Technical Information

SPEEDWIRE DEVICE DISCOVERY

1 Information on this Document

1.1 Validity

This document is valid for all SMA products with SMA Speedwire or SMA Speedwire/Webconnect.

No further support available

Please note that, except for the information available here, we cannot provide you with any additional development support for Speedwire Device Discovery.

1.2 Target Group

The activities described in this document must be performed by qualified persons only. Qualified persons must have the following skills:

- Knowledge of IP-based network protocols
- Training in the installation and configuration of IT systems
- Experience of working with Ethernet based fieldbuses
- Knowledge of and compliance with this document and all safety information

2 Speedwire

2.1 General Information on Speedwire

Speedwire is a wired, Ethernet based fieldbus for the implementation of powerful communication networks in decentralized PV systems.

Speedwire uses the internationally established Ethernet standard, the Ethernet based IP protocol as well as the communication protocol SMA Data2+ optimized for PV systems. This enables a consistent 100/1000 Mbit/s data transmission to the inverter as well as reliable monitoring, control, and regulation of the PV system.

The Speedwire network can be set up optionally in line, star or tree topology (see technical information "SMA Speedwire Fieldbus" at www.SMA-Solar.com).

2.2 Speedwire Device Discovery

With Speedwire Device Discovery, the IPv4 addresses of all the SMA products present in the network are queried. Each product equipped with SMA Speedwire can be detected in the local network by means of the Speedwire Device Discovery query function.

SMA products with Speedwire receive their IPv4 address as follows:

- Via DHCP from a router located in the network
- Via IPv4LL
- Via a manually set, non-adjustable configuration

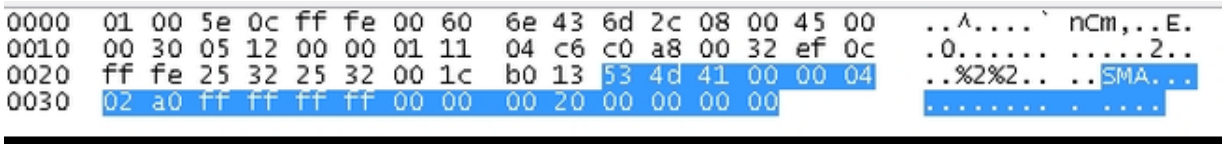
SMA products are set by default to the IPv4 address via DHCP. Consequently, the IPv4 address can change during operation depending on the router used. The Speedwire Device Discovery query function enables the IPv4 addresses of SMA products to be identified. If the IPv4 address is known and the Modbus® communication of the SMA product is activated, all other data of the product can be determined via Modbus® (see section 4 “Additional Information”, page 6).

The Speedwire fieldbus is structured so that the Speedwire Device Discovery service and other basic services configured as IP/UDP telegrams use the port 9522.

- Port 9522 is registered with the IANA (Internet Assigned Numbers Authority) for SMA Speedwire communication.
- One type of telegram used are multicast telegrams.

3 Discovery Query and Discovery Response

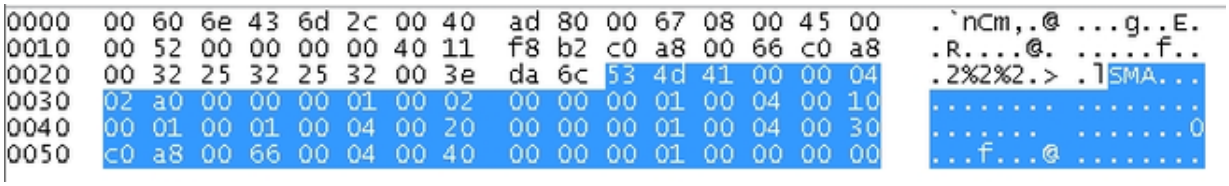
To prompt SMA products within a local network to report their presence, the following UDP datagram must be sent to the multicast address 239.12.255.254 via port 9522 (section marked in blue):



All SMA products which fulfill the following criteria will return a UDP datagram:

- ☐ A Speedwire interface must be present
- ☐ The queried product must be located in the same network as the end device from which the query is sent.
- ☐ It must be possible for the transmitted multicast to reach the product

The UDP datagram of the responding SMA products is structured as shown in the following figure (section marked in blue):



The IP address of the SMA products can be derived from the response. We recommend analyzing the response in order to exclude any errors.

Procedure:

- Analyze the UDP datagram up to and including the 18th byte (within the section marked in blue).

  ☑ If the sub-array "534d4100000402A00000000100020000000 1" is received as response, this indicates an SMA product.

  You can recognize the IPv4 addresses of SMA products by the packet originator addresses (see Section 2.1 "Code example in Java", page 3 and Section 2.2 "Code example in Python", page 4).

