Accessing MTOR's Core Services: A Developer's Guide

Introduction

MTOR follows an API-first design philosophy, providing access to its powerful internal services through its event-driven architecture. Once you've implemented the MTOR adapter from our previous guide, you can seamlessly access all of MTOR's core AI services.

This guide provides comprehensive instructions and examples for leveraging MTOR's internal services:

- Speech-to-Text (Whisper)
- Text-to-Speech (BARK and fallback systems)
- Vision Processing
- Weather Services
- Gmail Integration

Throughout this guide, we'll show you exactly how to structure your events, understand the response formats, and incorporate these powerful services directly into your applications.

Prerequisites

- Completed MTOR adapter implementation (from previous guide)
- Active WebSocket connection to MTOR
- Basic understanding of JSON event formats
- Familiarity with asynchronous programming

Core Concepts

MTOR's internal services follow the same event-driven patterns as all other MTOR interactions:

1. **Event-based Requests**: All service calls are made by sending events of specific types
2. **Asynchronous Processing**: Services process requests non-blockingly and return results via events
3. **Stateless Design**: Each request contains all necessary context
4. **Universal JSON Format**: Consistent patterns for requests and responses

Let's explore how to access each core service.

1. Speech-to-Text (Whisper)

MTOR uses OpenAI's Whisper model for high-quality speech recognition. Here's how to use it.

Request Format

```python
async def transcribe_audio(adapter, audio_base64):
    """Send audio for transcription using MTOR's Whisper integration."""
    event = {
        "type": "speech_to_text",
        "audio": audio_base64,  # Base64-encoded audio data
        "id": f"transcribe_{int(time.time())}"  # Unique request ID
    }

    await adapter.send_event(event)
    # Response will be handled by your event listener
```

Response Format

When transcription is complete, MTOR will send a response event:

```json
{
  "type": "transcription_result",
  "text": "This is the transcribed text from the audio.",
  "response_to": "transcribe_1614532345"
}
```

Complete Example

Here's a complete example showing how to record audio from a microphone, send it for transcription, and handle the response:

```
python
```

```python
import asyncio
import base64
import json
import pyaudio
import wave
import time
from mtor_adapter import MTORAdapter

class AudioTranscriptionService:
    def __init__(self, mtor_adapter):
        self.adapter = mtor_adapter
        self.adapter.register_handler("transcription_result", self.handle_transcription)
        self.pending_transcriptions = {}

    async def record_and_transcribe(self, duration=5):
        """Record audio and send for transcription."""
        # Record audio
        audio_data = await self.record_audio(duration)

        # Convert to base64
        audio_base64 = base64.b64encode(audio_data).decode('utf-8')

        # Create a unique ID and future for this request
        request_id = f"transcribe_{int(time.time())}"
        future = asyncio.Future()
        self.pending_transcriptions[request_id] = future

        # Send transcription request
        await self.adapter.send_event({
            "type": "speech_to_text",
            "audio": audio_base64,
            "id": request_id
        })

        # Wait for response (with timeout)
        try:
            result = await asyncio.wait_for(future, timeout=30)
            return result
        except asyncio.TimeoutError:
            del self.pending_transcriptions[request_id]
            raise TimeoutError("Transcription request timed out")

    async def handle_transcription(self, event):
```

```
    """Handle transcription result events."""
    request_id = event.get("response_to")
    if request_id in self.pending_transcriptions:
        future = self.pending_transcriptions[request_id]
        future.set_result(event.get("text"))
        del self.pending_transcriptions[request_id]

async def record_audio(self, duration=5, sample_rate=16000):
    """Record audio from microphone."""
    chunk = 1024
    audio_format = pyaudio.paInt16
    channels = 1

    p = pyaudio.PyAudio()
    stream = p.open(format=audio_format,
                    channels=channels,
                    rate=sample_rate,
                    input=True,
                    frames_per_buffer=chunk)

    print(f"Recording for {duration} seconds...")
    frames = []

    for i in range(0, int(sample_rate / chunk * duration)):
        data = stream.read(chunk)
        frames.append(data)

    print("Recording finished")
    stream.stop_stream()
    stream.close()
    p.terminate()

    # Save to in-memory file
    audio_data = b''
    mem_file = io.BytesIO()
    wf = wave.open(mem_file, 'wb')
    wf.setnchannels(channels)
    wf.setsampwidth(p.get_sample_size(audio_format))
    wf.setframerate(sample_rate)
    wf.writeframes(b''.join(frames))
    wf.close()

    mem_file.seek(0)
```

```
        audio_data = mem_file.read()
        return audio_data
```

Sending Pre-recorded Audio

If you already have audio files, you can send them as follows:

python

```python
async def transcribe_audio_file(adapter, file_path):
    """Transcribe an existing audio file using MTOR."""
    # Read file
    with open(file_path, 'rb') as audio_file:
        audio_data = audio_file.read()

    # Convert to base64
    audio_base64 = base64.b64encode(audio_data).decode('utf-8')

    # Send for transcription
    request_id = f"transcribe_file_{int(time.time())}"
    await adapter.send_event({
        "type": "speech_to_text",
        "audio": audio_base64,
        "id": request_id
    })

    # Response will come through the event listener
```

2. Text-to-Speech (BARK)

MTOR provides high-quality text-to-speech using the BARK model, with automatic fallback to other TTS engines for longer content.

