NUR2 MIGRATION DOCUMENT

VERSION HISTORY

VERSION	DATE	AUTHOR	CHANGES
0.1	2017-02-05	ML	Initial draft
0.2	2018-11-9	JRaut	Update latest status

TABLE OF CONTENTS

1. SCOPE

This document provides information how to migrate application from NUR based devices to NUR2 devices.

If you have any questions regarding to migration or NUR2 features, please contact support@nordicid.com

This document applies to NUR2-1W firmware version 7.2-A

2. NEW IN NUR2

2.1. MODULE TYPE DETECTION

NUR2 module can be detected using NURAPI device capabilities.

New value in `NUR_DEVICECAPS.chipVersion`:

```
/** Chip version R2000 */
NUR_CHIPVER_R2000 = 3
```

New value in `NUR_DEVICECAPS.moduleType`:

```
/** Module type NUR2-1W */
NUR_MODULETYPE_NUR2_1W = 5
```

2.2. RF PROFILE

NUR2 RFID low level settings, such as link frequency, miller, modulation are controlled via new rfProfile member in NURAPI module setup.

2.2.1. SETTING RF PROFILE

`NUR_MODULESETUP.rfProfile` accepted values:

```
/** Robust RF profile. This profile is recommended to use in noisy RF environments. */
#define NUR_RFPROFILE_ROBUST                    0
/** Nominal RF profile. This profile works good in most environments. */
#define NUR_RFPROFILE_NOMINAL                   1
/** High speed RF profile. This profile provides best throughput, but is prone to RF
interference. */
#define NUR_RFPROFILE_HIGHSPEED                 2
```

For example:

```
moduleSetup.rfProfile = NUR_RFPROFILE_HIGHSPEED;

NurApiSetModuleSetup(hApi, NUR_SETUP_RFPROFILE, &moduleSetup, sizeof(moduleSetup));
```

2.2.2. BACKWARDS COMPATIBILITY

For backwards compatibility with old code, it is still allowed to set any existing module setup members, such as link frequency, miller, modulation.

However, they do not change RFID behavior in any way.

You can still use old API to change rfProfile via linkFreq backward compatibility mode,

see section 3.1.1. LINKFREQ BACKWARDS COMPATIBILITY.

2.2.3. DETECT RF PROFILE SUPPORT

RF Profile support can be detected using NURAPI device capabilities.

New value in `NUR_DEVICECAPS.flagSet1`:

```
/* The module FW supports RF profile setting. */

NUR_DC_RFPROFILE      = (1<<24)
```

2.3. TAG PHASE INFO

NUR2 can report tag phase angle in units of tenths of degrees. If enabled, value is stored in inventoried tag meta data timestamp field.

2.3.1. ENABLING TAG PHASE INFO

Tag phase info can be enabled by setting `NUR_OPFLAGS_EN_TAG_PHASE` in `NUR_MODULESETUP.opFlags`.

For example:

```
moduleSetup.opFlags |= NUR_OPFLAGS_EN_TAG_PHASE;

NurApiSetModuleSetup(hApi, NUR_SETUP_OPFLAGS, &moduleSetup, sizeof(moduleSetup));
```

2.3.2. DETECT TAG PHASE INFO SUPPORT

Tag phase info support can be detected using NURAPI device capabilities.

New value in `NUR_DEVICECAPS.flagSet1`:

```
/* This module FW supports tag phase info. */

NUR_DC_TAGPHASE     = (1<<26)
```

2.4. DIAGNOSTICS

NUR2 can report health diagnostics information to host. This is useful especially for debugging purposes.

