AI Integration Guide

EdgarTools is designed from the ground up to work seamlessly with AI agents and Large Language Models (LLMs). This guide covers all AI-native features and best practices for integrating EdgarTools with your AI applications.

Table of Contents

Overview

EdgarTools provides three levels of AI integration:

1. **Interactive Documentation**: Rich, searchable docs via `.docs` property
2. **Token-Efficient Text Export**: AI-optimized context via `.text()` methods
3. **Specialized Skills**: Curated workflows for AI agents via Skills system
4. **MCP Integration**: Native support for Claude Desktop and MCP clients

All AI features are **optional dependencies**:

```
# Install with AI features
pip install edgartools[ai]

# Core EdgarTools works without AI dependencies
pip install edgartools
```

latest

Interactive Documentation (.docs Property)

Every major EdgarTools object includes comprehensive API documentation accessible via the `.docs` property.

Basic Usage

```
from edgar import Company

company = Company("AAPL")

# Display rich documentation in terminal
company.docs
```

This displays beautifully formatted documentation with: - Complete API reference (methods, parameters, return types) - Usage examples - Best practices - Related concepts

Searching Documentation

Use BM25 semantic search to find relevant information instantly:

```
# Search for specific functionality
results = company.docs.search("get financials")

# Search returns ranked results
for result in results:
    print(result)
```

Available Documentation

Documentation is available on these objects:

Object	Documentation Size	Coverage
`Company`	~1,070 lines	Complete class reference
`EntityFiling`	~557 lines	Filing access methods
`EntityFilings`	~671 lines	Collection operations
`XBRL`	~587 lines	XBRL parsing and statements
`Statement`	~567 lines	Financial state

Total: 3,450+ lines of comprehensive API documentation

Documentation Structure

Each documentation file includes:

```
# Class Overview
Brief description and purpose

## Key Concepts
Important concepts and terminology

## Basic Usage
Quick start examples

## Methods
### method_name(param1, param2)
Description, parameters, returns, examples

## Properties
### property_name
Description, return type, examples

## Advanced Usage
Complex scenarios and patterns

## Common Patterns
Frequently used workflows

## Examples
Real-world usage examples
```

AI-Optimized Text Output (.text() Methods)

The `.text()` method provides token-efficient context optimized for LLM consumption.

Basic Usage

```
from edgar import Company

company = Company("AAPL")

# Get AI-optimized context output
context = company.to_context()
print(context)
```

> **Note**: The older `company.text()` method is deprecated. Use `company.to_context()` for consistent naming across all EdgarTools classes.

Detail Levels

Three progressive disclosure levels for different use cases:

```
# Minimal - Just the essentials (~100-200 tokens)
minimal = company.to_context(detail='minimal')

# Standard - Balanced overview (~300-500 tokens)
standard = company.to_context(detail='standard')

# Detailed - Comprehensive information (~800-1200 tokens)
detailed = company.to_context(detail='detailed')
```

Token Limiting

Control output size for LLM context windows:

```
# Limit to specific token count
context = company.to_context(max_tokens=500)

# Automatically truncates while preserving structure
```

Output Format: Markdown-KV

EdgarTools uses **markdown-kv format** (research-backed as optimal for LLM comprehension):

```
## Company Information
name: Apple Inc.
ticker: AAPL
cik: 0000320193
sic: 3571 - Electronic Computers

## Recent Financials
revenue_fy2023: $383,285,000,000
net_income_fy2023: $96,995,000,000
```

Why markdown-kv? - Research shows it's most effective for LLM understanding - Combines human readability with machine parseability - Preserves hierarchical structure - Token-efficient compared to JSON or prose

Available Text Methods

```
# Company context
company_text = company.to_context(detail='standard')

# Filing context
filing = company.get_filings(form="10-K").latest()
filing_text = filing.text(detail='standard')

# XBRL context
xbrl = filing.xbrl()
```

```
xbrl_text = xbrl.to_context(detail='standard')

# Statement context
income = xbrl.statements.income_statement()
statement_text = income.text(detail='standard')
```

AI Skills System

Skills package documentation and helper functions for specialized SEC analysis.