Request Format

python

```python
async def synthesize_speech(adapter, text):
    """Convert text to speech using MTOR's TTS services."""
    event = {
        "type": "text_to_speech",
        "text": text,
        "voice": "v2/en_speaker_6",  # Optional voice selection
        "id": f"tts_{int(time.time())}"
    }

    await adapter.send_event(event)
    # Response will be handled by your event listener
```

Response Format

MTOR will send an audio response event:

json

```json
{
  "type": "speech_result",
  "audio": "base64_encoded_audio_data_here",
  "response_to": "tts_1614532345",
  "format": "wav"
}
```

Complete Example

Here's how to request speech synthesis and play the resulting audio:

```
python
```

```python
import asyncio
import base64
import json
import io
import time
import pygame
from mtor_adapter import MTORAdapter

class TextToSpeechService:
    def __init__(self, mtor_adapter):
        self.adapter = mtor_adapter
        self.adapter.register_handler("speech_result", self.handle_speech)
        self.pending_speech = {}
        pygame.mixer.init()

    async def say_text(self, text, voice=None):
        """Convert text to speech and play it."""
        # Create a unique ID and future for this request
        request_id = f"tts_{int(time.time())}"
        future = asyncio.Future()
        self.pending_speech[request_id] = future

        # Prepare the event
        event = {
            "type": "text_to_speech",
            "text": text,
            "id": request_id
        }

        # Add voice if specified
        if voice:
            event["voice"] = voice

        # Send the request
        await self.adapter.send_event(event)

        # Wait for response (with timeout)
        try:
            audio_data = await asyncio.wait_for(future, timeout=30)
            # Play the audio
            self.play_audio(audio_data)
            return True
        except asyncio.TimeoutError:
```

```
            del self.pending_speech[request_id]
            raise TimeoutError("Speech synthesis request timed out")

    async def handle_speech(self, event):
        """Handle speech result events."""
        request_id = event.get("response_to")
        if request_id in self.pending_speech:
            future = self.pending_speech[request_id]
            future.set_result(event.get("audio"))
            del self.pending_speech[request_id]

    def play_audio(self, audio_base64):
        """Play audio from base64 data."""
        # Decode base64 audio
        audio_data = base64.b64decode(audio_base64)

        # Save to in-memory file
        audio_io = io.BytesIO(audio_data)

        # Play using pygame
        pygame.mixer.music.load(audio_io)
        pygame.mixer.music.play()

        # Wait for audio to finish
        while pygame.mixer.music.get_busy():
            pygame.time.Clock().tick(10)
```

Speech Synthesis Options

MTOR supports additional options for its TTS service:

python

```python
async def advanced_tts(adapter, text, options):
    """Request TTS with advanced options."""
    event = {
        "type": "text_to_speech",
        "text": text,
        "options": {
            "voice": options.get("voice", "v2/en_speaker_6"),
            "speed": options.get("speed", 1.0),
            "temperature": options.get("temperature", 0.7),
            "priority": options.get("priority", "quality")  # "quality" or "speed"
        },
        "id": f"tts_adv_{int(time.time())}"
    }

    await adapter.send_event(event)
```

3. Vision Processing

MTOR provides computer vision capabilities, allowing you to send images for analysis and description.

Request Format

python

```python
async def process_image(adapter, image_base64, prompt=None):
    """Send an image for vision processing."""
    event = {
        "type": "vision",
        "image": image_base64,
        "prompt": prompt or "Describe this image in detail",
        "id": f"vision_{int(time.time())}"
    }

    await adapter.send_event(event)
```

Response Format

MTOR will return a text description of the image:

json

```
{
  "type": "query_result",
  "result": "The image shows a mountain landscape with a lake in the foreground...",
  "result_type": "text",
  "processing_time": 2.45,
  "cost": 0.0012,
  "response_to": "vision_1614532345"
}
```

