IDE Agent Wizard - Comprehensive Guide

IDE Agent Wizard Team

2026-02-23

- IDE Agent Wizard - Comprehensive Guide
  - Table of Contents
  - Overview
    - Use Cases
  - Architecture
    - System Overview
    - Data Flow
  - Setup Options
    - Flowchart: Setup Decision
    - Option Details
  - Configuration Management
    - New Configuration
    - Managing Existing Configuration
  - Memory System
    - Architecture
    - Database Schema
    - How Retrieval Works
    - Memory Types
    - Memory Synchronization
    - Memory Commands
  - Telegram Integration
    - Architecture
    - Security Model
    - Configuration
    - User Authorization
  - IDE Integration
    - Supported IDEs
    - How It Works
    - Context Files
  - Templates
    - Available Templates
    - Template Structure
    - Customizing Templates
  - Providers
    - Supported Providers
    - Configuration
    - Switching Providers
  - Docker Services
    - Services
    - File Mounts
    - Commands
    - Kimi Agent Patch
  - Security
    - Data Protection
    - Best Practices
    - PII Protection
  - Troubleshooting
    - Setup Issues
    - Telegram Issues
    - Memory Issues
    - Configuration Issues
  - API Reference
    - IDE Connector
    - Memory Store
  - Appendix
    - File Structure Reference
    - Environment Variables
    - Configuration Schema

IDE Agent Wizard - Comprehensive Guide

Version: v1.1.0-RC1 | **Date:** 2026-02-23

Table of Contents

1. Overview
2. Architecture
3. Setup Options
4. Configuration Management
5. Memory System

6. Telegram Integration
7. IDE Integration
8. Templates
9. Providers
10. Docker Services
11. Security
12. Troubleshooting
13. API Reference

Overview

IDE Agent Wizard is a universal AI agent framework that works with **any IDE** and **any LLM provider**. It provides:

- **Multi-Provider Support** - Kimi, Anthropic, OpenAI, OpenRouter
- **IDE Agnostic** - Works with Kimi Code, Claude Code, Cursor, etc.
- **Telegram Bot** - Full Telegram integration with memory sync
- **Persistent Memory** - SQLite-based with automatic context retrieval
- **Smart Configuration** - Add/remove features without recreating

Use Cases

Scenario	Recommended Setup
IDE coding assistant	IDE only
Mobile + desktop access	IDE + Telegram
Team shared agent	IDE + Telegram (restricted user)
Quick experimentation	IDE only → add Telegram later

Architecture

System Overview

```
┌─────────────────────────────────────────────────────────────────┐
│                       USER INTERFACES                           │
├─────────────────┬─────────────────┬─────────────────────────────┤
│   Kimi Code     │  Claude Code    │     Telegram App            │
│   Cursor        │  VS Code +      │     (Mobile/Desktop)        │
│   Any IDE       │  Continue.dev   │                             │
└─────────────────┴─────────────────┴─────────────────────────────┘
         │                 │                       │
         └─────────────────┼───────────────────────┘
                           │
                    ┌──────▼──────┐
                    │ workspace/  │
                    │ ├── SOUL.md │  ← Agent identity
                    │ ├── USER.md │  ← User profile
                    │ └── memory/ │  ← SQLite database
                    └──────┬──────┘
                           │
         ┌─────────────────┼─────────────────┐
         │                 │                 │
    ┌────▼────┐       ┌────▼────┐      ┌─────▼─────┐
    │  CLI    │       │  Core   │      │  Docker   │
    │  Tools  │◄─────►│  Agent  │◄────►│ Services  │
    └────┬────┘       └────┬────┘      └─────┬─────┘
         │                 │                 │
         │    ┌────────────┴──────────┐      │
         │    │                       │      │
    ┌────▼────▼──┐              ┌─────▼──────▼──┐
    │ Providers  │              │ Telegram      │
    │ • Kimi     │              │ Bot Service   │
    │ • Claude   │              │               │
    │ • GPT      │              │ • Bot         │
    │ • etc      │              │ • Kimi Agent  │
    └────────────┘              └───────────────┘
```

Data Flow

IDE Mode

```
User Input → IDE Connector → Memory (context) → Provider API → Response
                                    ↓
                              Store interaction
```

Telegram Mode

```
Telegram Message → Bot Container → Kimi Agent (8081) → API
                        ↓               ↓
                   SQLite Memory ←── Shared Volume
```