Listing Available Skills

```
from edgar.ai import list_skills

# Get all available skills
skills = list_skills()
print(skills)
# [EdgarToolsSkill(name='EdgarTools')]
```

Getting a Specific Skill

```
from edgar.ai import get_skill

skill = get_skill("EdgarTools")
print(skill)
# Skill: EdgarTools
# Description: Query and analyze SEC filings and financial statements...
# Documents: 4
# Helper Functions: 5
```

Using Helper Functions

Skills provide pre-built workflow wrappers:

```
# Get helper functions from skill
helpers = skill.get_helpers()

# Use helper function for common workflow
get_revenue_trend = helpers['get_revenue_trend']
income = get_revenue_trend("AAPL", periods=3)
print(income)
```

Available Helper Functions

latest

The EdgarTools skill includes:

```
from edgar.ai.helpers import (
    get_filings_by_period,      # Get published filings for specific quarter
    get_today_filings,           # Get recent filings (last ~24 hours)
    get_revenue_trend,           # Get multi-period income statement
    get_filing_statement,        # Get statement from specific filing
    compare_companies_revenue,   # Compare revenue across companies
)

# Get filings for Q1 2023
filings = get_filings_by_period(2023, 1, form="10-K")

# Get today's filings
current = get_today_filings()

# Get 3-year revenue trend
income = get_revenue_trend("AAPL", periods=3)

# Get specific statement
balance = get_filing_statement("AAPL", 2023, "10-K", "balance")

# Compare companies
comparison = compare_companies_revenue(["AAPL", "MSFT", "GOOGL"], periods=3)
```

Skill Documentation

Access skill documentation:

```
# List available documents
docs = skill.get_documents()
print(docs)
# ['skill', 'objects', 'workflows', 'readme']

# Get specific document content
skill_content = skill.get_document_content("skill")
print(skill_content)
```

Exporting Skills

Export skills for use with AI tools like Claude Desktop and Claude Code.

Simple API (Recommended)

```
from edgar.ai import install_skill, package_skill

# Install to ~/.claude/skills/ for automatic discovery
install_skill()
# Output: /Users/username/.claude/skills/edgartools

# Create ZIP for Claude Desktop upload
package_skill()
# Output: edgartools.zip (ready to upload)
```

```
# Custom locations
install_skill(to="~/my-skills")
package_skill(output="~/Desktop")
```

Advanced API

```
from edgar.ai import export_skill, edgartools_skill

# Official Claude Skills format (installs to ~/.claude/skills/)
path = export_skill(
    edgartools_skill,
    format="claude-skills"  # Default format
)

# Claude Desktop upload format (creates ZIP with SKILL.md)
zip_path = export_skill(
    edgartools_skill,
    format="claude-desktop"
)

# Export to custom location
path = export_skill(
    edgartools_skill,
    format="claude-skills",
    output_dir="~/my-skills",
    install=False
)
```

Export Formats

claude-skills (Official Anthropic Format): - Installs to `~/.claude/skills/` by default - Main file: `SKILL.md` (uppercase, per Anthropic spec) - Includes all supporting markdown files - Includes API reference documentation in `api-reference/` directory - YAML frontmatter with name and description - Ready for Claude Desktop and Claude Code

claude-desktop (Claude Desktop Upload Format): - Creates ZIP file by default (required by Claude Desktop upload UI) - Main file: `SKILL.md` (uppercase, per Anthropic spec) - All supporting files and API reference included - Ready for upload via Claude Desktop's skill upload interface - Can export as directory with `create_zip=False`

Exported skills include: - Tutorial documentation (skill.md/SKILL.md, workflows.md, objects.md) - API reference documentation (api-reference/ directory) - YAML frontmatter for tool integration - Helper function documentation

Model Context Protocol (MCP) Server

Run EdgarTools as an MCP server for Claude Desktop and other MCP c

latest

Installation

```
pip install edgartools[ai]
```

Starting the Server

```
# Start MCP server
python -m edgar.ai

# The server will start and wait for MCP client connections
```

Claude Desktop Configuration

Add to `~/Library/Application Support/Claude/claude_desktop_config.json`:

```
{
  "mcpServers": {
    "edgartools": {
      "command": "python",
      "args": ["-m", "edgar.ai"],
      "env": {
        "EDGAR_IDENTITY": "Your Name your.email@example.com"
      }
    }
  }
}
```

Important: The `EDGAR_IDENTITY` environment variable is required by SEC regulations.

