum;		///< backup node num.(0 - standalone)
	int		commState;		///< comm.channel states (bitwise or)
};
```

Chapter 3. Execom node communication subsystem component

3.1.Summary

- Name
  execom
- Description
  node communication subsystem (ethernet and serial line)
- Author
  Dejan Hrnjica
  hrnjica@unicontrols.cz
- Reviewer
  Zdenek Kolisek
- Layer

Application

- Version
  1.0
- Status
  stable
- Dependencies
  gcc 3.3.2, X86 and PPC platform
- Release date (milestone)
  7.10.2003

3.2 Description

Execom communication subsystem allows number of clients to send data to another client in another or the same node and receive data.

3.3 API / Compatibility

Class Interface
Methods:

- ***GetPriority***
- ***SendFrame***
- ***RcvFrame***

Třída Chanel
Methods:

- ***SendFrame***
- ***OnRcv***
- ***OnTmo***
- ***IsAlive***
- ***Alive***
- ***GetPriority***

Class NodeChanels
Methods:

- ***SendFrame***
- ***ChangeChanel***
- ***AnyAlive***
- ***IsAlive***
- ***OnRcv***
- ***OnTmo***

Chapter 4. CanIO CANopen remote IO subsystem manager

4.1.Summary

- Name
  canio
- Description
  CANopen remote IO subsystem manager
- Author
  Pavel Mraz
  mraz@unicontrols.cz
- Reviewer
  Petr Cvachoucek
- Layer
  Application
- Version
  1.0
- Status
  under developement
- Dependencies
  gcc 3.3.2, X86 and PPC platform
- Release date (milestone)
  31.1.2004

4.2 Description

CanIO subsystem interconnects interpreter with remote IO using virtual can API component.

Proces canio has this basic functions in node:

- creates and initializes data modules for CANopen network (based on parametrisation in RIO)
- starts and controls driver/manager for CANopen networks (copman),
- responds to node manager commands

Chapter 5. Tests and validation

5.1 Test 1: Validation of proper interpretation on X86 type node

Validation criteria: Check of proper interpretation of objects implemented in FPD and check of interpretation of commands of programing language ST on X86 type node.

Expected result: All the tests successfully acomplished.

Step 1: on host developement environment configure ODBC for running of database ApdGATD
Step 2: on host developement environment start UniCap projection system, compile database (note: in the TASK form have to be node 61 selected for compilation)
Step 3: run UniTun mode
Step 4: check successful acomplishing of all tests in block TEST on X86 (signalised by green color of displaying element)

5.2 Test 2: Validation of communication with remote IO

Validation criteria: Check communication of boards with remote inputs/outputs (digital and analog)

Expected result: Successful acomplishing of all tests in IO_TESTY block

Configuration on proces control node:
Step 1: configuration of boards with remote inputs and outputs
Step 2: configuration of VMOD-2D board with piggyback VCAN
Step 3: interconnect inputs and outputs acording to Figure 2

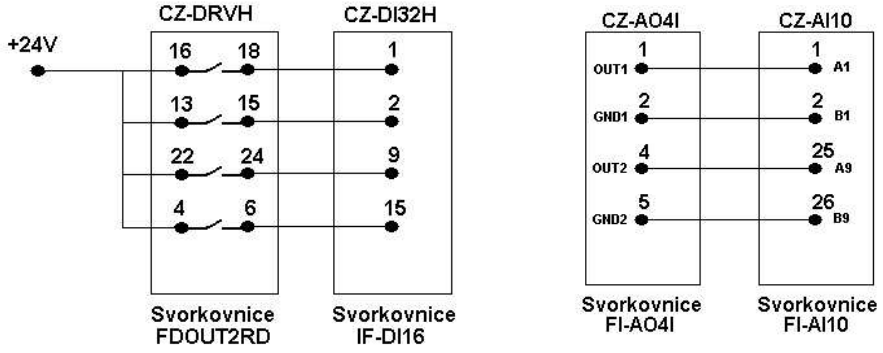

Figure 2 - interconnection of digital and analog IO

Step 4: configure SW in proces control station (properly set up ethernet communication and CANopen subsystem)

Configuration of host development environment:
Step 5: configure ODBC for database ApdGATD running
Step 6: configure communication channel in execom.cfdg file
Step 7: interconnect host development environment with proces control node
Step 8: on host developement environment start UniCap projection system, compile database (note: in the TASK form have to be node 1 selected for compilation).
Step 9: load compiled database to the proces control node
Step 10: save compiled database to the proces control node
Step 11: reset proces control node
Step 12: run UniTun mode after proces control node start
Step 13: check successful acomplishing of all tests in block IO_TESTY (signalised by green color of displaying element)

Remark: Acomplishing of digital IO tests lasts approximately 30 sec.

5.3 Results and comments

No.	Test	Result
1	Validation of proper interpretation on X86 type node	passed
2	Validation of communication with remote IO	under testing

Chapter 6. Installation instructions

List of processes and utilites

For proces control node runtime are necessary those processes:

ndman	node manager
execom	node communication subsystem (ethernet and serial line)
execont	control algorithm interpreter
canio	CANopen remote IO subsystem
igsssrv	IGSS visualisation runtime server

Node manager configuration

Proces ndman acts as node manager and has following command line options:

```
usage: ndman [options]
options:
```

```
  -node=<num> node number (override value from cfg)
  -mode=<m>   operational mode (override value from cfg)
  -cfg=<file> load cfg from file <file>
  -cfm=<mod>  load cfg from module <mode>, def. is 'nodecfg'
  -quiet      do not print anything to stdout or stderr
  -?          print program usage
modes:
  standalone  runs as stand-alone unicon node
  backup      runs as member of backup pair of nodes
```

If there is no node number and operational mode specified on command line, it is read from particular items in configuration module or file.

Node initialisation – subsystem start

During node startup are particular subsystems started by node manager according to given configuration. Node manager counts timeout for each subsystem's start.

Subsustems are divided into separate startup phases. Subsustems in the same startup phase are started in parallel, phases are performed sequentially. Startup phase is finished in the moment when all subsystems started in this phase successfully started – they have to announce ready status – or timeout for their start expired.

Subsystem division into startup phases allows to create right sequence of sybsystem starting in case subsystems are dependent on each other.

Node control from command line

In order to control node in runtime from command line, utility ndview is used.

```
usage: ndview [options]
options:
  -subs       print subsystems table
  -net        print node table
  -dir        print directory contents
  -add        add module to directory
  -load       load module from file to directory
  -name=<n>   name of loaded/added module
  -remove     remove module from directory
  -mid=<mid>  module id of removed module
  -type=<t>   type of loaded module
  -key=<k>    key value of loaded module
  -ver=<v>    version of loaded module
  -status=<s> status of loaded module
  -save       save module directory to disk
  -flash      flash module directory to rom
  -stop       stop running node
```

```
  -run         run stopped node
  -watch       run in watch mode (periodic update)
  -per=<per>   set watch period in msec (def. 1 sec)
  -?           print program usage
```

Start/stop of node runtime

In order to stop node use command:

```
ndview -stop
```

In order to start node use command:

```
ndview -run
```