HTTP/HTTPS Upload API Documentation

Overview

This document describes the HTTP/HTTPS upload interface that third-party servers need to implement to receive file uploads from TermiTec C1P terminals. The API supports two different upload formats: JSON and Multipart Form Data.

Version: 1.0
Protocol: HTTP/HTTPS
Methods: POST
Content Types: `application/json` or `multipart/form-data`

Authentication

The API supports two authentication methods:

1. HTTP Basic Authentication

Header:

```
Authorization: Basic {base64(username:password)}
```

Example:

```
Authorization: Basic dXNlcm5hbWU6cGFzc3dvcmQ=
```

2. Bearer Token (API Key)

Header:

```
Authorization: Bearer {api_key}
```

Example:

```
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

Upload Formats

Format 1: JSON Upload

Endpoint: `POST {your-server-url}`

Content-Type: `application/json`

Request Body

```
{
  "device_id": "string",
  "device_name": "string",
  "serial_number": "string",
  "timestamp": "string (RFC3339)",
  "files": [
    {
      "name": "string",
      "size": 1234,
      "path": "string",
      "modified": "string (RFC3339)",
      "content": "string (file content)"
    }
  ],
  "nfc_files": [
    {
      "name": "string",
      "size": 1234,
      "path": "string",
      "modified": "string (RFC3339)",
      "content": "string (file content)"
    }
  ]
}
```

Field Descriptions

Field	Type	Description
`device_id`	string	Typ identifier of the terminal device
`device_name`	string	Network name of the terminal device

Field	Type	Description
serial_number	string	Serialnumer of the terminal device
timestamp	string	Upload timestamp in RFC3339 format (e.g., "2025-01-15T14:30:00Z")
files	array	Array of tachograph files
nfc_files	array	Array of NFC card data files
files[].name	string	Original filename
files[].size	integer	File size in bytes
files[].path	string	Relative path of the file
files[].modified	string	File modification timestamp (RFC3339)
files[].content	string	File content as plain text or base64 encoded

Example Request

```
POST /api/upload HTTP/1.1
Host: your-server.com
Content-Type: application/json
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...

{
  "device_id": "TermiTec C1P Rev 1",
  "device_name": "termitec",
  "serial_number": "A1234567BC",
  "timestamp": "2025-01-15T14:30:00Z",
  "files": [
    {
      "name": "V123456.ddd",
      "size": 12345,
      "path": "V123456.tgd",
      "modified": "2025-01-15T14:25:00Z",
      "content": "..."
    }
  ],
  "nfc_files": [
    {
      "name": "card_789.nfc",
      "size": 4567,
      "path": "card_789.nfc",
```

```
      "modified": "2025-01-15T14:28:00Z",
      "content": "..."
    }
  ]
}
```

Format 2: Multipart Form Upload

Endpoint: `POST {your-server-url}`

Content-Type: `multipart/form-data`

Form Fields

Field Name	Type	Description
`device_id`	string	Typ identifier of the terminal device
`device_name`	string	Network name of the terminal device
`serial_number`	string	Serialnumer of the terminal device
`timestamp`	string	Upload timestamp in RFC3339 format
`files`	file(s)	One or more tachograph files
`nfc_files`	file(s)	One or more NFC card data files

Example Request

```
POST /api/upload HTTP/1.1
Host: your-server.com
Content-Type: multipart/form-data; boundary=----WebKitFormBoundary7MA4YWx
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...

------WebKitFormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="device_id"

TermiTec C1P Rev 1
------WebKitFormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="device_name"

termitec
------WebKitFormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="serial_number"
```

```
A1234567BC
------WebKitFormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="timestamp"

2025-01-15T14:30:00Z
------WebKitFormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="files"; filename="V123456.ddd"
Content-Type: application/octet-stream

[Binary file content]
------WebKitFormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="nfc_files"; filename="card_789.nfc"
Content-Type: application/octet-stream

[Binary file content]
------WebKitFormBoundary7MA4YWxkTrZu0gW--
```

Response Format

The server should respond with a JSON object containing upload status information.

Success Response

HTTP Status: `200 OK` or `201 Created` or `202 Accepted`

```
{
  "success": true,
  "message": "Upload successful",
  "files_uploaded": 5
}
```

Error Response

HTTP Status: `400 Bad Request` or `401 Unauthorized` or `500 Internal Server Error`