Available MCP Tools

Once configured, Claude Desktop can use these tools:

1. **edgar_company_research** - Get comprehensive company intelligence
2. Company profile and metadata
3. 3-year financial trends
4. Last 5 filings
5. Recent activity summary
6. **edgar_analyze_financials** - Detailed financial statement analysis
7. Income statement, balance sheet, or cash flow
8. Annual or quarterly periods
9. Multi-period trend analysis

Example Prompts

After configuring the MCP server, try these prompts in Claude Desktop:

```
"Research Apple Inc with financials"

"Analyze Tesla's revenue trends over the last 4 quarters"

"Compare Microsoft and Google's cash flow statements"

"Get Nvidia's balance sheet trends for the past 3 years"

"Show me today's 10-K filings"
```

Other MCP Clients

EdgarTools MCP server works with any MCP-compatible client:

- **Claude Desktop** (MacOS/Windows)
- **Cline** (VS Code extension)
- **Continue.dev** (IDE integration)
- Any other MCP v1.0 compatible client

See MCP Quickstart for complete setup instructions.

Helper Functions

Pre-built functions for common SEC analysis workflows.

Filing Access Helpers

```
from edgar.ai.helpers import get_filings_by_period, get_today_filings

# Get published filings for specific period
filings = get_filings_by_period(
    year=2023,
    quarter=1,
    form="10-K",
    filing_date=None  # Optional: filter by specific date
)

# Get current filings (last ~24 hours)
current = get_today_filings()
print(f"Found {len(current)} recent filings")
```

latest

Financial Analysis Helpers

```
from edgar.ai.helpers import (
    get_revenue_trend,
    get_filing_statement,
    compare_companies_revenue
)

# Get revenue trend analysis
income = get_revenue_trend(
    ticker="AAPL",
    periods=3,
    quarterly=False  # Set True for quarterly data
)

# Get specific statement from filing
balance = get_filing_statement(
    ticker="AAPL",
    year=2023,
    form="10-K",
    statement_type="balance"  # or "income", "cash_flow"
)

# Compare companies
comparison = compare_companies_revenue(
    tickers=["AAPL", "MSFT", "GOOGL"],
    periods=3
)
for ticker, income in comparison.items():
    print(f"\n{ticker} Revenue Trend:")
    print(income)
```

Best Practices

1. Choose the Right Tool for the Job

```
# For interactive learning -> Use .docs
company.docs.search("get filings")

# For AI context -> Use .text()
context = company.to_context(detail='standard')

# For specialized workflows -> Use helpers
from edgar.ai.helpers import get_revenue_trend
income = get_revenue_trend("AAPL", periods=3)

# For AI agents -> Use MCP server or Skills
```

2. Use Progressive Disclosure


```
# Start minimal, add detail as needed
overview = company.to_context(detail='minimal')  # Quick overview
```

```
# Need more context?
standard = company.to_context(detail='standard')  # Balanced view

# Need everything?
detailed = company.to_context(detail='detailed')  # Comprehensive
```

3. Respect Token Limits

```
# Always specify max_tokens for LLM context
context = company.to_context(max_tokens=500)

# For multiple objects, budget tokens:
company_context = company.to_context(max_tokens=300)
filing_context = filing.text(max_tokens=200)
total_context = company_context + "\n\n" + filing_context
# Total: ~500 tokens
```

4. Cache Expensive Operations

```
# Cache company objects
from functools import lru_cache

@lru_cache(maxsize=100)
def get_company(ticker: str):
    return Company(ticker)

# Reuse cached company
apple = get_company("AAPL")
```

5. Use Entity Facts API for Multi-Period Data

```
# More efficient than fetching multiple filings
company = Company("AAPL")
income = company.income_statement(periods=3)
# Single API call, ~500 tokens

# Less efficient:
# filing1 = company.get_filings(form="10-K")[0]
# filing2 = company.get_filings(form="10-K")[1]
# filing3 = company.get_filings(form="10-K")[2]
# 3 API calls, ~3,750 tokens
```

Token Optimization

Understanding Token Costs

latest

Approximate token sizes for common operations:

Operation	Tokens (approx)
`Company.text(detail='minimal')`	100-200
`Company.text(detail='standard')`	300-500
`Company.text(detail='detailed')`	800-1200
`Filing.text(detail='standard')`	200-400
`Statement.to_dataframe()` (3 periods)	500-1000
`XBRL.text(detail='standard')`	400-600

