THREADSYNC — MAGIC RUNTIME

Magic Runtime — Error Catalog

February 2026 · v2.0 · magic.threadsync.io

Error Catalog

Every error is deterministic, categorized, and actionable. No mystery 500s.

- 25 documented errors
- 5 categories
- Structured JSON responses
- Actionable resolutions

Error Response Format

All errors return a consistent JSON envelope. Clients can parse the `code` field to programmatically handle any error condition.

Standard Error Response

```
{
  "error": {
    "code": "E1003",
    "category": "CONTRACT",
    "message": "Input validation failed: field 'email' does not match
pattern",
    "request_id": "req_abc123",
    "timestamp": "2026-02-12T18:00:00Z",
    "details": {
      "field": "email",
      "constraint": "pattern",
      "expected": "^[\\w.-]+@[\\w.-]+$",
      "received": "not-an-email"
    }
  }
}
```

FIELD	TYPE	DESCRIPTION
`code`	string	Unique error code (e.g., `E1003`). Stable across versions.
`category`	string	Error category: CONTRACT, CAPABILITY, RUNTIME, DEPLOYMENT, AUTH
`message`	string	Human-readable description of what went wrong
`request_id`	string	Unique request identifier for correlation and support
`timestamp`	string	ISO 8601 timestamp of when the error occurred
`details`	object	Error-specific context (varies by error code)

CONTRACT Errors (E1xxx)

Contract errors occur when a controller's contract definition is invalid, or when request/response data fails schema validation.

Contract Validation Failures

E1001 – E1005

5 errors

E1001 Invalid contract schema HTTP 400

The controller's contract file does not conform to the Magic contract specification. This is caught at deploy time and prevents the controller from being registered.

EXAMPLE RESPONSE


```
{
  "error": {
    "code": "E1001",
    "category": "CONTRACT",
    "message": "Invalid contract schema: 'capabilities' must be an array",
    "request_id": "req_7f2a9b",
    "timestamp": "2026-02-12T18:00:00Z",
    "details": {
      "path": "$.capabilities",
      "expected_type": "array",
      "received_type": "string",
      "schema_version": "2.0"
    }
  }
}
```


Resolution

1. Validate your contract against the schema: `magic validate contract.json`
2. Check the `details.path` field to locate the exact issue in your contract
3. Refer to the Contracts documentation for the full specification

E1002 Missing required field HTTP 400

A required field is missing from the controller contract. Every contract must include `name`, `version`, `input`, and `output` sections.

EXAMPLE RESPONSE

```
{
  "error": {
    "code": "E1002",
    "category": "CONTRACT",
    "message": "Missing required field: 'output' schema is required",
    "request_id": "req_3e8c1d",
    "timestamp": "2026-02-12T18:00:00Z",
    "details": {
      "missing_field": "output",
      "contract_name": "ProcessOrder",
      "required_fields": ["name", "version", "input", "output"]
    }
  }
}
```

Resolution

1. Check the `details.missing_field` to identify which field is absent
2. Add the missing field to your contract JSON
3. Run `magic validate contract.json` to confirm all required fields are present

E1003 Input validation failed HTTP 422

The request body does not match the controller's declared input schema. The runtime validates all inputs before execution begins -- the controller code never sees invalid data.

EXAMPLE RESPONSE

```
{
  "error": {
    "code": "E1003",
    "category": "CONTRACT",
    "message": "Input validation failed: field 'email' does not match
pattern",
    "request_id": "req_abc123",
    "timestamp": "2026-02-12T18:00:00Z",
    "details": {
      "field": "email",
      "constraint": "pattern",
      "expected": "^[\\w.-]+@[\\w.-]+$",
      "received": "not-an-email"
    }
  }
}
```

Resolution

1. Inspect `details.field` and `details.constraint` to understand what failed
2. Fix the request payload to conform to the input schema
3. Common issues: missing required fields, wrong types, pattern mismatches, values outside min/max range

E1004 Output validation failed HTTP 502

The controller returned a response that does not match its declared output schema. This indicates a bug in the controller code -- the runtime blocks the malformed response from reaching the caller.

