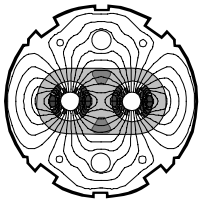
CERN
CH-1211 Geneva 23
Switzerland



the
Large
Hadron
Collider
project

LHC Project Document No.
LHC-UNICOS

CERN Div./Group or Supplier/Contractor Document No.
AB/CO

EDMS Document No.

Date: 2004-6-22

FUNCTIONAL SPECIFICATION

UNICOS SYSTEMINTEGRITY DEVICE TYPE

Abstract

This document describes the internal structure of the systemIntegrity device type, its configuration, the internal behavior and the procedure to add new systemIntegrity device types.

Prepared by:	***Checked by:***	***Approved by:***
UNICOS core team AB/CO		

History of Changes

Rev. No.	*Date*	*Pages*	*Description of Changes*
1.1 Draft	22-Jun-2004		First version

Table of Contents

1. INTRODUCTION

1.1 PURPOSE OF THIS DOCUMENT

The unicos-pvss package provides systemIntegrity device type dedicated to the cryogenics applications. New systemIntegrity device type can be added. This document describes the internal organization of the unicos-pvss package for the systemIntegrity device types. This document describes also the procedure to be followed to add a new systemIntegrity device type.

1.2 DEFINITIONS, ACRONYMS AND ABBREVIATIONS

From this point onwards, the following acronyms will be used:

- DPT: PVSS data point type.
- DP: PVSS data point, instance of the DPT.
- DPE: PVSS data point element.
- DPE: PVSS data point element
- Device: this is a DPT in PVSS.

1.3 CONTACT AT CERN

Any problem can be reported to the following email address: UNICOS.Support@cern.ch.

2. SYSTEMINTEGRITY: CONCEPT

2.1 SYSTEMINTEGRITY COMPONENT

2.1.1 CONCEPT

The systemIntegrity component is composed of:

- PVSS scripts: scripts/unSystemIntegrity.ctl
- PVSS lib: scripts/libs/unSystemIntegrity.ctl
- A PVSS panel for configuration: panels/vision/systemIntegrity/systemIntegrityConfiguration.pnl (Figure 1)
- A PVSS panel for diagnostic: panels/vision/systemIntegrity/systemIntegrityOperation.pnl (Figure 2)

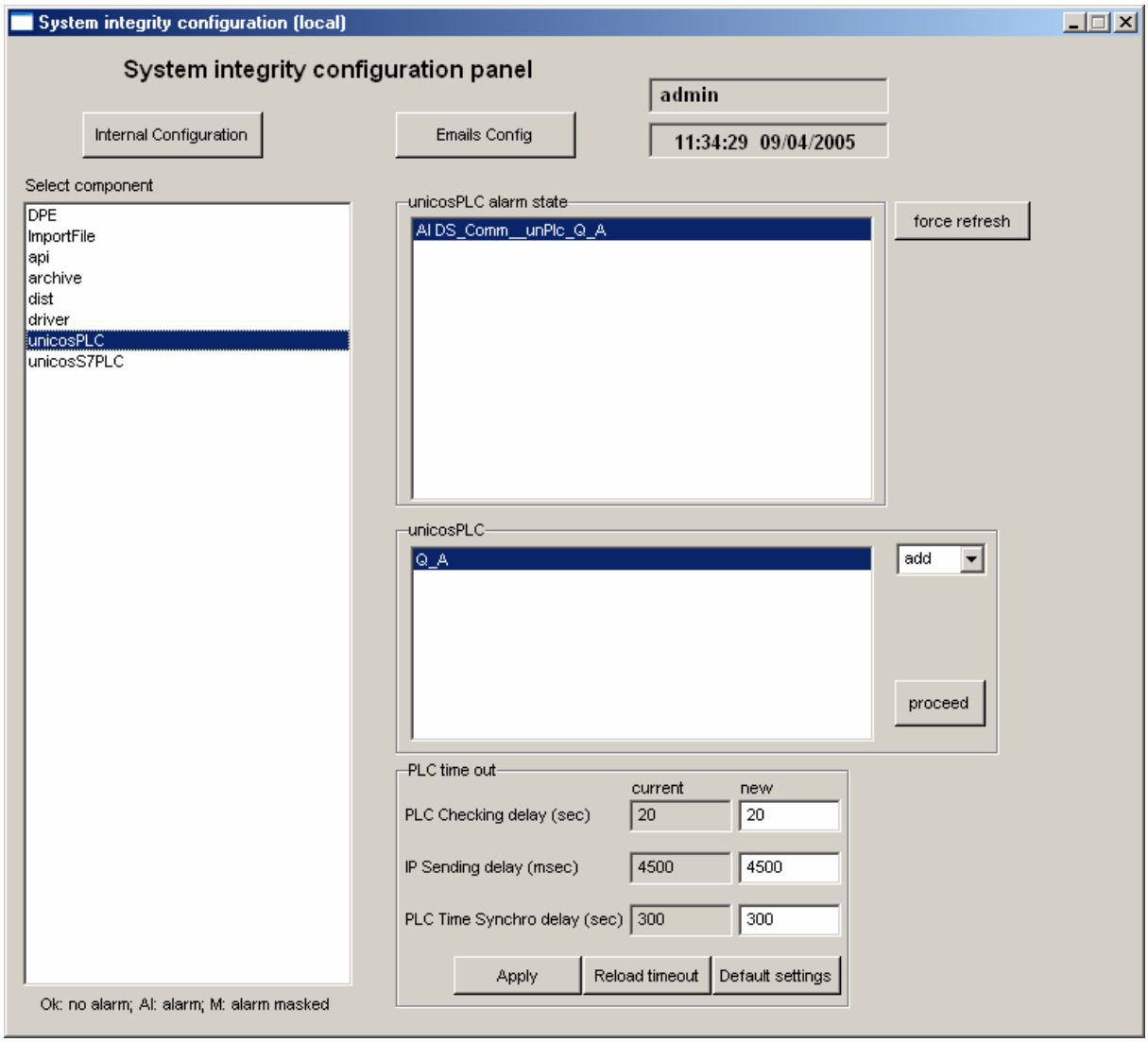


Figure 1: systemIntegrity configuration panel.

