NURAPI ANTENNA MAPPING

SW APPLICATION NOTE

VERSION HISTORY

Version	Date	Author	Changes
1	2016-05-30	ML	Initial release
2	2016-05-30	Mky	Extra beams

TABLE OF CONTENTS

1. ANTENNA MAPPING API

NurApi used to use logical antenna id's to control which antennas are enabled for RF operations. As of NurApi version 1.7.9.0, antenna api is extended to support physical to logical antenna mapping. Antenna mapping is used to map physical antenna name to NurApi logical antenna id.

Each device provides its own physical to logical antenna mapping. This is more convenient to use for programmer.

1.1. PREREQUISITES FOR NEW ANTENNA API

- At least native NurApi.dll v1.7.9
- At least C# NurApiDotNet.dll v.1.7.9.0
- Nur reader firmware v5.0-A

2. READER ANTENNA MAPPINGS

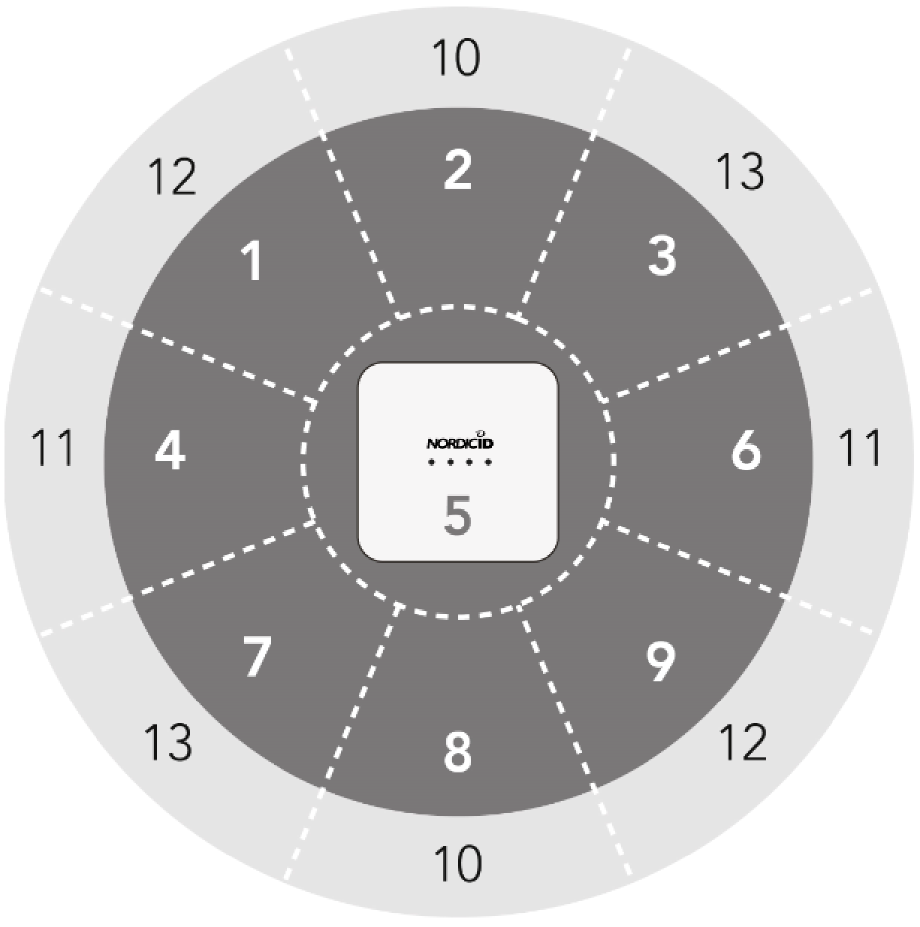
2.1. NORDIC ID AR55 MAPPING

Nordic ID AR55 with far beams provides 13 dual-linear polarized reading sectors, 26 beams in total. First 9 sectors are taken into consideration when calculating the tag positions. The last 4 sectors are for providing a larger read range i.e. extra beams.

The first models of AR55 provides 9 dual-linear polarized reading sectors, 18 beams in total.

