ООО «Группа индустриальных технологий»

УТВЕРЖДАЮ

Генеральный директор

_____ К.Н. Мигун

«____» _____ 2023 г.

УЗЕЛ ПЕЧАТНЫЙ R 1 DXC 03 M

Встроенное программное обеспечение

Текст программы

РОФ.ГРЛМ.03002-01 12 01

Листов 17

2023

Подп. и дата

Инв. № дубл.

Взам. инв. №

Подп. и дата

Инв. № подл.

АННОТАЦИЯ

В данном документе содержится текст программы встроенного программного обеспечения узла печатного R 1 DXC 03 M.

В разделе «Необходимый комплект оборудования и ПО» указаны технические и программные средства, необходимые для просмотра электронной записи исходного кода встроенного программного обеспечения узла печатного R 1 DXC 03 M.

В разделе «Характеристики записи текста программы» указаны характеристики исходного кода встроенного программного обеспечения узла печатного R 1 DXC 03 M, включая язык программирования, на котором написана программа, и место размещения электронной версии текста программы.

В разделе «Фрагмент текста программы» приведен фрагмент исходного кода встроенного программного обеспечения узла печатного.

1 НЕОБХОДИМЫЙ КОМПЛЕКТ ОБОРУДОВАНИЯ И ПО

Для просмотра файла, содержащего текст программы с исходным кодом, требуется следующий комплект оборудования и программного обеспечения:

- персональный компьютер с операционной системой Windows не ниже Windows 7;
- установленное программное обеспечение QT Creator, включающее в себя редактор кода, справочник, графические средства разработки интерфейса Qt Designer и графический интерфейс отладчика приложений;
- текстовый редактор для просмотра ASCII-текстовых файлов.

2 ХАРАКТЕРИСТИКИ ЗАПИСИ ТЕКСТА ПРОГРАММЫ

Исходный код программы написан на языках C++, Python.

Текст программы включает в себя исходный код встроенного программного обеспечения узла печатного R 1 DXC 03 M с комментариями, в которых указаны функциональное назначение представленных процедур.

Текст программы оформлен в форме электронного текстового файла, выполненного с использованием стандартной кодировки ASCII.

