Galactic Archives Hackathon API Documentation

API Version: 2.1.0
Base URL: `https://api.galacticarchives.space`
OpenAPI Spec: `https://api.galacticarchives.space/openapi.json`
Swagger UI: `https://api.galacticarchives.space/docs`

Table of Contents

Overview

The Galactic Archives Hackathon API provides access to space-related data across three specialized tracks:

- **Alien Archives**: NASA astronomy and exploration data (APOD, Mars Rover photos, Near Earth Objects)
- **Celestial Cartographer**: SpaceX missions and ISS tracking data
- **Astrogators AI**: Space weather and satellite monitoring (Solar storms, Starlink)

All endpoints return JSON responses and are accessible via standard HTTP GET methods.

Authentication

The API is **publicly accessible** with no authentication requirements. All endpoints can be called without API keys or headers.

Core Endpoints

Health Check

Endpoint: `GET /health`

Verify API availability and connectivity.

Response (200 OK):
```
{
"status": "healthy",
"timestamp": "2025-11-26T20:04:00Z",
"version": "2.1.0"
}
```

Python Example:
```
import requests

response = requests.get("https://api.galacticarchives.space/health")
data = response.json()
print(f"Status: {data['status']}")
print(f"Version: {data['version']}")
```

Dashboard Metrics

Endpoint: `GET /api/dashboard`

Retrieve cached aggregated metrics and dashboard data from all tracks.

Response (200 OK):
```
{
"alien_archives": {
"total_requests": 1250,
"last_updated": "2025-11-26T20:00:00Z"
},
"celestial_cartographer": {
"total_requests": 980,
"last_updated": "2025-11-26T20:00:00Z"
},
"astrogators_ai": {
"total_requests": 750,
"last_updated": "2025-11-26T20:00:00Z"
},
"total_api_calls": 2980,
"cache_timestamp": "2025-11-26T20:00:00Z"
}
```

Python Example:
```
import requests
import json
```

```
response = requests.get("https://api.galacticarchives.space/api/dashboard")
dashboard_data = response.json()

print("Dashboard Metrics:")
print(f"Total API Calls: {dashboard_data['total_api_calls']}")
print(f"Last Cache Update: {dashboard_data['cache_timestamp']}")

for track, metrics in dashboard_data.items():
if isinstance(metrics, dict) and 'total_requests' in metrics:
print(f"{track}: {metrics['total_requests']} requests")
```

Alien Archives Track

NASA Astronomy Picture of the Day

Endpoint: `GET /api/artifact/apod`

Retrieve NASA's Astronomy Picture of the Day with description and media information.

Query Parameters:

Parameter	Type	Required	Description
`count`	integer	No	Number of random APODs to retrieve (default: 1)
date	string	No	Specific date in YYYY-MM-DD format

Response (200 OK):
```
{
"title": "The Orion Nebula",
"explanation": "The Orion Nebula is a stellar nursery...",
"url": "https://apod.nasa.gov/apod/image/2511/orion_hubble.jpg",
"media_type": "image",
"date": "2025-11-26",
"copyright": "NASA/ESA"
}
```

Python Example:
```
import requests
from datetime import datetime, timedelta
```

Get today's APOD

```
response = requests.get("https://api.galacticarchives.space/api/artifact/apod")
apod = response.json()

print(f"Title: {apod['title']}")
print(f"Date: {apod['date']}")
print(f"URL: {apod['url']}")
print(f"Explanation: {apod['explanation'][:200]}...")
```

Get multiple random APODs

```
response = requests.get(
"https://api.galacticarchives.space/api/artifact/apod",
params={"count": 5}
)
apods = response.json()
print(f"\nRetrieved {len(apods)} random APODs")
```

Get APOD from specific date

```
yesterday = (datetime.now() - timedelta(days=1)).strftime("%Y-%m-%d")
response = requests.get(
"https://api.galacticarchives.space/api/artifact/apod",
params={"date": yesterday}
)
historic_apod = response.json()
print(f"Yesterday's APOD: {historic_apod['title']}")
```

Mars Rover Photos

Endpoint: `GET /api/artifact/marsphotos`

Retrieve images and data from NASA's Mars rovers (Curiosity, Perseverance).