Each beam provides horizontal (X) and vertical (Y) polarized beams.

2.1.1. PHYSICAL ANTENNA LOCATIONS



■ tag coordinate available

□ tag coordinate not available

2.1.2. LOGICAL TO PHYSICAL MAPPING

Logical antenna id	Physical antenna name	Logical antenna id	Physical antenna name
0	Beam1.X	14	Beam8.X
1	Beam1.Y	15	Beam8.Y
2	Beam2.X	16	Beam9.X
3	Beam2.Y	17	Beam9.Y
4	Beam3.X	18	Beam10.X*
5	Beam3.Y	19	Beam10.Y*
6	Beam4.X	20	Beam11.X*
7	Beam4.Y	21	Beam11.Y*
8	Beam5.X	22	Beam12.X*
9	Beam5.Y	23	Beam12.Y*
10	Beam6.X	24	Beam13.X*
11	Beam6.Y	25	Beam13.Y*
12	Beam7.X	26	AUX1*
13	Beam7.Y	27	AUX2*

* If the device is not equipped with the extra beam antennas, logical antenna id 18 = AUX1, 19 = AUX2 and 20 = AUX3 and the rest are not used.

2.2. NORDIC ID SAMPO S1 MAPPING

Nordic ID Sampo S1 provides one internal antenna and 3 (optional) external antenna ports.

2.2.1. LOGICAL TO PHYSICAL MAPPING

Logical antenna id	Physical antenna name
0	Internal
1	AUX1
2	AUX2
3	AUX3

2.3. NORDIC ID STIX MAPPING

Nordic ID Stix provides one internal antenna.

2.3.1. LOGICAL TO PHYSICAL MAPPING

Logical antenna id	Physical antenna name
0	Internal

2.4. NORDIC ID MERLIN MAPPING

Nordic ID Merlin handheld provides two different antenna configurations.

- Single linear antenna configuration
- Cross dipole antenna configuration. Control for both horizontal (X) and vertical (Y) antenna polarizations.

2.4.1. LOGICAL TO PHYSICAL MAPPING

Nordic ID Merlin UHF RFID

Logical antenna id	Physical antenna name
0	Linear

Nordic ID Merlin UHF RFID Cross Dipole

Logical antenna id	Physical antenna name
0	CrossDipole.X
1	CrossDipole.Y

2.5. NORDIC ID MORPHIC MAPPING

Nordic ID Morphic handheld provides cross dipole antenna configuration.

Cross dipole antenna configuration contains antennas for both horizontal (X) and vertical (Y) polarizations.

NOTE: Antennas in Morphic are tilted 45 degrees.

2.5.1. LOGICAL TO PHYSICAL MAPPING

Nordic ID Morphic UHF RFID Cross Dipole

Logical antenna id	Physical antenna name
0	CrossDipole.X
1	CrossDipole.Y

2.6. NORDIC ID MEDEA MAPPING

Nordic ID Medea handheld provides three different antenna configurations.

- Single linear antenna configuration
- Cross dipole antenna configuration. Control for both horizontal (X) and vertical (Y) antenna polarizations.
- Multipurpose antenna configuration. In addition to cross dipole configuration this configurations adds circular and proximity antennas.

2.6.1. LOGICAL TO PHYSICAL MAPPING

Nordic ID Medea UHF RFID

Logical antenna id	Physical antenna name
0	Linear

Nordic ID Medea UHF RFID Cross Dipole

Logical antenna id	Physical antenna name
0	CrossDipole.X
1	CrossDipole.Y

Nordic ID Medea UHF RFID One

Logical antenna id	Physical antenna name
0	CrossDipole.X
1	CrossDipole.Y
2	Proximity
3	Circular

3. PROGRAMMING

This sections describes new functionality added in NurApi v1.7.9.0 to support new physical antenna mapping api.