EXAMPLE RESPONSE


```
{
  "error": {
    "code": "E1004",
    "category": "CONTRACT",
    "message": "Output validation failed: 'total' must be a number, got
string",
    "request_id": "req_d91f4a",
    "timestamp": "2026-02-12T18:00:00Z",
    "details": {
      "field": "total",
      "constraint": "type",
      "expected": "number",
      "received": "string",
      "controller": "ProcessOrder",
      "controller_version": "1.2.0"
    }
  }
}
```


Resolution

1. This is a controller bug -- the controller code returns data that violates its own contract
2. Review the controller's return value and ensure it matches the output schema
3. Test locally with `magic test ProcessOrder --validate-output`

E1005 Contract version mismatch HTTP 409

The request targets a specific contract version that does not match the deployed version. This prevents silent breaking changes when clients expect a particular interface.

EXAMPLE RESPONSE


```
{
  "error": {
    "code": "E1005",
    "category": "CONTRACT",
    "message": "Contract version mismatch: requested v1.0, deployed v2.0",
    "request_id": "req_82e6f0",
    "timestamp": "2026-02-12T18:00:00Z",
    "details": {
      "controller": "ProcessOrder",
      "requested_version": "1.0.0",
      "deployed_version": "2.0.0",
      "breaking_changes": true
    }
  }
}
```


Resolution

1. Update your client to target the deployed contract version
2. If you need the old version, check if it is available via versioned routing: `/v1/controllers/ProcessOrder`
3. Review the Releases page for migration guides between versions

---

CAPABILITY Errors (E2xxx)

Capability errors are raised when a controller attempts to access a resource it has not declared in its contract. Magic uses a default-deny model -- everything not explicitly permitted is blocked.

Capability Violations

E2001 – E2005

5 errors

E2001 Capability not declared HTTP 403

The controller attempted to use a capability (database, network, secrets, etc.) that was not declared in its contract. All external access must be explicitly declared.

EXAMPLE RESPONSE


```
{
  "error": {
    "code": "E2001",
    "category": "CAPABILITY",
    "message": "Capability not declared: 'db:write' is not in controller
capabilities",
    "request_id": "req_f14a2c",
    "timestamp": "2026-02-12T18:00:00Z",
    "details": {
      "capability": "db:write",
      "controller": "ReadOnlyReport",
      "declared_capabilities": ["db:read"],
      "action": "db.insert('orders', ...)"
    }
  }
}
```


Resolution

1. Add the missing capability to your contract's `capabilities` array
2. Include appropriate scoping (table allowlist, domain allowlist, etc.)
3. Re-deploy the controller with `magic deploy`

E2002 Resource limit exceeded HTTP 429

The controller exceeded a resource quota defined in its capability scope, such as maximum rows returned, maximum request size, or per-minute call limits.

EXAMPLE RESPONSE


```
{
  "error": {
    "code": "E2002",
    "category": "CAPABILITY",
    "message": "Resource limit exceeded: query returned 15,000 rows (max:
10,000)",
    "request_id": "req_c87d3e",
    "timestamp": "2026-02-12T18:00:00Z",
    "details": {
      "resource": "db:read",
      "limit_type": "max_rows",
      "limit_value": 10000,
      "actual_value": 15000,
      "controller": "BulkExport"
    }
  }
}
```


Resolution

1. Add pagination to your query to stay within row limits
2. If higher limits are needed, update the capability scope in your contract
3. Contact your platform admin if limits are enforced at the organization level

E2003 Network egress denied HTTP 403

The controller attempted to make an outbound HTTP request to a domain not in its declared allowlist. All network egress is blocked by default and must be explicitly permitted per-domain.