Размещение файла: http://gitlab.git-holding.ru:9071/git/meta-git

3 ФРАГМЕНТ ТЕКСТА ПРОГРАММЫ

```
/**
 * \file
 * \brief  Implementation of DXC-Core application (main thread)
 *
 * \author (c) GIT Moscow
 *
 * \date   01.01.2017
 */

#include <log4cxx/logger.h>
#include <log4cxx/level.h>
#include <log4cxx/fileappender.h>
#include <log4cxx/patternlayout.h>
#include <log4cxx/simplelayout.h>
#include <log4cxx/xml/domconfigurator.h>
#include <boost/thread.hpp>
#include <boost/statechart/fifo_scheduler.hpp>

#include <sys/types.h>
#include <sys/stat.h>

#include <errno.h>
#include <unistd.h>
#include <signal.h>

#include <iostream>
```

```
#include <fstream>

#include "version.h"
#include "gendefs.h"
#include "globals.h"
#include "CoreDb.h"


#define LOG_OFF(logger, message) \
{\
    LOG4CXX_LOG (logger, log4cxx::Level::getOff(), message);\
}


//! defines for config check result (bit-coded)
#define CONFIG_CHECK_ERROR             0x01   // used by config
checker
#define CONFIG_CHECK_WARNINGS_ONLY     0x02   // used by windows
config checker only
#define ACTIVATION_CHECK_ERROR         0x04
#define ACTIVATION_ID_MISMATCH_ERROR   0x08
#define SCRIPT_CHECK_ERROR             0x80   // used by script
checker (see
webinterface/webinterface/lib/archiveManager/archiveManager.py)


//! \weakgroup DXCCORE_MAIN
//!\{

// define the default logger path
std::string loggerPath("/etc/gitcomm/logger.xml");

// define the storage for the basePath
std::string basePath;

// definition of global MemoryPool for Logger
log4cxx::helpers::Pool gLoggerPool;

// definition for global root logger
log4cxx::LoggerPtr rootLogger(log4cxx::Logger::getRootLogger());

// definition for logger output
static log4cxx::LoggerPtr
statusLogger(log4cxx::Logger::getLogger("SystemStatus"));

//definition of global Event Logger
log4cxx::LoggerPtr
appEventLogger(log4cxx::Logger::getLogger("AppEventLogging"));

//definition of global Customer Loggers
log4cxx::LoggerPtr
gUserActionLogger(log4cxx::Logger::getLogger("UserActionLogging"));
log4cxx::LoggerPtr
gCustomerLogger(log4cxx::Logger::getLogger("CustomerLogging"));

//definition of global test support logger
```

```
log4cxx::LoggerPtr
gTestSupportLogger(log4cxx::Logger::getLogger("TestSupportLogging"));

// default path and file name of persistent IP Device Role Assignment
XML file
static std::string
gIpStationRoleAssignmentFilePath("/etc/gitcomm/IpStationRoleAssignment
.xml");

// global marker to run the threads
bool gThreadRun = true;

// global variable which contains the startup time
std::string gStartupTime;

// global variable which contains compile time
std::string gDxcCoreBuildDate;

// DXC version and build date/time info for logging dxcCore
std::string gDxcVersionBuildString;

// Global bitfield for special logging for test purposes
ServiceLoggingTYPE gServiceLogging;

// IP address of device, where the corresponding HAL process runs
// usually localhost, can be set to remote device for debugging
purposes via command line
static std::string sRemoteHost = "127.0.0.1";
std::string getRemoteHost() { return sRemoteHost; }
static void setRemoteHost(const std::string &remoteHost) { sRemoteHost
= remoteHost; }

// TCP port no the modbus server listens to
// can be set via command line
static int sModbusServerPort = 0;   // 0: use default port, >0 use
this port
int getModbusServerPort() { return sModbusServerPort; }
static void setModbusServerPort(int port) { sModbusServerPort = port;
}

// TCP port no the CanOpen gateway
// can be set via command line
static int sCanOpenGatewayPort = 0;   // 0: use default port, >0 use
this port
int getCanOpenGatewayPort() { return sCanOpenGatewayPort; }
//static void setCanOpenGatewayPort(int port) { sCanOpenGatewayPort =
port; }

// information, if DXC is currently running with default config
static bool sDefaultConfigActive = false;
bool isDefaultConfigActive()  { return sDefaultConfigActive; }
void setDefaultConfigActive() { sDefaultConfigActive = true; }

// our own slot number and access method
static uint8_t sOwnSlotId = SLOT_NO_OWN;
uint8_t getOwnSlotId()   { return sOwnSlotId; }
```

```
void setOwnSlotId(uint8_t slotId) { sOwnSlotId = slotId; }

static bool sSystemStartupFinished = false;
bool    isSystemStartupFinished() { return sSystemStartupFinished; }
void    setSystemStartupFinished(bool systemStartupFinished) {
sSystemStartupFinished = systemStartupFinished; }

// get and set protection switching is pending
static bool sProtectionSwitchIsPending = false;