3.1. NEW IN NURAPI V1.7.9.0

- Support for logical antennas up to 32 antennas.
- Support for physical antenna mappings

3.1.1. NURAPI C API CHANGES

Functions added:

```
This function retrieves the antenna mapping from the device i.e. the
physical names of the antennas.

int NurApiGetAntennaMap(HANDLE hApi, struct NUR_ANTENNA_MAPPING
*antennaMap, int *nrMappings, int maxnMappings, DWORD szMapping);
```

```
This function enables the physical antennas that are specified as comma
separated string parameter.

int NurApiEnablePhysicalAntenna(HANDLE hApi, BOOL disableOthers, const
TCHAR *commaSeparated);
```

```
This function disables the physical antennas that are specified as comma
separated string parameter.

int NurApiDisablePhysicalAntenna(HANDLE hApi, const TCHAR
*commaSeparated);
```

```
This function return NUR_NO_ERROR if all physical antennas specified in
comma separated string are enabled, otherwise error is returned.

int NurApiIsPhysicalAntennaEnabled(HANDLE hApi, const TCHAR
*commaSeparated);
```

```
This function maps given logical antenna identifier to a physical name of
an antenna.

int NurApiAntennaIdToPhysicalAntenna(HANDLE hApi, int antennaId, TCHAR
*name, int maxNameLen);
```

```
This function maps given physical antenna name to an logical antenna
identifier.

int NurApiPhysicalAntennaToAntennaId(HANDLE hApi, const TCHAR *name, int
*antennaId);
```

Fields added to struct NUR_MODULESETUP:

```
Bitmask of of enabled antennas, support up to 32 antennas. Value 0x1 -
0xFFFFFFFF. Each bit represents logical antenna id, bit0 = first antenna
id.

DWORD antennaMaskEx;
```

Flags added to enum NUR_MODULESETUP_FLAGS:

```
antennaMaskEx field in struct NUR_MODULESETUP is valid

NUR_SETUP_ANTMASKEX
```

See more detailed information in NurApi documentation, in file “NurApi C Documentation.chm”

3.1.2. NURAPI C# API CHANGES

Properties and functions added to NurApi class

Get/Set bitmask of enabled antennas, support up to 32 antennas. Value 0x1 - 0xFFFFFFFF. Each bit represents logical antenna id, bit0 = first antenna id. `uint AntennaMaskEx`
Get list of available physical antennas supported by connected reader. `List<string> AvailablePhysicalAntennas`
Get or sets list of enabled physical antennas in connected reader. `List<string> EnabledPhysicalAntennas`

This function enables the physical antennas that are specified as comma separated string parameter.
 	
`void EnablePhysicalAntenna(string commaSeparated , bool disableOthers)`
 	
`void EnablePhysicalAntenna(string commaSeparated)`

This function disables the physical antennas that are specified as comma separated string parameter. `void DisablePhysicalAntenna(string commaSeparated)`
This function returns true if all physical antennas specified in comma separated string are enabled, otherwise false is returned. `bool IsPhysicalAntennaEnabled(string commaSeparated)`
This function maps given logical antenna identifier to a physical name of an antenna. `string NurAntennaIdToPhysicalAntenna(int nurAntennaId)`
This function maps given physical antenna name to an logical antenna identifier. `int NurPhysicalAntennaToAntennaId(string name)`

Properties added to NurApi.Tag class

The physical antenna's name from where the last best RSSI was detected. `string PhysicalAntenna`

See more detailed information in NurApi C# documentation, in file “NurApi .NET Documentation.chm”

3.2. C# EXAMPLES

In C# examples the *hNur* is considered to be a valid and connected NurApi object.

3.2.1. CONTROLLING ANTENNAS

C# Controlling antennas

```
// AR55: Enable middle beams (4,5,6) both polarities. Disables other beams.
hNur.EnablePhysicalAntenna("Beam4,Beam5,Beam6", true);

// AR55: Enable middle beams (4,5,6) only X polarities. Disables other beams.
hNur.EnablePhysicalAntenna("Beam4.X,Beam5.X,Beam6.X", true);

// Sampo: Enable internal and external port 3. Disables other antennas.
hNur.EnablePhysicalAntenna("Internal,AUX3", true);

// Sampo/Module: Enable  external port 2. leave other antennas untouched.
hNur.EnablePhysicalAntenna("AUX2");

// All models: Enable all available antennas
hNur.EnablePhysicalAntenna("All", true);

// AR55: Disable beam 1.
hNur.DisablePhysicalAntenna("Beam1");

// AR55: Test if beam 4 both polarities are enabled
if (hNur.IsPhysicalAntennaEnabled("Beam4"))
{
    // Enabled, do something
}
```