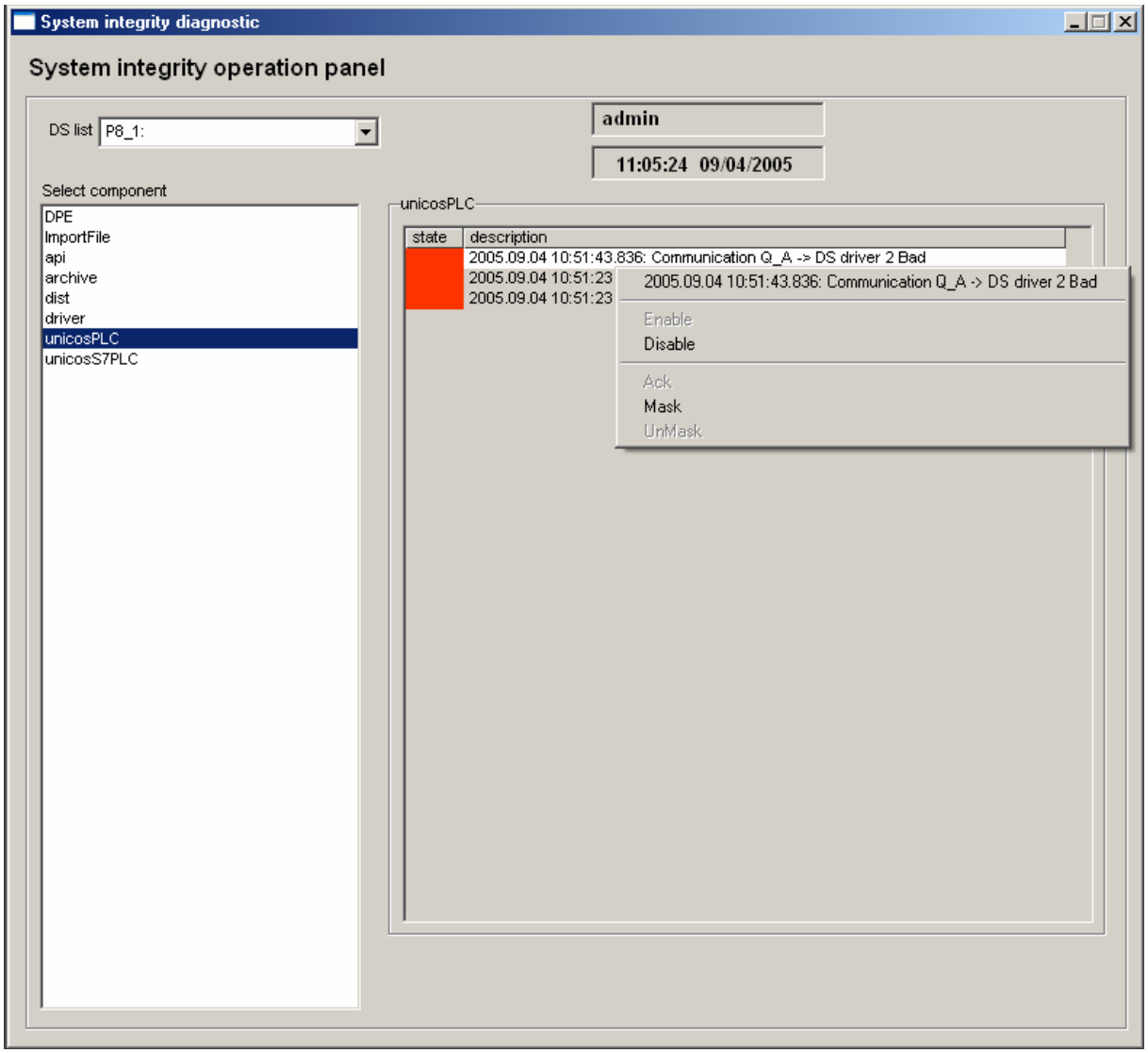


Figure 2: systemIntegrity diagnostic panel.

The systemIntegrity components in PVSS are data point of type `_UnSystemIntegrity` (Figure 3) with a name `deviceType_systemIntegrityInfo` (Figure 4). A systemIntegrity device type depends on the state of the device; it handles (set/reset) instances of `_UnSystemAlarm` data point type (Figure 5) for each device. The `_UnSystemAlarm` data point is usually used to set the state of the device for the systemIntegrity. The data point element `alarm` of the _UnSystemAlarm (Figure 5) represents the state of the alarm (this depends on the alert_hdl config), for instance: 0 no alarm, > 10 alarm. The `enabled` data point element of the _UnSystemAlarm (Figure 5) represents the state of the systemAlarm: true for systemAlarm enabled and false for systemAlarm disabled. The system integrity alarm list (Figure 7) shows the `_UnSystemAlarm` alarms. The convention for the `_UnSystemAlarm` data point name is `_unSystemAlarm_devicePattern_deviceName`, where _unSystemAlarm is the prefix, devicePattern depends on the device type and device name is the name of the current device, for instance for driver devicePattern is “drv” and device name is the driver manager number, for the archive devicePattern is “archive” and device name is the data point name of the archive: _ValueArchive_6. For the DPE component, the

`_UnSystemAlarm` is set to 10 whenever the DPE has not been updated since a certain period.

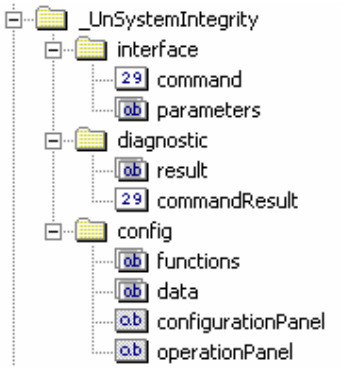


Figure 3: systemIntegrity data point type.

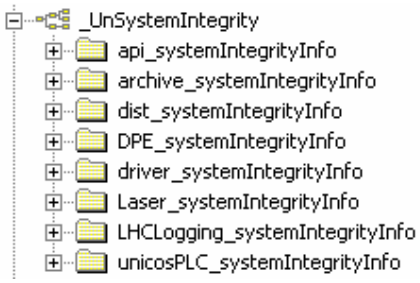


Figure 4: systemIntegrity devices.

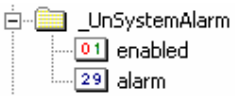


Figure 5: _UnSystemAlarm data point type.

