

# Sub-Functions & Command Chaining Architecture

## Command Sub-Structure Within Functions













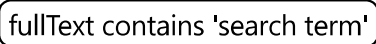
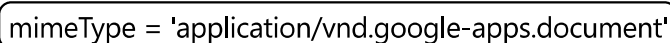
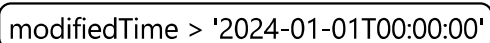
### 1. Command Parameters

```
json
{
  "command": "create" | "update" | "rewrite",
  "type": "application/vnd.ant.react" | "text/html" | "text/markdown" | "application/vnd.ant.code",
  "language": "python" | "javascript" | "sql" | "html" | etc.,
  "id": "unique_identifier",
  "title": "Display Name",
  "content": "Full content",
  "old_str": "Text to replace (update only)",
  "new_str": "Replacement text (update only)"
}
```

### 2. Query Sub-Commands

```
json
{
  "api_query": "name contains 'project' and fullText contains 'AI'",
  "semantic_query": "artificial intelligence research documents",
  "order_by": "createdTime desc" | "relevance desc" | "modifiedTime desc",
  "page_size": 10,
  "request_page_token": false
}
```

#### Query Operators:

-  contains,  =,  !=,  <,  <=,  >,  >=,  in,  and,  or,  not,  has
-  fullText contains 'search term'
-  mimeType = 'application/vnd.google-apps.document'
-  modifiedTime > '2024-01-01T00:00:00'

### 3. Multi-Operation Execution

```
javascript
```

```
// Multiple sub-operations in single call
import * as math from 'mathjs';
import Papa from 'papaparse';

// File operations
const data = await window.fs.readFile('data.csv', { encoding: 'utf8' });

// Data processing
const parsed = Papa.parse(data, { header: true, dynamicTyping: true });

// Analysis operations
const results = math.evaluate('sqrt(16) + log(10)');

// Output operations
console.log("Processing complete:", results);
```

## Function Chaining Patterns

### Sequential Workflow Pattern

Search → Analyze → Create  
 web\_search → repl → artifacts

#### Example Implementation:

1. **Research Phase:** `web_search("AI architecture patterns")`
2. **Analysis Phase:** `repl(process_search_results())`
3. **Creation Phase:** `artifacts(create_productivity_tool())`

### Iterative Refinement Pattern

Search → Fetch → Analyze → Update → Repeat  
 web\_search → web\_fetch → repl → artifacts(update) → web\_search

### Multi-Source Aggregation Pattern

Drive Search + Web Search → Analysis → Synthesis  
 google\_drive\_search + web\_search → repl → artifacts

## Advanced Chaining Examples

## Research-to-Product Pipeline

yaml

### Step 1: Information Gathering

- **web\_search**: "productivity tools AI"
- **google\_drive\_search**: "project requirements"
- **web\_fetch**: specific\_research\_papers

### Step 2: Data Processing

- **repl**: analyze\_search\_results()
- **repl**: extract\_patterns()
- **repl**: generate\_recommendations()

### Step 3: Product Development

- **artifacts**: create\_react\_component()
- **artifacts**: update\_with\_features()
- **artifacts**: rewrite\_for\_optimization()

## Content Analysis Workflow

yaml

### Discovery Phase:

- **google\_drive\_search**: "fullText contains 'architecture'"
- **web\_search**: "current AI trends 2025"

### Processing Phase:

- **repl**: parse\_documents()
- **repl**: sentiment\_analysis()
- **repl**: extract\_key\_themes()

### Synthesis Phase:

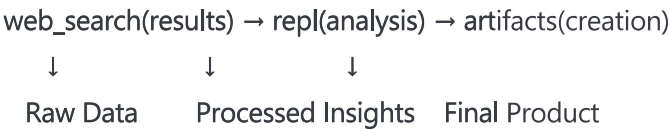
- **artifacts**: create\_summary\_report()
- **artifacts**: update\_recommendations()

## Context Preservation Across Chains

### State Management in Chaining

- **Search Results** → Stored in function\_results
- **Analysis Outputs** → Logged in repl console
- **Artifact State** → Maintained across updates

# Information Flow



## Productivity Tool Implementation Pattern

Based on the analyzed AI architecture patterns, here's the recommended function chaining:

### Phase 1: Foundation (RAG + Prompt Engineering)

```
yaml
Research:
  - web_search: "RAG implementation patterns"
  - google_drive_search: "existing knowledge base"

Analysis:
  - repl: analyze_content_structure()
  - repl: optimize_retrieval_strategy()

Development:
  - artifacts: create_rag_interface()
  - artifacts: update_prompt_templates()
```

### Phase 2: Orchestration (Multi-Agent Coordination)

```
yaml
Discovery:
  - web_search: "multi-agent AI systems"
  - web_fetch: specific_framework_docs

Implementation:
  - repl: design_agent_workflows()
  - artifacts: create_orchestration_dashboard()
```

### Phase 3: Scaling (Federated Learning)

```
yaml
```

#### Research:

- `web_search`: "federated learning productivity"
- `google_drive_search`: "team collaboration patterns"

#### Deployment:

- `repl`: `calculate_scaling_metrics()`
- `artifacts`: `create_federated_interface()`

## Available Sub-Function Libraries

### In `repl` Environment:

- **Data Processing:** `papaparse`, `sheetjs`, `lodash`
- **Math/Analysis:** `mathjs`, `d3`
- **Visualization:** `plotly`, `chart.js`
- **File Access:** `window.fs.readFile()`

### In `artifacts` Environment:

- **React Components:** `useState`, `useEffect`, `lucide-react`
- **Charts:** `recharts`
- **Styling:** `tailwindcss` utility classes
- **3D Graphics:** `three.js` (r128)

## Command Chaining Best Practices

### Efficiency Patterns

1. **Combine Related Operations** in single function calls
2. **Use Semantic Queries** for better search results
3. **Process Data in Chunks** for large datasets
4. **Cache Results** in variables for reuse

### Error Handling in Chains

javascript

```
// In repl - robust chaining
try {
  const searchData = await processSearchResults();
  const analysis = await analyzeData(searchData);
  console.log("Chain completed:", analysis);
} catch (error) {
  console.error("Chain failed at:", error.step);
}
```

## Performance Optimization

- **Parallel Processing:** Run independent searches simultaneously
- **Selective Fetching:** Only fetch full content when needed
- **Incremental Updates:** Use `artifacts(update)` vs `rewrite`

## Real Example: This Document's Creation

This document itself demonstrates function chaining:

1. `web_search` → Found AI architecture patterns
2. `repl` → Analyzed 16 patterns across 3 categories
3. `artifacts` → Created this comprehensive guide

### Function Chain Used:

```
web_search("AI architecture patterns")
  → repl(analyze_patterns())
  → artifacts(create_guide())
```

The sub-functions enabled complex orchestration while maintaining modularity and reusability across the entire workflow.

---

**Summary:** Sub-functions provide granular control within each tool, while chaining patterns enable sophisticated workflows that transform raw information into actionable productivity solutions.