Parkzone API Documentation

Version: 1.3.4
Last Updated: February 2026
Base URL: `https://{tenant}.parkzone.app/api.php`

Introduction

The Parkzone API enables external services and applications to integrate parking management functionality into their systems. This RESTful API provides comprehensive endpoints for managing parking reservations, ad-hoc bookings, user administration, and tenant configuration.

Key Features

- Multi-tenant architecture with subdomain-based tenant identification
- Secure authentication via hash-based or token-based mechanisms
- Real-time parking spot availability management
- User and reservation management
- Configuration customization per tenant
- Complete audit trail via history logging

Target Audience

This documentation is designed for developers integrating third-party applications with the Parkzone platform, including property management systems, mobile applications, and enterprise resource planning tools.

Architecture Overview

Multi-Tenant Model

The API operates on a multi-tenant architecture where each tenant is identified by their subdomain:

https://company-a.parkzone.app/api.php
https://company-b.parkzone.app/api.php

Each tenant maintains isolated data including:

- Configuration settings
- User accounts
- Parking reservations
- Ad-hoc bookings
- History logs

Request Format

All API requests use **POST** method with JSON payloads. The action parameter can be provided either as a query parameter or within the JSON body.

Example Request:
POST /api.php?action=sync
Content-Type: application/json

Or:
POST /api.php
Content-Type: application/json

{
"action": "sync"
}

Response Format

All responses are returned in JSON format with UTF-8 encoding.

Success Response:
{

```
"success": true,
"data": { ... }
}
```

Error Response:

```
{
"success": false,
"error": "Error message description"
}
```

Authentication

The API supports multiple authentication mechanisms depending on the endpoint and use case.

Authentication Methods

1. **Admin Hash Authentication** - Full administrative access using SHA-256 hash
2. **Support Hash Authentication** - Support-level access using SHA-256 hash
3. **Support Token Authentication** - Time-limited token-based support access
4. **User Code Authentication** - User-specific apartment code for reservations

Admin Authentication

Admin authentication provides full access to all management endpoints.

Required Parameter:

```
{
"isAdmin": true,
"hash": "admin_sha256_hash"
}
```

Support Authentication

Support authentication provides limited administrative access.

Hash-based (Legacy):
```
{
"isSupport": true,
"hash": "support_sha256_hash"
}
```

Token-based (Recommended):
```
{
"isSupport": true,
"supportToken": "time_limited_token"
}
```

Security Considerations

- All authentication hashes are stored as SHA-256 values
- Hash comparison uses timing-safe `hash_equals()` function
- Support tokens have expiration mechanisms
- Failed authentication returns HTTP 403 Forbidden
- Authentication credentials should never be transmitted in plain text

API Endpoints

Sync - Retrieve Complete Tenant State

Retrieves the complete current state of a tenant including configuration, users, reservations, and ad-hoc bookings.

Endpoint: `action=sync`

Authentication: None (public read access)

Request:
```
{
"action": "sync"
}
```

Response:

```
{
"config": {
"headingTitle": "Company Parking",
"numPlaces": 10,
"guestTime": 90,
"authMode": false,
"maxResDays": 14,
"resGapMin": 0,
"pronoun": "du"
},
"adhoc": {
"1": {
"k": "AB-CD-1234",
"ts": 1708012800
}
},
"users": [
{
"code": "user_password",
"apt": "101",
"comment": "Main entrance"
}
],
"res": {
"2026-02-15": {
"2": {
"id": 123,
"n": "John Doe",
"k": "XY-ZZ-5678"
}
}
}
}
```

Response Fields:

Field	Type	Description
config	Object	Tenant configuration settings
adhoc	Object	Current ad-hoc bookings (key = parking spot number)
users	Array	List of registered users with apartment codes
res	Object	Reservations organized by date (YYYY-MM-DD)

Table 1: Sync endpoint response structure

Login Admin - Administrative Login

Validates administrative credentials.

Endpoint: `action=loginadmin`

Authentication: Admin hash required

Request:

```
{
"action": "loginadmin",
"hash": "admin_sha256_hash"
}
```

Response:

```
{
"success": true
}
```

Login Support (Token-based) - Support Access

Validates time-limited support token for support operations.

Endpoint: `action=loginsupporttoken`

Authentication: Support token required

Request:

```
{
"action": "loginsupporttoken",
```

"supportToken": "generated_time_limited_token"
}

Response:
{
"success": true
}

Save Configuration - Update Tenant Settings

Updates tenant-specific configuration parameters.

