

Contents

RADIANT Error Codes Reference	1
Overview	1
Error Code Format	1
Authentication Errors (1xxx)	2
Authorization Errors (2xxx)	2
Validation Errors (3xxx)	2
Resource Errors (4xxx)	3
Rate Limiting Errors (5xxx)	3
AI/Model Errors (6xxx)	3
Billing Errors (7xxx)	4
Storage Errors (8xxx)	4
Internal Errors (9xxx)	4
Usage in Code	5
TypeScript/JavaScript	5
Response Format	5
Client Handling	5
Retry Logic	5
Error Display	6
Adding New Error Codes	6
See Also	7

RADIANT Error Codes Reference

Standardized error codes for consistent API responses across all RADIANT services.

Overview

All RADIANT errors follow a consistent format:

```
{  
    "error": {  
        "code": "RADIANT_AUTH_1001",  
        "message": "Invalid authentication token. Please sign in again.",  
        "category": "authentication",  
        "retryable": false,  
        "timestamp": "2024-12-25T10:30:00.000Z"  
    }  
}
```

Error Code Format

RADIANT_<CATEGORY>_<NUMBER>

- **RADIANT** - Prefix for all error codes
- **CATEGORY** - Short category identifier (AUTH, VAL, RES, etc.)
- **NUMBER** - Unique 4-digit number within category

Authentication Errors (1xxx)

Errors related to authentication and identity.

Code	HTTP	Retryable	Description
RADIANT_AUTH_10011			Invalid authentication token
RADIANT_AUTH_10021			Token has expired
RADIANT_AUTH_10031			Missing authentication token
RADIANT_AUTH_10041			Invalid API key
RADIANT_AUTH_10051			API key has expired
RADIANT_AUTH_10061			API key has been revoked
RADIANT_AUTH_10073			Insufficient API key scope
RADIANT_AUTH_10081			Multi-factor authentication required
RADIANT_AUTH_10091			Session has expired

Authorization Errors (2xxx)

Errors related to permissions and access control.

Code	HTTP	Retryable	Description
RADIANT_AUTHZ_2001	403		Forbidden - access denied
RADIANT_AUTHZ_2002	403		Tenant ID mismatch
RADIANT_AUTHZ_2003	403		Required role not assigned
RADIANT_AUTHZ_2004	403		Permission denied
RADIANT_AUTHZ_2005	403		Resource access denied
RADIANT_AUTHZ_2006	403		Subscription tier insufficient

Validation Errors (3xxx)

Errors related to input validation.

Code	HTTP	Retryable	Description
RADIANT_VAL_3001	400		Required field is missing
RADIANT_VAL_3002	400		Invalid field format
RADIANT_VAL_3003	400		Value out of allowed range
RADIANT_VAL_3004	400		Invalid data type
RADIANT_VAL_3005	400		Constraint violation
RADIANT_VAL_3006	400		Schema mismatch
RADIANT_VAL_3007	400		Invalid JSON in request body
RADIANT_VAL_3008	400		Maximum length exceeded
RADIANT_VAL_3009	400		Minimum length required

Resource Errors (4xxx)

Errors related to resources and entities.

Code	HTTP	Retryable	Description
RADIANT_RES_4001	404		Resource not found
RADIANT_RES_4002	409		Resource already exists
RADIANT_RES_4003	410		Resource has been deleted
RADIANT_RES_4004	423		Resource is locked
RADIANT_RES_4005	409		Resource conflict
RADIANT_RES_4006	429		Resource quota exceeded

Rate Limiting Errors (5xxx)

Errors related to rate limiting and throttling.

Code	HTTP	Retryable	Description
RADIANT_RATE_5001	429		Rate limit exceeded
RADIANT_RATE_5002	429		Tenant rate limit exceeded
RADIANT_RATE_5003	429		User rate limit exceeded
RADIANT_RATE_5004	429		API key rate limit exceeded
RADIANT_RATE_5005	429		Model rate limit exceeded
RADIANT_RATE_5006	429		Burst limit exceeded

Retry-After Header: Rate limit errors include `Retry-After` header with seconds to wait.

AI/Model Errors (6xxx)

Errors related to AI models and inference.