Query Parameters:

Parameter	Type	Required	Description
`rover`	string	No	Rover name: `curiosity` or `perseverance` (default: `curiosity`)
sol	integer	No	Mars day number (default: latest available)
`camera`	string	No	Camera code (e.g., `FHAZ`, `RHAZ`, `MAST`, `CHEMCAM`, `HAZCAM`)
page	integer	No	Pagination (default: 1)

Response (200 OK):

```
{
"rover": "Curiosity",
"mission_start": "2011-08-05",
"max_date": "2025-11-25",
"photos": [
{
"id": 1478999,
"sol": 4038,
"camera": {
"id": 20,
"name": "FHAZ",
"rover_id": 1
},
"img_src": "https://mars.nasa.gov/msl/api/v1/rover_photos/6789.jpg",
"earth_date": "2025-11-25"
}
],
"total_photos": 854932
}
```

Python Example:

```
import requests
```

Get latest Curiosity photos

```
response = requests.get(
"https://api.galacticarchives.space/api/artifact/marsphotos",
params={"rover": "curiosity"}
)
mars_data = response.json()

print(f"Rover: {mars_data['rover']}")
print(f"Mission Start: {mars_data['mission_start']}")
print(f"Latest Photos Date: {mars_data['max_date']}")
print(f"Total Photos Available: {mars_data['total_photos']}")
```

Iterate through photos

```
for photo in mars_data['photos'][:3]:
print(f"\nSol {photo['sol']} ({photo['earth_date']})")
print(f"Camera: {photo['camera']['name']}")
print(f"URL: {photo['img_src']}")
```

Get Perseverance photos from specific sol

```
response = requests.get(
"https://api.galacticarchives.space/api/artifact/marsphotos",
params={"rover": "perseverance", "sol": 500}
)
perseverance_data = response.json()
print(f"\nPerseverance photos on Sol 500: {len(perseverance_data['photos'])}")
```

Get specific camera photos

```
response = requests.get(
"https://api.galacticarchives.space/api/artifact/marsphotos",
params={"rover": "curiosity", "camera": "CHEMCAM"}
)
chemcam_data = response.json()
print(f"ChemCam photos: {len(chemcam_data['photos'])}")
```

Near Earth Objects

Endpoint: `GET /api/artifact/neo`

Retrieve NASA's Near Earth Objects (asteroids and comets) close approach data.

Query Parameters:

Parameter	Type	Required	Description
`start_date`	string	No	Start date in YYYY-MM-DD format
end_date	string	No	End date in YYYY-MM-DD format
`page`	integer	No	Pagination (default: 1)

Response (200 OK):

```
{
"element_count": 25,
"near_earth_objects": [
{
"id": "3822519",
"neo_reference_id": "3822519",
"name": "2018 LF16",
"absolute_magnitude_h": 22.89,
"estimated_diameter_km": {
"min": 0.073,
"max": 0.163
},
"is_potentially_hazardous_asteroid": false,
"close_approach_data": [
{
"close_approach_date": "2025-11-28",
"relative_velocity_km_per_sec": 14.2,
"miss_distance_km": 7812345
}
]
}
]
}
```

Python Example:

```
import requests
from datetime import datetime, timedelta
```

Get NEOs for the next 7 days

```
today = datetime.now().strftime("%Y-%m-%d")
next_week = (datetime.now() + timedelta(days=7)).strftime("%Y-%m-%d")

response = requests.get(
"https://api.galacticarchives.space/api/artifact/neo",
params={"start_date": today, "end_date": next_week}
)
neo_data = response.json()

print(f"NEOs in next 7 days: {neo_data['element_count']}")

for neo in neo_data['near_earth_objects']:
print(f"\nNEO: {neo['name']} (ID: {neo['id']})")
print(f"Absolute Magnitude: {neo['absolute_magnitude_h']}")
print(f"Estimated Diameter (km): {neo['estimated_diameter_km']['min']:.3f} -
{neo['estimated_diameter_km']['max']:.3f}")
print(f"Potentially Hazardous: {neo['is_potentially_hazardous_asteroid']}")
```

```
for approach in neo['close_approach_data']:
    print(f"\nClose Approach: {approach['close_approach_date']}")
    print(f"Velocity: {approach['relative_velocity_km_per_sec']} km/s")
    print(f"Miss Distance: {approach['miss_distance_km']} km")
```

Filter for potentially hazardous asteroids

```
hazardous = [
neo for neo in neo_data['near_earth_objects']
if neo['is_potentially_hazardous_asteroid']
]
print(f"\nPotentially Hazardous: {len(hazardous)}")
```

Celestial Cartographer Track

SpaceX Launches

Endpoint: `GET /api/cartographer/spacexlaunches`

Retrieve SpaceX launch history and upcoming missions.