Optimization Strategies

1. Use .head() to limit collections:

```
# Limit filing output
filings = company.get_filings(form="10-K")
print(filings.head(5))  # Only show 5 filings
```

2. Prefer Entity Facts API:

```
# Efficient: Single API call
income = company.income_statement(periods=3)  # ~500 tokens

# Less efficient: Multiple filing accesses
# ~3,750 tokens for 3 separate filings
```

3. Filter before displaying:

```
# Filter first
tech_companies = filings.filter(ticker=["AAPL", "MSFT", "GOOGL"])
print(tech_companies)  # Much smaller output
```

4. Use specific statements vs full XBRL:

```
# Efficient
income = xbrl.statements.income_statement()  # ~1,250 tokens

# Less efficient
print(xbrl)  # Full XBRL object, ~2,500 tokens
```

5. Control detail levels:

```
# For overview tasks
overview = company.to_context(detail='minimal')

# For detailed analysis
analysis = company.to_context(detail='detailed')
```

Token Counting

```
from edgar.ai.core import TokenOptimizer

# Estimate tokens before sending to LLM
content = company.to_context()
token_count = TokenOptimizer.estimate_tokens(content)
print(f"Estimated tokens: {token_count}")

# Optimize for specific token limit
optimized = TokenOptimizer.optimize_for_tokens(
    content,
    max_tokens=1000
)
```

Examples

Example 1: Building LLM Context

```
from edgar import Company

# Get company
company = Company("AAPL")

# Build multi-level context
context_parts = []

# 1. Company overview
context_parts.append("# Company Overview")
context_parts.append(company.to_context(detail='minimal', max_tokens=200))

# 2. Latest filing
filing = company.get_filings(form="10-K").latest()
context_parts.append("\n# Latest 10-K Filing")
context_parts.append(filing.text(detail='standard', max_tokens=300))

# 3. Financial statements
xbrl = filing.xbrl()
income = xbrl.statements.income_statement()
context_parts.append("\n# Income Statement")
context_parts.append(income.text(max_tokens=500))

# Combine for LLM
```

```
llm_context = "\n".join(context_parts)
print(f"Total context: ~{len(llm_context.split())*1.3:.0f} tokens")
```

Example 2: Interactive Documentation Assistant

```
from edgar import Company

company = Company("AAPL")

# Search for relevant documentation
query = "how do I get historical financials"
results = company.docs.search(query)

# Display top results
print(f"Search results for: {query}\n")
for i, result in enumerate(results[:3], 1):
    print(f"{i}. {result}")
```

Example 3: Batch Processing with Token Budget

```
from edgar import get_filings

# Get filings
filings = get_filings(2023, 1, form="10-K")

# Process with token budget
token_budget = 5000
tokens_used = 0

results = []
for filing in filings:
    # Check token budget
    filing_text = filing.text(detail='minimal')
    estimated_tokens = len(filing_text.split()) * 1.3

    if tokens_used + estimated_tokens > token_budget:
        break

    results.append(filing_text)
    tokens_used += estimated_tokens

print(f"Processed {len(results)} filings using ~{tokens_used:.0f} tokens")
```

Troubleshooting

Issue: "AI features not available"

latest

Solution: Install AI dependencies:

```
pip install edgartools[ai]
```

Issue: MCP server won't start

Solution: Check that EDGAR_IDENTITY is set:

```
export EDGAR_IDENTITY="Your Name your.email@example.com"
python -m edgar.ai
```

Issue: Documentation not displaying

Solution: Documentation requires optional AI dependencies. Install with:

```
pip install edgartools[ai]
```

Issue: Token counts seem high

Solution: Use lower detail levels or max_tokens:

```
# Instead of:
text = company.to_context(detail='detailed')

# Try:
text = company.to_context(detail='standard', max_tokens=500)
```

Additional Resources

- EdgarTools Documentation
- MCP Quickstart Guide
- AI Skills README
- GitHub Issues
- GitHub Discussions

Contributing

We welcome contributions to improve AI integration:

- Suggest new helper functions
- Create specialized skills for external packages
- Improve documentation

- Report integration issues

See Contributing Guide for details.

Need Help? Open an issue on GitHub or start a discussion.