Code	HTTP	Retryable	Description
RADIANT_AI_6001	404		Model not found
RADIANT_AI_6002	503		Model temporarily unavailable
RADIANT_AI_6003	503		Model overloaded
RADIANT_AI_6004	502		AI provider error
RADIANT_AI_6005	400		Context length exceeded
RADIANT_AI_6006	400		Content filtered by safety
RADIANT_AI_6007	400		Invalid AI request
RADIANT_AI_6008	500		Streaming error
RADIANT_AI_6009	504		AI request timeout
RADIANT_AI_6010	503		Model is cold (warming up)

Billing Errors (7xxx)

Errors related to billing, credits, and subscriptions.

Code	HTTP	Retryable	Description
RADIANT_BILL_7001	402		Insufficient credits
RADIANT_BILL_7002	402		Payment required
RADIANT_BILL_7003	402		Payment failed
RADIANT_BILL_7004	402		Subscription expired
RADIANT_BILL_7005	402		Subscription cancelled
RADIANT_BILL_7006	400		Invalid coupon code
RADIANT_BILL_7007	429		Usage quota exceeded

Storage Errors (8xxx)

Errors related to file storage.

Code	HTTP	Retryable	Description
RADIANT_STOR_8001	413		Storage quota exceeded
RADIANT_STOR_8002	413		File too large
RADIANT_STOR_8003	415		Invalid file type
RADIANT_STOR_8004	500		Upload failed
RADIANT_STOR_8005	404		File not found

Internal Errors (9xxx)

Internal server errors and system failures.

Code	HTTP	Retryable	Description
RADIANT_INT_9001	500		Internal server error
RADIANT_INT_9002	500		Database error
RADIANT_INT_9003	500		Cache error
RADIANT_INT_9004	500		Queue processing error
RADIANT_INT_9005	503		Service unavailable
RADIANT_INT_9006	502		Dependency failure
RADIANT_INT_9007	500		Configuration error
RADIANT_INT_9008	504		Request timeout

Usage in Code

TypeScript/JavaScript

```
import {
  ErrorCodes,
  RadianError,
  createNotFoundError,
  createValidationError,
  isRetryableError
} from '@radiant/shared';

// Using factory functions (recommended)
throw createNotFoundError('User', userId);
throw createValidationError('Email is required', 'email');

// Direct construction
throw new RadianError(ErrorCodes.AUTH_INVALID_TOKEN, 'Custom message', {
  details: { tokenPrefix: 'rad_...' },
  requestId: context.awsRequestId,
});

// Check if retryable
if (isRetryableError(error.code)) {
  // Implement retry logic
}
```

Response Format

```
// RadianError automatically formats responses
const error = new RadianError(ErrorCodes.RESOURCE_NOT_FOUND);
return error.toResponse();

// Returns:
//{
//  statusCode: 404,
//  headers: { 'Content-Type': 'application/json' },
//  body: '{"error":{"code":"RADIANT_RES_4001","message":"Resource not found."}}, "category": "re
// }
```

Client Handling

Retry Logic

```
async function callWithRetry(fn: () => Promise<Response>, maxRetries = 3) {
  for (let i = 0; i < maxRetries; i++) {
    try {
      const response = await fn();
```

```

    if (response.ok) return response;

    const error = await response.json();
    if (!error.error.retryable) throw error;

    const retryAfter = response.headers.get('Retry-After') || '5';
    await sleep(parseInt(retryAfter) * 1000);
} catch (e) {
    if (i === maxRetries - 1) throw e;
}
}
}

```

Error Display

```

function getUserMessage(error: RadiantError): string {
    // Error codes include user-friendly messages
    return error.message;
}

function shouldShowRetryButton(error: RadiantError): boolean {
    return error.retryable;
}

```

Adding New Error Codes

1. Add the code to packages/shared/src/errors/codes.ts:

```

export const ErrorCodes = {
    // ... existing codes
    MY_NEW_ERROR: 'RADIANT_CAT_NNNN',
} as const;

```

2. Add metadata:

```

export const ErrorCodeMetadata: Record<ErrorCode, {...}> = {
    // ... existing metadata
    [ErrorCodes.MY_NEW_ERROR]: {
        httpStatus: 400,
        category: 'category',
        retryable: false,
        userMessage: 'User-friendly error message.',
    },
};

```

3. Update this documentation.
-

See Also

- [API Reference](#)
- [Contributing Guide](#)
- [Troubleshooting](#)