NOTE 49

ASI2816/RMT416 HD Radio Images over SNMP

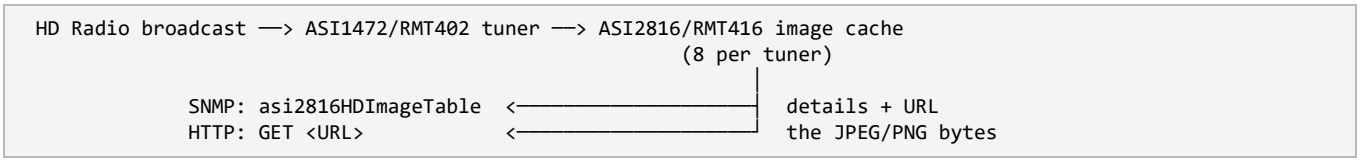
Rev 1

Applies to: ASI2816/RMT416 with ASI1472/RMT402 HD Radio tuner modules, firmware **0.22.0 or later** (AUDIOSCIENCE-ASI2816-MIB revision 10, "HD image fetch URL"). The ASI2816 and RMT416 run the same firmware and use the same MIB, AUDIOSCIENCE-ASI2816-MIB, so everything below applies to either.

HD Radio stations broadcast two kinds of picture:

- **Station logos**: one per HD program (HD1, HD2, ...), repeated every few minutes.
- **Artist Experience images**: album art or a picture for the song or advert playing now.

The ASI2816/RMT416 decodes these pictures and keeps the most recent ones for each tuner. **SNMP does not carry the picture itself.** For each cached picture, SNMP gives you a row of details (what it is, which program, which station) and a URL. You download the picture from that URL with an ordinary HTTP GET.



1. Before you start

Requirement	Details
Firmware	0.22.0 or later. Earlier firmware has no image URL column.
SNMP	SNMPv2c on UDP port 161. The read community is `public` unless it has been changed. The web interface's built-in SNMP browser uses `public`.
HTTP	The SNMP manager must also reach the ASI2816/RMT416 on TCP port 80. Image URLs need **no login**.
Tuning	The tuner must be on an HD Radio station that sends images. Pictures arrive over the air, so allow a few minutes after tuning. Stations repeat each file several times over about two minutes.
Network address	The URLs use the device's own IPv4 address. If the device does not know its address, the URL column is empty.

Load the MIB files (optional but recommended)

The device serves its own MIB files, so nothing else needs to be installed:

- Web interface: **SNMP** panel, **Download MIB files** button, or
- `http://<device-ip>/ui2/snmp/mibs.zip` (no login needed).

For net-snmp, unzip the files into `~/.snmp/mibs` and add `-m +AUDIOSCIENCE-ASI2816-MIB` to your commands. Every example below also gives the numeric OID, so the MIB files are not required.

2. The SNMP tables involved

All OIDs below start with the ASI2816/RMT416 tuner subtree:

```
1.3.6.1.4.1.58172.24.2.2816.2.1      (written as <T> below)
```

Table	OID	Row index	What you use it for
`asi2816TunerCtrlTable`	`<T>.2`	tuner	Which tuner is which (slot, tuner number), plus its current band and frequency
`asi2816HDTunerCtrlTable`	`<T>.3`	tuner	Which HD program the tuner is playing now
`asi2816HDProgramTable`	`<T>.10`	tuner . program	Which logo belongs to each HD program
`asi2816TunerPSDTable`	`<T>.11`	tuner	Which album-art picture belongs to the song playing now
`asi2816HDImageTable`	`<T>.14`	tuner . image slot	Every cached picture, with its download URL

The image table: `asi2816HDImageTable` (`<T>.14.1.<column>.<tuner>.<slot>`)

Each tuner has up to 8 image slots. Empty slots have no row.