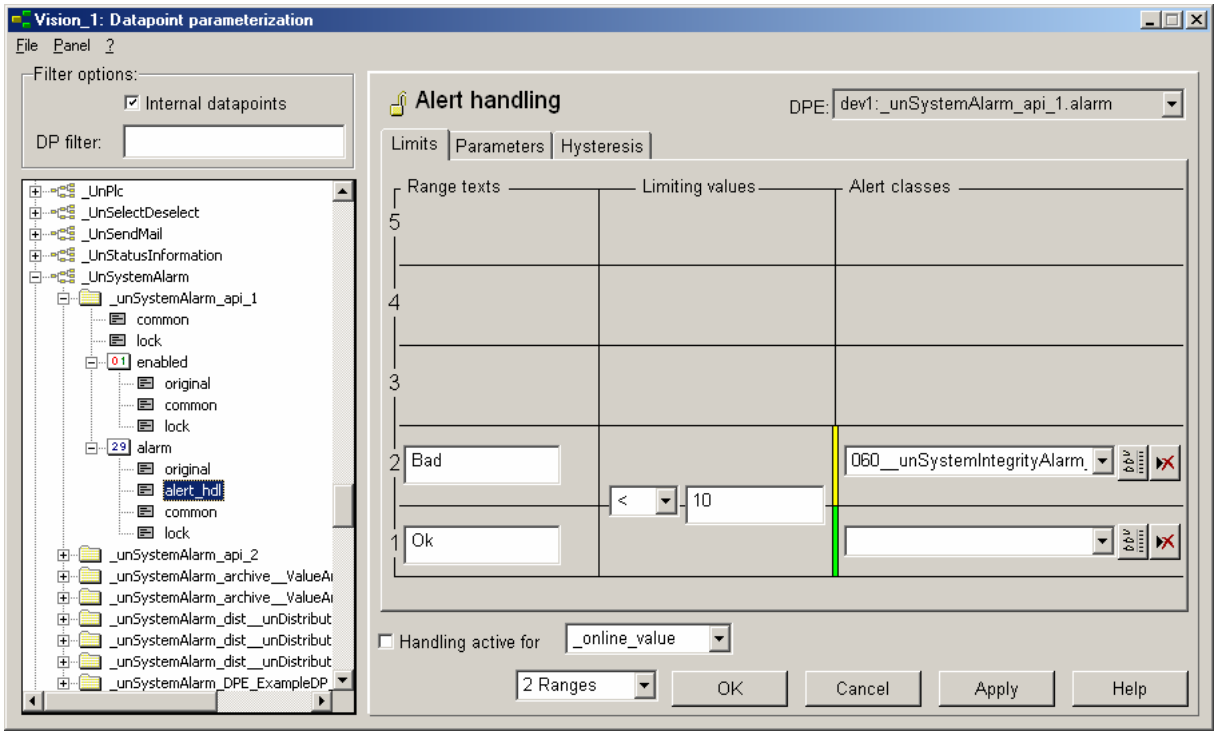


Figure 6: Parameterization of the alarm data point element of the _UnSystemAlarm.

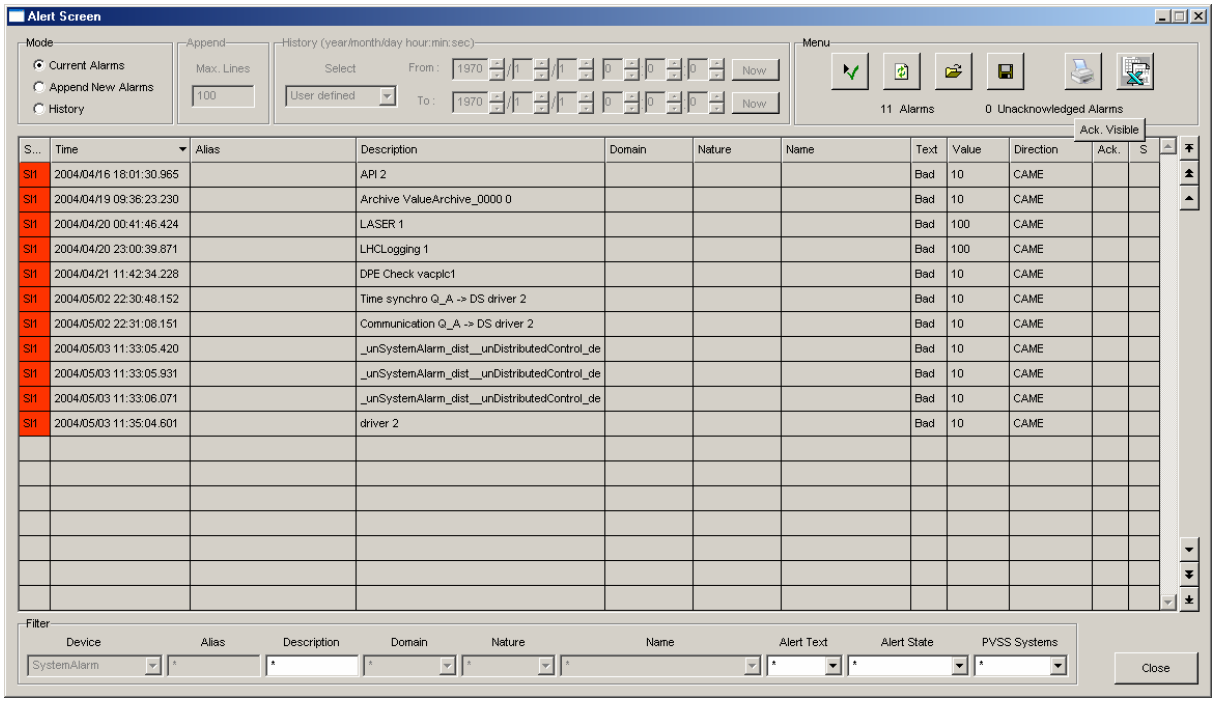


Figure 7: systemIntegrity alarm list.

2.1.2 SYSTEMINTEGRITY INTERNAL CONFIGURATION

The following configuration of a systemIntegrity device type can be set (Figure 8, Section 2.2):

1. The function called during the initialization of the systemIntegrity of this device type. This function returns the list of enabled `_UnSystemAlarm` of this device type.

2. The function called whenever a request is sent to this device type:

- Add new `_UnSystemAlarm(s)`
- Delete `_UnSystemAlarm(s)`
- Enable `_UnSystemAlarm(s)`
- Disable `_UnSystemAlarm(s)`
- Diagnostic: returns the list of enabled `_UnSystemAlarm(s)`

Typically add/enable will setup callback function(s) (`dpConnect`) or periodic check on the state of a device.

3. The function called whenever the configuration data of this device type are modified.

4. The configuration data of this device type. The data is usually used for the checking.

5. The panel to configure the systemIntegrity of the devices of this device type. The $parameter of this panel is the component name listed in the list on the left.

6. The panel to show the status of the systemIntegrity of the devices of this device type. The $parameter of this panel is the component name listed in the list on the left.

The functions must be in a PVSS lib file.

Empty function (“”) can be set and therefore no function will be called.

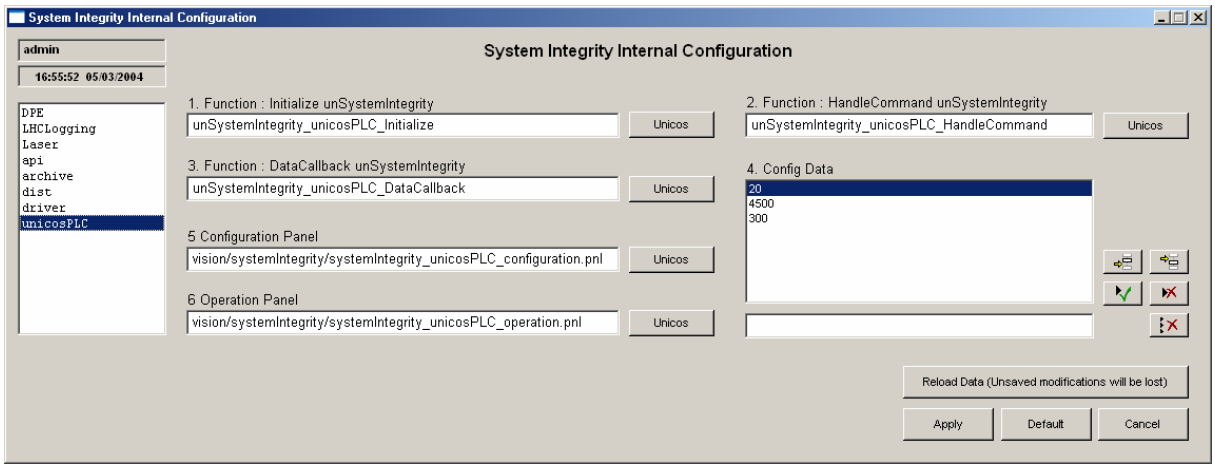


Figure 8: systemIntegrity device type configuration.