Selecting the Right Ethernet Adapter

The response of SMA products to the Speedwire Device Discovery query can be sent to the multicast address 239.12.255.254 or the address of the Ethernet adapter.

If several Ethernet adapters are used, make sure that the query is directed to the correct Ethernet adapter.

In case of problems, contact your network administrator.

3.1 Code Example in Java

The following example shows how the Speedwire Device Discovery query is used in Java. The emitting Ethernet adapter is hard coded.

```
import java.io.*;
import java.net.*;
import java.util.Arrays;

public class SPWDisco {
  /**
   * client to demonstrate the usage of UDP multicast sockets.
   * @throws IOException
 * @throws InterruptedException
   */
  public void multicast() throws IOException, InterruptedException {
    try {
      InetAddress multicastAddress = InetAddress.getByName("239.12.255.254"); // MC Address
      InetAddress adpAdr           = InetAddress.getByName("192.168.0.50");   // host adapter
      final int port = 9522;                                                  // standard port for SPW
      MulticastSocket socket = new MulticastSocket(port);

      socket.setInterface(adpAdr);
      socket.setReuseAddress(true);
      socket.setSoTimeout(5000);
      socket.joinGroup(multicastAddress);

      // send discovery command
      byte[] txbuf = javax.xml.bind.DatatypeConverter.parseHexBinary("534d4100000402a0ffffffff0000002000000000");

      DatagramPacket hi = new DatagramPacket(txbuf, txbuf.length, multicastAddress, port);
      socket.send(hi);
      System.out.println("SPW discover sent\n");


      do {
          byte[] rxbuf = new byte[8192];
          DatagramPacket packet = new DatagramPacket(rxbuf, rxbuf.length);
          socket.receive(packet);
          scanPacket(packet, adpAdr);
        } while (true);                 // should leave loop by SocketTimeoutException
      } catch (SocketTimeoutException e) {
        System.out.println("\n5 sec w/o receiving answers - Timeout, terminating\n");
      }
    }

  private void scanPacket(DatagramPacket packet, InetAddress hostAdr) throws IOException {
      boolean found = true;
      // expected answer:
      byte[] refArr   = javax.xml.bind.DatatypeConverter.parseHexBinary  ("534d4100000402A000000001000200000001");
      byte[] buf      = packet.getData();
      for( int index = 0; index < refArr.length; ++index ) {
           if( buf[index] != refArr[index]) {found = false; }
      }
      if (found==true) {
         InetAddress addr = packet.getAddress();
         if ( Arrays.equals( addr.getAddress(), hostAdr.getAddress())) {
             System.out.println("Response from host Adapter: " + addr);
         }
         else
         {
            System.out.println("Response from Device: " + addr);
         }
      }
  }

      /**
       * MAIN
       */
      public static void main(String[] args) throws Exception {
          SPWDisco client = new SPWDisco();
        client.multicast();
      }
    }
```

3.2 Code Example in Python

The following example shows how the Speedwire Device Discovery query is used in Python. The emitting Ethernet adapter is hard coded.

```
#
# Example to perform a speedwire discovery using python
#
from twisted.internet.protocol import DatagramProtocol
from twisted.internet import reactor
from twisted.application.internet import MulticastServer

# Change this address to the address of your local network interface
localInterface = '10.2.3.222'

# Nothing to change below this line
spwMCastAdr = '239.12.255.255'
spwPort = 9522

discoveryRequest  = '534d4100000402a0ffffffff0000002000000000'
discoveryResponse = '534d4100000402a000000001000200000001'

class MulticastClientUDP(DatagramProtocol):

   def startProtocol(self):
      print "Joining speedwire multicast group."
      self.transport.joinGroup(spwMCastAdr)
      self.transport.setOutgoingInterface(localInterface)

      print "Sending discovery request."
      data = discoveryRequest.decode('hex')
      self.transport.write(data, (spwMCastAdr, spwPort))

   def datagramReceived(self, datagram, (srcAddress, port)):
      data = datagram.encode('hex')
      if (data.startswith(discoveryResponse)):
         print "Found device: " + srcAddress

def stopReactor():
    print "Discovery finished."
    reactor.stop()

reactor.listenMulticast(spwPort, MulticastClientUDP(), listenMultiple=True)
reactor.callLater(10, stopReactor)
reactor.run()
```

4 Additional Information

SMA Documents

Links to additional information can be found at www.SMA-Solar.com:

Document title	Document type
SMA Modbus® Interface	Technical Information
SunSpec® Modbus® Interface	Technical Information
SMA Speedwire Fieldbus	Technical Information

Additional Documents

Document title	Source
Service Name and Transport Protocol Port Number Registry	http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xml
Modbus® Application Protocol Specification	http://www.modbus.org/specs.php
Modbus® Messaging Implementation Guide	http://www.modbus.org/specs.php