EXAMPLE RESPONSE


```
{
  "error": {
    "code": "E2003",
    "category": "CAPABILITY",
    "message": "Network egress denied: 'evil.example.com' not in allowlist",
    "request_id": "req_a4e2b1",
    "timestamp": "2026-02-12T18:00:00Z",
    "details": {
      "destination": "evil.example.com",
      "allowed_domains": ["api.stripe.com", "*.amazonaws.com"],
      "controller": "PaymentProcessor"
    }
  }
}
```


Resolution

1. If the domain is legitimate, add it to the `http:egress.allow` list in your contract
2. Wildcard patterns are supported (e.g., `*.stripe.com`)
3. If this was unexpected, investigate your controller for unintended network calls

E2004 Database table not in allowlist HTTP 403

The controller attempted to query a database table that is not in its declared table allowlist. Each controller can only access the specific tables listed in its contract.

EXAMPLE RESPONSE


```
{
  "error": {
    "code": "E2004",
    "category": "CAPABILITY",
    "message": "Database table not in allowlist: 'users_private' is not
accessible",
    "request_id": "req_91bc4f",
    "timestamp": "2026-02-12T18:00:00Z",
    "details": {
      "table": "users_private",
      "allowed_tables": ["orders", "products", "order_items"],
      "operation": "SELECT",
      "controller": "OrderLookup"
    }
  }
}
```


Resolution

1. Add the table to your contract's `db:read.tables` or `db:write.tables` allowlist
2. Consider principle of least privilege -- does this controller really need access to this table?
3. Re-deploy after updating the contract

E2005 Secret key not authorized HTTP 403

The controller attempted to read a secret that is not listed in its `secrets:read` capability scope. Secrets are strictly scoped per-controller with no cross-controller access.

EXAMPLE RESPONSE

```
{
  "error": {
    "code": "E2005",
    "category": "CAPABILITY",
    "message": "Secret key not authorized: 'DATABASE_URL' is not in allowed
secrets",
    "request_id": "req_e63a8b",
    "timestamp": "2026-02-12T18:00:00Z",
    "details": {
      "secret_key": "DATABASE_URL",
      "allowed_keys": ["STRIPE_SECRET_KEY", "WEBHOOK_SIGNING_KEY"],
      "controller": "PaymentProcessor"
    }
  }
}
```

Resolution

1. Add the secret key to your contract's `secrets:read.keys` allowlist
2. Ensure the secret exists in the runtime's secret store
3. Review access: controllers should only access secrets they genuinely need

---

RUNTIME Errors (E3xxx)

Runtime errors occur during controller execution when resource limits are hit or the sandbox environment fails to initialize properly.

Execution Failures

E3001 – E3005

5 errors

E3001 Controller timeout exceeded HTTP 504

The controller did not complete within its declared timeout limit. The process is terminated with SIGKILL after the deadline. Default timeout is 30 seconds; maximum configurable is 300 seconds.

EXAMPLE RESPONSE

```
{
  "error": {
    "code": "E3001",
    "category": "RUNTIME",
    "message": "Controller timeout exceeded: execution killed after 30s",
    "request_id": "req_4b7c9a",
    "timestamp": "2026-02-12T18:00:00Z",
    "details": {
      "controller": "DataAggregator",
      "timeout_seconds": 30,
      "elapsed_seconds": 30.002,
      "signal": "SIGKILL"
    }
  }
}
```

Resolution

1. Optimize controller logic -- reduce database queries, add caching, avoid unnecessary computation
2. Increase the timeout in your contract's `resources.timeout` field (max 300s)
3. Consider breaking long-running work into smaller async steps

E3002 Memory limit exceeded HTTP 503

The controller process was OOM-killed by the cgroup memory enforcer. Default limit is 256Mi; maximum configurable is 2Gi.