2.1.3 SYSTEMINTEGRITY DATA FLOW

The behavior PVSS scripts scripts/unSystemIntegrity.ctl (Figure 9):

- Call the initialize function (if declared and defined) of each declared device type to get the list of enabled devices and set the `interface.command` DPE (Figure 3) to add and the `interface.parameters` DPE to the list of enabled devices.

- Register the HandleCommand function (if declared and defined) of each declared device type to handle any command on this device type by mean of setting the HandleCommand function as a callback (dpConnect) on the `interface.command` DPE and the `interface.parameters` DPE (Figure 3) of this device type. The first time the callback is called with the list returned by the initialize function.
- Register the DataCallback function (if declared and defined) of each declared device type to handle any change in the configuration data of this device type by mean of setting the `DataCallback` function as a callback (`dpConnect`) on the `config.data` DPE (Figure 3) of this device type. The first time the callback is called with the value saved in the `config.data` DPE.

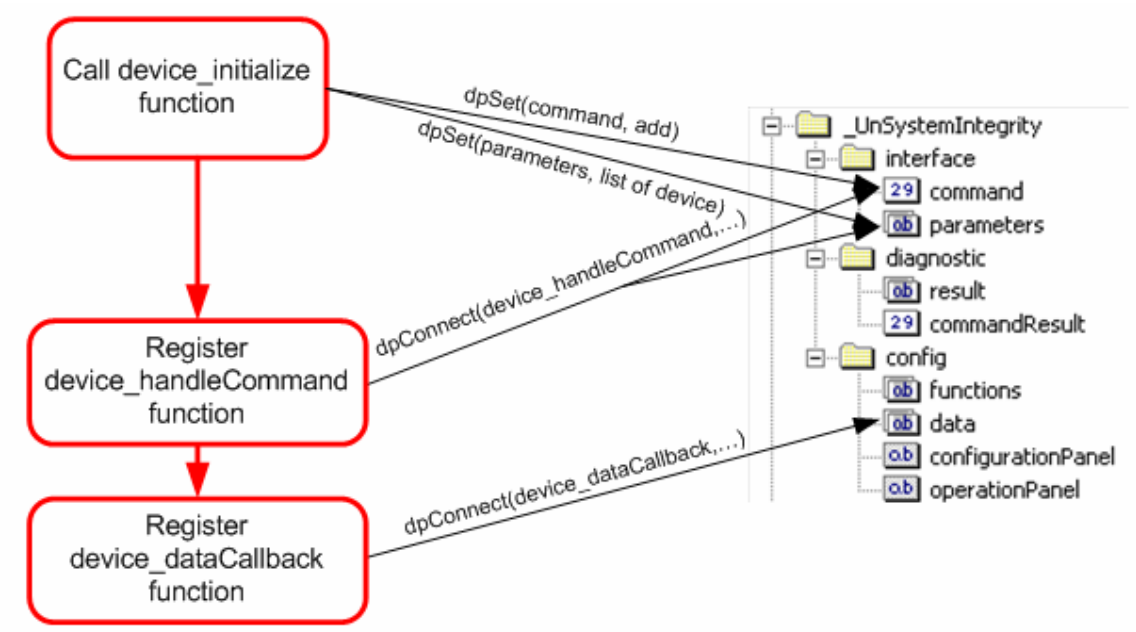


Figure 9: systemIntegrity data flow.

The panel panels/vision/systemIntegrity/systemIntegrityConfiguration.pnl (Figure 1) is used to configure the device systemIntegrity by mean of adding the device_systemIntegrity configuration panel (Section 2.2).

The panel panels/vision/systemIntegrity/systemIntegrityOperation.pnl (Figure 2) is used to operate the device systemIntegrity by mean of adding the device_systemIntegrity operation panel (Section 2.2).

2.2 SYSTEMINTEGRITY DEVICE TYPE

2.2.1 OVERVIEW

The systemIntegrity device type component is composed of:

- An instance of the data point of type `_UnSystemIntegrity` (Figure 3) with a name `deviceType_systemIntegrityInfo` (Figure 4).
- A PVSS library (2.2.4): `scripts/libs/systemIntegrity_device.ctl`.
- A PVSS panel for configuration (Figure 10 and Section 2.2.2). The panel name and path is configurable.

- A PVSS panel for operation/diagnostic (Figure 11 and Section 2.2.3).The panel name and path is configurable.

A device can be included into or excluded from the systemIntegrity device type.

The systemIntegrity device type can be:

- A periodic check on the state of the device. For example the DPE system integrity is a periodic check on the DPE value, if the value is not modified within a certain time, the DPE systemIntegrity generates a `_UnSystemAlarm`.
- A callback triggered if the state of a device is modified. For example the unicosPLC front-end (PLC) the systemIntegrity is:
  - Sending periodically the local TCP/IP number to the front-end partner.
  - Checking the state of the front-end by checking that a DPE is modified periodically.
  - Checking of the MODBUS errors and the connection state of the front-end.
  - Synchronization of date and time of the front-end periodically.

Three configuration data are defined for the systemIntegrity of unicosPLC:

  - The periodicity for sending the TCP/IP number to the front-end partner.
  - The periodicity for checking the state of the front-end.
  - The periodicity for checking the MODBUS errors and the connection state of the front-end.
- A callback triggered if the value of a DPE is modified.
- Etc.