Complete Example

Here's how to capture an image from a webcam and process it with MTOR's vision service:

```
python
```

```python
import asyncio
import base64
import json
import cv2
import time
import numpy as np
from mtor_adapter import MTORAdapter

class VisionService:
    def __init__(self, mtor_adapter):
        self.adapter = mtor_adapter
        self.adapter.register_handler("query_result", self.handle_vision_result)
        self.pending_vision = {}

    async def capture_and_analyze(self, prompt=None):
        """Capture image from webcam and send for analysis."""
        # Capture image
        image_data = self.capture_image()

        # Convert to base64
        _, buffer = cv2.imencode('.jpg', image_data)
        image_base64 = base64.b64encode(buffer).decode('utf-8')

        # Create request ID and future
        request_id = f"vision_{int(time.time())}"
        future = asyncio.Future()
        self.pending_vision[request_id] = future

        # Send vision request
        await self.adapter.send_event({
            "type": "vision",
            "image": image_base64,
            "prompt": prompt or "Describe this image in detail",
            "id": request_id
        })

        # Wait for response
        try:
            result = await asyncio.wait_for(future, timeout=30)
            return result
        except asyncio.TimeoutError:
            del self.pending_vision[request_id]
            raise TimeoutError("Vision processing request timed out")
```

```
async def handle_vision_result(self, event):
    """Handle vision result events."""
    request_id = event.get("response_to")
    if request_id in self.pending_vision and request_id.startswith("vision_"):
        future = self.pending_vision[request_id]
        future.set_result(event.get("result"))
        del self.pending_vision[request_id]

def capture_image(self):
    """Capture image from webcam."""
    cap = cv2.VideoCapture(0)

    if not cap.isOpened():
        raise Exception("Could not open webcam")

    # Allow camera to warm up
    for i in range(5):
        ret, frame = cap.read()
        time.sleep(0.1)

    # Capture frame
    ret, frame = cap.read()

    # Release camera
    cap.release()

    if not ret:
        raise Exception("Could not capture image")

    return frame
```

Processing Existing Images

To analyze an existing image:

python

```python
async def analyze_image_file(adapter, file_path, prompt=None):
    """Analyze an existing image file using MTOR's vision service."""
    # Read image file
    with open(file_path, 'rb') as img_file:
        img_data = img_file.read()

    # Convert to base64
    image_base64 = base64.b64encode(img_data).decode('utf-8')

    # Send for analysis
    request_id = f"vision_file_{int(time.time())}"
    await adapter.send_event({
        "type": "vision",
        "image": image_base64,
        "prompt": prompt or "Describe this image in detail",
        "id": request_id
    })
```

4. Weather Services

MTOR's weather service provides current weather information based on the user's location.

Request Format

python

```python
async def get_weather(adapter, location=None):
    """Get weather information from MTOR."""
    event = {
        "type": "weather",
        "id": f"weather_{int(time.time())}"
    }

    # Add location if specified, otherwise it uses geolocation
    if location:
        event["location"] = location

    await adapter.send_event(event)
```

Response Format

json


```
{
  "type": "weather_response",
  "weather": {
    "location": "New York, NY",
    "temperature": 72,
    "description": "partly cloudy",
    "humidity": 65,
    "wind_speed": 8,
    "forecast": [
      {"day": "Today", "high": 75, "low": 68, "description": "partly cloudy"},
      {"day": "Tomorrow", "high": 77, "low": 65, "description": "sunny"}
    ]
  },
  "response_to": "weather_1614532345"
}
```


Complete Example

```
python
```

```python
import asyncio
import json
import time
from mtor_adapter import MTORAdapter

class WeatherService:
    def __init__(self, mtor_adapter):
        self.adapter = mtor_adapter
        self.adapter.register_handler("weather_response", self.handle_weather)
        self.pending_weather = {}

    async def get_current_weather(self, location=None):
        """Get current weather information."""
        # Create request ID and future
        request_id = f"weather_{int(time.time())}"
        future = asyncio.Future()
        self.pending_weather[request_id] = future

        # Create event
        event = {
            "type": "weather",
            "id": request_id
        }

        # Add location if specified
        if location:
            event["location"] = location

        # Send request
        await self.adapter.send_event(event)

        # Wait for response
        try:
            weather_data = await asyncio.wait_for(future, timeout=15)
            return weather_data
        except asyncio.TimeoutError:
            del self.pending_weather[request_id]
            raise TimeoutError("Weather request timed out")

    async def handle_weather(self, event):
        """Handle weather response events."""
        request_id = event.get("response_to")
        if request_id in self.pending_weather:
```

```
            future = self.pending_weather[request_id]
            future.set_result(event.get("weather"))
            del self.pending_weather[request_id]

    def format_weather_report(self, weather_data):
        """Format weather data into a readable report."""
        location = weather_data.get("location", "Unknown location")
        temp = weather_data.get("temperature", "N/A")
        desc = weather_data.get("description", "N/A")

        report = f"Weather report for {location}: {temp}°F, {desc}."

        if "humidity" in weather_data:
            report += f" Humidity: {weather_data['humidity']}%."

        if "wind_speed" in weather_data:
            report += f" Wind: {weather_data['wind_speed']} mph."

        if "forecast" in weather_data:
            report += "\n\nForecast:"
            for day in weather_data["forecast"]:
                report += f"\n{day['day']}: {day['description']}, high of {day['high']}°F, low

        return report
```

5. Gmail Integration

MTOR provides Gmail integration, allowing applications to read and interact with email.

