Eva AI Tool Development Guide

Table of Contents

Overview

Eva AI uses a tag-based tool system where the AI can invoke various functions by using XML-like tags in its responses. The system is designed to be modular, allowing you to easily add new capabilities without modifying the core chatbot logic.

How Tools Work

1. Eva generates a response containing tool tags (e.g., `<t>Hello</t>`, `<s>weather today</s>`)
2. The `parse_ai_response()` function extracts these tags using regex patterns
3. Each tool executes its corresponding Python function
4. Results are captured and fed back to Eva for follow-up responses

---

Tool Architecture

Core Components

1. Tool Tag Definition

- Format: `<tag_name>content</tag_name>`
- Case-insensitive with DOTALL flag support
- Content can be multi-line

2. Regex Pattern

```python
tool_pattern = re.compile(r'<tag_name>(.+?)</tag_name>', re.IGNORECASE | re.DOTALL)
```

3. Processing Logic

- Extract all matches using `findall()`
- Process each match individually
- Capture outputs for Eva's awareness
- Handle errors gracefully

4. Response Integration

- Add results to `captured_outputs[]`
- Append to `session_history` as user input
- Set `needs_followup = True` for Eva to respond

Step-by-Step Tool Creation

Step 1: Define Your Tool Function

Create the actual functionality first:

```python
def your_new_function(parameter):
    """Your tool's core functionality"""
    try:
        # Implement your logic here
        result = do_something(parameter)
        return f"✅ Success: {result}"
    except Exception as e:
        return f"❌ Error: {e}"
```

Step 2: Add Tool to Eva's Prompt

Update the `EVA_PROMPT` variable to include your new tool:

```python
```

```
EVA_PROMPT = """
...existing prompt...

Your Tools (USE THE SAME FORMAT):
<t> text </t>: [what you want to say to Abdelrhman - always use this to communicate]
<your_tool> parameter </your_tool>: [description of what your tool does]
...other existing tools...
"""
```

Step 3: Add Regex Pattern in parse_ai_response()

Add your tool's processing logic in the `parse_ai_response()` method:

python

```python
def parse_ai_response(self, response_text, enhanced_system_prompt=None):
    # ... existing code ...

    # YOUR NEW TOOL - Process ALL instances
    your_tool_pattern = re.compile(r'<your_tool>(.+?)</your_tool>', re.IGNORECASE | re.DOTALL)
    your_tool_matches = your_tool_pattern.findall(response_text)
    if your_tool_matches:
        print(f"🔧 Eva is using YOUR TOOL ({len(your_tool_matches)} instances)")
        for parameter in your_tool_matches:
            parameter = parameter.strip()
            print(f"  Processing: {parameter}")
            try:
                result = your_new_function(parameter)
                print(f"  Result: {result}")
                captured_outputs.append(result)

                # Optional: Add to session history for Eva's awareness
                self.session_history.append({
                    "role": "user",
                    "content": f"tool_result: {result}"
                })
                needs_followup = True

            except Exception as e:
                error_msg = f"❌ Error in your tool: {e}"
                print(error_msg)
                captured_outputs.append(error_msg)
```

Step 4: Test Your Tool

1. Add example usage to Eva's prompt (optional)
2. Test with various inputs
3. Verify error handling
4. Check console output and user feedback

Tool Types & Patterns

1. Simple Action Tools

Pattern: Execute and return status

```python
# Example: <open_app>notepad</open_app>
app_pattern = re.compile(r'<open_app>(.+?)</open_app>', re.IGNORECASE | re.DOTALL)
```

2. Data Retrieval Tools

Pattern: Get data and provide to Eva for processing

```python
# Example: <get_weather>Cairo</get_weather>
weather_pattern = re.compile(r'<get_weather>(.+?)</get_weather>', re.IGNORECASE | re.DOTALL)
# Always set needs_followup = True for data retrieval
```

3. Trigger/Update Tools

Pattern: No parameters, just triggers an action

```python
# Example: <task_update>
task_pattern = re.compile(r'<task_update>', re.IGNORECASE | re.DOTALL)
if task_pattern.search(response_text):
    # Handle trigger
```

4. Complex Parameter Tools

Pattern: Multiple parameters separated by delimiters

```python
# Example: <send_email>user@email.com | Subject | Body</send_email>
email_pattern = re.compile(r'<send_email>(.+?)</send_email>', re.IGNORECASE | re.DOTALL)
# Parse: parts = parameter.split('|')
```

Integration Requirements

1. Import Dependencies

Add any required imports at the top of the file:

```python
import your_required_library
from some_module import specific_function
```