bool    isProtectionSwitchPending() { return
sProtectionSwitchIsPending; }
void    setProtectionSwitchPending(bool protectionSwitchIsPending) {
sProtectionSwitchIsPending = protectionSwitchIsPending; }

// string identifying the network this DXC is part of
static std::string gNetworkIdentifier("-ERROR not set-");
// access method
const std::string & getNetworkIdentifier(void) { return
gNetworkIdentifier; };

// access method: returns string containing path and file name of
persistent IP Device Role Assignment XML file
const std::string & getIpStationRoleAssignmentFilePath(void) { return
gIpStationRoleAssignmentFilePath; };

extern int mainFrame(uint8_t ownShelfId,
    const std::string & configFilePath,
    const std::string & roleAssignmentFilePath,
    const bool processSchema);

extern int checkConfig(const std::string& configFilePath, const
std::string& warnConfigFilePath, uint8_t ownShelfId, bool
skipTwBackCheck);

//! defines the index in ConfigCheckerStatusTextTYPE
enum ConfigCheckerStatusTextLanguageTYPE
{
    CC_ST_LANG_EN = 0, //!< English
    CC_ST_LANG_DE = 1, //!< Deutsch
    CC_ST_LANG_MAX, //!< for comparison purposes
    // size of mapping table
    CC_ST_LANG_DEFAULT = 0, //!< default language is English
//
};

extern ConfigCheckerStatusTextLanguageTYPE gStatusTextLangType;


/**

*****************************************************************
********
 * \brief      configureLogger
 *
 * Configure some basic loggers used during dxcCore startup.
 *
```

```
 * \param[in]  loggerXmlPath    path to xml configuration file for
logger configuration

*********************************************************************
********
 */
static void configureLogger(std::string loggerXmlPath)
{
    // configure the Log4cxx logger
    log4cxx::xml::DOMConfigurator::configure(loggerXmlPath);

    // set appender for the root logger
    log4cxx::FileAppenderPtr rootAppender = rootLogger-
>getAppender("DeveloperRollingAppender");

    // check for available appender
    if (rootAppender != 0)
    {
        rootAppender->setFile("/var/log/gitcomm/dxcCore.log");
        rootAppender->activateOptions(gLoggerPool);
    }
    return;
}


void mainFrameStub(void)
{
    LOG4CXX_INFO (rootLogger, "mainFrameStub...");

    gCoreDb.setDxcApplVersionMajor (getCoreDbSystemKey(),
DXC_APPLICATION_VERSION_MAJOR);
    gCoreDb.setDxcApplVersionMinor (getCoreDbSystemKey(),
DXC_APPLICATION_VERSION_MINOR);
    gCoreDb.setDxcApplVersionInfo  (getCoreDbSystemKey(),
DXC_APPLICATION_VERSION_INFO);

}


/**

*********************************************************************
********
 * \brief      Function to handle all caught signals
 *
 * \param[in]  sig  signal to handle

*********************************************************************
********
 */
void signalHandler(int sig)
{
    // select handling on received signal
    switch (sig)
    {
        case SIGUSR1:
```

```
        {
            // storage for new logger control file
            std::string loggerSource;

            // build different source paths
            if (basePath.size() > 0)
            {
                struct stat fileValid;

                // build source for debug path
                loggerSource = basePath + "/debuglogger.xml";

                // check for existent file
                if (stat(loggerSource.c_str(), &fileValid) == -1)
                {
                    // setup default
                    loggerSource = loggerPath;
                }
            }
            else
            {
                // setup default
                loggerSource = loggerPath;
            }

            // read the logger configuration
            LOG4CXX_INFO (rootLogger, "Setup new Logger: " <<
loggerSource);

            // commands to configure root logger
            configureLogger(loggerSource);
        }
            break;

        case SIGUSR2:
            // print linecard compatibility list
#if 0
            LOG4CXX_INFO (statusLogger, "\n" <<
Card::getCompatibilityList());
            // print linecard stati to console
            LOG4CXX_INFO (statusLogger, "\n" <<
gCoreDb.getSlotStatusString(ResKey()) << "\n")
            ;

            // print terminal compatibility list
            LOG4CXX_INFO (statusLogger, "\n" <<
Terminal::getCompatibilityList());
            // print terminal stati to console
            LOG4CXX_INFO (statusLogger, "\n" <<
gCoreDb.getTerminalStatusString(ResKey()) << "\n")
            ;

            // print IP role stati to console