3.2.2. ONE BY ONE INVENTORY

C# Perform one by one inventory on each main (all polarities) antennas

```
// Get all available physical antennas (including all polarities of antennas)
List<string> inventoryAntennas = hNur.AvailablePhysicalAntennas;

// Strip off polarities and remove duplicates from list (distinct)
for (int i = inventoryAntennas.Count-1; i >= 0; i--)
{
    // Look for polarity and strip if found
    int pos = inventoryAntennas[i].IndexOf('.');
    if (pos > 0)
        inventoryAntennas[i] = inventoryAntennas[i].Remove(pos);

    // Remove duplicates
    if (i < (inventoryAntennas.Count-
1) && inventoryAntennas[i] == inventoryAntennas[i + 1])
        inventoryAntennas.RemoveAt(i + 1);
}

// Use always autoswitch antenna, so inventory goes through all polarities.
hNur.SelectedAntenna = NurApi.ANTENNAID_AUTOSELECT;

// Loop through all antennas (w/o polarities) and perform inventory
foreach (string physAntenna in inventoryAntennas)
{
    // Enable only current antenna
    hNur.EnablePhysicalAntenna(physAntenna, true);

    // Clear module and api tag storage
    hNur.ClearTagsEx();

    // Perform inventory (with module stored settings)
    NurApi.InventoryResponse invResp = hNur.Inventory();
    System.Diagnostics.Debug.WriteLine(string.Format("{0}: Found {1} tags", physAntenna,
 invResp.numTagsFound));

    if (invResp.numTagsFound > 0)
    {
        // Fetch tags from module to api tag storage
        hNur.FetchTags();

        // Loop through tags
        foreach (NurApi.Tag tag in hNur.GetTagStorage())
        {
            System.Diagnostics.Debug.WriteLine(string.Format("EPC '{0}' RSSI {1} dBm", t
ag.GetEpcString(), tag.rssi));
        }
    }
}
```

3.2.3. INVENTORY STREAM

C# start/stop inventory stream and read tag's physical antenna in streaming event

```
// Event is fired after asynchronous inventory finished in the reader
void MyInventoryStreamEvent(object sender, NurApi.InventoryStreamEventArgs e)
{
    // Loop through tags
    foreach (NurApi.Tag tag in hNur.GetTagStorage())
    {
        System.Diagnostics.Debug.WriteLine(string.Format("EPC '{0}' RSSI {1} dBm ANT '{2
}'", tag.GetEpcString(), tag.rssi, tag.PhysicalAntenna));
    }

    // Restart inventory if stopped by reader
    if (e.data.stopped)
        hNur.StartInventoryStream();
}

// Call this to start asynchronous tag streaming
void StartInventory()
{
    // Setup event for tag stream
    hNur.InventoryStreamEvent += MyInventoryStreamEvent;

    // Enable all antennas
    hNur.EnabledPhysicalAntennas = hNur.AvailablePhysicalAntennas;

    // Automatically switch between all enabled antennas
    hNur.SelectedAntenna = NurApi.ANTENNAID_AUTOSELECT;

    // Start inventory tag streaming (with module stored inventory settings)
    hNur.StartInventoryStream();
}

// Call this to stop asynchronous tag streaming
void StopInventory()
{
    // Stop inventory tag streaming
    hNur.StopInventoryStream();

    // Remove event for tag stream
    hNur.InventoryStreamEvent -= MyInventoryStreamEvent;
}
```

3.2.4. MORE C# EXAMPLES

More C# NurApi examples found in NurDistribution/Samples folder.

3.3. C/C++ EXAMPLES

In C/C++ calls the *hNur* parameter is considered to be valid handle to the currently connected NUR API handle.

