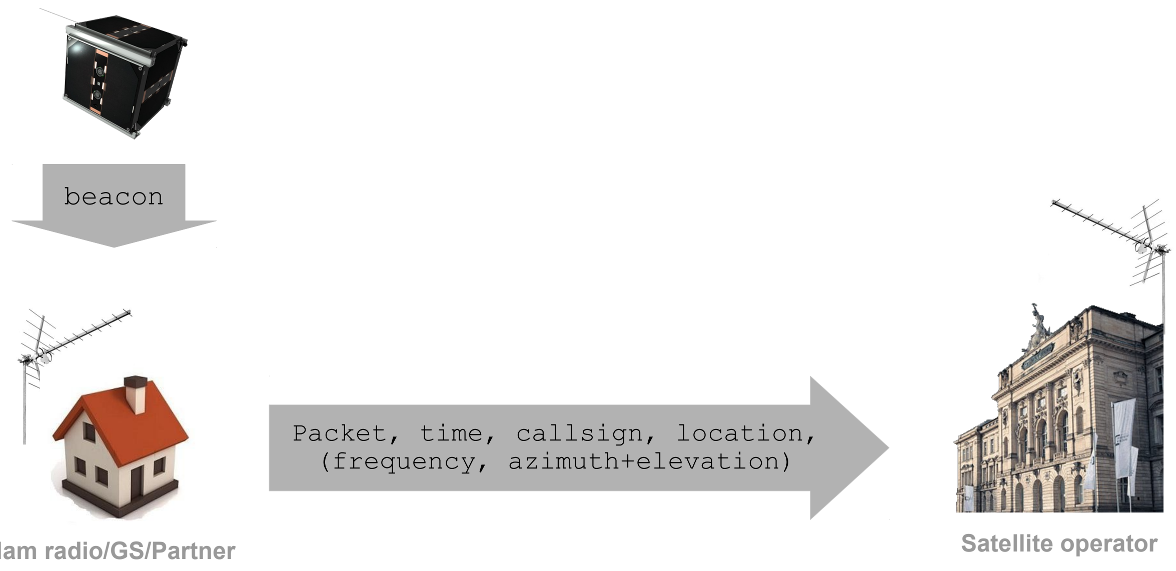
Simple Downlink Share Convention v0.9 (SiDS)

Author	Slavi Dombrovski / University of Würzburg (Germany)
E-mail	dombrovski@informatik.uni-wuerzburg.de

Date	15.01.2015
Version	0.9

1. Introduction

This document describes an interface which has been defined to enable live automatic packet forwarding from any receipt station (ham, ground station, university etc.) to the appropriate satellite operator's server. The technique has been introduced by the University of Würzburg during the operations phase of the UWE-3 satellite. Many radio amateurs worldwide were enabled to automatically forward received satellite packets to the university's operations server. For this purpose the beacon decoding software from Mike Rupprecht (which is widely used by the ham radio community) was extended to comply with this convention.


beacon
Packet, time, callsign, location, (frequency, azimuth+elevation)
Ham radio/GS/Partner
Satellite operator

2. Forwarder (ham radio, partner university)

The use of specific software components for handling the transceiver, interfacing the TNC, pass propagation or antenna movement is ***not*** part of this convention. Instead the ability to successfully receive packets from one or more known satellites is assumed at this point.

2.1. What does the forwarder need to know about the target server?

Basically the forwarder needs to know the Norad-ID, the downlink frequency and the operator's URL. Norad-ID is used to find the appropriate TLE (e.g. celestrak) which in turn is required to calculate satellite passes and Doppler corrections during an over-pass. The downlink frequency is used to set the center frequency (plus Doppler correction) of a transceiver or a software defined radio. If a packet has been received, it can be forwarded as described further down to the given URL. Usually the URL contains the address of a PHP script which is used on the server-side to handle the submitted data.