            LOG4CXX_INFO(statusLogger, "\n" <<
gCoreDb.getIpRoleStatusString(ResKey()) << "\n");
```

```
            // print IP terminal role assignment DB
            LOG4CXX_INFO(statusLogger, "\n" <<
gCoreDb.getIpDevRoleAssignmentStatusString(ResKey()) << "\n");
#endif
            break;

            // deleting User Logfile
        case SIGCONT:
        {
            LOG4CXX_INFO (rootLogger, "Received SIGCONT -> Deleting
EventLog");
            log4cxx::helpers::Pool p;
            log4cxx::FileAppenderPtr appEventAppender =
appEventLogger->getAppender("AppEventFileAppender");
            appEventAppender->setFile(appEventAppender->getFile(),
false, false, 0, p);
            appEventAppender->activateOptions(p);

            LOG4CXX_LOG (appEventLogger, log4cxx::Level::getOff(),
"##############################################");
            LOG4CXX_LOG (appEventLogger, log4cxx::Level::getOff(),
"############ START EVENT LOGGING ############");
            LOG4CXX_LOG (appEventLogger, log4cxx::Level::getOff(),
"##############################################");
            // evaluate startup time
            time_t rawtime;
            time(&rawtime);
            LOG4CXX_LOG (appEventLogger, log4cxx::Level::getOff(),
"timestamp: " << ctime(&rawtime));
        }
            break;

        default:
        {
            // change global run variable
            gThreadRun = false;
        }
    }
    return;
}

/**

*****************************************************************************
********
 * \brief      log boot string with important information
 *
 * \param[in]  ownShelfId         own shelf ID
 * \param[in]  configFilePath     XML configuration file currently
used

*****************************************************************************
********
 */
void logBootString(const uint8_t ownShelfId, const std::string &
configFilePath)
```

```
{
    LOG_OFF(gTestSupportLogger,
"#########################################################################
#######");
    LOG_OFF(gTestSupportLogger, "###                              B O O T I
N G . . .                              ###");
    LOG_OFF(gTestSupportLogger, gDxcVersionBuildString);
    LOG_OFF(gTestSupportLogger, "Network Identifier: '" <<
getNetworkIdentifier() << "' ShelfID: " << int(ownShelfId));
    LOG_OFF(gTestSupportLogger, "Config File: " << configFilePath);
    LOG_OFF(gTestSupportLogger, "DXC core starting at " <<
gStartupTime << " ...");

}


/**

************************************************************************
********
 * \brief      Usage function to display the possible options
 *
 * Print dxcCore command line options to std::cerr.
 *
 * \param[in]  argv0 process command line and start working threads
 * \return     none

************************************************************************
********
 */
static void usage(char *argv0)
{
    // display the usage message
    std::cerr << argv0 << "\n";
    std::cerr << gDxcVersionBuildString << "\n";
    std::cerr << "usage ShelfId NetworkIdent [optional parameter]:\n";
    std::cerr << "  ShelfId                 - mandatory argument -
unique shelf number\n";
    std::cerr << "  NetworkIdent            - mandatory argument -
network identifier string\n";
    std::cerr << "  [-c configFilePath]     - absolute path/file name
of XML configuration file\n";
    std::cerr << "  [-d]                    - start process as
daemon\n";
    std::cerr << "  [-h host]               - host name/IP-address of
HAL (default:127.0.0.1)\n";
    std::cerr << "  [-l logger.xml]         - absolute path/file name
of logger control file\n";
    std::cerr << "  [-r ]                   - absolute path/file name
of the IP station role assignment file
(IpStationRoleAssignment.xml)\n";
    std::cerr << "  [-R ]                   - create template
IpRoleAssignment file from XML configuration file and exit (for
BulkTest purposes only)\n";
    std::cerr << "  [-m portNo]             - use portNo for Modbus
server\n";
```

```
    std::cerr << "  [-s]                    - skip TW call-back checks
during XML configuration file verification\n";
    std::cerr << "  [-v]                    - verify XML configuration
file syntax / semantic only (ShelfId -c -l -s -t -w options only)\n";
    std::cerr << "  [-w warnConfigFilePath] - absolute path/file name
of XML configuration file of warning sequences\n";
    std::cerr << "  [-t]                    - language for XML
configuration file verification status messages {ge, en}\n";