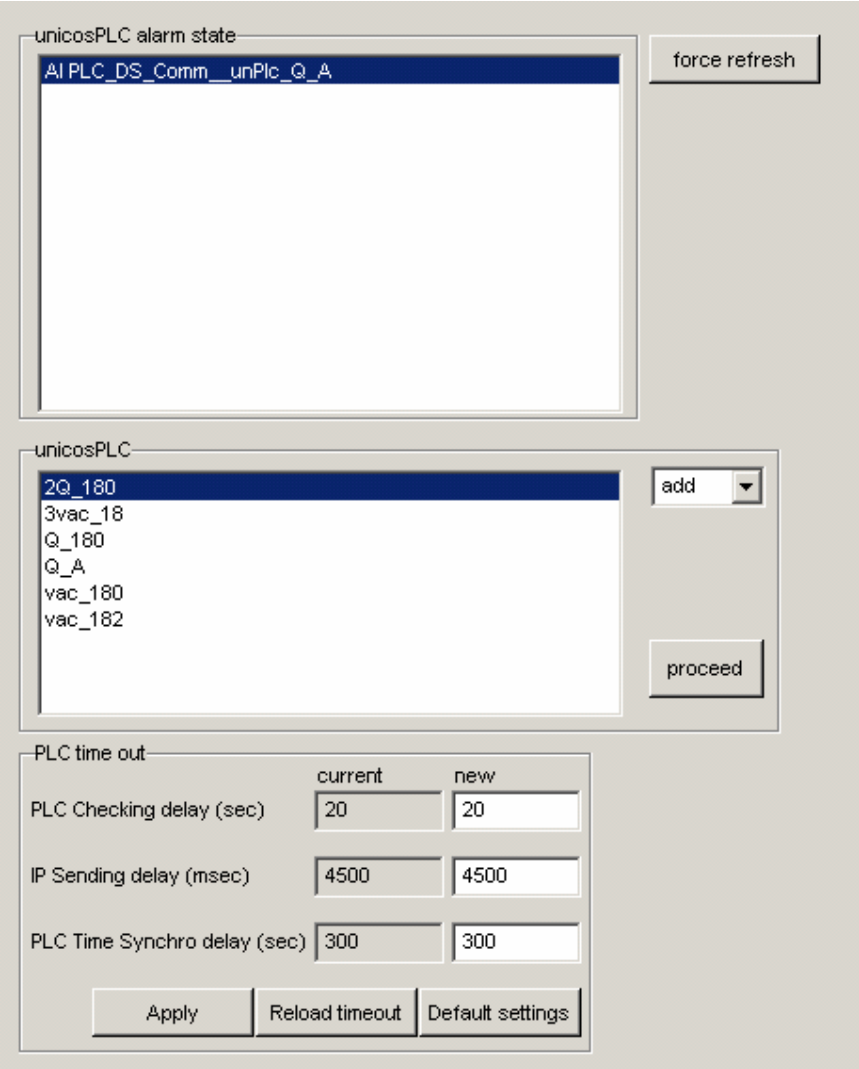


Figure 10: systemIntegrity device type configuration panel.

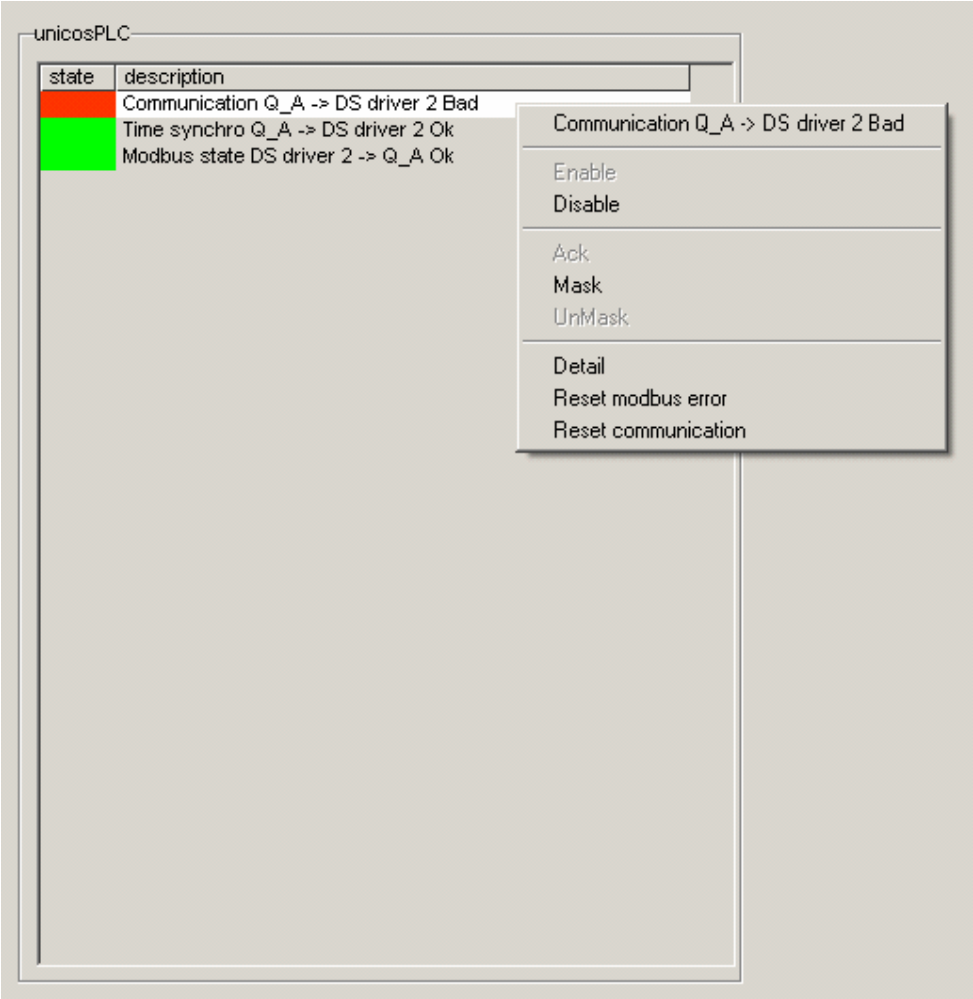


Figure 11: systemIntegrity device type operation panel.

2.2.2 SYSTEMINTEREGRITY DEVICE TYPE CONFIGURATION PANEL

Via the systemIntegrity device type configuration panel (Figure 10 and Figure 12), it is possible to:

- Add new device (Section 2.2.4)
- Delete configured devices (Section 2.2.4)
- Disable configured devices (Section 2.2.4)
- Enable configured devices (Section 2.2.4).

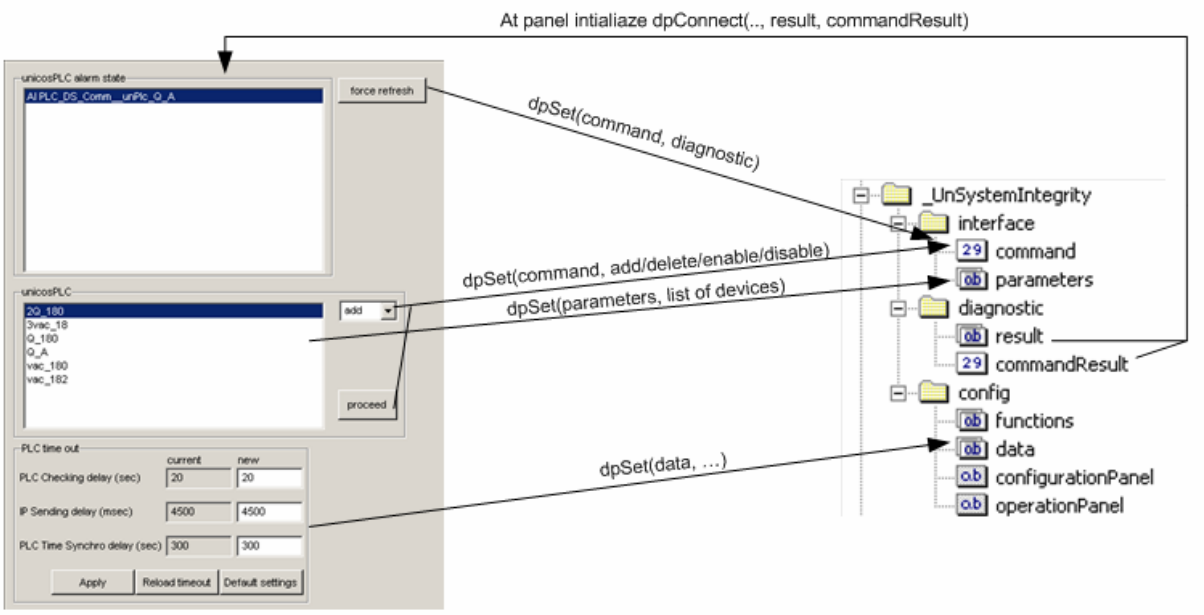


Figure 12: systemIntegrity device type configuration panel data flow.