Authentication

Gmail integration requires OAuth authentication. The first step is to request authorization:

python

```python
async def request_gmail_auth(adapter):
    """Request Gmail authorization through MTOR."""
    event = {
        "type": "gmail_auth_request",
        "id": f"gmail_auth_{int(time.time())}"
    }

    await adapter.send_event(event)
    # This will trigger a browser window opening for the user to authorize
```

Reading Emails

Once authenticated, you can read emails:

python

```python
async def read_emails(adapter, max_emails=10):
    """Request to read Gmail emails."""
    event = {
        "type": "gmail_read",
        "max_emails": max_emails,
        "id": f"gmail_read_{int(time.time())}"
    }

    await adapter.send_event(event)
```

Response Format

json

```
{
  "type": "gmail_response",
  "emails": [
    {
      "from": "sender@example.com",
      "subject": "Meeting Tomorrow",
      "date": "2023-04-30T15:23:45Z",
      "snippet": "Let's discuss the project..."
    },
    ...
  ],
  "response_to": "gmail_read_1614532345"
}
```

Complete Example

```
python
```

```python
import asyncio
import json
import time
from mtor_adapter import MTORAdapter

class GmailService:
    def __init__(self, mtor_adapter):
        self.adapter = mtor_adapter
        self.adapter.register_handler("gmail_auth_response", self.handle_auth)
        self.adapter.register_handler("gmail_response", self.handle_emails)
        self.pending_requests = {}
        self.is_authenticated = False

    async def authenticate(self):
        """Authenticate with Gmail."""
        if self.is_authenticated:
            return True

        # Create request ID and future
        request_id = f"gmail_auth_{int(time.time())}"
        future = asyncio.Future()
        self.pending_requests[request_id] = future

        # Send auth request
        await self.adapter.send_event({
            "type": "gmail_auth_request",
            "id": request_id
        })

        # Wait for authentication response
        try:
            auth_result = await asyncio.wait_for(future, timeout=60)  # Longer timeout for user
            self.is_authenticated = auth_result.get("success", False)
            return self.is_authenticated
        except asyncio.TimeoutError:
            del self.pending_requests[request_id]
            raise TimeoutError("Gmail authentication timed out")

    async def read_recent_emails(self, max_emails=10):
        """Read recent emails from Gmail."""
        # Ensure we're authenticated
        if not self.is_authenticated:
            auth_success = await self.authenticate()
```

```
        if not auth_success:
            raise Exception("Failed to authenticate with Gmail")

    # Create request ID and future
    request_id = f"gmail_read_{int(time.time())}"
    future = asyncio.Future()
    self.pending_requests[request_id] = future

    # Send read request
    await self.adapter.send_event({
        "type": "gmail_read",
        "max_emails": max_emails,
        "id": request_id
    })

    # Wait for response
    try:
        emails = await asyncio.wait_for(future, timeout=30)
        return emails
    except asyncio.TimeoutError:
        del self.pending_requests[request_id]
        raise TimeoutError("Gmail read request timed out")

async def handle_auth(self, event):
    """Handle Gmail authentication response."""
    request_id = event.get("response_to")
    if request_id in self.pending_requests and request_id.startswith("gmail_auth_"):
        future = self.pending_requests[request_id]
        future.set_result(event.get("audio"))
        del self.pending_requests[request_id]

# Utility methods
async def record_audio(self, duration=5, sample_rate=16000):
    """Record audio from microphone."""
    # Implementation details as in earlier example
    pass

def capture_image(self):
    """Capture image from webcam."""
    # Implementation details as in earlier example
    pass

def play_audio(self, audio_base64):
    """Play audio from base64 data."""
```

```
        # Implementation details as in earlier example
        pass
```

7. MTOR Event Bus Architecture

To fully understand how to interact with MTOR's internal services, it's important to understand the event bus architecture that powers all MTOR interactions.

Event Structure

All events in MTOR follow a consistent structure:

```json
{
  "type": "event_type",          // The type of event
  "id": "unique_request_id",     // A unique identifier for the request
  "response_to": "request_id",   // In responses, the ID of the original request
  "data": { ... }                // Event-specific data payload
}
```

Event Flow

1. **Request Events**: Your adapter sends a request event to MTOR
2. **Processing**: MTOR processes the request asynchronously
3. **Response Events**: MTOR sends one or more response events back to your adapter
4. **Event Handling**: Your adapter's registered handlers process the response events

Event Types

Here's a summary of the core MTOR service event types:

Service	Request Event Type	Response Event Type	Description
Speech-to-Text	`speech_to_text`	`transcription_result`	Convert audio to text using Whisper
Text-to-Speech	`text_to_speech`	`speech_result`	Convert text to speech using BARK
Vision	`vision`	`query_result`	Process and describe images
Weather	`weather`	`weather_response`	Get current weather information
Gmail	`gmail_auth_request`	`gmail_auth_response`	Authenticate with Gmail
Gmail	`gmail_read`	`gmail_response`	Read emails from Gmail
Chat	`chat`	`query_result`	General chat interactions

8. Advanced Integration Patterns

Parallel Processing

For efficiency, you can send multiple service requests in parallel:

python

```python
async def multimodal_analysis(adapter, image_base64, audio_base64):
    """Process image and audio in parallel."""
    # Create futures for both requests
    vision_future = asyncio.Future()
    speech_future = asyncio.Future()

    # Generate request IDs
    vision_id = f"vision_{int(time.time())}"
    speech_id = f"speech_{int(time.time())}"

    # Store futures
    adapter.pending_requests[vision_id] = vision_future
    adapter.pending_requests[speech_id] = speech_future

    # Send requests in parallel
    await asyncio.gather(
        adapter.send_event({
            "type": "vision",
            "image": image_base64,
            "prompt": "Describe this image",
            "id": vision_id
        }),
        adapter.send_event({
            "type": "speech_to_text",
            "audio": audio_base64,
            "id": speech_id
        })
    )

    # Wait for both results
    vision_result, speech_result = await asyncio.gather(
        asyncio.wait_for(vision_future, timeout=30),
        asyncio.wait_for(speech_future, timeout=30)
    )

    # Combine the results
    return {
        "vision": vision_result,
        "speech": speech_result
    }
```

Event Streaming

For long-running processes, you can implement event streaming:

```
python
```

```python
class StreamingAdapter(MTORAdapter):
    """Adapter that handles streaming events."""

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.streams = {}

    def register_stream_handler(self, stream_id, handler):
        """Register a handler for a specific stream."""
        self.streams[stream_id] = handler

    async def start_stream(self, stream_type, params=None):
        """Start a new event stream."""
        stream_id = f"stream_{stream_type}_{int(time.time())}"

        await self.send_event({
            "type": f"start_{stream_type}_stream",
            "stream_id": stream_id,
            "params": params or {},
            "id": f"start_{stream_id}"
        })

        return stream_id

    async def stop_stream(self, stream_id):
        """Stop an active stream."""
        await self.send_event({
            "type": "stop_stream",
            "stream_id": stream_id,
            "id": f"stop_{stream_id}"
        })

        if stream_id in self.streams:
            del self.streams[stream_id]

    async def _handle_event(self, event_type, event):
        """Override to handle stream events."""
        # Check if this is a stream event
        if event_type == "stream_data" and "stream_id" in event:
            stream_id = event["stream_id"]
            if stream_id in self.streams:
                handler = self.streams[stream_id]
                await handler(event)
```

```
            return None

        # Otherwise, use normal event handling
        return await super()._handle_event(event_type, event)
```

This could be used for streaming speech recognition:

```
python
```

```python
async def continuous_speech_recognition():
    """Demonstrate continuous speech recognition."""
    adapter = StreamingAdapter(mtor_websocket_url, "streaming_demo", "")
    await adapter.connect()

    # Stream handler function
    async def handle_stream(event):
        transcript = event.get("transcript", "")
        is_final = event.get("is_final", False)

        if is_final:
            print(f"FINAL: {transcript}")
        else:
            print(f"Interim: {transcript}")

    # Start streaming
    stream_id = await adapter.start_stream("speech", {
        "interim_results": True,
        "single_utterance": False
    })

    # Register handler
    adapter.register_stream_handler(stream_id, handle_stream)

    # Start microphone capture
    # ... (microphone setup code) ...

    try:
        # Run for 60 seconds
        for _ in range(60):
            # Send audio chunks
            chunk = await get_audio_chunk()
            audio_base64 = base64.b64encode(chunk).decode('utf-8')

            await adapter.send_event({
                "type": "stream_audio",
                "stream_id": stream_id,
                "audio": audio_base64,
                "id": f"chunk_{time.time()}"
            })

            await asyncio.sleep(1)
    finally:
```

```
        # Stop streaming
        await adapter.stop_stream(stream_id)
        await adapter.disconnect()
```

9. Error Handling and Resilience

When working with MTOR's services, implement proper error handling:

Service-Specific Error Events

MTOR services may return error events:

```json
{
  "type": "error",
  "error": "Error message describing what went wrong",
  "error_code": "SERVICE_SPECIFIC_ERROR_CODE",
  "response_to": "original_request_id"
}
```

Implementing Retries

For transient errors, implement a retry mechanism:

```python
async def retry_with_backoff(coro, max_retries=3, initial_delay=1):
    """Execute a coroutine with exponential backoff retries."""
    retries = 0
    delay = initial_delay

    while True:
        try:
            return await coro
        except (TimeoutError, ConnectionError) as e:
            retries += 1
            if retries > max_retries:
                raise Exception(f"Failed after {max_retries} attempts: {str(e)}")

            # Exponential backoff
            await asyncio.sleep(delay)
            delay *= 2
```