Column	Name	Type	Meaning
2	`tunerHDImageLotKnown`	TruthValue	`true(1)` when the LOT id is known. Zero is a valid LOT id.
3	`tunerHDImageLotId`	Integer32	The station's file number (LOT id) for this picture
4	`tunerHDImageMime`	String	`image/jpeg` or `image/png`
5	`tunerHDImageKind`	Integer	`ae(1)` = album art, `logo(2)` = station logo, `none(0)` = not stated
6	`tunerHDImagePort`	Integer32	HD data port the file arrived on
7	`tunerHDImageProgramIndex`	Integer32	HD program the picture belongs to (1 = HD1 ... 8 = HD8, 0 = not stated)
8	`tunerHDImageBand`	Integer	Band when the picture was received: `am(1)`, `fm(2)`
9	`tunerHDImageFreqHz`	Integer32	Frequency in Hz when the picture was received
10	`tunerHDImageUrl`	String	Download URL. Empty when the picture's bytes are not held.

Columns that say which picture is current

OID	Name	Meaning
`<T>.2.1.2.<tuner>`	`tunerCtrlModuleSlot`	Module slot (1-4)
`<T>.2.1.3.<tuner>`	`tunerCtrlTunerNumber`	Tuner on that module (1-4)
`<T>.2.1.4.<tuner>`	`tunerCtrlFreqHz`	Current frequency in Hz
`<T>.2.1.5.<tuner>`	`tunerCtrlBand`	Current band: `am(1)`, `fm(2)`, `wb(3)`, `dab(4)`
`<T>.3.1.2.<tuner>`	`tunerCtrlHDProgramIndex`	HD program playing now (1-8)
`<T>.10.1.2.<tuner>.<program>`	`tunerHDProgramLabel`	Program name, e.g. the station's HD2 name
`<T>.10.1.3.<tuner>.<program>`	`tunerHDProgramLogoKnown`	`true(1)` when the program has a logo
`<T>.10.1.4.<tuner>.<program>`	`tunerHDProgramLogoLotId`	LOT id of that program's logo
`<T>.11.1.8.<tuner>`	`tunerCtrlPSDAELotKnown`	`true(1)` when the current song has album art
`<T>.11.1.9.<tuner>`	`tunerCtrlPSDAELotId`	LOT id of the current song's album art

3. Step by step with net-snmp and curl

In these examples the device is `192.168.1.50`. Replace it with the address of your ASI2816/RMT416.

Step 1: Find the tuner's index

Tuners are numbered 1, 2, 3, ... across all modules. List the module slot and the tuner number on that module for each index:

```
snmpwalk -v2c -c public 192.168.1.50 1.3.6.1.4.1.58172.24.2.2816.2.1.2.1.2
```

```
snmpwalk -v2c -c public 192.168.1.50 1.3.6.1.4.1.58172.24.2.2816.2.1.2.1.3
```

The last number of each returned OID is the tuner index. For example, if the row ending `.5` shows slot 2 and tuner number 1, then tuner index 5 is the first tuner on the module in slot 2.

Step 2: List the pictures cached on that tuner

Walk the image table and read the rows for your tuner (index 5 here):

```
snmpwalk -v2c -c public -m +AUDIOSCIENCE-ASI2816-MIB 192.168.1.50 AUDIOSCIENCE-ASI2816-MIB::asi2816HDImageTable
```

Without the MIB files, walk the numeric OID instead:

```
snmpwalk -v2c -c public 192.168.1.50 1.3.6.1.4.1.58172.24.2.2816.2.1.14
```

Example output for one slot (illustrative values; yours will differ):

```
tunerHDImageLotKnown.5.1 = INTEGER: true(1)
tunerHDImageLotId.5.1 = INTEGER: 1234
tunerHDImageMime.5.1 = STRING: image/png
tunerHDImageKind.5.1 = INTEGER: logo(2)
tunerHDImagePort.5.1 = INTEGER: 4096
tunerHDImageProgramIndex.5.1 = INTEGER: 1
tunerHDImageBand.5.1 = INTEGER: fm(2)
tunerHDImageFreqHz.5.1 = INTEGER: 101100000
tunerHDImageUrl.5.1 = STRING: http://192.168.1.50/dyn/v1/blob/<64 hex digits>
```

Step 3: Download the picture

Fetch the URL with any HTTP client. No login is needed:

```
curl -o tuner5_logo.png "http://192.168.1.50/dyn/v1/blob/<64 hex digits>"
```

The reply's `Content-Type` matches `tunerHDImageMime`. Pictures are at most 32 KB.