```
{
  "success": false,
  "message": "Error description",
  "files_uploaded": 0,
```

```
  "errors": [
    "Detailed error message 1",
    "Detailed error message 2"
  ]
}
```

Response Fields

Field	Type	Required	Description
`success`	boolean	Yes	Indicates whether the upload was successful
`message`	string	Yes	Human-readable status message
`files_uploaded`	integer	No	Number of successfully uploaded files
`errors`	array	No	Array of error messages (only if errors occurred)

HTTP Status Codes

Status Code	Meaning	Description
`200 OK`	Success	Upload completed successfully
`201 Created`	Success	Files created successfully
`202 Accepted`	Success	Upload accepted for processing
`400 Bad Request`	Error	Invalid request format or parameters
`401 Unauthorized`	Error	Authentication failed
`403 Forbidden`	Error	Access denied
`413 Payload Too Large`	Error	File size exceeds limit
`500 Internal Server Error`	Error	Server-side error
`503 Service Unavailable`	Error	Server temporarily unavailable

Client Behavior

Retry Logic

The terminal will automatically retry failed uploads with the following parameters:

- **Retry Attempts:** 3 (default, configurable)
- **Retry Delay:** 5 seconds (default, configurable)
- **Retry Condition:** Any HTTP status code >= 400 or network error

Upload Interval

- **Default:** 30 seconds
- **Configurable:** 10 seconds to 300 seconds
- Files are scanned and uploaded automatically at the configured interval

TLS Configuration

- **TLS Verification:** Enabled by default
- **Configurable:** Can be disabled for self-signed certificates (not recommended)

Timeout

- **Default:** 30 seconds
- **Configurable:** 10 seconds to 300 seconds

Proxy Support

The client supports HTTP/HTTPS proxy with optional authentication:

- **Proxy URL:** Format `http://proxy.example.com:8080`
- **Proxy Authentication:** Basic authentication supported

File Types

Supported file formats:

Extensions	Typ	Description
`.tgd`, `.TGD`	Tachograph	Tachograph Digital Files
`.ddd`, `.DDD`	Tachograph	Driver Download Data
`.v1b`, `.V1B`	Tachograph	VU Files (Generation 1)
`.c1b`, `.C1B`	Tachograph	Card Files (Generation 1)
`.v1a`, `.V1A`	Tachograph	VU Files (Generation 1 Alternative)

Extensions	Typ	Description
`.c1a`, `.C1A`	Tachograph	Card Files (Generation 1 Alternative)
`.v2b`, `.V2B`	Tachograph	VU Files (Generation 2)
`.c2b`, `.C2B`	Tachograph	Card Files (Generation 2)
`.v2a`, `.V2A`	Tachograph	VU Files (Generation 2 Alternative)
`.c2a`, `.C2A`	Tachograph	Card Files (Generation 2 Alternative)
`.esv`, `.ESV`	Tachograph	ESV Files
`.dsr`, `.DSR`	Tachograph	DSR Files
`.ntf`, `.NTF`	NFC	NFC Card Data
`.txt`, `.TXT`	Textfile	Test Files

Tachograph Files

- **Extension:** `.tgd`, `.ddd`, `.v1b`, `.c1b`, `.esv`, etc.
- **Content:** Binary or text format depending on file type

NTF/NFC Card Files

- **Extension:** `.ntf`
- **Content:** Binary or text format

```
{
   "nfcType": "ISO14443A/Mifare Ultralight",
   "nfcDate": "2025-10-14T12:40:30.798629",
   "nfcId": "04c4120a1d7280",
   "nfcData": "deFK0E80721D0A12C4049F0BB922",
   "nfcParity": 17,
   "nfcLock": "Argon2",
   "nfcAuth": "zI1NiIsIn"
}
```

Field	Type	Required	Description
`nfcType`	string	Yes	Type of tag read
`nfcDate`	string	Yes	Original timestamp of the file

Field	Type	Required	Description
`nfcId`	string	Yes	Tag-ID
`nfcData`	string	No	Tag content (Payload)
`nfcParity`	int	No	Checksum over the content of the tag
`nfcLock`	string	No	Authentication method
`nfcAuth`	string	No	Authkey of the tag

Security Considerations

1. Authentication

- Always use HTTPS in production environments
- Implement strong authentication (Bearer tokens recommended)
- Rotate API keys regularly
- Never expose credentials in logs or error messages

2. Data Validation

- Validate file names (prevent path traversal attacks)
- Check file sizes (prevent DoS attacks)
- Validate file content types
- Sanitize all input data

3. Rate Limiting

- Implement rate limiting per device or IP address
- Recommended: 120 requests per hour per device

4. TLS/SSL

- Use TLS 1.2 or higher
- Use valid SSL certificates (Let's Encrypt, etc.)
- Disable insecure cipher suites

Implementation Examples

Example 1: Node.js/Express Server (JSON Format)