2.2.3 SYSTEMINTEGRITY DEVICE TYPE OPERATION PANEL

Via the systemIntegrity device type operation panel (Figure 11 and Figure 13), it is possible to get the list of configured devices (Section 2.2.4)

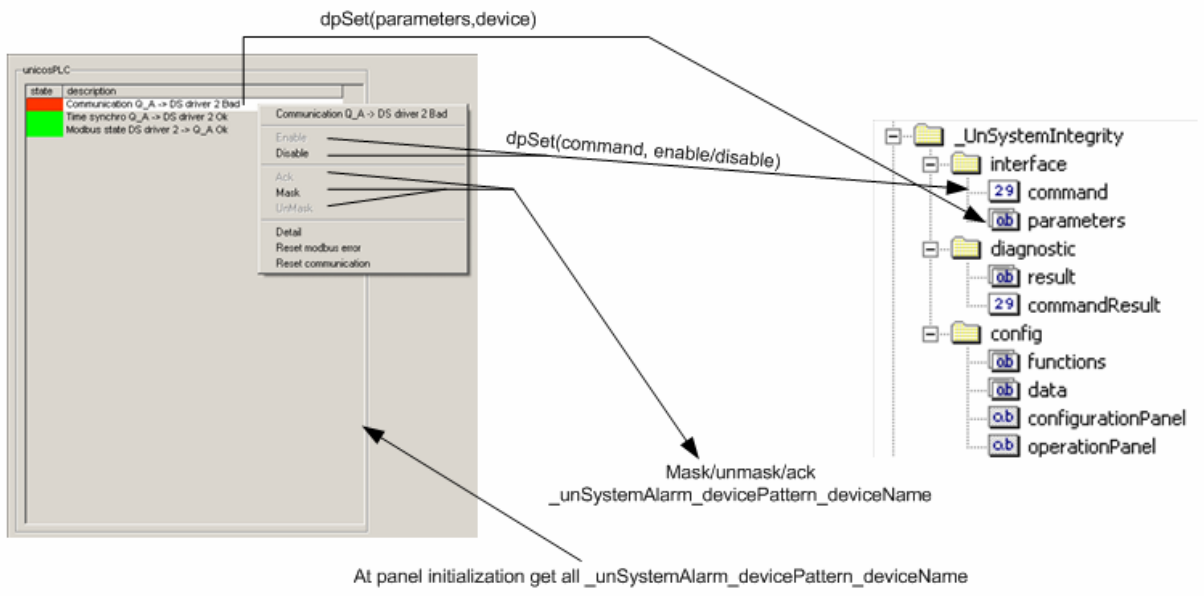


Figure 13: systemIntegrity device type operation panel data flow.

2.2.4 SYSTEMINTEGRITY DEVICE TYPE LIB

The systemIntegrity device type library contains the following (the DPE device type is given as example in Appendix A: systemIntegrity DPE component library):

- Declaration part: declaration of the device pattern, device type name, any other constant declaration, global variables if needed for instance variables to keep the configuration data.
- unSystemIntegrity_device_Initialize: return the list of enabled _unSystemAlarm_devicePattern_deviceName. Usually this function retrieves the configuration data and set the global variables, otherwise there is a risk of un-initialized global variables.
- unSystemIntegrity_device_handleCommand: handle the add/delete/enable/disable/diagnostic command set in the deviceType_systemIntegrityInfo (Figure 4). There are 5 types of command:
  - UN_SYSTEMINTEGRITY_ADD: to add a new device into the systemIntegrity device type by means of adding _UnSystemAlarm (Figure 5) and set-up the device type systemIntegrity for the device. The function unSystemIntegrity_createSystemAlarm from the library unSystemIntegrity.ctl can be used; it creates the systemAlarm and set up the alert_hdl config. The systemAlarm is disabled. This function can start the thread for periodic check, set-up dpConnect for callback function, etc.
  - UN_SYSTEMINTEGRITY_DELETE: to remove the device from the systemIntegrity device type by means of removing the _UnSystemAlarm from any location (alert_hdl PVSS summary list, etc.) and then deleting them. This function must do the opposite of what was done in the UN_SYSTEMINTEGRITY_ADD or UN_SYSTEMINTEGRITY_ENABLE, for instance stop the thread for periodic check, set-up dpDisconnect for callback function, etc.
  - UN_SYSTEMINTEGRITY_ENABLE: to enable the device into the systemIntegrity device type by means of enabling the _UnSystemAlarm and unmask the alert_hdl config. This function can start the thread for periodic check, set-up dpConnect for callback function, etc.
  - UN_SYSTEMINTEGRITY_DISABLE: to disable the device into the systemIntegrity device type by means of disabling the _UnSystemAlarm and mask the alert_hdl config. This function must do the opposite of what was done in the UN_SYSTEMINTEGRITY_ADD or UN_SYSTEMINTEGRITY_ENABLE, for instance stop the thread for periodic check, set-up dpDisconnect for callback function, etc.
  - UN_SYSTEMINTEGRITY_DIAGNOSTIC: returns the list of enabled devices. Usually a global variable is used to keep the list of enabled devices.
- unSystemIntegrity_device_DataCallback: this function is not mandatory. It can be used for instance in case a periodic check is done on the state of the device to set the checking periodicity. Usually this function sets the configuration data into global variables. This will allow the use of the configuration data in other functions like the one(s) started by the UN_SYSTEMINTEGRITY_ADD or UN_SYSTEMINTEGRITY_ENABLE commands of the unSystemIntegrity_device_handleCommand.

The systemIntegrity DPE is checking that a data point value is periodically updated into PVSS. A typical example is the vacuum supervision, there is one DPE per PLC and each PLC updates periodically each DPE. One _UnSystemAlarm is created per PLC (Figure 14).

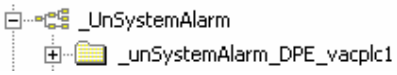


Figure 14: _UnSystemAlarm for vacuum.

3. ADDING A NEW SYSTEMINTEGRITY COMPONENT

The systemIntegrity for DPE (Appendix A: systemIntegrity DPE component library) can be used as an example. To add a new systemIntegrity component:

1. Create a data point of type _UnSystemIntegrity (Figure 3) with the suffix systemIntegrityInfo and configure it (Figure 8).