EXAMPLE RESPONSE


```
{
  "error": {
    "code": "E3002",
    "category": "RUNTIME",
    "message": "Memory limit exceeded: process killed (OOM) at 256Mi",
    "request_id": "req_d3f1e8",
    "timestamp": "2026-02-12T18:00:00Z",
    "details": {
      "controller": "ImageProcessor",
      "memory_limit": "256Mi",
      "peak_usage": "256Mi",
      "enforcement": "cgroup_oom_kill"
    }
  }
}
```


Resolution

1. Profile your controller's memory usage locally
2. Reduce in-memory data (stream instead of buffering, use pagination)
3. Increase the memory limit in your contract's `resources.memory` field (max 2Gi)

E3003 CPU limit exceeded HTTP 503

The controller was throttled or killed after exceeding its CPU allocation for a sustained period. This protects other controllers sharing the same host.

EXAMPLE RESPONSE


```
{
  "error": {
    "code": "E3003",
    "category": "RUNTIME",
    "message": "CPU limit exceeded: throttled at 0.5 cores for 10s",
    "request_id": "req_7a2e9c",
    "timestamp": "2026-02-12T18:00:00Z",
    "details": {
      "controller": "CryptoHasher",
      "cpu_limit": "0.5 cores",
      "throttle_duration_ms": 10000,
      "enforcement": "cgroup_cpu_max"
    }
  }
}
```


Resolution

1. Optimize CPU-intensive operations (algorithmic improvements, avoiding busy loops)
2. Increase the CPU limit in your contract's `resources.cpu` field (max 2.0 cores)
3. Offload heavy computation to external services via `http:egress`

E3004 Sandbox initialization failed HTTP 500

The runtime failed to create the isolated execution environment for the controller. This is typically an infrastructure issue, not a controller bug.

EXAMPLE RESPONSE


```
{
  "error": {
    "code": "E3004",
    "category": "RUNTIME",
    "message": "Sandbox initialization failed: unable to create cgroup",
    "request_id": "req_b5c8d2",
    "timestamp": "2026-02-12T18:00:00Z",
    "details": {
      "stage": "cgroup_creation",
      "reason": "cgroup hierarchy not available",
      "host": "worker-03"
    }
  }
}
```


Resolution

1. This is an infrastructure issue -- check that the host supports cgroups v2
2. Verify the runtime has sufficient permissions to create process namespaces
3. Retry the request; if persistent, check `magic health` for system status

E3005 Controller not found HTTP 404

No controller is registered with the given name, or the controller has been undeployed. The request cannot be routed.

EXAMPLE RESPONSE


```
{
  "error": {
    "code": "E3005",
    "category": "RUNTIME",
    "message": "Controller not found: 'SendInvoice' is not deployed",
    "request_id": "req_1f9a7e",
    "timestamp": "2026-02-12T18:00:00Z",
    "details": {
      "controller": "SendInvoice",
      "available_controllers": ["ProcessOrder", "LookupCustomer",
"GenerateReport"]
    }
  }
}
```


Resolution

1. Verify the controller name is spelled correctly (names are case-sensitive)
2. Check deployed controllers with `magic list`
3. If the controller was recently deployed, it may still be initializing -- wait a few seconds and retry

---

DEPLOYMENT Errors (E4xxx)

Deployment errors occur during controller lifecycle operations: deploy, hot-reload, rollback, and registry management.

Deploy & Lifecycle Failures

E4001 – E4005

5 errors

E4001 Deployment validation failed HTTP 400

The deployment package failed pre-flight validation. This includes contract schema validation, dependency checks, and capability verification. The deployment is rejected before any changes are made.

EXAMPLE RESPONSE

```
{
  "error": {
    "code": "E4001",
    "category": "DEPLOYMENT",
    "message": "Deployment validation failed: controller entrypoint not
found",
    "request_id": "req_8c4f2a",
    "timestamp": "2026-02-12T18:00:00Z",
    "details": {
      "controller": "NewFeature",
      "validation_errors": [
        "entrypoint 'handler.py' not found in package",
        "dependency 'numpy' not in approved list"
      ]
    }
  }
}
```

Resolution

1. Review all validation errors in `details.validation_errors`
2. Ensure the entrypoint file exists and matches the contract declaration
3. Run `magic validate --pre-deploy ./controller/` locally before deploying

E4002 Controller already exists HTTP 409

A controller with the same name and version is already deployed. To update, either increment the version or use the `--force` flag to replace.