Endpoint: `action=saveconfig`

Authentication: Admin, Support (hash or token)

Request:
{
"action": "saveconfig",
"isAdmin": true,
"hash": "admin_hash",
"config": {
"headingTitle": "New Parking Area",
"numPlaces": 15,
"guestTime": 120,
"authMode": true,
"maxResDays": 30,
"resGapMin": 1,
"pronoun": "sie"
}
}

Configuration Parameters:

Parameter	Type	Description
headingTitle	String	Display title for parking area
numPlaces	Integer	Total number of parking spots
guestTime	Integer	Guest parking duration in minutes
authMode	Boolean	Require authentication for reservations
maxResDays	Integer	Maximum reservation duration in days
resGapMin	Integer	Minimum gap between reservations (days)
pronoun	String	UI formality level ("du" or "sie")

Table 2: Configuration parameters

Response:
```
{
"success": true
}
```

Change Password - Update Authentication Credentials

Updates admin or support password hashes.

Endpoint: `action=changepw`

Authentication: Admin or Support (hash or token)

Request:
```
{
"action": "changepw",
"isAdmin": true,
"hash": "current_admin_hash",
"newAdminHash": "new_sha256_hash",
"newSupportHash": "new_sha256_hash"
}
```

Response:
```
{
```

"success": true
}

Save User - Create New User Account

Creates a new user account with apartment association.

Endpoint: `action=saveuser`

Authentication: Admin or Support (hash or token)

Request:
{
"action": "saveuser",
"isAdmin": true,
"hash": "admin_hash",
"apt": "102",
"code": "user_password",
"comment": "Second floor, west wing"
}

Response:
{
"success": true
}

Error Cases:

- Apartment number already exists
- Missing required fields (apt, code)

Delete User - Remove User Account

Deletes a user account by authentication code.

Endpoint: `action=deleteuser`

Authentication: Admin or Support (hash or token)

Request:
{
"action": "deleteuser",
"isAdmin": true,

"hash": "admin_hash",
"code": "user_password"
}

Response:
{
"success": true
}

Save Reservation - Create Parking Reservation

Creates a new parking reservation for a specific time period.

Endpoint: `action=savereservation`

Authentication: Optional (user code if authMode enabled)

Request:
{
"action": "savereservation",
"platz": 5,
"vonDT": "2026-02-20T08:00",
"bisDT": "2026-02-22T18:00",
"name": "John Doe",
"kennzeichen": "AB-CD-1234",
"authId": "user_password"
}

Parameters:

Parameter	Type	Description
platz	Integer	Parking spot number (1-indexed)
vonDT	String	Start date-time (ISO 8601 format)
bisDT	String	End date-time (ISO 8601 format)
name	String	Name of person making reservation
kennzeichen	String	Vehicle license plate number
authId	String	User code (required if authMode is true)

Table 3: Reservation parameters

Response:
```
{
"success": true,
"code": "4567"
}
```

The returned code is a 4-digit cancellation code for the reservation.

Validation Rules:

- Reservation must be in the future
- Duration cannot exceed maxResDays configuration
- Spot must not be occupied by ad-hoc booking
- No overlapping reservations allowed
- Respects resGapMin buffer period between reservations

Error Cases:

- Invalid date range
- Past date reservation attempt
- Exceeds maximum duration
- Spot already reserved
- Ad-hoc booking conflict
- Invalid authentication code (when authMode enabled)
- Insufficient gap between reservations

Cancel Booking - Delete Reservation

Cancels an existing parking reservation.

Endpoint: `action=cancelbooking`

Authentication: Admin/Support OR reservation cancellation code

Admin/Support Request:
```
{
"action": "cancelbooking",
"isAdmin": true,
"hash": "admin_hash",
"id": 123
}
```

User Request (with code):
```
{
"action": "cancelbooking",
"id": 123,
"code": "4567"
}
```

The `code` parameter accepts either:

- The 4-digit cancellation code returned during reservation creation
- The apartment authentication code used to create the reservation

Response:
```
{
"success": true
}
```

Save Ad-Hoc Booking - Create Immediate Parking

Creates an immediate, short-term parking spot occupation (typically for guests).