Circuit Breaker Pattern

For more advanced resilience, implement a circuit breaker:

python

```python
class CircuitBreaker:
    """Circuit breaker for MTOR service calls."""

    def __init__(self, failure_threshold=5, reset_timeout=60):
        self.failure_count = 0
        self.failure_threshold = failure_threshold
        self.reset_timeout = reset_timeout
        self.state = "closed"  # closed, open, half-open
        self.last_failure_time = 0

    async def execute(self, coro):
        """Execute a coroutine with circuit breaker protection."""
        if self.state == "open":
            # Check if it's time to try again
            if time.time() - self.last_failure_time > self.reset_timeout:
                self.state = "half-open"
            else:
                raise Exception("Circuit breaker is open")

        try:
            result = await coro

            # Success - reset if in half-open state
            if self.state == "half-open":
                self.state = "closed"
                self.failure_count = 0

            return result
        except Exception as e:
            # Failure
            self.last_failure_time = time.time()
            self.failure_count += 1

            if self.failure_count >= self.failure_threshold:
                self.state = "open"

            raise e
```

Usage with services:

```python
# Create circuit breakers for different services
speech_breaker = CircuitBreaker(failure_threshold=3, reset_timeout=30)
vision_breaker = CircuitBreaker(failure_threshold=3, reset_timeout=30)

async def transcribe_with_protection(adapter, audio_base64):
    """Transcribe audio with circuit breaker protection."""
    async def transcribe():
        # ... transcription code ...
        pass

    try:
        return await speech_breaker.execute(transcribe())
    except Exception as e:
        if "Circuit breaker is open" in str(e):
            return "Speech service is currently unavailable. Please try again later."
        else:
            raise
```

10. Real-World Integration Examples

Here are some complete examples that show how to integrate MTOR's core services into real applications:

Voice Assistant App

```
python
```

```python
class MTORVoiceAssistant:
    """Voice assistant powered by MTOR services."""

    def __init__(self, mtor_adapter):
        self.adapter = mtor_adapter
        self.setup_handlers()

    def setup_handlers(self):
        """Register event handlers."""
        self.adapter.register_handler("transcription_result", self.handle_transcription)
        self.adapter.register_handler("speech_result", self.handle_speech)
        self.adapter.register_handler("query_result", self.handle_query)
        self.adapter.register_handler("weather_response", self.handle_weather)

    async def run_assistant(self):
        """Run the voice assistant."""
        # Start listening loop
        while True:
            try:
                # Listen for wake word
                wake_word_detected = await self.detect_wake_word()
                if not wake_word_detected:
                    continue

                # Play activation sound
                self.play_activation_sound()

                # Listen for command
                command = await self.listen_for_command()
                if not command:
                    continue

                # Process command
                await self.process_command(command)

            except Exception as e:
                print(f"Error in assistant loop: {str(e)}")

    async def detect_wake_word(self):
        """Listen for wake word."""
        # In a real implementation, this would use a local wake word detector
        # For this example, we'll simulate it
        return True
```

```python
async def listen_for_command(self):
    """Listen for and transcribe a command."""
    audio_data = await self.record_audio(5)  # 5 seconds

    # Convert to base64
    audio_base64 = base64.b64encode(audio_data).decode('utf-8')

    # Create tracking future
    future = asyncio.Future()
    request_id = f"cmd_{int(time.time())}"
    self.pending[request_id] = future

    # Send to MTOR for transcription
    await self.adapter.send_event({
        "type": "speech_to_text",
        "audio": audio_base64,
        "id": request_id
    })

    # Wait for result
    try:
        result = await asyncio.wait_for(future, timeout=10)
        return result
    except asyncio.TimeoutError:
        return None

async def process_command(self, command):
    """Process a voice command."""
    command = command.lower()

    if "weather" in command:
        await self.get_weather()
    elif "what do you see" in command or "take a picture" in command:
        await self.describe_surroundings()
    elif "read my email" in command or "check my mail" in command:
        await self.read_emails()
    else:
        # Default to general chat
        await self.chat_response(command)

async def speak(self, text):
    """Convert text to speech and play it."""
    # Create tracking future
```

```
        future = asyncio.Future()
        request_id = f"speak_{int(time.time())}"
        self.pending[request_id] = future

        # Send to MTOR for speech synthesis
        await self.adapter.send_event({
            "type": "text_to_speech",
            "text": text,
            "id": request_id
        })

        # Wait for result
        try:
            audio_base64 = await asyncio.wait_for(future, timeout=15)
            # Play the audio
            self.play_audio(audio_base64)
        except asyncio.TimeoutError:
            print("Speech synthesis timed out")

    # Event handlers and other methods would be implemented here
```