Step 4: Pick the picture for what is playing now

The image table holds whatever the tuner has received recently. It can include logos for other HD programs and, for a while after retuning, pictures from the previous station. To show the right picture:

Current station logo

1. Read `tunerCtrlHDProgramIndex.<tuner>`. This is the program playing now, *P*.
2. Read `tunerHDProgramLogoKnown.<tuner>.<P>`. If it is `true(1)`, read `tunerHDProgramLogoLotId.<tuner>.<P>`. This is the logo's LOT id, *L*.
3. In the image table, find the row for this tuner where:
   - `tunerHDImageLotId` = *L*;
   - `tunerHDImageKind` = `logo(2)`;
   - `tunerHDImageBand` and `tunerHDImageFreqHz` equal the tuner's current `tunerCtrlBand` and `tunerCtrlFreqHz`.
4. That row's `tunerHDImageUrl` is the logo.

Current album art

1. Read `tunerCtrlPSDAELotKnown.<tuner>`. If it is `true(1)`, read `tunerCtrlPSDAELotId.<tuner>`. This is the picture's LOT id, *A*.
2. In the image table, find the row for this tuner where:

- `tunerHDImageLotId` = *A*;
- `tunerHDImageKind` = `ae(1)`;
- the band and frequency match the tuner's current band and frequency.

3. That row's `tunerHDImageUrl` is the album art.

Why match band and frequency as well? LOT ids are small numbers that stations reuse, and two stations can use the same LOT id for different files. Matching the band and frequency makes sure the picture came from the station the tuner is on now.

If no row matches, the picture has been announced but not yet received. Check again later.