Setup Options

Flowchart: Setup Decision

```
                    ┌─────────────────┐
                    │   Run setup.sh  │
                    └────────┬────────┘
                             │
                    ┌────────▼────────┐
                    │ init.yaml       │
                    │ exists?         │
                    └────────┬────────┘
                             │
              ┌──────────────┼──────────────┐
              │ NO                          │ YES
              ▼                             ▼
     ┌─────────────────┐           ┌─────────────────┐
     │  NEW SETUP      │           │  EXISTING SETUP │
     └────────┬────────┘           └────────┬────────┘
              │                             │
     ┌────────▼────────┐           ┌────────▼────────┐
     │ [1] IDE only    │           │ [1] Add/Remove  │
     │ [2] IDE +       │           │     Telegram    │
     │     Telegram    │           │ [2] Edit        │
     └─────────────────┘           │     Settings    │
                                   │ [3] Start Fresh │
                                   └─────────────────┘
```

Option Details

1. IDE Only

Best for: Users who only need IDE assistance

Features: - Direct API calls to LLM providers - Local SQLite memory - Full context from SOUL.md and USER.md - No Docker required

Use case:

```
./setup.sh
# Select: IDE only
# Use with: Kimi Code, Claude Code, etc.
```

2. IDE + Telegram

Best for: Users who want mobile and desktop access

Features: - Everything from IDE mode - Telegram bot with shared memory - Docker auto-starts - Access from anywhere

Requirements: - Docker installed - Telegram bot token - Kimi API key

Use case:

```
./setup.sh
# Select: IDE + Telegram
# Provide: Bot token, API key
# Result: Docker starts automatically
```

Configuration Management

The setup wizard intelligently detects existing configurations and offers appropriate options.

New Configuration

When `init.yaml` doesn't exist:

```yaml
# wizard presents:
mode_selection:
  - ide_only: "Works with Kimi Code, Claude Code, Cursor, etc."
  - ide_telegram: "Both IDE and Telegram bot with shared memory"
```

Managing Existing Configuration

When `init.yaml` exists, the wizard shows:

```
Current setup: IDE only  (or: IDE + Telegram)

[1]    Add Telegram          (if no Telegram)
[1]    Remove Telegram       (if has Telegram)
    Add/remove Telegram bot

[2]     Edit Settings
    Change: Agent Identity / User Profile / Both

[3]    Start Fresh
    Delete existing and create new configuration
```

Add Telegram to Existing

Scenario: You started with IDE only, now want Telegram.

```
./setup.sh
# Select: Add Telegram
# Provide: Bot token, API key
# Result: init.yaml updated, .env created
```

What happens: 1. Wizard reads existing `init.yaml` 2. Prompts for Telegram credentials 3. Updates `mode.telegram.enabled = true` 4. Creates `.env` with API keys 5. Docker starts automatically

Remove Telegram

Scenario: You want to disable Telegram temporarily.

```
./setup.sh
# Select: Remove Telegram
```

What happens: 1. Sets `mode.telegram.enabled = false` 2. Keeps all other settings 3. IDE mode continues working 4. Telegram bot stops (run `docker compose down`)

Edit Settings

Scenario: You want to change your agent's name or template.

```
./setup.sh
# Select: Edit Settings
# Choose: Agent Identity / User Profile / Both
```

Options: - **Agent Identity:** Change name, tone, template - **User Profile:** Change your name, role, preferences - **Both:** Update everything

---

Memory System

Architecture

The memory system uses **SQLite** (relational database) with keyword-based retrieval. It is **NOT** a graph database or vector store.

```
┌─────────────────────────────────────────────────┐
│                 MEMORY SYSTEM                   │
├─────────────────────────────────────────────────┤
│                                                 │
│  ┌──────────────┐      ┌─────────────────────┐  │
│  │   Memory     │      │  IDE Connector      │  │
│  │   Store      │◄────►│                     │  │
│  │              │      │ • get_context()     │  │
│  │ • store()    │      │ • store_interaction │  │
│  │ • recall()   │      │ • store_fact()      │  │
│  │ • get_stats()│      │ • recall()          │  │
│  └──────────────┘      └─────────────────────┘  │
│         ▲                                       │
│         │ SQLite                                │
│         ▼                                       │
│  ┌──────────────────────────────────────────┐   │
│  │    workspace/memory/agent_memory.db      │   │
│  └──────────────────────────────────────────┘   │
└─────────────────────────────────────────────────┘
```