Example: UWE-3 satellite and the address of the UWE-3 operations server

NoradID	39446
Downlink frequency [Hz]	436395200
URL	http://robotik.informatik.uni-wuerzburg.de/uwe/report_frame.php

2.2. What should be forwarded?

Assuming a satellite transmits a beacon as an AX.25 frame. The receiver should forward the whole frame (not only the AX.25 payload) as is to the server. That is, if additional protocol was applied to the frame after the receive (e.g. KISS, which is normally used as a protocol between the TNC and the Workstation), it should be removed before send. Since the convention is not limited to a specific protocol (such as AX.25), if a TNC is used – it should be set to KISS mode to prevent it from processing the data. Simply speaking, all bytes received by the station should be forwarded as is to the target server.

2.3. Format of the forwarded message

In order to forward a received packet a special URL-link is constructed. During the construction process some additional values are added – whereat some are essential and some optional. The computed link begins with the basic URL (e.g. http://robotik.informatik.uni-wuerzburg.de/uwe/report_frame.php), followed by '**?**' and then parameter/value-Pairs separated by '**&**'. All available parameters and the appropriate value formats are listed in the Table 1.

```
BASIC_URL?PARAM_1=VALUE_1&PARAM_2=VALUE_2&...
```

Field	*Type*	*Required*	*Description*	*Example*
noradID	Integer	**yes**	Norad ID of the spacecraft	39446
source	String	**yes**	e.g. Callsign of the receiver	DK3WN
timestamp	String	**yes**	UTC timestamp (*see ISO 8601*)	2014-05-01T10:21:33.560Z
frame	HEXString	**yes**	Received binary data as hexadecimal string. Whitespaces are *optional*	88 88 60 AA...
locator	String	**yes**	Type of the given receiver's location. Currently only 'longLat' is supported which defines the *WGS84* standard	longLat
longitude	String	**yes**	Longitude of the receiver's location (*WGS84*)	8.95564E
latitude	String	**yes**	Latitude of the receiver's location (*WGS84*)	49.73145N
tncPort	Integer	no	Port at which the packet was received (e.g. the defined Channel in the KISS packet)	0
azimuth	Float	no	Azimuth degree of a directional antenna (if available)	10.5
elevation	Float	no	Elevation degree of a directional antenna (if available)	85.0
fDown	Integer	no	Frequency of the transceiver's downlink channel during the receive (with doppler!)	436399000

Table 1: Required fields for the forwarding

Using the example values from the Table 1, the final URL becomes:

```
http://robotik.informatik.uni-wuerzburg.de/uwe/report_frame.php?
noradID=39446&source=DK3WN&timestamp=2014-05-01T10:21:33.560Z&frame=88 88 60 AA AE 8A 60 88 A0
60 AA AE 8E E1 03 F0 C0 D7 00 00 00 05 40 02 2A
68&locator=longLat&longitude=8.95564E&latitude=49.73145N&tncPort=0&azimuth=10.5&elevation=85.0
&fDown=436399000
```

For testing purposes the final URL can be copied into a web browser. The operator's server will give a positive feedback (Content="OK") if the forwarded data could be accepted or negative feedback if one or more fields are malformed (see Receiver section for more information).

All higher programming languages such as VisualStudio, Java etc. offer the ability of using Web-components. In the simplest case it is a graphical component which can show contents of a given URL. The forwarding can be achieved by using such a component (off-screen) by "opening" the generated final URL to submit the received data.

Hints:

- The required WGS84 coordinates of the own location can be derived under http://gpso.de/maps/
- The appropriate TLE elements can be found under http://celestrak.com/NORAD/elements/, e.g. http://celestrak.com/NORAD/elements/cubesat.txt
- Float numbers should use '.' as a decimal separator

3. Receiver (operator, university)

In order to enable the reception of forwarded packets a web-server must be established and made accessible by the sender. As described in the *Forwarder* section, the data set is transferred completely within the URL (URL parameter) – thus requiring only simple data processing. The server component itself (Apache/Java etc.) as well as the processing language (PHP/Java) can be chosen freely. In case of University of Würzburg *Apache+PHP+MySQL* combination is used to receive, handle and store external packets.

The receiver needs to reply with an appropriate answer packet. That is, an acknowledgment if the submitted data fulfills the convention or a negative answer if one or more fields are malformed. The packet description can be found in the Table 2.

HTTP code	***Plain Content***	***Description***
200	OK	Submitted data fulfills the convention
400	Error: ...	One or more fields are malformed, empty or non-existent. The error text should be human-readable

Table 2: Web server answer

The example implementation of a PHP script and an optional SQL table definition can be found in the appendix.