4. Example script (Python)

This script downloads the current logo and album art for one tuner. It needs only Python 3.6 or later: it uses the standard library for both the SNMP queries and the HTTP downloads, so there is nothing else to install and it runs on Windows, macOS and Linux. Save it as `asi2816_hd_images.py`.

```
#!/usr/bin/env python3
"""Download the current HD Radio station logo and album art from one
ASI2816/RMT416 tuner (firmware 0.22.0 or later).

Uses only the Python standard library (Python 3.6 or later): SNMPv2c over
UDP for the details, HTTP for the pictures.

usage: asi2816_hd_images.py <device-ip> <tuner-index> [community]
"""
import socket
import sys
import urllib.error
import urllib.request

T = '1.3.6.1.4.1.58172.24.2.2816.2.1'   # ASI2816/RMT416 tuner subtree



# --- Minimal SNMPv2c client (GET and GETNEXT) -------------------------------

def _tlv(tag, value):
    n = len(value)
    if n < 0x80:
        size = bytes([n])
    else:
        b = n.to_bytes((n.bit_length() + 7) // 8, 'big')
        size = bytes([0x80 | len(b)]) + b
    return bytes([tag]) + size + value



def _int(n):
    return _tlv(0x02, n.to_bytes(n.bit_length() // 8 + 1, 'big', signed=True))



def _oid(oid):
    parts = [int(p) for p in oid.strip('.').split('.')]
    out = bytearray([40 * parts[0] + parts[1]])
    for p in parts[2:]:
        chunk = [p & 0x7F]
        p >>= 7
        while p:
            chunk.append(0x80 | (p & 0x7F))
            p >>= 7
        out += bytes(reversed(chunk))
    return _tlv(0x06, bytes(out))
```

```
def _items(buf):
    """(tag, value) of each TLV in buf."""
    i = 0
    while i < len(buf):
        tag, n = buf[i], buf[i + 1]
        i += 2
        if n & 0x80:
            k = n & 0x7F
            n = int.from_bytes(buf[i:i + k], 'big')
            i += k
        yield tag, buf[i:i + n]
        i += n


def _oid_str(b):
    parts = [b[0] // 40, b[0] % 40]
    n = 0
    for c in b[1:]:
        n = (n << 7) | (c & 0x7F)
        if not c & 0x80:
            parts.append(n)
            n = 0
    return '.'.join(str(p) for p in parts)


def _value(tag, val):
    if tag == 0x02:                          # INTEGER
        return int.from_bytes(val, 'big', signed=True)
    if tag in (0x41, 0x42, 0x43, 0x46):      # Counter32, Gauge32, TimeTicks, Counter64
        return int.from_bytes(val, 'big')
    if tag == 0x04:                          # OCTET STRING
        return val.decode('utf-8', 'replace')
    if tag == 0x06:                          # OBJECT IDENTIFIER
        return _oid_str(val)
    return None                              # NULL, noSuchObject/Instance, endOfMibView


class Snmp:
    def __init__(self, host, community='public', timeout=2.0, retries=2):
        self.addr = (host, 161)
        self.community = community.encode()
        self.timeout = timeout
        self.retries = retries
        self.request_id = 0

    def _request(self, pdu_type, oid):
        """(oid, value) answered to a GET (0xA0) or GETNEXT (0xA1)."""
        self.request_id = (self.request_id + 1) & 0x7FFFFFFF
        varbinds = _tlv(0x30, _tlv(0x30, _oid(oid) + b'\x05\x00'))
        pdu = _tlv(pdu_type, _int(self.request_id) + _int(0) + _int(0) + varbinds)
        msg = _tlv(0x30, _int(1) + _tlv(0x04, self.community) + pdu)  # 1 = v2c
        with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as s:
            s.settimeout(self.timeout)
            for _ in range(self.retries + 1):
                s.sendto(msg, self.addr)
                try:
                    reply = s.recv(65535)
                except socket.timeout:
                    continue
                body = next(_items(reply))[1]
                resp = list(_items(list(_items(body))[2][1]))
                if _value(*resp[0]) != self.request_id:
                    continue                 # a late reply to an earlier request
                if _value(*resp[1]):         # error-status
                    return oid, None
```

```
                varbind = next(_items(resp[3][1]))[1]
                (_, name), (tag, val) = _items(varbind)
                return _oid_str(name), _value(tag, val)
        raise TimeoutError('no SNMP reply from %s' % self.addr[0])

    def get(self, oid):
        return self._request(0xA0, oid)[1]

    def walk(self, prefix):
        """(oid, value) of every instance under prefix."""
        oid = prefix
        while True:
            next_oid, value = self._request(0xA1, oid)
            if value is None or not next_oid.startswith(prefix + '.'):
                return
            yield next_oid, value
            oid = next_oid


# --- HD Radio pictures ------------------------------------------------------

def main():
    if len(sys.argv) < 3:
        sys.exit(__doc__)
    host, t = sys.argv[1], int(sys.argv[2])
    snmp = Snmp(host, sys.argv[3] if len(sys.argv) > 3 else 'public')

    band = snmp.get('%s.2.1.5.%d' % (T, t))      # tunerCtrlBand
    freq = snmp.get('%s.2.1.4.%d' % (T, t))      # tunerCtrlFreqHz
    prog = snmp.get('%s.3.1.2.%d' % (T, t))      # tunerCtrlHDProgramIndex
    if band is None or prog is None:
        sys.exit('tuner %d: no HD tuner at this index' % t)

    def image(column, slot):                     # asi2816HDImageTable cell
        return snmp.get('%s.14.1.%d.%d.%s' % (T, column, t, slot))

    def find_slot(kind, lot_id):
        """Image slot on this tuner holding LOT id `lot_id` of `kind`
        (1 = ae, 2 = logo), received on the tuner's current band and
        frequency."""
        for oid, lot in snmp.walk('%s.14.1.3.%d' % (T, t)):   # tunerHDImageLotId
            slot = oid.rsplit('.', 1)[1]
            if (lot == lot_id and image(2, slot) == 1 and image(5, slot) == kind
                    and image(8, slot) == band and image(9, slot) == freq):
                return slot
        return None

    def fetch(name, kind, lot_id):
        slot = find_slot(kind, lot_id)
        if slot is None:
            print('%s: not received yet' % name)
            return
        url, mime = image(10, slot), image(4, slot)
        if not url:
            print('%s: picture bytes not held' % name)
            return
        ext = {'image/png': 'png', 'image/jpeg': 'jpg'}.get(mime, 'bin')
        out = 'tuner%d_%s.%s' % (t, name, ext)
        try:
            with urllib.request.urlopen(url, timeout=10) as resp:
                data = resp.read()
        except urllib.error.URLError as e:
            print('%s: download failed (%s): %s' % (name, url, e))
            return
        with open(out, 'wb') as f:
            f.write(data)
        print('%s: %s -> %s' % (name, url, out))
```

```
    if snmp.get('%s.10.1.3.%d.%d' % (T, t, prog)) == 1:        # tunerHDProgramLogoKnown
        fetch('logo', 2, snmp.get('%s.10.1.4.%d.%d' % (T, t, prog)))
    else:
        print('logo: none announced for HD%d' % prog)

    if snmp.get('%s.11.1.8.%d' % (T, t)) == 1:                  # tunerCtrlPSDAELotKnown
        fetch('albumart', 1, snmp.get('%s.11.1.9.%d' % (T, t)))
    else:
        print('albumart: none announced')


if __name__ == '__main__':
    main()
```