Database Schema

Table: `memories`

Column	Type	Description
`id`	INTEGER PK	Auto-increment primary key
`content`	TEXT	Memory content (Q&A or fact)
`category`	TEXT	Type: conversation, fact, preference…
`importance`	TEXT	low, medium, high
`metadata`	TEXT	JSON: timestamp, source, user, agent
`created_at`	TIMESTAMP	When stored
`access_count`	INTEGER	How many times recalled
`last_accessed`	TIMESTAMP	Last retrieval time

Indexes: - `idx_memories_category` - For faster category queries

How Retrieval Works

Algorithm (simplified):

```
1. Get last 15 memories (ORDER BY created_at DESC)
2. Split query into keywords
3. For each memory:
   - score = count of keywords found in content
4. Sort by (score, created_at) descending
5. Return top 5
6. UPDATE access_count, last_accessed
```

Important: This is **keyword matching**, NOT semantic search. It looks for exact word matches, not meaning similarity.

Memory Types

1. Conversation History

Automatically stores all Q&A pairs.

```
connector.store_interaction(
    question="How do I create a React component?",
    answer="You can create a React component by..."
)
```

2. Facts

Important information extracted from conversations.

```
connector.store_fact("User prefers TypeScript over JavaScript")
```

3. Context Retrieval

Automatically retrieves relevant context before responding.

```
context = connector.get_context(user_message)
# Returns: Relevant past conversations and facts
```

Memory Synchronization

IDE Mode

- Direct SQLite access
- Immediate read/write
- No latency

Telegram Mode

- Both containers mount `./workspace/memory/`
- SQLite file shared via Docker volume
- Automatic synchronization

```
# docker-compose.yml
volumes:
  - ../workspace/memory:/app/memory:rw  # Shared
```

Memory Commands

Via CLI:

```
python scripts/backup-memory.py    # Backup database
```

Via Telegram Bot:

```
/memory    # Show statistics
/clear     # Clear conversation history
```

Telegram Integration

Architecture

Two-container system for security and modularity:

```
┌─────────────────────────────────────────────────────────┐
│                  TELEGRAM ARCHITECTURE                  │
├─────────────────────────┬───────────────────────────────┤
│   telegram-bot          │   kimi-agent                  │
│   (Python)              │   (Python/FastAPI)            │
│                         │                               │
│ • Receives messages     │ • Has API key                 │
│ • Manages memory        │ • Processes LLM requests      │
│ • Syncs with IDE        │ • Loads AGENTS.md             │
│                         │                               │
│ Port: -                 │ Port: 8081                    │
└──────────┬──────────────┴───────────────┬───────────────┘
           │                              │
           │      ┌──────────────┐        │
           └─────►│   Telegram   │◄───────┘
                  │   API        │
                  └──────────────┘
```

Security Model

1. **Kimi Agent** has API key (secure container)
2. **Telegram Bot** has bot token (separate container)
3. **Memory** shared via volume (SQLite)
4. **AGENTS.md** loaded by Kimi Agent on startup

Configuration

Required

```yaml
# init.yaml
mode:
  telegram:
    enabled: true
    bot_token: "YOUR_BOT_TOKEN"
```

```
# .env
KIMI_API_KEY=your_key_here
TELEGRAM_BOT_TOKEN=your_token_here
```

Optional

```yaml
# init.yaml
mode:
  telegram:
    user_id: "123456789"      # Restrict to specific user
    webhook_url: ""           # For webhook mode (default: polling)
```

User Authorization

Restrict bot to specific Telegram user:

1. Get your Telegram ID from @userinfobot
2. Set during setup wizard
3. Bot ignores messages from other users

IDE Integration

Supported IDEs

IDE	Integration Type	Notes
Kimi Code	Native	Full context support
Claude Code	Native	Full context support
Cursor	Native	Full context support
VS Code + Continue	Extension	Via tool support
Any IDE	File-based	Read SOUL.md, USER.md

How It Works

The agent reads context files automatically:

```
# Agent initialization reads:
workspace/SOUL.md     # Who am I?
workspace/USER.md     # Who is the user?
init.yaml             # Configuration
```

Context Files

SOUL.md (Agent Identity)

```
# SOUL - Assistant

## Identity
**Name:** Klaus
**Role:** Software Architect
**Specialization:** System design, cloud architecture

## Core Philosophy
> "Build systems that scale"

## Personality
**Tone:** professional
**Style:** detailed
```

USER.md (User Profile)

```
# USER - John

## Profile
**Name:** John Doe
**Role:** Senior Developer
**Experience Level:** advanced

## Preferences
- **Communication:** detailed
- **Code Style:** clean
```

Templates

Templates define agent personality and capabilities.