Query Parameters:

Parameter	Type	Required	Description
`limit`	integer	No	Maximum results (default: 10)
status	string	No	Filter by status: success, failure, upcoming
`rocket`	string	No	Rocket model filter

Response (200 OK):
```
{
"total_launches": 287,
"launches": [
{
"id": "5e9d0d95eda59f758e7a3b14",
"name": "Starlink 1-145",
"date_utc": "2025-11-26T23:47:00Z",
"status": "success",
"rocket": {
"id": "5e9d0d95eda59f7a78618c6d",
"name": "Falcon 9"
},
"launchpad": {
"name": "CCAFS SLC 40",
"location": "Cape Canaveral, Florida"
},
"mission": {
"name": "Starlink 1-145",
"type": "Communications"
}
}
]
}
```

Python Example:
```
import requests
from datetime import datetime
```

Get recent SpaceX launches

```
response = requests.get(
"https://api.galacticarchives.space/api/cartographer/spacexlaunches",
params={"limit": 20, "status": "success"}
)
launches_data = response.json()

print(f"Total SpaceX Launches: {launches_data['total_launches']}")
print(f"Recent Successful Launches: {len(launches_data['launches'])}\n")

for launch in launches_data['launches'][:5]:
launch_date = datetime.fromisoformat(launch['date_utc'].replace('Z', '+00:00'))
print(f"Mission: {launch['name']}")
print(f"Date: {launch_date.strftime('%Y-%m-%d %H:%M:%S UTC')}")
print(f"Rocket: {launch['rocket']['name']}")
print(f"Launchpad: {launch['launchpad']['name']} - {launch['launchpad']['location']}")
print(f"Mission Type: {launch['mission']['type']}")
print()
```

Get upcoming launches

```
response = requests.get(
"https://api.galacticarchives.space/api/cartographer/spacexlaunches",
params={"limit": 5, "status": "upcoming"}
)
upcoming = response.json()
print(f"Upcoming Launches: {len(upcoming['launches'])}")
```

Get Falcon Heavy launches

```
response = requests.get(
"https://api.galacticarchives.space/api/cartographer/spacexlaunches",
params={"limit": 100, "rocket": "Falcon Heavy"}
)
falcon_heavy = response.json()
print(f"Falcon Heavy Launches: {len(falcon_heavy['launches'])}")
```

Latest SpaceX Launch

Endpoint: `GET /api/cartographer/spacexlatest`

Retrieve details of the most recent SpaceX launch.

Response (200 OK):

```
{
"id": "5e9d0d95eda59f758e7a3b14",
"name": "Starlink 1-145",
"date_utc": "2025-11-26T23:47:00Z",
"status": "success",
"rocket": {
"id": "5e9d0d95eda59f7a78618c6d",
"name": "Falcon 9",
"stages": 2
},
"launchpad": {
"name": "CCAFS SLC 40",
"location": "Cape Canaveral, Florida",
"latitude": 28.5535,
"longitude": -80.5771
},
"mission": {
"name": "Starlink 1-145",
"type": "Communications"
},
"details": "Deployed 53 Starlink satellites to orbit"
}
```

Python Example:

```
import requests
from datetime import datetime
```

Get latest SpaceX launch

```
response = requests.get("https://api.galacticarchives.space/api/cartographer/spacexlatest")
latest = response.json()

print(f"Latest SpaceX Launch:")
print(f"Mission: {latest['name']}")
print(f"Date: {latest['date_utc']}")
print(f"Rocket: {latest['rocket']['name']} (Stages: {latest['rocket']['stages']})")
print(f"Status: {latest['status'].upper()}")
print(f"Location: {latest['launchpad']['location']}")
print(f"Details: {latest['details']}")
```

Calculate days since last launch

```
launch_date = datetime.fromisoformat(latest['date_utc'].replace('Z', '+00:00'))
days_ago = (datetime.now(launch_date.tzinfo) - launch_date).days
print(f"Days since launch: {days_ago}")
```

ISS Current Location

Endpoint: `GET /api/cartographer/isslocation`

Get real-time location and velocity data for the International Space Station.