2.4.1. DIAGNOSTICS DATA

NUR2 diagnostics data structure.

```
struct NUR_DIAG_REPORT
{
  DWORD flags;             /**< Report flags. see enum NUR_DIAG_REPORT_FLAGS */
  DWORD uptime;            /**< Uptime in milliseconds */
  DWORD rfActiveTime;      /**< RF on time in milliseconds */
  int   temperature;       /**< Temperature in celcius. 1000 if not supported */
  DWORD bytesIn;           /**< Number of bytes in to module */
  DWORD bytesOut;          /**< Number of bytes out from module */
  DWORD bytesIgnored;      /**< Number of ignored (invalid) bytes */
  DWORD antennaErrors;     /**< Number of bad antenna errors */
  DWORD hwErrors;          /**< Number of automatically recovered internal HW failures */
  DWORD invTags;           /**< Number of successfully inventoried tags */
  DWORD invColl;           /**< Number of collisions during inventory */
  DWORD readTags;          /**< Number of successfully read tag commands */
  DWORD readErrors;        /**< Number of failed read tag commands */
  DWORD writeTags;         /**< Number of successfully write tag commands */
  DWORD writeErrors;       /**< Number of failed write tag commands */
  DWORD errorConds;    /**< Number of temporary error conditions (over temp, low voltage) occured */
  DWORD setupErrs;         /**< Number of invalid setup errors */
  DWORD invalidCmds;       /**< Number of invalid (not supported) commands received */
};
```

2.4.2. GETTING DATA

NUR2 diagnostics data can be fetched from module with NurApiDiagGetReport() function.

For example:

```
struct NUR_DIAG_REPORT report;
NurApiDiagGetReport(hApi, 0, &report, sizeof(report));
```

See also in documentation:

```
int NurApiDiagSetConfig(HANDLE hApi, DWORD flags, DWORD interval);

int NurApiDiagGetConfig(HANDLE hApi, DWORD *flags, DWORD *interval);

int NurApiDiagGetReport(HANDLE hApi, DWORD flags, struct NUR_DIAG_REPORT
*report, DWORD reportSize);
```

3. NOT SUPPORTED IN NUR2

Some of the NUR05W and NUR10W module series functionality is not supported in NUR2 based modules.

3.1. MODULE SETUP FIELDS

Following fields in module setup (`struct NUR_MODULESETUP`) are not supported, setting them does not affect in any way

- rxDecoding
    - Automatically controlled by rfProfile
- txModulation
    - Automatically controlled by rfProfile
- autotune
    - Always enabled in NUR2
- rxSensitivity
    - Not needed. Read range can be controlled better via txLevel in NUR2

3.1.1. LINKFREQ BACKWARDS COMPATIBILITY

NUR2 module has backwards compatibility enabled with linkFreq field. You can use linkFreq field to change rfProfile.

- linkFreq 160000 = rfProfile 0 (robust)
- linkFreq 256000 = rfProfile 1 (nominal)
- linkFreq 320000 = rfProfile 2 (high speed)

3.2. XTID INVENTORY

Reading XTID memory during inventory automatically is not supported in NUR2.

Following functions are not supported

- `int NurApiConfigXTIDInventory(HANDLE hApi, BOOL dataOnly, BOOL includeHeader);`

3.3. CUSTOM EXCHANGE

NUR2 devices does not support custom GEN2 commands.

Following functions are not supported

- `int NurApiCustomReadSingulatedTag32(HANDLE hApi, DWORD rdCmd, BYTE cmdBits, DWORD rdBank, BYTE bankBits, DWORD passwd, BOOL secured, BYTE sBank, DWORD sAddress, int sMaskBitLength, BYTE *sMask, DWORD rdAddress, int rdByteCount, BYTE *rdBuffer);`
- `int NurApiCustomReadTagByEPC(HANDLE hApi, DWORD rdCmd, BYTE cmdBits, DWORD rdBank, BYTE bankBits, DWORD passwd, BOOL secured, BYTE *epcBuffer, DWORD epcBufferLen, DWORD rdAddress, int rdByteCount, BYTE *rdBuffer);`
- `int NurApiCustomReadTag32(HANDLE hApi, DWORD rdCmd, BYTE cmdBits, DWORD rdBank, BYTE bankBits, DWORD passwd, BOOL secured, DWORD rdAddress, int rdByteCount, BYTE *rdBuffer);`
- `int NurApiDisableCustomReselect(HANDLE hApi);`
- `int NurApiCustomWriteSingulatedTag32(HANDLE hApi, DWORD wrCmd, BYTE cmdBits, DWORD wrBank, BYTE bankBits, DWORD passwd, BOOL secured, BYTE sBank, DWORD sAddress, int sMaskBitLength, BYTE *sMask, DWORD wrAddress, int wrByteCount, BYTE *wrBuffer);`
- `int NurApiCustomWriteTagByEPC(HANDLE hApi, DWORD wrCmd, BYTE cmdBits, DWORD wrBank, BYTE bankBits, DWORD passwd, BOOL secured, BYTE *epcBuffer, DWORD epcBufferLen, DWORD wrAddress, int wrByteCount, BYTE *wrBuffer);`
- `int NurApiCustomWriteTag32(HANDLE hApi, DWORD wrCmd, BYTE cmdBits, DWORD wrBank, BYTE bankBits, DWORD passwd, BOOL secured, DWORD wrAddress, int wrByteCount, BYTE *wrBuffer);`