2. Create the configuration panel of the systemIntegrity device type (Section 2.2.2).

3. Create the operational panel of the systemIntegrity device type (Section 2.2.3).

4. Create the device systemIntegrity device type library (Section 2.2.4).

APPENDIX A: SYSTEMINTEGRITY DPE COMPONENT LIBRARY

```
/**@name LIBRARY: unSystemIntegrity_DPE.ctl

@author: Herve Milcent (AB-CO)

Creation Date: 12/07/2002

Modification History:

version 1.0

External Functions:

Internal Functions:

Purpose:
	Library of the DPE component used by the systemIntegrity.

Usage: Public

PVSS manager usage: CTRL (PVSS00CTRL)

Constraints:
	. global variable: the following variables are global to the script
	. data point type needed: _UnSystemIntegrity
	. data point: DPE_systemIntegrity
	. PVSS version: 2.12.1
	. operating system: W2000, NT and Linux, but tested only under W2000 and Linux.
	. distributed system: yes.
*/

//------------------------------------------------------------------------------------------------
------------------------------
// constant declaration
//------------------
const string UN_SYSTEM_INTEGRITY_DPE = "DPE";
const string UN_SYSTEM_INTEGRITY_DPE_check = "unSystemIntegrity_DPECheck";
const string DPE_pattern = "DPE_";

//
//------------------

// global declaration
global dyn_string g_unSystemIntegrity_DPEList; // list of the DPE dp to check
global dyn_int g_unSystemIntegrity_DPE_ThreadId; // list of the thread Id checking the DPE
global int g_unSystemIntegrity_DPECheckingDelay;

//@{

//------------------------------------------------------------------------------------------------
----------------------

// unSystemIntegrity_DPE_Initialize
/**
Purpose:
Get the list of DPE check by the systemIntegrity. The ones that are enabled.

@param dsResult: dyn_string, output, the list of enabled DPE that are checked

Usage: Public

PVSS manager usage: CTRL (PVSS00CTRL)

Constraints:
	. PVSS version: 2.12.1
	. operating system: W2000, NT and Linux, but tested only under W2000 and Linux.
	. distributed system: yes.
*/
unSystemIntegrity_DPE_Initialize(dyn_string &dsResult)
{
	dyn_string dsList;
```

```
        int len, i;
        string dpToCheck;
        bool enabled;

// get the configuration data and initialized the global variables: call the callback.
        dpGet(UN_SYSTEM_INTEGRITY_DPE+UN_SYSTEMINTEGRITY_EXTENSION+".config.data", dsList);
        unSystemIntegrity_DPE_DataCallback("", dsList);

        dsList = dpNames(c_unSystemAlarm_dpPattern+DPE_pattern+"*", c_unSystemAlarm_dpType);
//DebugN("DPE:", dsList);
        len = dynlen(dsList);
        for(i = 1; i<=len; i++) {
                dpToCheck = dpSubStr(dsList[i], DPSUB_DP);
                dpToCheck = substr(dpToCheck,
strlen(c_unSystemAlarm_dpPattern+DPE_pattern),strlen(dpToCheck));
                dpGet(dsList[i]+".enabled", enabled);
                if(enabled)
                        dynAppend(dsResult, dpToCheck);
        }
//DebugN(dsResult);
}

//------------------------------------------------------------------------------------------------
----------------------

// unSystemIntegrity_DPE_HandleCommand
/**
Purpose:
handle any systemIntegrity command.

@param sDpe1: string, input, dpe
@param command: int, input, the systemIntegrity command
@param sDpe2: string, input, dpe
@param parameters: dyn_string, input, the list of parameters of the systemIntegrity command

Usage: Public

PVSS manager usage: CTRL (PVSS00CTRL)

Constraints:
        . PVSS version: 2.12.1
        . operating system: W2000, NT and Linux, but tested only under W2000 and Linux.
        . distributed system: yes.
*/
unSystemIntegrity_DPE_HandleCommand(string sDpe1, int command, string sDpe2, dyn_string parameters)
{
        dyn_string exceptionInfo;
        int i, len =dynlen(parameters);

        switch(command) {
                case UN_SYSTEMINTEGRITY_ADD: // add the _UnSystemAlarm DP
                        for(i=1; i<=len; i++) {
                                unSystemIntegrity_DPE_checking(parameters[i], true, true,
exceptionInfo);
                        }
                        break;
                case UN_SYSTEMINTEGRITY_DELETE: // delete the _UnSystemAlarm DP
                        for(i=1; i<=len; i++) {
                                unSystemIntegrity_DPE_checking(parameters[i], false, false,
exceptionInfo);
                                // remove dp from the alert list of applicationDP if it is in

        _unSystemIntegrity_modifyApplicationAlertList(c_unSystemAlarm_dpPattern+DPE_pattern+paramete
rs[i], false, exceptionInfo);
                                // delete the _UnSystemAlarm dps
                                if(dpExists(c_unSystemAlarm_dpPattern+DPE_pattern+parameters[i]))
                                        dpDelete(c_unSystemAlarm_dpPattern+DPE_pattern+parameters[i]);
                        }
                        break;
                case UN_SYSTEMINTEGRITY_ENABLE: // enable the _UnSystemAlarm DP
                        for(i=1; i<=len; i++) {
                                if(dpExists(c_unSystemAlarm_dpPattern+DPE_pattern+parameters[i]))
                                        unSystemIntegrity_DPE_checking(parameters[i], false, true,
exceptionInfo);
                        }
```

```
                        break;
                case UN_SYSTEMINTEGRITY_DISABLE: // disable the _UnSystemAlarm DP
                        for(i=1; i<=len; i++) {
                                if(dpExists(c_unSystemAlarm_dpPattern+DPE_pattern+parameters[i]))
                                        unSystemIntegrity_DPE_checking(parameters[i], false, false,
exceptionInfo);
                        }
                        break;
                case UN_SYSTEMINTEGRITY_DIAGNOSTIC: // give the list of _UnSystemAlarm DP and the
state

        dpSet(UN_SYSTEM_INTEGRITY_DPE+UN_SYSTEMINTEGRITY_EXTENSION+".diagnostic.commandResult",
command,

        UN_SYSTEM_INTEGRITY_DPE+UN_SYSTEMINTEGRITY_EXTENSION+".diagnostic.result",
g_unSystemIntegrity_DPEList);
                        break;
                default:
                        break;
        }

        if(dynlen(exceptionInfo) > 0) {
                if(isFunctionDefined("unMessageText_sendException")) {
                        unMessageText_sendException("*", "*", "unSystemIntegrity_DPE_HandleCommand",
"user", "*", exceptionInfo);
                }
// handle any error uin case the send message failed
                if(dynlen(exceptionInfo) > 0) {
                        DebugN(getCurrentTime(), exceptionInfo);
                }
        }
//      DebugN(sDpe1, command, parameters, g_unSystemIntegrity_DPEList);
}

//------------------------------------------------------------------------------------------------------------
----------------------

// unSystemIntegrity_DPE_DataCallback
/**
Purpose:
handle any systemIntegrity command.

@param sDpe1: string, input, dpe
@param dsConfigData: dyn_string, input, the config data

Usage: Public

PVSS manager usage: CTRL (PVSS00CTRL)

Constraints:
        . PVSS version: 2.12.1
        . operating system: W2000, NT and Linux, but tested only under W2000 and Linux.
        . distributed system: yes.
*/
unSystemIntegrity_DPE_DataCallback(string sDpe1, dyn_string dsConfigData)
{
        int DPECheckingDelay;

        if(dynlen(dsConfigData) > 0)
                DPECheckingDelay = dsConfigData[1];
//DebugN(DPECheckingDelay , dsConfigData);
        g_unSystemIntegrity_DPECheckingDelay = DPECheckingDelay;
        if(g_unSystemIntegrity_DPECheckingDelay <= 0)