Endpoint: `action=saveadhoc`

Authentication: None (public access)

Request:
```
{
"action": "saveadhoc",
"platz": 3,
"kennzeichen": "XY-ZZ-9876"
}
```

Response:
```
{
"success": true,
"code": "7890"
}
```

The returned `code` is a 4-digit code to release the ad-hoc booking.

Validation Rules:

- Spot must not already have an ad-hoc booking
- Spot must not have a reservation for current date
- Ad-hoc bookings expire automatically based on `guestTime` configuration

Error Cases:

- Spot already occupied by ad-hoc booking
- Spot has reservation for today

Free Ad-Hoc Booking - Release Parking Spot

Releases an ad-hoc parking spot occupation.

Endpoint: `action=freeadhoc`

Authentication: Admin/Support OR ad-hoc release code

Admin/Support Request (by spot):
```
{
"action": "freeadhoc",
"isAdmin": true,
"hash": "admin_hash",
"platz": 3
}
```

Admin/Support Request (by code):

```
{
"action": "freeadhoc",
"isAdmin": true,
"hash": "admin_hash",
"code": "7890"
}
```

User Request:

```
{
"action": "freeadhoc",
"code": "7890"
}
```

Response:

```
{
"success": true
}
```

Data Models

Configuration Object

Field	**Type**	**Default**	**Description**
headingTitle	String	"Parkzone"	Display title
numPlaces	Integer	10	Number of parking spots
guestTime	Integer	90	Guest duration (minutes)
authMode	Boolean	false	Authentication required
headingIcon	String	""	Icon identifier
maxResDays	Integer	14	Max reservation days
resGapMin	Integer	0	Gap between reservations
pronoun	String	"du"	UI formality level

Table 4: Configuration object fields

User Object

Field	**Type**	**Description**
code	String	User authentication password
apt	String	Apartment or unit identifier
comment	String	Additional user information

Table 5: User object fields

Reservation Object

Field	**Type**	**Description**
id	Integer	Unique reservation identifier
n	String	Name of reservation holder
k	String	Vehicle license plate

Table 6: Reservation object fields (in sync response)

Ad-Hoc Booking Object

Field	Type	Description
k	String	Vehicle license plate
ts	Integer	Unix timestamp of booking creation

Table 7: Ad-hoc booking object fields

Error Handling

Error Response Structure

All API errors return a consistent JSON structure:

```
{
"success": false,
"error": "Human-readable error message"
}
```

HTTP Status Codes

Code	Status	Description
200	OK	Request successful (check JSON for success field)
403	Forbidden	Authentication failed
500	Internal Error	Server-side error occurred

Table 8: HTTP status codes

Common Error Messages

- "Admin Passwort falsch" - Invalid admin authentication
- "Support Passwort falsch" - Invalid support authentication
- "Support Token ungültig/abgelaufen" - Invalid or expired support token
- "Keine Berechtigung" - Insufficient permissions
- "Platz ungültig" - Invalid parking spot number
- "Zeitraum bereits belegt!" - Time period already occupied

- "Maximal X Tage reservierbar" - Exceeds maximum reservation duration
- "Buchung in der Vergangenheit nicht möglich" - Cannot book in the past
- "Code ungültig" - Invalid cancellation or authentication code

Integration Best Practices

Polling vs. Webhooks

The current API version supports polling via the `sync` endpoint. For real-time updates, implement periodic polling with appropriate intervals:

- **Dashboard applications:** 30-60 second intervals
- **Mobile applications:** 60-120 second intervals
- **Backend integrations:** 5-15 minute intervals

Rate Limiting

While no explicit rate limits are enforced, we recommend:

- Maximum 1 request per second per tenant
- Implement exponential backoff on errors
- Cache sync data locally to minimize requests

Tenant Identification

The API automatically identifies tenants via subdomain. Ensure your application:

1. Uses the correct tenant subdomain for all requests
2. Does not share authentication credentials across tenants
3. Implements proper tenant isolation in multi-tenant integrations

Security Recommendations

1. **HTTPS Only** - Always use HTTPS for API communications
2. **Credential Storage** - Store authentication hashes securely (encrypted at rest)
3. **Token Rotation** - Regularly rotate support tokens
4. **Input Validation** - Validate all user inputs before sending to API

5. **Error Handling** - Never expose authentication credentials in error logs
6. **Code Security** - Treat 4-digit cancellation codes as sensitive data

Data Synchronization Strategy

For applications requiring real-time parking availability:

1. **Initial Load:** Call `sync` endpoint on application start
2. **Periodic Updates:** Poll `sync` at regular intervals
3. **Local Cache:** Maintain local state between sync calls
4. **Optimistic UI:** Update UI immediately on user actions, confirm with API response
5. **Conflict Resolution:** Handle booking conflicts gracefully with user-friendly messages

Example Use Cases

Use Case 1: Property Management Integration

A property management system integrates Parkzone to provide residents with parking management:

Implementation Steps:

1. Property manager configures tenant via `saveconfig` with building-specific settings
2. Bulk import residents using `saveuser` endpoint
3. Enable `authMode` to require apartment codes for reservations
4. Integrate sync data into property dashboard
5. Provide residents with direct booking interface

Use Case 2: Mobile Parking App

A mobile application provides on-the-go parking management:

Features:

- Real-time availability display via periodic sync calls
- Quick ad-hoc booking for visitors with QR code generation
- Advance reservation management
- Push notifications for reservation confirmations

Use Case 3: Enterprise Access Control

Integration with physical access control systems:

Workflow:

1. Sync endpoint polled every 60 seconds
2. Ad-hoc bookings trigger immediate gate access
3. Reservation data synced with barrier control system
4. License plate recognition validates against booking data
5. History logs provide audit trail for security

Migration and Onboarding

New Tenant Setup

When onboarding a new tenant:

1. **Create Tenant:** Tenant is auto-created on first subdomain access
2. **Configure Settings:** Use `saveconfig` to customize tenant configuration
3. **Set Passwords:** Use `changepw` to set secure admin and support passwords
4. **Import Users:** Bulk create users via `saveuser` endpoint
5. **Test Integration:** Validate sync, booking, and cancellation workflows

Data Migration

For migrating from existing parking systems:

- Export user data in CSV format
- Map user fields to API structure (apt, code, comment)
- Programmatically create users via API
- Import future reservations as needed
- Validate data integrity via sync endpoint

API Changelog

Version 1.0 (Current)

Initial Release Features:

- Multi-tenant support via subdomain isolation
- Complete CRUD operations for reservations and ad-hoc bookings
- User management endpoints
- Configuration customization per tenant
- Hash-based and token-based authentication
- History logging for audit trails

Future Roadmap Considerations:

- Webhook notifications for real-time events
- Extended reporting and analytics endpoints
- Bulk operation endpoints
- GraphQL support
- OpenAPI specification

Support and Contact

For technical support, integration assistance, or API access requests, please contact:

Heitzer - Professional Consulting Services
Website: https://www.heitzer.info
Product Website: https://parkzone.app

Technical Support & Sales:
Email: christian@heitzer.info
Phone: +49 6142 4769324

Appendix A: Quick Reference

Endpoint Summary

Action	Auth	Purpose
sync	None	Get complete tenant state
loginadmin	Admin	Validate admin credentials
loginsupporttoken	Token	Validate support token
saveconfig	Admin/Support	Update configuration
changepw	Admin/Support	Change passwords
saveuser	Admin/Support	Create user
deleteuser	Admin/Support	Delete user
savereservation	Optional	Create reservation
cancelbooking	Admin/Code	Cancel reservation
saveadhoc	None	Create ad-hoc booking
freeadhoc	Admin/Code	Release ad-hoc spot

Table 9: Endpoint quick reference

Date-Time Format

All date-time parameters use ISO 8601 format:

- **Format:** `YYYY-MM-DDTHH:MM` or `YYYY-MM-DDTHH:MM:SS`
- **Example:** `2026-02-20T08:00`
- **Timezone:** Local time (no timezone designation)

Code Generation

The API generates 4-digit numeric codes for:

- Reservation cancellation codes (returned in `savereservation`)
- Ad-hoc booking release codes (returned in `saveadhoc`)
- Range: 1000-9999

Appendix B: Glossary

- **Tenant** - An isolated customer instance with dedicated data and configuration
- **Ad-Hoc Booking** - Short-term immediate parking occupation (typically for guests)
- **Reservation** - Scheduled parking spot booking for future date range
- **Platz** - Parking spot number (German: "place/spot")
- **Kennzeichen** - Vehicle license plate number (German: "license plate")
- **authMode** - Authentication mode requiring user codes for reservations
- **Hash** - SHA-256 password hash for authentication
- **Support Token** - Time-limited token for support access