    std::cerr << "  [-x]                    - skip XML configuration
file syntax verification (-v overwrites this option)\n";

    return;
}

/**

*****************************************************************
********
 * \brief      main
 *
 * Main function called on application start.
 *
 * \param[in]  argc
 * \param[in]  argv
 * \return   0 in case of normal return, > 0 in case of wrong input
parameters, -1 otherwise

*****************************************************************
********
 */
int main(int argc, char **argv)
{
    int result = -1;
    int c;

    uint8_t ownShelfId = SHELF_NO_INVALID;

    struct sigaction sig;

    bool startDaemon = false;
    pid_t ppid;

    bool verifyXmlOnly(false);
    bool skipTwBackCheck(false);
    bool processSchema(true);

    // bulk test support
    bool bCreateRoleAssignmentTemplate(false);

    // define the default path of the XML config file
    std::string configFilePath("/etc/gitcomm/gitcommConfig.xml");
    std::string warnConfigFilePath(""); // must be set by -w cmd line
parameter!

    // Evaluate the compile time of the DXC core
    // (used by ConfigFileParser to store it in the database)
```

```
    gDxcCoreBuildDate = std::string(__DATE__) + std::string(" ") +
std::string(__TIME__);

    // create global string containing all version information for
this DXC core build (referenced by DXC core, config checker ...)
    std::stringstream tmp;
    tmp << "DXC core version: " << int(DXC_APPLICATION_VERSION_MAJOR)
<< "." << int(DXC_APPLICATION_VERSION_MINOR) << ".";
    tmp << DXC_APPLICATION_VERSION_INFO << ", compiled at " <<
gDxcCoreBuildDate;
    gDxcVersionBuildString = tmp.str();

    // parse command line arguments
    while ((c = getopt(argc, argv, "c:dh:l:m:Rr:st:vw:x")) >= 0)
    {
        // handle the optional parameters
        switch (c)
        {
        case 'c': // set new location and name for XML configuration
file
            configFilePath = optarg;
            break;

        case 'd': // start DXC Core in Daemon mode
            startDaemon = true;
            break;

        case 'h': // set new host name for HAL
            setRemoteHost(optarg);
            break;

        case 'l': // set new location and name for logging control
xml-file
            loggerPath = optarg;
            break;
        case 'm': // use portNo for Modbus server
            setModbusServerPort( atoi(optarg) );
            break;
        case 'r':
            gIpStationRoleAssignmentFilePath = optarg;
            break;
        case 'R': // test support: create template
IpStationRoleAssignment.xml file from XML configuration file
            bCreateRoleAssignmentTemplate = true;
            break;
        case 'v': // verify XML file syntax/semantic only
            verifyXmlOnly = true;
            break;
        case 's': // skip TW back direction check during XML
syntax/semantic verification
         skipTwBackCheck = true;
         break;

        case 't': // set language for StatusLine text output (XML
config check only)
        {
```

```
            std::string lang = optarg;
            if ("ge" == lang)
                gStatusTextLangType = CC_ST_LANG_DE;
            else if ("en" == lang)
                gStatusTextLangType = CC_ST_LANG_EN;
            else
            {
                usage(argv[0]);
                return 2;
            }
        }
            break;
        case 'w': // set new location and name for XML configuration
file of warning sequences
            warnConfigFilePath = optarg;
            break;
        case 'x': // skip XML schema validation
            processSchema = false;
            break;

        case '?':
        default:
            usage(argv[0]);
            return 0;
        }
    }

    // get (mandatory) command line arguments to set: {unique shelf
ID, NetworkIdentifier}
    if (verifyXmlOnly || bCreateRoleAssignmentTemplate)
    {
        if (optind == (argc - 1))   // last command line parameter
        {
            // get ownShelfId from command line
            ownShelfId = atoi(argv[optind]); // numeric
        }
        else if (optind == (argc - 2))   // second-last command line
parameter
        { // get ownShelfId, ignore gNetworkIdentifier

            // get ownShelfId from command line
            ownShelfId = atoi(argv[optind]); // numeric

            // get gNetworkIdentifier from command line
            gNetworkIdentifier = std::string((argv[optind + 1])); //
string
        }
        else