Response (200 OK):

```
{
```

```
"name": "ISS (ZARYA)",
"timestamp": "2025-11-26T20:15:33Z",
"latitude": 51.6442,
"longitude": -63.4891,
"altitude_km": 407.8,
"velocity_kmh": 27600,
"visibility": "daylight",
"footprint": 2400,
"solar_array_angle": 15.2,
"eclipse": false
}
```

Python Example:

```
import requests
import math
```

Get ISS location

```
response = requests.get("https://api.galacticarchives.space/api/cartographer/isslocation")
iss = response.json()

print(f"ISS Status Report")
print(f"Timestamp: {iss['timestamp']}")
print(f"Position: {iss['latitude']:.4f}°, {iss['longitude']:.4f}°")
print(f"Altitude: {iss['altitude_km']} km")
print(f"Velocity: {iss['velocity_kmh']} km/h ({iss['velocity_kmh']/3.6:.1f} m/s)")
print(f"Visibility: {iss['visibility']}")
print(f"In Shadow: {iss['eclipse']}")
print(f"Ground Coverage Footprint: {iss['footprint']} km radius")
```

Calculate speed in orbit per minute

```
speed_per_minute = (iss['velocity_kmh'] / 60)
print(f"\nSpeed per minute: {speed_per_minute:.2f} km")
print(f"Orbital period: ~90 minutes")
print(f"Distance per orbit: ~{int(iss['velocity_kmh'] * 1.5)} km")
```

Determine hemisphere

```
hemisphere = "Northern" if iss['latitude'] > 0 else "Southern"
print(f"Currently over {hemisphere} Hemisphere")
```

People in Space

Endpoint: `GET /api/cartographer/peopleinspace`

Get count and details of astronauts currently in orbit.

Response (200 OK):

```
{
"number": 7,
"people": [
{
"name": "Aleksandr Grebenkin",
"craft": "ISS",
"country": "Russia",
"launch_date": "2024-09-11T11:19:00Z"
},
{
"name": "Oleg Kononenko",
"craft": "ISS",
"country": "Russia",
"launch_date": "2023-09-15T12:34:00Z"
}
]
}
```

Python Example:

```
import requests
from datetime import datetime
```

Get current people in space

```
response = requests.get("https://api.galacticarchives.space/api/cartographer/peopleinspace")
space_crew = response.json()

print(f"People Currently in Space: {space_crew['number']}")
print(f"{'Name':<25} {'Craft':<10} {'Country':<15} {'Days in Space'}")
print("-" * 70)

now = datetime.now(datetime.timezone.utc)

for person in space_crew['people']:
launch = datetime.fromisoformat(person['launch_date'].replace('Z', '+00:00'))
days_in_space = (now - launch).days
```

```
print(f"{person['name']:<25} {person['craft']:<10} {person['country']:<15} {days_i
```

Count by country

```
countries = {}
for person in space_crew['people']:
country = person['country']
countries[country] = countries.get(country, 0) + 1
```

```
print(f"\nAstronauts by Country:")
for country, count in sorted(countries.items(), key=lambda x: x[1], reverse=True):
print(f" {country}: {count}")
```

Astrogators AI Track

Solar Storm Alert

Endpoint: `GET /api/advisor/solarstorm`

Retrieve current solar activity and space weather alerts.