EXAMPLE RESPONSE


```
{
  "error": {
    "code": "E4002",
    "category": "DEPLOYMENT",
    "message": "Controller already exists: 'ProcessOrder@2.0.0' is deployed",
    "request_id": "req_5d3b7c",
    "timestamp": "2026-02-12T18:00:00Z",
    "details": {
      "controller": "ProcessOrder",
      "version": "2.0.0",
      "deployed_at": "2026-02-10T14:30:00Z",
      "deployed_by": "admin@company.com"
    }
  }
}
```


Resolution

1. Increment the version number in your contract if this is a new release
2. Use `magic deploy --force` to replace the existing version (caution: no automatic rollback)
3. Use `magic deploy --hot-reload` for zero-downtime updates to the same version

E4003 Hot-reload failed HTTP 500

The runtime could not hot-swap the controller code while maintaining availability. The previous version remains active and continues serving requests.

EXAMPLE RESPONSE


```
{
  "error": {
    "code": "E4003",
    "category": "DEPLOYMENT",
    "message": "Hot-reload failed: health check did not pass within 15s",
    "request_id": "req_2e8f4d",
    "timestamp": "2026-02-12T18:00:00Z",
    "details": {
      "controller": "ProcessOrder",
      "previous_version": "1.9.0",
      "target_version": "2.0.0",
      "reason": "health_check_timeout",
      "rollback_status": "automatic_rollback_complete"
    }
  }
}
```


Resolution

1. The previous version is still running -- no downtime occurred
2. Check your controller's initialization logic for errors or slow startup
3. Review logs with `magic logs ProcessOrder --since 5m` for details

E4004 Rollback target not found HTTP 404

The requested rollback version does not exist in the deployment history. The runtime retains a configurable number of previous versions (default: 5).

EXAMPLE RESPONSE


```
{
  "error": {
    "code": "E4004",
    "category": "DEPLOYMENT",
    "message": "Rollback target not found: version '1.0.0' not in history",
    "request_id": "req_6a1c3e",
    "timestamp": "2026-02-12T18:00:00Z",
    "details": {
      "controller": "ProcessOrder",
      "requested_version": "1.0.0",
      "available_versions": ["2.0.0", "1.9.0", "1.8.0", "1.7.0", "1.6.0"],
      "retention_policy": 5
    }
  }
}
```


Resolution

1. Check available rollback targets in `details.available_versions`
2. Use `magic history ProcessOrder` to see the full deployment timeline
3. If the version was purged, re-deploy from source

E4005 Registry full HTTP 507

The controller registry has reached its maximum capacity. No new controllers can be deployed until existing ones are removed or the registry limit is increased.

EXAMPLE RESPONSE


```
{
  "error": {
    "code": "E4005",
    "category": "DEPLOYMENT",
    "message": "Registry full: 100/100 controller slots used",
    "request_id": "req_9b5d1f",
    "timestamp": "2026-02-12T18:00:00Z",
    "details": {
      "current_count": 100,
      "max_count": 100,
      "plan": "standard",
      "upgrade_url": "https://magic.threadsync.io/enterprise.html"
    }
  }
}
```


Resolution

1. Remove unused controllers with `magic undeploy ControllerName`
2. Review deployed controllers with `magic list --sort-by=last-invoked` to find stale entries
3. Upgrade to Enterprise for higher registry limits

AUTH Errors (E5xxx)

Authentication and authorization errors occur when API credentials are invalid, expired, or lack sufficient permissions for the requested operation.

Authentication & Authorization

E5001 – E5005

5 errors

E5001 Invalid API key HTTP 401

The provided API key is not recognized. It may have been revoked, never existed, or is malformed. The key is not logged for security reasons.