Multimodal Search App

```
python
```

```python
class MTORMultimodalSearch:
    """App that can search using text, voice, or images."""

    def __init__(self, mtor_adapter):
        self.adapter = mtor_adapter
        self.setup_handlers()

    def setup_handlers(self):
        """Register event handlers."""
        # Register handlers for the events your app will receive
        pass

    async def search_by_voice(self):
        """Search using voice input."""
        # Record audio
        audio_data = self.record_audio()
        audio_base64 = base64.b64encode(audio_data).decode('utf-8')

        # Transcribe
        transcription = await self.transcribe_audio(audio_base64)

        # Use transcription for search
        await self.search_by_text(transcription)

    async def search_by_image(self):
        """Search using an image."""
        # Capture or select image
        image_data = self.capture_image()
        image_base64 = base64.b64encode(image_data).decode('utf-8')

        # Get image description
        description = await self.describe_image(image_base64)

        # Use description for search
        await self.search_by_text(description)

    async def search_by_text(self, query):
        """Search using text input."""
        # Send chat query to MTOR
        response = await self.chat_query(query)

        # Speak the response
        await self.speak_result(response)
```

```
    async def transcribe_audio(self, audio_base64):
        """Transcribe audio using MTOR's speech-to-text."""
        # Implementation details
        pass

    async def describe_image(self, image_base64):
        """Get image description using MTOR's vision service."""
        # Implementation details
        pass

    async def chat_query(self, query):
        """Send a chat query to MTOR."""
        # Implementation details
        pass

    async def speak_result(self, text):
        """Speak a result using MTOR's text-to-speech."""
        # Implementation details
        pass
```

Conclusion

MTOR's internal services provide a powerful foundation for building sophisticated, multimodal applications. By following the patterns and examples in this guide, you can leverage MTOR's speech, vision, weather, and email capabilities through its event-driven architecture.

The key benefits of this approach include:

1. **Unified Interface**: Access diverse AI capabilities through a consistent event-based interface
2. **Stateless Design**: Clean, maintainable applications with minimal state management
3. **Non-blocking Operations**: All operations happen asynchronously, keeping your application responsive
4. **Composability**: Combine services in powerful ways to create unique experiences

Remember that all requests to MTOR's services should be structured as events with the appropriate event type and a unique request ID. Responses will come back as events with a `response_to` field matching your request ID.

By implementing the patterns demonstrated in this guide, you'll be able to create sophisticated applications that leverage the full power of MTOR's core services while maintaining the event-driven, stateless philosophy that makes MTOR so flexible and performant.

Additional Resources

- MTOR Event Bus Documentation
- WebSockets in Python
- AsyncIO Programming
- BARK Text-to-Speech
- Whisper Speech-to-Text (event) del self.pending_requests[request_id] async def handle_emails(self, event): """Handle Gmail email response.""" request_id = event.get("response_to") if request_id in self.pending_requests and request_id.startswith("gmail_read_"): future = self.pending_requests[request_id] future.set_result(event.get("emails", [])) del self.pending_requests[request_id] def format_email_summary(self, emails): """Format emails into a readable summary.""" if not emails: return "No emails found."

```
    summary = f"Found {len(emails)} recent emails:\n\n"

    for i, email in enumerate(emails, 1):
        from_addr = email.get("from", "Unknown sender")
        subject = email.get("subject", "No subject")
        date = email.get("date", "Unknown date")
        snippet = email.get("snippet", "")

        summary += f"{i}. From: {from_addr}\n"
        summary += f"   Subject: {subject}\n"
        summary += f"   Date: {date}\n"
        if snippet:
            summary += f"   Preview: {snippet[:100]}...\n"
        summary += "\n"

    return summary
```