2. Configuration Variables

Add configuration at the top with other constants:

```python
YOUR_API_KEY = "your_api_key_here"
YOUR_ENDPOINT = "https://api.example.com"
```

3. Error Handling Standards

Always include proper error handling:

```python
try:
    result = your_function(parameter)
    success_msg = f"✅ Tool succeeded: {result}"
    captured_outputs.append(success_msg)
except Exception as e:
    error_msg = f"❌ Tool failed: {e}"
    print(error_msg)
    captured_outputs.append(error_msg)
```

4. Logging Standards

Use consistent logging:

```python
print(f"🔧 Eva is using YOUR TOOL ({len(matches)} instances)")
print(f"   Processing: {parameter}")
print(f"   Result: {result}")
```

Testing & Debugging

1. Console Output

Monitor console for:

- Tool detection messages
- Parameter processing
- Results and errors
- Eva's follow-up responses

2. Debug Information

Add debug prints during development:

```python
print(f"DEBUG - Raw parameter: '{parameter}'")
print(f"DEBUG - Parsed parts: {parts}")
print(f"DEBUG - Function result: {result}")
```

3. Test Cases

Create comprehensive test scenarios:

- Valid inputs
- Invalid inputs
- Edge cases
- Error conditions
- Multiple instances

4. Integration Testing

- Test with real Eva conversations

- Verify follow-up responses work
- Check memory integration
- Test with other tools

Best Practices

1. Tool Design

- ✅ **Single Responsibility**: Each tool should do one thing well
- ✅ **Consistent Naming**: Use clear, descriptive tag names
- ✅ **Parameter Validation**: Always validate inputs
- ✅ **Graceful Errors**: Never let tools crash the system

2. Response Handling

- ✅ **User Feedback**: Always provide clear success/error messages
- ✅ **Eva Awareness**: Use `captured_outputs` for important results
- ✅ **Follow-up Logic**: Set `needs_followup = True` for data retrieval
- ✅ **Consistent Format**: Use emojis and consistent message structure

3. Performance

- ✅ **Async Operations**: Use threading for long-running operations
- ✅ **Rate Limiting**: Implement proper API rate limiting
- ✅ **Resource Cleanup**: Clean up resources properly
- ✅ **Caching**: Cache results when appropriate

4. Security

- ✅ **Input Sanitization**: Validate and sanitize all inputs
- ✅ **API Security**: Secure API keys and credentials
- ✅ **Permission Checks**: Verify user permissions for sensitive operations
- ✅ **Error Messages**: Don't expose sensitive information in errors

---

Examples

Example 1: Simple Status Tool

```python
# 1. Function Definition
def check_system_status():
    """Check system health"""
    try:
        cpu_usage = psutil.cpu_percent(interval=1)
        memory = psutil.virtual_memory()
        return f"CPU: {cpu_usage}%, Memory: {memory.percent}%"
    except Exception as e:
        return f"Error getting status: {e}"

# 2. Add to EVA_PROMPT
# <system_status>: [check system health]

# 3. Add to parse_ai_response()
status_pattern = re.compile(r'<system_status>', re.IGNORECASE | re.DOTALL)
if status_pattern.search(response_text):
    print("🖥️ Eva is checking system status")
    try:
        result = check_system_status()
        success_msg = f"🖥️ System Status: {result}"
        print(success_msg)
        captured_outputs.append(success_msg)
    except Exception as e:
        error_msg = f"❌ System status error: {e}"
        print(error_msg)
        captured_outputs.append(error_msg)
```

Example 2: Parameter-Based Tool

```python
```

```python
# 1. Function Definition
def convert_currency(amount, from_curr, to_curr):
    """Convert currency using external API"""
    try:
        # Implementation here
        rate = get_exchange_rate(from_curr, to_curr)
        result = float(amount) * rate
        return f"{amount} {from_curr} = {result:.2f} {to_curr}"
    except Exception as e:
        return f"Currency conversion error: {e}"

# 2. Add to EVA_PROMPT
# <convert_currency> amount from_currency to_currency </convert_currency>: [convert currencies]

# 3. Add to parse_ai_response()
currency_pattern = re.compile(r'<convert_currency>(.+?)</convert_currency>', re.IGNORECASE | re.DOTALL)
currency_requests = currency_pattern.findall(response_text)
if currency_requests:
    print(f"💱 Eva is using CURRENCY CONVERTER ({len(currency_requests)} conversions)")
    for request in currency_requests:
        try:
            parts = request.strip().split()
            if len(parts) != 3:
                raise ValueError("Format: amount from_currency to_currency")

            amount, from_curr, to_curr = parts
            result = convert_currency(amount, from_curr, to_curr)
            success_msg = f"💱 {result}"
            print(f"   {success_msg}")
            captured_outputs.append(success_msg)

        except Exception as e:
            error_msg = f"❌ Currency conversion error: {e}"
            print(error_msg)
            captured_outputs.append(error_msg)
```