3.3.1. CONTROLLING ANTENNAS

C/C++ Controlling antennas

```
// AR55: Enable middle beams (4,5,6) both polarities. Disables other
beams.
NurApiEnablePhysicalAntenna(hNur, _T("Beam4,Beam5,Beam6"), TRUE);

// AR55: Enable middle beams (4,5,6) only X polarities. Disables other
beams.
NurApiEnablePhysicalAntenna(hNur, _T("Beam4.X,Beam5.X,Beam6.X"), TRUE);

// Sampo: Enable internal and external port 3. Disables other antennas.
NurApiEnablePhysicalAntenna(hNur, _T("Internal,AUX3"), TRUE);

// All models: Enable all available antennas
NurApiEnablePhysicalAntenna(hNur, _T("All"), TRUE);

// AR55: Disable beam 1.
NurApiDisablePhysicalAntenna(hNur, _T("Beam1"));

// AR55: Test if beam 4 both polarities are enabled
if (NurApiIsPhysicalAntennaEnabled(hNur, _T("Beam4")) == NUR_NO_ERROR)
{
    // Enabled, do something
}
```

3.3.2. ONE BY ONE INVENTORY

C++ Perform one by one inventory on each main (all polarities) antennas

```
struct NUR_ANTENNA_MAPPING map[NUR_MAX_ANTENNAS_EX];
int nrMappings = 0;
int error, i;
TCHAR *lastAntenna = NULL;

// Get list of available antennas on reader
error = NurApiGetAntennaMap(hApi, map, &nrMappings, _countof(map),
sizeof(map[0]));
if (error != NUR_NO_ERROR)
     return; // Handle error..

// Use always autoswitch antenna, so inventory goes through all
polarities.
struct NUR_MODULESETUP setup;
setup.selectedAntenna = NUR_ANTENNAID_AUTOSELECT;
error = NurApiSetModuleSetup(hApi, NUR_SETUP_SELECTEDANT, &setup,
sizeof(setup));
if (error != NUR_NO_ERROR)
     return; // Handle error..

// Loop through all antennas (w/o polarities) and perform inventory
for (i=0; i<nrMappings; i++)
{
     struct NUR_INVENTORY_RESPONSE invResp;
     TCHAR *polarityPtr;

     // Check for polarity mark '.' in antenna name
     polarityPtr = _tcschr(map[i].name, '.');
     if (polarityPtr) *polarityPtr = '\0';

     // If last antenna is same as current without polarity, skip
     // We want to do inventory only once per main antenna (all polarities)
     if (lastAntenna && _tcscmp(lastAntenna, map[i].name) == 0)
          continue;

     // Store last antenna string pointer
     lastAntenna = map[i].name;

    // Enable only current antenna
     NurApiEnablePhysicalAntenna(hApi, map[i].name, TRUE);

     // Clear module and api tag storage
     NurApiClearTags(hApi);

     // Perform inventory (with module stored settings)
     NurApiSimpleInventory(hApi, &invResp);
     _tprintf(_T("%s: Found %d tags\r\n"), map[i].name,
invResp.numTagsFound);

/*....continues in next page....*/
/*....from previous page....*/

     if (invResp.numTagsFound > 0)
     {
          int tagCount, tagIdx;
          TCHAR epcStr[128];

          // Fetch tags from module to api tag storage
          NurApiFetchTags(hApi, TRUE, NULL);

          // Get tag count from api tag storage
          NurApiGetTagCount(hApi, &tagCount);

          // Loop through tags
          for (tagIdx=0; tagIdx<tagCount; tagIdx++)
          {
               struct NUR_TAG_DATA_EX tagData;
               error = NurApiGetTagDataEx(hApi, tagIdx, &tagData,
sizeof(tagData));
               if (error != NUR_NO_ERROR)
                    return; // Handle error..
               EpcToString(tagData.epc, tagData.epcLen, epcStr);
               _tprintf(_T("EPC '%s' RSSI %d dBm\r\n"), epcStr,
tagData.rssi);
          }
     }
}
```

3.3.3. MORE C/C++ EXAMPLES

More C/C++ NurApi examples found in NurDistribution/Samples/NurApiExample folder.