3.4. GEN2V2 FUNCTIONALITY

NUR2 does not support Gen2V2 commands.

Following functions are not supported

- `int NurApiGen2v2Untraceable(HANDLE hApi, DWORD passwd, struct NUR_UNTRACEABLE_PARAM *pUtrace, DWORD szParam);`
- `int NurApiGen2v2UntraceableByEPC(HANDLE hApi, DWORD passwd, BYTE *epcBuffer, DWORD epcBufferLen, struct NUR_UNTRACEABLE_PARAM *pUtrace, DWORD szParam);`
- `int NurApiGen2v2Untraceable32(HANDLE hApi, DWORD passwd, BYTE sBank, DWORD sAddress, int sMaskBitLength, BYTE *sMask, struct NUR_UNTRACEABLE_PARAM *pUtrace, DWORD szParam);`
- `int NurApiGen2v2Authenticate32(HANDLE hApi, BOOL secured, DWORD passwd, BYTE sBank, DWORD sAddress, int sMaskBitLength, BYTE *sMask, struct NUR_AUTHENTICATE_PARAM *pAuth, DWORD szParam, struct NUR_AUTHENTICATE_RESP *pResp);`
- `int NurApiGen2v2AuthenticateByEPC(HANDLE hApi, BOOL secured, DWORD passwd, BYTE *epcBuffer, DWORD epcBufferLen, struct NUR_AUTHENTICATE_PARAM *pAuth, DWORD szParam, struct NUR_AUTHENTICATE_RESP *pResp);`
- `int NurApiGen2v2Authenticate(HANDLE hApi, BOOL secured, DWORD passwd, struct NUR_AUTHENTICATE_PARAM *pAuth, DWORD szParam, struct NUR_AUTHENTICATE_RESP *pResp);`
- `int NurApiGen2v2ReadBuffer(HANDLE hApi, BOOL secured, DWORD passwd, WORD bitAddress, WORD bitCount, BYTE *buffer, int *actualBits);`
- `int NurApiGen2v2ReadBuffer32(HANDLE hApi, BOOL secured, DWORD passwd, BYTE sBank, DWORD sAddress, DWORD sMaskBitLength, BYTE *sMask, WORD bitAddress, WORD bitCount, BYTE *buffer, int *actualBits);`
- `int NurApiGen2v2ReadBufferByEPC(HANDLE hApi, BOOL secured, DWORD passwd, BYTE *epcBuffer, DWORD epcBufferLen, WORD bitAddress, WORD bitCount, BYTE *buffer, int *actualBits);`

3.5. PROPRIETARY EXTENSIONS

NUR2 does not support tag vendor's proprietary extensions.

Following functions are not supported

- `int NurApiNXPReadProtect(HANDLE hApi, DWORD passwd, BOOL protect, BYTE sBank, DWORD sAddress, int sMaskBitLength, BYTE *sMask);`
- `int NurApiNXPEAS(HANDLE hApi, DWORD passwd, BOOL set, BYTE sBank, DWORD sAddress, int sMaskBitLength, BYTE *sMask);`
- `int NURAPICONV NurApiNXPAlarm(HANDLE hApi);`
- `int NURAPICONV NurApiNXPStartAlarmStream(HANDLE hApi);`
- `int NURAPICONV NurApiNXPStopAlarmStream(HANDLE hApi);`
- `BOOL NURAPICONV NurApiIsNXPAlarmStreamRunning(HANDLE hApi);`