```
const express = require('express');
const app = express();

app.use(express.json({ limit: '50mb' }));

// Authentication middleware
app.use((req, res, next) => {
  const authHeader = req.headers.authorization;

  if (!authHeader) {
    return res.status(401).json({
      success: false,
      message: 'Missing authorization header'
    });
  }

  // Verify Bearer token or Basic auth
  if (authHeader.startsWith('Bearer ')) {
    const token = authHeader.substring(7);
    if (!isValidToken(token)) {
      return res.status(401).json({
        success: false,
        message: 'Invalid token'
      });
    }
  }

  next();
});

// Upload endpoint
app.post('/api/upload', (req, res) => {
  try {
    const { device_id, timestamp, files, nfc_files } = req.body;

    // Validate request
    if (!device_id || !timestamp) {
      return res.status(400).json({
        success: false,
        message: 'Missing required fields'
      });
```

```
    }

    // Process files
    let filesProcessed = 0;

    if (files && Array.isArray(files)) {
      files.forEach(file => {
        // Save file to storage
        saveFile('files', file);
        filesProcessed++;
      });
    }

    if (nfc_files && Array.isArray(nfc_files)) {
      nfc_files.forEach(file => {
        // Save file to storage
        saveFile('nfc', file);
        filesProcessed++;
      });
    }

    // Return success
    res.status(200).json({
      success: true,
      message: 'Upload successful',
      files_uploaded: filesProcessed
    });

  } catch (error) {
    res.status(500).json({
      success: false,
      message: 'Server error: ' + error.message
    });
  }
});

function isValidToken(token) {
  // Implement your token validation logic
  return token === 'your-secret-token';
}

function saveFile(category, file) {
  // Implement your file storage logic
  const fs = require('fs');
```

```
  const path = require('path');

  const dir = path.join(__dirname, 'uploads', category);
  if (!fs.existsSync(dir)) {
    fs.mkdirSync(dir, { recursive: true });
  }

  const filePath = path.join(dir, file.name);
  fs.writeFileSync(filePath, file.content);
}

app.listen(443, () => {
  console.log('Upload server listening on port 443');
});
```

Example 2: Python/Flask Server (Multipart Format)

```
from flask import Flask, request, jsonify
import os
from werkzeug.utils import secure_filename

app = Flask(__name__)
app.config['MAX_CONTENT_LENGTH'] = 50 * 1024 * 1024  # 50 MB limit

UPLOAD_FOLDER_FILES = '/var/data/uploads/files'
UPLOAD_FOLDER_NFC = '/var/data/uploads/nfc'

# Create upload directories
os.makedirs(UPLOAD_FOLDER_FILES, exist_ok=True)
os.makedirs(UPLOAD_FOLDER_NFC, exist_ok=True)

def check_auth(auth_header):
    """Verify authentication"""
    if not auth_header:
        return False

    if auth_header.startswith('Bearer '):
        token = auth_header[7:]
        return token == 'your-secret-token'

    return False
```

```python
@app.route('/api/upload', methods=['POST'])
def upload_files():
    # Check authentication
    auth_header = request.headers.get('Authorization')
    if not check_auth(auth_header):
        return jsonify({
            'success': False,
            'message': 'Unauthorized'
        }), 401

    try:
        # Get form fields
        device_id = request.form.get('device_id')
        timestamp = request.form.get('timestamp')

        if not device_id or not timestamp:
            return jsonify({
                'success': False,
                'message': 'Missing required fields'
            }), 400

        files_uploaded = 0

        # Process tachograph files
        if 'files' in request.files:
            files = request.files.getlist('files')
            for file in files:
                if file.filename:
                    filename = secure_filename(file.filename)
                    filepath = os.path.join(UPLOAD_FOLDER_FILES, filename
                    file.save(filepath)
                    files_uploaded += 1