Example run, for tuner index 5:

```
python3 asi2816_hd_images.py 192.168.1.50 5
```

On Windows, use `py` in place of `python3`. The output names each picture's URL and the file it was saved to (illustrative values):

```
logo: http://192.168.1.50/dyn/v1/blob/<64 hex digits> -> tuner5_logo.jpg
albumart: http://192.168.1.50/dyn/v1/blob/<64 hex digits> -> tuner5_albumart.jpg
```

To use the script's SNMP functions in your own application, import it: `Snmp(host).get(oid)` returns one value, and `Snmp(host).walk(oid)` yields every `(oid, value)` pair under an OID.

5. Behaviour worth knowing

- **The URL changes when the picture changes.** The last part of the URL is the SHA-256 hash of the picture's bytes. A new song's album art therefore gets a new URL, and two slots holding the same file share one URL.
- **Cache freely.** The device marks each reply `Cache-Control: public, max-age=31536000, immutable`, because the bytes behind a URL never change. Your application only needs to download a URL it has not seen before.
- **Old URLs return 404.** Once the device drops a picture from its cache, its URL answers `404 Not Found`. It never returns a different picture.
- **8 slots per tuner.** When the cache is full, the device reuses slots in this order, oldest picture first within each group:
    1. empty slots;
    2. pictures from other stations, and logos that no program refers to any more;
    3. album art;
    4. other programs' logos.
- **Empty URL.** `tunerHDImageUrl` is empty when the picture has been announced but its bytes are not held (still arriving, or over the 32 KB limit), or when the device does not know its own IPv4 address.
- **How quickly SNMP sees changes.** The device checks each tuner about once a second, and the SNMP values change as soon as the device's own state does. How fast new pictures arrive depends mostly on how often the station sends them.
- **Not over SNMP.** Picture bytes are never sent over SNMP, only the URL. There are no SNMP traps for new pictures, so poll the PSD and program tables, or the image table, at whatever rate suits your application. Every few seconds is plenty.

- **AM and FM HD only.** Pictures come from HD Radio on AM and FM. DAB slideshow images are not supported.

Tech Support

If at any point you need help with this information please contact our Technical Support department at 585-271-8870 between the hours of 9AM and 5PM EST or by email at support@audioscience.com