Example GDNP messages in JSON form

The assumption is that each GDNP message is a single JSON object containing an array of components (i.e. an ordered list of values). In every message, the first value is the Session ID, just as in the TLV version of the protocol. In most cases, the remaining values are all JSON objects, but in a couple of cases they are simple JSON values.

All these examples have been validated at http://jsonlint.com/. The comments have to be stripped out for valid JSON.

```
{"disc": [121212, {"trust_relay": null}]}
                              //Discovery of a standardized objective.
                              //121212 is the Session ID (nonce).
                              //"trust_relay" names the objective.
                              //Its proprietary domain is null.

{"resp": [121212, {"v6a":"fe80000000000000c0daac175f6d8e76"},
                {"trust_relay": [null, "draft-pritikin"]}]}
                              //On-link discovery response contains
                              //the link-local address of trust relay,
                              //and a value for the objective, i.e.
                              //the name of the trust bootstrap method.

{"disc": [12345, {"money": "bank.example.com"}]}
                              //Discovery, sent by initiator, may be
                              //relayed by gateways.
                              //12345 is the Session ID (nonce).
                              //"money" is the objective, proprietary to
                              //bank.example.com.

{"resp": [12345, {"v6a": "fd00beefdeadbeefc0daac175f6d8e76"}]}
                              //Routeable discovery response (ULA example).

{"req": [54321, {"money": ["bank.example.com", {"ct": 5}, 100]}]}
                              //Request negotiation of $100, loop count 5.
                              //54321 is the Session ID (nonce).

{"neg": [54321, {"money": ["bank.example.com", {"ct": 4}, 50]}]}
                              //Continue negotiation.
                              //Peer offers $50.

{"neg": [54321, {"money": ["bank.example.com", {"ct": 3}, 75]}]}
                              //Continue negotiation.
                              //Initiator requests $75.

{"wait": [54321, 120000]}     //Peer sets 2 minute timeout.

{"end": [54321, true]}        //Peer accepts result.
```

Grammar:

A GDNP message is a valid JSON object according to the following grammar.

NOTE: This is ABNF, except
(1) ABNF tokens are in upper case; JSON text is in lower case.
(2) JSON symbols and reserved words have their JSON significance.
(3) In particular, [ ] have their JSON significance (array) and *not* their ABNF significance (optional). We use *1 in ABNF for optional items.

We rely on the fact that JSON arrays are ordered.

```
;Discovery Message

DISCOVERY = {"disc": [SID, 1*OOBJ]}

;Response Message

RESPONSE = {"resp": [SID, 1*LOCATOR / DIVERT / OOBJ]}

;Request Message (starts a new negotiation)

REQUEST = {"req": [SID, OOBJ]}

;Negotiation Message

NEGOTIATION = {"neg": [SID, OOBJ]}

;Negotiation-Ending Message

ENDING = {"end": [SID, true/false]} ; true for accept, false for reject

;Confirm-Waiting message:

WAITING = {"wait": [SID, int]}  ; waiting time in milliseconds

;Session ID

SID = int       ; a nonce that is unique for a given GDNP
                ; session, created by the session initiator,
                ; must be an integer <2^32.

;Divert Option (only allowed in {resp} messages, probably not needed).

DIVERT = {"div": LOCATOR / [ 2*LOCATOR]}


;Locators

LOCATOR  = V4ADDR / V6ADDR / FQDN

; Locators are for machine consumption, so human-readable
```

```
; formats are unnecessary. Using human-readable formats would
; need frequent calls to inet_pton and inet_ntop.


V4ADDR = {"v4a": int}  ; IPv4 address of the target, expressed
                       ; as an integer <2^32. For socket calls,
                       ; this needs to be mapped into uint32_t.

V6ADDR = {"v6a": DQUOTE 32HEXDIG DQUOTE}
                       ; IPv6 address of the target expressed as a
                       ; string of exactly 32 hexadecimal digits. This
                       ; is the best we can do in JSON. For socket calls,
                       ; this needs to be mapped into
                       ; struct in6_addr {unsigned char s6_addr[16];}

FQDN = {"fqdn": string}  ; FQDN of the target


;Device_ID Option (may be no use, placeholder)

DID = {"did": value}

;Generic objective option

OOBJ = {OBJECTIVE_NAME: DOMAIN / [ DOMAIN, FLAGS *1LOOP, *1value] }

; value depends on the specific objective, any JSON value is allowed

OBJECTIVE_NAME = <whatever string is chosen for this objective>

DOMAIN = null / string    ; null if a standardized objective,
                          ; FQDN if a proprietary objective

FLAGS =                   ; flags are used only when disambiguation needed
        *1({"disc_OK": true},) ; objective valid for discovery
        *1({"neg_OK": true},)  ; objective valid for negotiation
        *1({"syn_OK": true},)  ; objective valid for synchronization

LOOP = {"ct": int}        ; max loop count for negotiation
                          ; if absent, use the default GDNP_DEF_LOOPCT
```

CBOR serialization

This would be straightforward except for the `{"v4a": value}` and `{"v6a": value}` constructs.

The value of `v4a` should become a 4-byte unsigned integer (CBOR initial byte 0x1a).

The value of `v6a` should become a 16-byte byte string (CBOR initial byte 0x50).

What happens if we try CBOR Diagnostic Notation instead of JSON?

Not much changes. CBOR DN is slightly extended from JSON, and it allows us to be more rational in representing binary values.

The minimum would be to redefine `V4ADDR` and `V6ADDR` to use the CBOR DN method of expressing byte strings. Then for example we would have messages like

```
{"resp": [12345, {"v6a": h'fd00beefdeadbeefc0daac175f6d8e76'}]}
                              //Routeable discovery response (ULA example).
```

We could go further by redefining the other integer fields as byte strings, which would tie down the binary representation more obviously. So then we'd get

```
{"resp": [h'00003039', {"v6a": h'fd00beefdeadbeefc0daac175f6d8e76'}]}
```

Normally however one doesn't design in CBOR DN; one designs in CDDL.