                g_unSystemIntegrity_DPECheckingDelay = c_unSystemIntegrity_defaultDPECheckDelay;
}

//------------------------------------------------------------------------------------------------------------
----------------------

// unSystemIntegrity_DPE_checking
/**
Purpose:
This function register/de-register the callback funtion of DPE manager. This function can also
create the _unSystemAlarm_DPE dp
with the alarm config but it cannot delete it.
```

```
        @param sDp: string, input, data point name
        @param bCreate: bool, input, true create the dp and the alarm config if it does not exist,
enable it
        @param bRegister: bool, input, true do a dpConnect, false do a dpDisconnect
        @param exceptionInfo: dyn_string, output, exception are returned here

Usage: Internal

PVSS manager usage: CTRL (PVSS00CTRL)

Constraints:
        . PVSS version: 2.12.1
        . operating system: W2000, NT and Linux, but tested only under W2000 and Linux.
        . distributed system: yes.
*/

unSystemIntegrity_DPE_checking(string sDp, bool bCreate, bool bRegister, dyn_string &exceptionInfo)
{
        string dp, description, alarmDp;
        int res, pos, thId;
        bool bError= false;

// remove the system name

        dp = dpSubStr(sDp, DPSUB_DP);
//DebugN("unSystemIntegrity_DPE_checking", dp, bCreate, bRegister);
        alarmDp = c_unSystemAlarm_dpPattern+DPE_pattern+dp;
        if(bCreate) {
// create the sDp and its alarm config if it is not existing
                if(!dpExists(alarmDp)) {
                        description = getCatStr("unSystemIntegrity", "DPE_DESCRIPTION")+dp;
                        unSystemIntegrity_createSystemAlarm(dp, DPE_pattern, description,
exceptionInfo);
                }
        }

        if(dynlen(exceptionInfo)<=0) {
                if(bRegister) {
//DebugN("Enable", g_unSystemIntegrity_DPE_ThreadId, "end");
                        pos = _unSystemIntegrity_isInList(g_unSystemIntegrity_DPEList, alarmDp);
                        if(pos <= 0) {
        // add it in the list, because one of the callback function needs it.

                                pos = _unSystemIntegrity_setList(g_unSystemIntegrity_DPEList, alarmDp,
true);

// start the thread for checking the counter
                                thId = startThread(UN_SYSTEM_INTEGRITY_DPE_check, alarmDp, dp);
                                g_unSystemIntegrity_DPE_ThreadId[pos] = thId;
//DebugN(thId);
                                if(thId<0)
                                        bError = true;

                                if(bError) {

                                        pos = _unSystemIntegrity_isInList(g_unSystemIntegrity_DPEList,
alarmDp);
                                        if(pos > 0) {
                // kill the threads if they were started
                                                res =
stopThread(g_unSystemIntegrity_DPE_ThreadId[pos]);
                                        }

                                        fwException_raise(exceptionInfo, "ERROR",

        "_unSystemIntegrity_set_DPE_checking():"+getCatStr("unSystemIntegrity",
"CANNOT_START_THREAD") +dp,"");
                                }
                                else {
//DebugN("Enable", g_unSystemIntegrity_DPE_ThreadId, "end");
                        // set the enable to true and activate the alarm if it is not activated.
                                        dpSet(alarmDp+".enabled", true);
                                        unAlarmConfig_mask(alarmDp+".alarm", false, exceptionInfo);
                                }
```

```
                        }
                }
                else {
//DebugN("Disable", g_unSystemIntegrity_DPE_ThreadId, "end");
                        pos = _unSystemIntegrity_isInList(g_unSystemIntegrity_DPEList, alarmDp);
                        if(pos > 0) {
// kill the thread
                                res = stopThread(g_unSystemIntegrity_DPE_ThreadId[pos]);
                                if(res<0)
                                        bError = true;

                        // set the enable to false and de-activate the alarm.
                                dpSet(alarmDp+".enabled", false, alarmDp+".alarm",
c_unSystemIntegrity_no_alarm_value);
                                unAlarmConfig_mask(alarmDp+".alarm", true, exceptionInfo);

                                if(bError) {
                                        fwException_raise(exceptionInfo, "ERROR",

        "_unSystemIntegrity_set_DPE_checking():"+getCatStr("unSystemIntegrity",
"CANNOT_STOP_THREAD") +dp,"");
                                }
                                else {
                        // remove from list
                                        pos = _unSystemIntegrity_setList(g_unSystemIntegrity_DPEList,
alarmDp, false);
//DebugN("rmv", res);
                                        dynRemove(g_unSystemIntegrity_DPE_ThreadId, pos);
                                }
                        }
                }
        }

}

//------------------------------------------------------------------------------------------------
----------------------

// unSystemIntegrity_DPECheck
/**
Purpose:
Function to check the value of a DPE.
Set an alarm if the value has not changed and if the time has not changed and the stime_inv bit is
set to false.
in the other case reset it.

        @param sAlarmDp: string, input, alarm dp
        @param dpToCheck: string, input, dpe to check

Usage: Internal

PVSS manager usage: CTRL (PVSS00CTRL)

Constraints:
        . PVSS version: 2.12.1
        . operating system: W2000, NT and Linux, but tested only under W2000 and Linux.
        . distributed system: yes.
*/

unSystemIntegrity_DPECheck(string sAlarmDp, string sDpToCheck)
{
        int waitingTime;
        int oldValue = -1, newValue, alarmValue, oldAlarmValue = -1;
//      time oldTime=0, newTime;
        bool bStimeInv;
        bool bEnabled;

//DebugN(g_unSystemIntegrity_DPECheckingDelay, sAlarmDp, sDpToCheck);
        // get the value of the DPE and the old time
        //dpGet(sDpToCheck+".", oldValue/*, sDpToCheck+".:_online.._stime", oldTime);
        dpGet(sDpToCheck+".", oldValue);
        while(true) {
        // if 0 set it to c_unSystemIntegrity_defaultDPECheckDelay
                waitingTime = g_unSystemIntegrity_DPECheckingDelay;
                if(waitingTime <= 0)
```

```
                        waitingTime = c_unSystemIntegrity_defaultDPECheckDelay;
        // wait
                delay(waitingTime);

        // get the value of the counter and the timeout PLCChecking
                dpGet(sDpToCheck+".", newValue/*, sDpToCheck+".:_online..._stime", newTime*/,
sDpToCheck+".:_online..._stime_inv", bStimeInv);

//              if( bStimeInv || ((newValue == oldValue) && (newTime == oldTime)) ){
                if( bStimeInv || (newValue == oldValue) ){
        // counter was not modified, set an alarm
                        alarmValue = c_unSystemIntegrity_alarm_value_level1;
                }
                else {
        // counter was modified, reset the alarm
                        alarmValue = c_unSystemIntegrity_no_alarm_value;
                }
                if(alarmValue != oldAlarmValue)
                        dpSet(sAlarmDp+".alarm", alarmValue);

//DebugN("unSystemIntegrity_DPECheck", sDpToCheck, oldValue, newValue, oldTime, newTime,
alarmValue);
                oldValue = newValue;
                oldAlarmValue = alarmValue;
//              oldTime = newTime;
        }
}

//@}
```

APPENDIX B: LIST OF FIGURES