Example 3: Data Retrieval Tool

python

```python
# 1. Function Definition
def get_news_headlines(category="general", count=5):
    """Get latest news headlines"""
    try:
        # API call implementation
        headlines = fetch_news_api(category, count)
        return headlines
    except Exception as e:
        return f"News fetch error: {e}"

# 2. Add to EVA_PROMPT
# <get_news> category count </get_news>: [get latest news headlines]

# 3. Add to parse_ai_response()
news_pattern = re.compile(r'<get_news>(.+?)</get_news>', re.IGNORECASE | re.DOTALL)
news_requests = news_pattern.findall(response_text)
if news_requests:
    print(f"📰 Eva is fetching news ({len(news_requests)} requests)")
    for request in news_requests:
        try:
            parts = request.strip().split()
            category = parts[0] if parts else "general"
            count = int(parts[1]) if len(parts) > 1 else 5

            headlines = get_news_headlines(category, count)
            captured_outputs.append(f"📰 Latest {category} news: {headlines}")

            # Important: Add to session for Eva's awareness
            self.session_history.append({
                "role": "user",
                "content": f"news_update: {headlines}"
            })
            needs_followup = True

        except Exception as e:
            error_msg = f"❌ News fetch error: {e}"
            print(error_msg)
            captured_outputs.append(error_msg)
```

Troubleshooting

Common Issues

1. Tool Not Triggering

- ✅ Check regex pattern syntax
- ✅ Verify tag name in Eva's prompt matches code
- ✅ Test regex pattern independently
- ✅ Check for case sensitivity issues

2. Parameters Not Parsing

- ✅ Add debug prints for raw parameters
- ✅ Check for unexpected whitespace
- ✅ Verify delimiter handling
- ✅ Test with simple inputs first

3. Eva Not Responding to Results

- ✅ Ensure `needs_followup = True` is set
- ✅ Check `session_history` additions
- ✅ Verify `captured_outputs` format
- ✅ Test follow-up logic independently

4. Multiple Instances Not Working

- ✅ Use `findall()` instead of `search()`
- ✅ Loop through all matches
- ✅ Process each instance separately
- ✅ Handle errors per instance

Debugging Checklist

```python
# Add these debug statements during development:
print(f"🔍 DEBUG - Raw response: {response_text}")
print(f"🔍 DEBUG - Pattern matches: {your_pattern.findall(response_text)}")
print(f"🔍 DEBUG - Parameter: '{parameter}'")
print(f"🔍 DEBUG - Parsed data: {parsed_data}")
print(f"🔍 DEBUG - Function result: {result}")
print(f"🔍 DEBUG - Captured outputs: {captured_outputs}")
print(f"🔍 DEBUG - Needs followup: {needs_followup}")
```

Advanced Topics

1. Asynchronous Tools

For long-running operations:

```python
import threading

def long_running_task(parameter):
    def task():
        # Long operation here
        result = expensive_operation(parameter)
        # Store result somewhere Eva can access

    threading.Thread(target=task, daemon=True).start()
    return "⏳ Task started in background..."
```

2. Stateful Tools

For tools that maintain state:

```python
class StatefulTool:
    def __init__(self):
        self.state = {}

    def process(self, parameter):
        # Use self.state for persistence
        pass

# Initialize once
stateful_tool = StatefulTool()
```

3. Tool Chaining

For tools that depend on other tools:

```python
```

```
# Check if prerequisite tools have run
if 'prerequisite_result' in captured_outputs:
    # Safe to run dependent tool
    pass
```

4. External Service Integration

```python
import requests
import json

def call_external_service(data):
    try:
        response = requests.post(
            'https://api.service.com/endpoint',
            headers={'Authorization': f'Bearer {API_KEY}'},
            json=data,
            timeout=10
        )
        response.raise_for_status()
        return response.json()
    except requests.exceptions.RequestException as e:
        raise Exception(f"Service call failed: {e}")
```

This guide should provide everything you need to create powerful new tools for Eva AI. Remember to test thoroughly and follow the established patterns for the best integration experience!