        {
            // check ALL shelfs
        }
    }
    else
    {
        if (optind == (argc - 2))
        { // standard operation: both arguments required
```

```
            // get ownShelfId from command line
            ownShelfId = atoi(argv[optind]); // numeric

            // check ownShelfId range (also detects not numeric
values)
            if ((ownShelfId < 1) || (ownShelfId > 250))
            {   // range error
                usage(argv[0]);
                return 1;
            }

            // get gNetworkIdentifier from command line
            gNetworkIdentifier = std::string((argv[optind + 1])); //
string
        }
        else
        {
            // MANDATORY arguments required!
            usage(argv[0]);
            return 1;
        }
    }

    { // evaluate startup time
        time_t rawtime;
        time(&rawtime);
        gStartupTime = ctime(&rawtime);
        // Remove the \n character at the end of the string
        gStartupTime.erase(gStartupTime.size() - 1, 1);
    }

    { // logger - configuration
        // search for last '/'
        size_t index = loggerPath.find_last_of('/');

        // build base path
        if (index < loggerPath.size())
            basePath = loggerPath.substr(0, index);

        // configure the Log4cxx logger
        configureLogger(loggerPath);
    }

    if (verifyXmlOnly)
    {
        return checkConfig(configFilePath, warnConfigFilePath,
ownShelfId, skipTwBackCheck);
    }

    if (bCreateRoleAssignmentTemplate)
    { // create a template IpStationRoleAssignmentFile.xml based on
all Roles in DB and exit
#if 0
```

```
IpRoleAssignmentFileCreator::createRoleAssignmentTemplate(configFilePa
th);
#endif
        return 1;
    }

    // check shelf number
    if (SHELF_NO_INVALID == ownShelfId)
    {
        usage(argv[0]);
        return 0;
    }

    // start in daemon mode
    if (startDaemon == true)
    {
        // do trace for daemon option
        std::cerr << "starting " << argv[0] << " as daemon !!!" <<
std::endl;

        // fork child process
        if ((ppid = fork()) == 0)
        {
            // this is the child process
            struct sigaction sig;

            // create signal handler for correct termination of all
threads
            memset(&sig, 0, sizeof(struct sigaction));

            sig.sa_handler = SIG_IGN;
            sig.sa_flags = SA_RESTART;

            // setup new signal handler
            (void) sigaction(SIGHUP, &sig, NULL); // Signal handler ->
terminal connection closed
            (void) sigaction(SIGINT, &sig, NULL); // Signal handler ->
terminal key [CTRL-C]
            (void) sigaction(SIGQUIT, &sig, NULL); // Signal handler -
> terminal key [CTRL-Q]

            // close all default file handles
            (void) close(0); // stdin
            (void) close(1); // stdout
            (void) close(2); // stderr

            // log boot string at dxcCore.log, service.log and
appEvent.log
            logBootString(ownShelfId, configFilePath);

            return mainFrame(ownShelfId, configFilePath,
gIpStationRoleAssignmentFilePath, processSchema);
        }
        else
        {
```

```
            // this is the parent process
            // check for an error
            if (ppid == -1)
            {
                // display the error message
                std::cerr << argv[0] << std::endl;
                std::cerr << "Create daemon: " << errno << std::endl;
                std::cerr << strerror(errno) << std::endl;
            }
            else
            {
                // successfully created
                result = 0;
            }
        }
    }
    else
    {
        // create signal handler for correct termination of all
threads
        memset(&sig, 0, sizeof(struct sigaction));
        sig.sa_handler = signalHandler;

        // Signal handler -> terminal key [CTRL-C]
        (void) sigaction(SIGINT, &sig, NULL);

        // call normal frame work
        return mainFrame(ownShelfId, configFilePath,
gIpStationRoleAssignmentFilePath, processSchema);
    }

    return result;
}

//!\}

/** \} group dxcCore_APP */
```

ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

Изм.	Номера листов (страниц)				Всего листов (страниц) в документе	№ докум.	Входящий № сопроводи тельного докум. и дата	Подпись	Дата
	изменен ных	заменен ных	новых	аннулиро ванных					

Подп. и дата
Инв. № дубл.
Взам. инв. №
Подп. и дата
Инв. № подл.

0	Нов.			
Изм.	Лист	№ докум.	Подп.	Дата

РОФ.ГРЛМ.03002-01 12 01