Available Templates

Template	Best For	Personality
`general`	Everyday tasks	Balanced, helpful
`architect`	System design	Strategic, technical
`developer`	Coding	Practical, efficient
`finance`	Financial analysis	Analytical, precise
`legal`	Legal work	Formal, thorough
`marketing`	Growth/marketing	Creative, persuasive
`ui`	Design work	Visual, user-focused

Template Structure

```
templates/
├── architect/
│   └── SOUL.md       # Personality definition
├── developer/
│   └── SOUL.md
└── ...
```

Customizing Templates

Edit `workspace/SOUL.md` after setup, or:

```
./setup.sh
# Select: Edit Settings → Agent Identity
```

Providers

Supported Providers

Provider	Environment Variable	Models
Kimi	`KIMI_API_KEY`	kimi-k2-5
Anthropic	`ANTHROPIC_API_KEY`	claude-3-opus, etc.
OpenAI	`OPENAI_API_KEY`	gpt-4, gpt-3.5
OpenRouter	`OPENROUTER_API_KEY`	Multiple
Gemini	`GEMINI_API_KEY`	gemini-pro

Configuration

```yaml
# init.yaml
provider:
  name: "kimi"           # Provider name
  api_key: ""            # From environment
  model:
    kimi: "kimi-k2-5"    # Model selection
  parameters:
    temperature: 0.7
    max_tokens: 4096
    top_p: 0.9
```

Switching Providers

Edit `init.yaml` or use setup wizard to change providers.

Docker Services

Services

Service	Image	Port	Purpose
`kimi-agent`	Custom	8081	LLM processing, AGENTS.md
`telegram-bot`	Custom	-	Telegram interface

File Mounts

Host Path	Container Path	Service
`./workspace/memory/`	`/app/memory`	Both
`./workspace/`	`/app/workspace`	Both
`./docs/`	`/app/docs`	Kimi Agent

Commands

```bash
# Start
docker compose -f docker/docker-compose.yml up -d

# Stop
docker compose -f docker/docker-compose.yml down

# View logs
docker compose -f docker/docker-compose.yml logs -f

# Restart
docker compose -f docker/docker-compose.yml restart
```

Kimi Agent Patch

The Kimi Agent uses a custom Dockerfile to: 1. Extend base `clawd-agent` image 2. Load AGENTS.md on startup 3. Mount additional volumes

See `docker/kimi-agent-patch/` for details.

Security

Data Protection

File	Contains	In Git?
`.env`	API keys, tokens	No
`init.yaml`	Configuration	No
`workspace/SOUL.md`	Agent identity	No
`workspace/USER.md`	User profile	No
`workspace/memory/`	SQLite data	No
`docs/AGENTS.md`	Agent guide	Yes

Best Practices

1. **Never commit** sensitive files
2. **Rotate API keys** regularly
3. **Restrict Telegram** to specific user ID
4. **Backup memory** periodically

PII Protection

All user data stored locally: - SQLite database in `workspace/memory/` - No cloud sync - No telemetry

---

Troubleshooting

Setup Issues

Problem: Setup fails with Python error

```
# Check version
python3 --version  # Must be 3.11+

# Check pip
pip3 --version

# Manual setup
python3 -m venv .venv
source .venv/bin/activate
pip install -r requirements.txt
python scripts/setup_wizard.py
```

Telegram Issues

Problem: Bot not responding

```
# Check containers
docker compose -f docker/docker-compose.yml ps

# View logs
docker compose -f docker/docker-compose.yml logs telegram-bot
docker compose -f docker/docker-compose.yml logs kimi-agent

# Restart
docker compose -f docker/docker-compose.yml restart
```

Problem: Kimi Agent unhealthy

```
Note: Kimi Agent shows "unhealthy" but works (no curl in container)
This is a known limitation, not an error.
```