4. Appendix

Example implementation of the operators web server (MySQL table definition + PHP script).

```sql
create table ExternalMessages (
    ID           INTEGER not null primary key AUTO_INCREMENT,
    RecvTime     TIMESTAMP,              -- receive time
    CreatedByIP  VARCHAR(50),            -- receivers IP
    CreatedBy    VARCHAR(50),            -- callsign of the receiver
    Langitude    VARCHAR(20),            -- location of the receiver
    Latitude     VARCHAR(20),            -- ...
    Frame        VARCHAR(250) CHARACTER SET binary, -- Packet (e.g. AX.25)
    TNCPort      INTEGER,

    FUp          INTEGER,                -- uplink radio frequency
    FDown        INTEGER,                -- downlink radio frequency
    Azimuth      FLOAT,                  -- Antenna orientation during the receive
    Elevation    FLOAT                   -- ...
    -- more optional fields can be added here
);
```

Listing 1: SQL Table definition (with optional AX.25

```php
<?php
// set error reporting and locale
error_reporting(E_ALL - E_WARNING - E_NOTICE);

// ###################################################
// ####### TODO: Change to your own locale ##########
// ###################################################
setlocale(LC_ALL, 'de_DE@euro', 'de_DE', 'deu_deu');

// read script parameters by first trying $_POST, then $_GET
$source    = read_env_string('source', '');
$timestamp = read_env_string('timestamp', '');
$frame     = str_replace    (' ', '', read_env_string('frame', ''));
$locator   = read_env_string('locator', '');
$longitude = read_env_string('longitude', '');
$latitude  = read_env_string('latitude', '');
$tnc_port  = read_env_int   ('tncPort', 0);

// initializations for mysqli instance and error message
$mysqli = NULL;
$error_msg = NULL;

// sanity checks on input data
do {
        if (strlen($source)    ==  0) { $error_msg = 'Field source not set!'; break; }
        if (strlen($source)    >  50) { $error_msg = 'Field source is invalid!'; break; }
        if (strlen($timestamp) ==  0) { $error_msg = 'Field timestamp not set!'; break; }
        if (! preg_match('/^[0-9]{4}-[0-9]{2}-[0-9]{2}T[0-9]{2}:[0-9]{2}:[0-9]{2}\.[0-9]{3}Z$/', $timestamp)) {
$error_msg = 'Field timestamp is invalid!'; break; }
        if (strlen($frame)     ==  0) { $error_msg = 'Field frame not set!'; break; }
        if (strlen($frame)     > 500) { $error_msg = 'Field frame is invalid!'; break; }
        if (preg_match('/[^0-9a-fA-F]/', $frame)) { $error_msg = 'Field frame is invalid!'; break; }
        if (strlen($locator)   ==  0) { $error_msg = 'Field locator not set!'; break; }
        if ($locator != 'longLat') { $error_msg = 'Field locator is invalid!'; break; }
        if (strlen($longitude) ==  0) { $error_msg = 'Field longitude not set!'; break; }
        if (strlen($longitude) >  20) { $error_msg = 'Field longitude is invalid!'; break; }
        if (! preg_match('/^(\+|-)?[0-9]{1,3}\.[0-9]{1,10}([eE]|[wW])$/', $longitude)) { $error_msg = 'Field longitude is
invalid!'; break; }
        if (strlen($latitude)  ==  0) { $error_msg = 'Field latitude not set!'; break; }
        if (strlen($latitude)  >  20) { $error_msg = 'Field latitude is invalid!'; break; }
        if (! preg_match('/^(\+|-)?[0-9]{1,3}\.[0-9]{1,10}([nN]|[sS])$/', $latitude)) { $error_msg = 'Field latitude is
invalid!'; break; }
} while (false);

// show error and exit if an error ocurred
if ($error_msg)
        error_exit($mysqli, $error_msg);

// create mysqli object and connect to database
// ###################################################
// ####### TODO: please enter own credentials ########
// ###################################################
$mysqli = new mysqli('192.168.XXX.YYY', 'USER', 'PASS', 'DB_NAME', 3306);
if ($mysqli->connect_error)
        error_exit($mysqli, "unable to connect to database");

// format timestamp to be compatible with MySQL TIMESTAMP
$ts = str_replace(array('T', 'Z'), array(' ', ''), $timestamp);

// convert supplied frame from hex to binary representation
$frame_bin = hex2bin($frame);

// prepare insert query
$stmt = $mysqli->prepare("INSERT INTO ExternalMessages (RecvTime, CreatedByIP, CreatedBy, Longitude, Latitude, Frame,
TNCPort) VALUES (?, ?, ?, ?, ?, ?, ?)");
if ($stmt === false)
        error_exit($mysqli, "unable to prepare insert statement");

do {
        // bind parameters to insert query
        if (! $stmt->bind_param('sssssi', $ts, $_SERVER['REMOTE_ADDR'], $source, $longitude, $latitude, $frame_bin,
$tnc_port)) { $error_msg = 'unable to bind params'; break; }
        // execute insert query
        if (! $stmt->execute()) { $error_msg = 'executing insert query failed'; break; }
        // check if only one row was affected
        if ($stmt->affected_rows != 1) { $error_msg = 'wrong number of rows affected'; break; }
} while (false);
$stmt->close();

// show error and exit if an error ocurred
if ($error_msg)
        error_exit($mysqli, $error_msg);

// we're fine, so output OK
echo "OK";

// close database connection
$mysqli->close();


// show error and exit, sends "Bad Request" header using HTTP status code 400
function error_exit($mysqli, $error_msg) {
        // send Bad Request (400) header
        header("Bad Request", true, 400);
        // close database connection if it was successfully opened
        if (is_object($mysqli)) {
                if (is_null($mysqli->connect_error))
                        $mysqli->close();
        }
        // output error message
        echo "Error: $error_msg";
        exit();
}


// read integer from environment, first trying $_POST, then $_GET
function read_env_int($var, $default) {
  $result = $default;
  if (isset($_POST[$var]))
    $result = intval($_POST[$var]);
  else
    if (isset($_GET[$var]))
      $result = intval($_GET[$var]);
  return $result;
}

// read decoded string from environment, first trying $_POST, then $_GET
function read_env_string($var, $default) {
  $result = $default;
  if (isset($_POST[$var]))
    $result = rawurldecode($_POST[$var]);
  else
    if (isset($_GET[$var]))
      $result = rawurldecode($_GET[$var]);
  return $result;
}
?>
```