EXAMPLE RESPONSE


```
{
  "error": {
    "code": "E5001",
    "category": "AUTH",
    "message": "Invalid API key: the provided key is not recognized",
    "request_id": "req_c2a4e6",
    "timestamp": "2026-02-12T18:00:00Z",
    "details": {
      "key_prefix": "mk_live_7f2a...",
      "auth_method": "api_key",
      "header": "X-API-Key"
    }
  }
}
```


Resolution

1. Verify the API key is correct and has not been rotated
2. Ensure you are using the correct header: `X-API-Key`
3. Generate a new key from the admin dashboard if the current key was revoked

E5002 Token expired HTTP 401

The JWT bearer token has expired. Tokens have a configurable TTL (default: 1 hour). The client must refresh the token or re-authenticate.

EXAMPLE RESPONSE


```
{
  "error": {
    "code": "E5002",
    "category": "AUTH",
    "message": "Token expired: JWT expired 15 minutes ago",
    "request_id": "req_a8d3f1",
    "timestamp": "2026-02-12T18:00:00Z",
    "details": {
      "expired_at": "2026-02-12T17:45:00Z",
      "token_type": "bearer",
      "issuer": "magic.threadsync.io"
    }
  }
}
```


Resolution

1. Refresh the token using your refresh token endpoint
2. Implement automatic token refresh in your client before expiry
3. If using SSO, re-authenticate through your identity provider

E5003 Insufficient permissions HTTP 403

The authenticated user or service account does not have the required permission for this operation. Administrative actions (deploy, rollback, config changes) require elevated roles.

EXAMPLE RESPONSE


```
{
  "error": {
    "code": "E5003",
    "category": "AUTH",
    "message": "Insufficient permissions: 'deploy' requires 'admin' role",
    "request_id": "req_f7b2c4",
    "timestamp": "2026-02-12T18:00:00Z",
    "details": {
      "action": "deploy",
      "required_role": "admin",
      "current_role": "viewer",
      "user": "dev@company.com"
    }
  }
}
```


Resolution

1. Request the required role from your organization admin
2. Check your current permissions with `magic whoami`
3. Use a service account with appropriate permissions for CI/CD pipelines

E5004 Rate limit exceeded HTTP 429

The caller has exceeded the rate limit for the API. Rate limits are applied per API key and vary by plan. The response includes a `Retry-After` header indicating when to retry.

EXAMPLE RESPONSE

```
{
  "error": {
    "code": "E5004",
    "category": "AUTH",
    "message": "Rate limit exceeded: 1000 requests/minute limit reached",
    "request_id": "req_3e9a1b",
    "timestamp": "2026-02-12T18:00:00Z",
    "details": {
      "limit": 1000,
      "window": "1m",
      "current": 1001,
      "retry_after_seconds": 42,
      "plan": "standard"
    }
  }
}
```

Resolution

1. Respect the `Retry-After` header and implement exponential backoff
2. Add client-side rate limiting to stay within your quota
3. Upgrade to Enterprise for higher rate limits

E5005 IP not allowlisted HTTP 403

The request originates from an IP address that is not in the organization's IP allowlist. When IP allowlisting is enabled, only requests from approved CIDRs are accepted.

EXAMPLE RESPONSE


```
{
  "error": {
    "code": "E5005",
    "category": "AUTH",
    "message": "IP not allowlisted: '203.0.113.42' is not in approved ranges",
    "request_id": "req_8d2f6a",
    "timestamp": "2026-02-12T18:00:00Z",
    "details": {
      "source_ip": "203.0.113.42",
      "allowed_ranges": ["10.0.0.0/8", "192.168.1.0/24", "198.51.100.0/24"],
      "policy": "organization"
    }
  }
}
```


Resolution

1. Add the source IP or CIDR range to your organization's IP allowlist
2. If connecting from a VPN or proxy, ensure the egress IP is allowlisted
3. Contact your organization admin to update the IP policy