Query Parameters:

Parameter	Type	Required	Description
`days_back`	integer	No	Historical data in past days (default: 7)

Response (200 OK):

```
{
"timestamp": "2025-11-26T20:00:00Z",
"solar_activity": {
"level": "moderate",
"kp_index": 6.2,
"solar_wind_speed": 450,
"bz_component": -5.2
},
"alerts": [
{
"type": "G2",
"severity": "moderate",
"description": "Moderate Geomagnetic Storm - Kp 6-7",
"issued": "2025-11-26T14:30:00Z",
"expected_duration": "12 hours"
}
],
"aurora_forecast": {
"visibility_latitude": 55,
"intensity": "moderate"
}
}
```

Python Example:

```
import requests
from datetime import datetime
```

Get current solar storm data

```
response = requests.get("https://api.galacticarchives.space/api/advisor/solarstorm")
space_weather = response.json()

print(f"Space Weather Report - {space_weather['timestamp']}")
print(f"\nSolar Activity:")
print(f" Level: {space_weather['solar_activity']['level'].upper()}")
print(f" Kp Index: {space_weather['solar_activity']['kp_index']}")
print(f" Solar Wind Speed: {space_weather['solar_activity']['solar_wind_speed']} km/s")
print(f" Bz Component: {space_weather['solar_activity']['bz_component']} nT")

print(f"\nAlerts ({len(space_weather['alerts'])}):")
for alert in space_weather['alerts']:
print(f" {alert['type']} - {alert['severity'].upper()}")
print(f" {alert['description']}")
print(f" Issued: {alert['issued']}")
print(f" Duration: {alert['expected_duration']}")

print(f"\nAurora Forecast:")
print(f" Visible at latitude: >{space_weather['aurora_forecast']['visibility_latitude']}°")
print(f" Intensity: {space_weather['aurora_forecast']['intensity']}")
```

Get 30-day history

```
response = requests.get(
"https://api.galacticarchives.space/api/advisor/solarstorm",
params={"days_back": 30}
)
historical = response.json()
print(f"\n30-Day Historical Alerts: {len(historical.get('alerts', []))}")
```

Starlink Satellites

Endpoint: `GET /api/advisor/starlink`

Get information about operational Starlink satellites.

Query Parameters:

Parameter	Type	Required	Description
`limit`	integer	No	Maximum satellites to return (default: 50)
active_only	boolean	No	Filter to active satellites only (default: true)

Response (200 OK):

```
{
"total_satellites": 6847,
"active_satellites": 6200,
"decommissioned": 647,
"satellites": [
{
"norad_id": 55912,
"name": "STARLINK-1368",
"status": "active",
"launch_date": "2022-12-26",
"orbital_altitude_km": 552,
"inclination": 97.41,
"latitude": 42.5,
"longitude": -115.3,
"velocity_kmh": 27300
}
]
}
```

Python Example:

```
import requests
```

Get active Starlink satellites

```
response = requests.get(
"https://api.galacticarchives.space/api/advisor/starlink",
params={"limit": 100, "active_only": True}
)
starlink_data = response.json()

print(f"Starlink Constellation Status")
print(f"Total Satellites: {starlink_data['total_satellites']}")
print(f"Active: {starlink_data['active_satellites']}")
print(f"Decommissioned: {starlink_data['decommissioned']}")

print(f"\nSample Satellites:")
print(f"{'Name':<20} {'Status':<10} {'Latitude':<12} {'Longitude':<12} {'Alt (km)':<10}")
print("-" * 70)

for sat in starlink_data['satellites'][:10]:
print(f"{sat['name']:<20} {sat['status']:<10} {sat['latitude']:<12.2f} {sat['longitude']:<12.2f}
{sat['orbital_altitude_km']:<10}")
```

Calculate coverage statistics

```
active = [s for s in starlink_data['satellites'] if s['status'] == 'active']
avg_altitude = sum(s['orbital_altitude_km'] for s in active) / len(active) if active else 0
print(f"\nAverage Orbital Altitude: {avg_altitude:.1f} km")
```

Group by inclination pattern

```
inclinations = {}
for sat in active:
inc = round(sat['inclination'], 1)
inclinations[inc] = inclinations.get(inc, 0) + 1
```

```
print(f"\nOrbital Planes (Inclination):")
for inc in sorted(inclinations.keys()):
print(f" {inc}°: {inclinations[inc]} satellites")
```

Testing Utilities

Test Any API

Endpoint: `GET /api/test/{api_name}`

Generic testing endpoint to validate any API endpoint's functionality.