6. Combining Services

One of MTOR's strengths is the ability to combine services. Here's an example that uses speech, vision, and text-to-speech together:

```python
import asyncio
import base64
import json
import time
import cv2
import pyaudio
import wave
import io
import pygame
from mtor_adapter import MTORAdapter

class MultimodalAssistant:
    def __init__(self, mtor_adapter):
        self.adapter = mtor_adapter
        self.setup_handlers()
        self.pending_requests = {}
        pygame.mixer.init()

    def setup_handlers(self):
        """Register event handlers."""
        self.adapter.register_handler("transcription_result", self.handle_transcription)
        self.adapter.register_handler("query_result", self.handle_query_result)
        self.adapter.register_handler("speech_result", self.handle_speech)

    async def voice_controlled_assistant(self):
        """Run a voice-controlled assistant that can see and speak."""
        while True:
            # Prompt the user
            await self.speak("How can I help you today?")

            # Listen for the user's command
            command = await self.listen()
            print(f"Heard: {command}")

            # Process based on the command
            if "take a picture" in command.lower() or "what do you see" in command.lower():
```

```
            await self.visual_description()
        elif "weather" in command.lower():
            await self.get_weather()
        elif "quit" in command.lower() or "exit" in command.lower():
            await self.speak("Goodbye!")
            break
        else:
            # Default to chat
            await self.chat_response(command)

async def listen(self):
    """Record audio and transcribe it."""
    # Record audio
    audio_data = await self.record_audio(5)  # 5 seconds

    # Convert to base64
    audio_base64 = base64.b64encode(audio_data).decode('utf-8')

    # Create request ID and future
    request_id = f"transcribe_{int(time.time())}"
    future = asyncio.Future()
    self.pending_requests[request_id] = future

    # Send transcription request
    await self.adapter.send_event({
        "type": "speech_to_text",
        "audio": audio_base64,
        "id": request_id
    })

    # Wait for transcription
    try:
        result = await asyncio.wait_for(future, timeout=30)
        return result
    except asyncio.TimeoutError:
        del self.pending_requests[request_id]
        return "I didn't hear anything clearly."

async def speak(self, text):
    """Convert text to speech and play it."""
    # Create request ID and future
    request_id = f"tts_{int(time.time())}"
    future = asyncio.Future()
    self.pending_requests[request_id] = future
```

```
    # Send TTS request
    await self.adapter.send_event({
        "type": "text_to_speech",
        "text": text,
        "id": request_id
    })

    # Wait for speech
    try:
        audio_data = await asyncio.wait_for(future, timeout=30)
        # Play the audio
        self.play_audio(audio_data)
        # Wait for audio to finish
        while pygame.mixer.music.get_busy():
            pygame.time.Clock().tick(10)
    except asyncio.TimeoutError:
        del self.pending_requests[request_id]
        print("Speech synthesis timed out")

async def visual_description(self):
    """Capture an image and get a description."""
    await self.speak("Taking a picture now.")

    # Capture image
    image_data = self.capture_image()

    # Convert to base64
    _, buffer = cv2.imencode('.jpg', image_data)
    image_base64 = base64.b64encode(buffer).decode('utf-8')

    # Create request ID and future
    request_id = f"vision_{int(time.time())}"
    future = asyncio.Future()
    self.pending_requests[request_id] = future

    # Send vision request
    await self.adapter.send_event({
        "type": "vision",
        "image": image_base64,
        "prompt": "Describe what you see in this image in detail.",
        "id": request_id
    })
```

```
        # Wait for description
        try:
            description = await asyncio.wait_for(future, timeout=30)
            await self.speak(description)
        except asyncio.TimeoutError:
            del self.pending_requests[request_id]
            await self.speak("I'm having trouble analyzing the image.")

    async def get_weather(self):
        """Get and speak the current weather."""
        request_id = f"weather_{int(time.time())}"
        future = asyncio.Future()
        self.pending_requests[request_id] = future

        # Send weather request
        await self.adapter.send_event({
            "type": "weather",
            "id": request_id
        })

        # Wait for weather data
        try:
            weather_data = await asyncio.wait_for(future, timeout=15)

            # Format weather info
            location = weather_data.get("location", "Unknown location")
            temp = weather_data.get("temperature", "Unknown")
            desc = weather_data.get("description", "Unknown")

            weather_text = f"The current weather in {location} is {temp} degrees and
{desc}."

            await self.speak(weather_text)
        except asyncio.TimeoutError:
            del self.pending_requests[request_id]
            await self.speak("I'm having trouble getting the weather information.")

    async def chat_response(self, query):
        """Get a chat response to a query."""
        request_id = f"chat_{int(time.time())}"
        future = asyncio.Future()
        self.pending_requests[request_id] = future

        # Send chat request
```

```
    await self.adapter.send_event({
        "type": "chat",
        "prompt": query,
        "id": request_id
    })

    # Wait for response
    try:
        response = await asyncio.wait_for(future, timeout=30)
        await self.speak(response)
    except asyncio.TimeoutError:
        del self.pending_requests[request_id]
        await self.speak("I'm having trouble coming up with a response.")

# Event handlers
async def handle_transcription(self, event):
    """Handle transcription result events."""
    request_id = event.get("response_to")
    if request_id in self.pending_requests and request_id.startswith("transcribe_"):
        future = self.pending_requests[request_id]
        future.set_result(event.get("text"))
        del self.pending_requests[request_id]

async def handle_query_result(self, event):
    """Handle query result events (chat, vision, etc.)."""
    request_id = event.get("response_to")
    if request_id in self.pending_requests:
        future = self.pending_requests[request_id]
        future.set_result(event.get("result"))
        del self.pending_requests[request_id]

async def handle_speech(self, event):
    """Handle speech result events."""
    request_id = event.get("response_to")
    if request_id in self.pending_requests and request_id.startswith("tts_"):
        future = self.pending_requests[request_id]
        future.set_result
```