Acknowledgments

I would like to thank Mike Rupprecht (DK3WN) for his cooperation and for implementing the interface in his beacon decoder software – which is widely used by radio amateurs. And many thanks to all the radio amateurs who have helped – and still help – us to receive satellite packets from all over the world:

Callsign	***#packets***	***Callsign***	***#packets***	***Callsign***	***#packets***
DK3WN	30111	CU2JX	3707	YC3BVG	100
Rainer	29132	R4UAB	3346	CX8AF	88
SP7THR	17796	__	2726	IK8OZV	88
PE0SAT	12609	VK5HI	2032	PA2EON	88
JA1GDE	8386	JA5BLZ	1806	EA1JM	49
EU1XX	7825	IW0HLG	1732	PY2SDR	41
LU4EOU	6314	JO1PTD	1307	PY4ZBZ	41
JA6PL	6048	JA0CAW	1111	JE1CVL	35
ON4HF	5060	JE9PEL	306	JF1EUY	26
G7GQW	4961	DG4YDF	193		

Currently known users (2015)

Callsign/Institute	***Forward***	***Receive***
DK3WN, Rainer, SP7THR, PE0SAT, JA1GDE, EU1XX, LU4EOU, JA6PL, ON4HF, G7GQW, CU2JX, R4UAB, VK5HI, JA5BLZ, IW0HLG, JO1PTD, JA0CAW, JE9PEL, DG4YDF, YC3BVG, CX8AF, IK8OZV, PA2EON, EA1JM, PY2SDR, PY4ZBZ, JE1CVL, JF1EUY	x	
University of Würzburg, UWE-3/UWE-4 team	x	x
G.A.U.S.S. Srl, UniSat-6		x