Path Parameters:

Parameter	Type	Required	Description
`api_name`	string	Yes	Name of API to test

Response (200 OK):

```
{
"test_name": "nasa_apod",
"status": "success",
"response_time_ms": 245,
"data_size_bytes": 1823,
"timestamp": "2025-11-26T20:15:00Z"
}
```

Python Example:

```
import requests
import json
```

Test various APIs

```
apis_to_test = [
"nasa_apod",
"mars_photos",
"spacex_launches",
"iss_location",
"starlink",
"solar_storm"
]
```

```
print("API Health Check Results:")
print(f"{'API Name':<25} {'Status':<10} {'Response Time (ms)':<20}")
print("-" * 55)
```

```
for api in apis_to_test:
try:
response = requests.get(f"https://api.galacticarchives.space/api/test/{api}")
data = response.json()
```

```
        status = data.get('status', 'unknown')
        response_time = data.get('response_time_ms', 'N/A')

        print(f"{api:<25} {status:<10} {response_time:<20}")
    except Exception as e:
        print(f"{api:<25} {'error':<10} {str(e):<20}")
```

Response Schemas

AggregatedData Schema

Represents aggregated metrics and dashboard data.

```
{
"alien_archives": {
"total_requests": "integer",
"last_updated": "string (ISO 8601)"
},
"celestial_cartographer": {
"total_requests": "integer",
"last_updated": "string (ISO 8601)"
},
"astrogators_ai": {
"total_requests": "integer",
"last_updated": "string (ISO 8601)"
},
"total_api_calls": "integer",
```

"cache_timestamp": "string (ISO 8601)"
}

HTTPValidationError Schema

Standard HTTP validation error response.

{
"detail": [
{
"type": "string",
"loc": ["string"],
"msg": "string",
"input": "any"
}
]
}

ValidationError Schema

Detailed validation error information.

{
"detail": [
{
"type": "string",
"loc": ["string"],
"msg": "string",
"input": "any",
"url": "string (URI)"
}
]
}

Error Handling

HTTP Status Codes

Status Code	Description	Example
200	Success	Successful API call with data
400	Bad Request	Invalid query parameters
404	Not Found	Endpoint does not exist
422	Unprocessable Entity	Validation error in request
500	Internal Server Error	Server error during processing
503	Service Unavailable	API temporarily unavailable

Common Error Responses

Invalid Query Parameter:

```
{
"detail": [
{
"type": "value_error",
"loc": ["query", "limit"],
"msg": "ensure this value is less than or equal to 100",
"input": 500
}
]
}
```

Endpoint Not Found:

```
{
"detail": "Not Found"
}
```

Error Handling Best Practices in Python

```
import requests
from requests.exceptions import RequestException, Timeout

def safe_api_call(endpoint, params=None, timeout=10):
"""Safely call Galactic Archives API with error handling"""
try:
response = requests.get(
f"https://api.galacticarchives.space{endpoint}",
params=params,
timeout=timeout
)
response.raise_for_status() # Raise exception for bad status codes
return response.json()
```

```
except Timeout:
    print(f"Request timed out after {timeout} seconds")
    return None

except requests.exceptions.HTTPError as e:
    print(f"HTTP Error: {e.response.status_code}")
    if e.response.status_code == 422:
        print(f"Validation Error: {e.response.json()}")
    elif e.response.status_code == 404:
        print("Endpoint not found")
    return None

except requests.exceptions.RequestException as e:
    print(f"Request failed: {e}")
    return None
```

Usage

data = safe_api_call("/api/artifact/apod", params={"count": 5})
if data:
print(f"Retrieved data: {data}")

Python Examples

Complete Hackathon Companion Script

""""
Galactic Archives Hackathon API Companion
Complete example utilizing all API tracks
""""

import requests
import json
from datetime import datetime, timedelta
from typing import Optional, Dict, List
import time