        # Process NFC files
        if 'nfc_files' in request.files:
            nfc_files = request.files.getlist('nfc_files')
            for file in nfc_files:
                if file.filename:
                    filename = secure_filename(file.filename)
                    filepath = os.path.join(UPLOAD_FOLDER_NFC, filename)
                    file.save(filepath)
                    files_uploaded += 1

        return jsonify({
```

```
            'success': True,
            'message': 'Upload successful',
            'files_uploaded': files_uploaded
        }), 200

    except Exception as e:
        return jsonify({
            'success': False,
            'message': f'Server error: {str(e)}'
        }), 500

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=443, ssl_context='adhoc')
```

Example 3: PHP Server (Both Formats)

```
<?php
header('Content-Type: application/json');

// Authentication
$authHeader = $_SERVER['HTTP_AUTHORIZATION'] ?? '';

if (empty($authHeader)) {
    http_response_code(401);
    echo json_encode([
        'success' => false,
        'message' => 'Missing authorization header'
    ]);
    exit;
}

if (strpos($authHeader, 'Bearer ') === 0) {
    $token = substr($authHeader, 7);
    if ($token !== 'your-secret-token') {
        http_response_code(401);
        echo json_encode([
            'success' => false,
            'message' => 'Invalid token'
        ]);
        exit;
    }
}
```

```
// Check content type
$contentType = $_SERVER['CONTENT_TYPE'] ?? '';

try {
    $filesUploaded = 0;

    if (strpos($contentType, 'application/json') !== false) {
        // JSON format
        $input = file_get_contents('php://input');
        $data = json_decode($input, true);

        if (!isset($data['device_id']) || !isset($data['timestamp'])) {
            http_response_code(400);
            echo json_encode([
                'success' => false,
                'message' => 'Missing required fields'
            ]);
            exit;
        }

        // Process files
        if (isset($data['files']) && is_array($data['files'])) {
            foreach ($data['files'] as $file) {
                $filename = basename($file['name']);
                $filepath = '/var/data/uploads/files/' . $filename;
                file_put_contents($filepath, $file['content']);
                $filesUploaded++;
            }
        }

        if (isset($data['nfc_files']) && is_array($data['nfc_files'])) {
            foreach ($data['nfc_files'] as $file) {
                $filename = basename($file['name']);
                $filepath = '/var/data/uploads/nfc/' . $filename;
                file_put_contents($filepath, $file['content']);
                $filesUploaded++;
            }
        }

    } elseif (strpos($contentType, 'multipart/form-data') !== false) {
        // Multipart format
        $deviceId = $_POST['device_id'] ?? null;
        $timestamp = $_POST['timestamp'] ?? null;
```

```
    if (empty($deviceId) || empty($timestamp)) {
        http_response_code(400);
        echo json_encode([
            'success' => false,
            'message' => 'Missing required fields'
        ]);
        exit;
    }

    // Process uploaded files
    if (isset($_FILES['files'])) {
        $files = $_FILES['files'];
        if (is_array($files['name'])) {
            for ($i = 0; $i < count($files['name']); $i++) {
                $filename = basename($files['name'][$i]);
                $tmpPath = $files['tmp_name'][$i];
                $destination = '/var/data/uploads/files/' . $filename
                move_uploaded_file($tmpPath, $destination);
                $filesUploaded++;
            }
        }
    }

    if (isset($_FILES['nfc_files'])) {
        $files = $_FILES['nfc_files'];
        if (is_array($files['name'])) {
            for ($i = 0; $i < count($files['name']); $i++) {
                $filename = basename($files['name'][$i]);
                $tmpPath = $files['tmp_name'][$i];
                $destination = '/var/data/uploads/nfc/' . $filename;
                move_uploaded_file($tmpPath, $destination);
                $filesUploaded++;
            }
        }
    }
}

http_response_code(200);
echo json_encode([
    'success' => true,
    'message' => 'Upload successful',
    'files_uploaded' => $filesUploaded
]);
```

```
} catch (Exception $e) {
    http_response_code(500);
    echo json_encode([
        'success' => false,
        'message' => 'Server error: ' . $e->getMessage()
    ]);
}
?>
```

Changelog

Version	Date	Changes
1.0	2025-10-21	Initial API documentation
1.1	2025-11-29	HTTP Parameters expanded

End of Documentation