Memory Issues

Problem: Memory not syncing between IDE and Telegram

```
# Check permissions
ls -la workspace/memory/

# Fix permissions
chmod 755 workspace/memory/
chmod 644 workspace/memory/agent_memory.db

# Restart Docker
docker compose -f docker/docker-compose.yml restart
```

Configuration Issues

Problem: Want to change settings

```
./setup.sh
# Select: Edit Settings
```

Problem: Want to start fresh

```
./reset.sh  #    Deletes everything!
./setup.sh
```

API Reference

IDE Connector

```python
from core.connectors.ide_connector import get_connector

connector = get_connector()
```

Methods

get_context(message)

```python
context = connector.get_context("How do I deploy?")
# Returns: Relevant memories and conversation history
```

store_interaction(question, answer)

```python
connector.store_interaction(
    question="What is Docker?",
    answer="Docker is a containerization platform..."
)
```

store_fact(fact)

```python
connector.store_fact("User prefers Python over JavaScript")
```

recall(query)

```python
memories = connector.recall("deployment")
# Returns: List of relevant memories
```

get_stats()

```python
stats = connector.get_stats()
# Returns: Memory statistics
```

Memory Store

```python
from core.memory import get_memory_store

store = get_memory_store()
```

Methods

store(key, value, type="fact")

```python
store.store("user_preference", "Python", type="preference")
```

recall(query, limit=5)

```python
results = store.recall("Python", limit=3)
```

get_stats()

```python
stats = store.get_stats()
```

Appendix

File Structure Reference

```
ide-agent-wizard/
├── setup.sh              # Main entry point
├── reset.sh              # Factory reset
├── init.yaml.example     # Config template
├── .env.example          # Environment template
├── requirements.txt      # Python dependencies
│
├── bot/
│   └── telegram_bot.py   # Telegram bot implementation
│
├── cli/
│   ├── agent-cli.py      # CLI interface
│   └── setup.py          # Setup utilities
│
├── core/
│   ├── agent.py          # Main agent logic
│   ├── memory.py         # Memory store
│   ├── connectors/
│   │   ├── ide_connector.py    # IDE integration
│   │   └── base.py            # Base connector
│   └── providers/
│       ├── kimi_provider.py
│       ├── anthropic_provider.py
│       ├── openrouter_provider.py
│       └── ...
│
├── docker/
│   ├── docker-compose.yml
│   ├── Dockerfile
│   └── kimi-agent-patch/
│       ├── Dockerfile
│       └── app.py
│
├── docs/
│   ├── README.md
│   ├── AGENTS.md
│   ├── RELEASE_NOTES.md
│   └── CHECKLIST.md
│
├── scripts/
│   ├── setup_wizard.py   # Interactive wizard
│   ├── initialize.py     # Post-setup init
│   ├── backup-memory.py  # Memory backup
│   └── reset.sh          # Reset script
│
├── templates/
│   ├── architect/
│   ├── developer/
│   ├── finance/
│   ├── general/
│   ├── legal/
│   ├── marketing/
│   └── ui/
│
└── workspace/            # User data (gitignored)
    ├── SOUL.md
    ├── USER.md
    ├── memory/
    └── projects/
```

Environment Variables

```
# Required for Telegram mode
KIMI_API_KEY=your_kimi_key
TELEGRAM_BOT_TOKEN=your_bot_token

# Optional
KIMI_AGENT_URL=http://localhost:8081
```

Configuration Schema

```yaml
# init.yaml
agent:
  name: "Klaus"
  template: "architect"
  personality:
    tone: "professional"
    style: "balanced"
    language: "en"

user:
  name: "John"
  role: "Developer"
  experience_level: "advanced"
  preferences:
    communication: "detailed"
    code_style: "clean"

mode:
  primary: "hybrid"  # or "ide"
  ide:
    enabled: true
  telegram:
    enabled: true
    bot_token: ""
    user_id: ""
    webhook_url: ""

provider:
  name: "kimi"
  api_key: ""
  model:
    kimi: "kimi-k2-5"
  parameters:
    temperature: 0.7
    max_tokens: 4096
    top_p: 0.9
```

Questions? Check `docs/AGENTS.md` for AI-specific guidance.

Ready to build? Run `./setup.sh` and let’s go!