class GalacticArchivesAPI:
""""Client for Galactic Archives Hackathon API""""

```
BASE_URL = "https://api.galacticarchives.space"
```

```
    def __init__(self, timeout=10):
        self.timeout = timeout
        self.session = requests.Session()

    def health_check(self) -> Optional[Dict]:
        """Check API health status"""
        try:
            response = self.session.get(
                f"{self.BASE_URL}/health",
                timeout=self.timeout
            )
            response.raise_for_status()
            return response.json()
        except Exception as e:
            print(f"Health check failed: {e}")
            return None

    def get_dashboard(self) -> Optional[Dict]:
        """Get cached dashboard metrics"""
        try:
            response = self.session.get(
                f"{self.BASE_URL}/api/dashboard",
                timeout=self.timeout
            )
            response.raise_for_status()
            return response.json()
        except Exception as e:
            print(f"Dashboard fetch failed: {e}")
            return None

    # Alien Archives Methods

    def get_apod(self, count=1, date=None) -> Optional[Dict]:
        """Get NASA Astronomy Picture of the Day"""
        params = {"count": count}
        if date:
            params["date"] = date
        try:
            response = self.session.get(
                f"{self.BASE_URL}/api/artifact/apod",
                params=params,
                timeout=self.timeout
            )
            response.raise_for_status()
            return response.json()
        except Exception as e:
            print(f"APOD fetch failed: {e}")
            return None

    def get_mars_photos(self, rover="curiosity", sol=None, camera=None) -> Option
        """Get Mars rover photos"""
        params = {"rover": rover}
        if sol:
            params["sol"] = sol
        if camera:
            params["camera"] = camera

        try:
            response = self.session.get(
                f"{self.BASE_URL}/api/artifact/marsphotos",
                params=params,
                timeout=self.timeout
            )
            response.raise_for_status()
            return response.json()
        except Exception as e:
            print(f"Mars photos fetch failed: {e}")
            return None

    def get_near_earth_objects(self, start_date=None, end_date=None) -> Optional[D
        """Get Near Earth Objects"""
        params = {}
        if start_date:
            params["start_date"] = start_date
        if end_date:
            params["end_date"] = end_date

        try:
            response = self.session.get(
                f"{self.BASE_URL}/api/artifact/neo",
                params=params,
                timeout=self.timeout
            )
            response.raise_for_status()
            return response.json()
        except Exception as e:
            print(f"NEO fetch failed: {e}")
            return None

    # Celestial Cartographer Methods

    def get_spacex_launches(self, limit=10, status=None) -> Optional[Dict]:
        """Get SpaceX launches"""
        params = {"limit": limit}
        if status:
            params["status"] = status

        try:
            response = self.session.get(
                f"{self.BASE_URL}/api/cartographer/spacexlaunches",
                params=params,
                timeout=self.timeout
            )
            response.raise_for_status()
            return response.json()
        except Exception as e:
            print(f"SpaceX launches fetch failed: {e}")
            return None

    def get_latest_spacex_launch(self) -> Optional[Dict]:
        """Get latest SpaceX launch"""
        try:
            response = self.session.get(
                f"{self.BASE_URL}/api/cartographer/spacexlatest",
                timeout=self.timeout
            )
            response.raise_for_status()
            return response.json()
        except Exception as e:
            print(f"Latest SpaceX launch fetch failed: {e}")
            return None

    def get_iss_location(self) -> Optional[Dict]:
        """Get ISS current location"""
        try:
            response = self.session.get(
                f"{self.BASE_URL}/api/cartographer/isslocation",
                timeout=self.timeout
            )
            response.raise_for_status()
            return response.json()
        except Exception as e:
            print(f"ISS location fetch failed: {e}")
            return None

    def get_people_in_space(self) -> Optional[Dict]:
        """Get people currently in space"""
        try:
            response = self.session.get(
                f"{self.BASE_URL}/api/cartographer/peopleinspace",
                timeout=self.timeout
            )
            response.raise_for_status()
            return response.json()
        except Exception as e:
            print(f"People in space fetch failed: {e}")
            return None

    # Astrogators AI Methods

    def get_solar_storm_alerts(self, days_back=7) -> Optional[Dict]:
            """Get solar storm alerts and space weather"""
            params = {"days_back": days_back}

            try:
                response = self.session.get(
                    f"{self.BASE_URL}/api/advisor/solarstorm",
                    params=params,
                    timeout=self.timeout
                )
                response.raise_for_status()
                return response.json()
            except Exception as e:
                print(f"Solar storm alerts fetch failed: {e}")
                return None

        def get_starlink_satellites(self, limit=50, active_only=True) -> Optional[Dict]:
            """Get Starlink satellite data"""
            params = {"limit": limit, "active_only": active_only}

            try:
                response = self.session.get(
                    f"{self.BASE_URL}/api/advisor/starlink",
                    params=params,
                    timeout=self.timeout
                )
                response.raise_for_status()
                return response.json()
            except Exception as e:
                print(f"Starlink satellites fetch failed: {e}")
                return None

        def test_api(self, api_name) -> Optional[Dict]:
            """Test specific API endpoint"""
            try:
                response = self.session.get(
                    f"{self.BASE_URL}/api/test/{api_name}",
                    timeout=self.timeout
                )
                response.raise_for_status()
                return response.json()
            except Exception as e:
                print(f"API test failed: {e}")
                return None

        def close(self):
            """Close the session"""
            self.session.close()
```

Example Usage

def main():
"""Demonstrate all API endpoints"""

```
    api = GalacticArchivesAPI()

    # Health check
    print("=== API Health Check ===")
    health = api.health_check()
    if health:
        print(f"Status: {health['status']}")
        print(f"Version: {health['version']}\n")

    # Dashboard
    print("=== Dashboard Metrics ===")
    dashboard = api.get_dashboard()
    if dashboard:
        print(f"Total API Calls: {dashboard['total_api_calls']}\n")

    # Alien Archives
    print("=== Alien Archives ===")
    apod = api.get_apod()
    if apod:
        print(f"APOD: {apod['title']}")
        print(f"Date: {apod['date']}\n")

    mars = api.get_mars_photos()
```

```
if mars:
    print(f"Mars Photos: {len(mars.get('photos', []))} found")
    print(f"Rover: {mars['rover']}\n")

neo = api.get_near_earth_objects()
if neo:
    print(f"Near Earth Objects: {neo['element_count']} in date range\n")

# Celestial Cartographer
print("=== Celestial Cartographer ===")
launches = api.get_spacex_launches(limit=5)
if launches:
    print(f"SpaceX Launches: {len(launches['launches'])} (recent)")
    latest = api.get_latest_spacex_launch()
    if latest:
        print(f"Latest: {latest['name']}\n")

iss = api.get_iss_location()
if iss:
    print(f"ISS Location: {iss['latitude']:.2f}°, {iss['longitude']:.2f}°")
    print(f"Altitude: {iss['altitude_km']} km\n")

people = api.get_people_in_space()
if people:
    print(f"People in Space: {people['number']}\n")

# Astrogators AI
print("=== Astrogators AI ===")
weather = api.get_solar_storm_alerts()
if weather:
    print(f"Solar Activity: {weather['solar_activity']['level']}")
    print(f"Kp Index: {weather['solar_activity']['kp_index']}\n")

starlink = api.get_starlink_satellites(limit=5)
if starlink:
    print(f"Starlink: {starlink['active_satellites']} active satellites\n")
```

```
  # Close session
  api.close()
```

if **name** == "**main**":
main()

Quick Reference

Base URL

https://api.galacticarchives.space

Documentation

https://api.galacticarchives.space/docs
https://api.galacticarchives.space/openapi.json

All Endpoints Summary

Track	Endpoint	Method	Purpose
Core	/health	GET	Health check
Core	/api/dashboard	GET	Dashboard metrics
Alien Archives	/api/artifact/apod	GET	NASA APOD
Alien Archives	/api/artifact/marsphotos	GET	Mars rover photos
Alien Archives	/api/artifact/neo	GET	Near Earth Objects
Celestial Cartographer	/api/cartographer/spacexlaunches	GET	SpaceX launches
Celestial Cartographer	/api/cartographer/spacexlatest	GET	Latest SpaceX launch
Celestial Cartographer	/api/cartographer/isslocation	GET	ISS location
Celestial Cartographer	/api/cartographer/peopleinspace	GET	People in space
Astrogators AI	/api/advisor/solarstorm	GET	Solar storm alerts
Astrogators AI	/api/advisor/starlink	GET	Starlink satellites
Testing	/api/test/{api_name}	GET	Test API endpoint

Support and Resources

- **API Documentation:** https://api.galacticarchives.space/docs
- **OpenAPI Specification:** https://api.galacticarchives.space/openapi.json
- **GitHub:** https://github.com/galacticarchives
- **Issue Tracker:** Report issues through GitHub

Document Version: 1.0
Last Updated: November 26, 2025
API Version: 2.1.0

This documentation provides comprehensive coverage of the Galactic Archives Hackathon API with production-ready Python examples suitable for hackathon development and integration.