🤖 Agent Context API

> **Real-time communication with imaginex platform agents**

Send messages to your imaginex platform agents in real-time, make them speak proactively, or add ba
their responses.

⚠️ **Note**: This API is for agents created and deployed on the imaginex.bithuman.ai platform, not for lo

✨ What is Agent Context API?

The Agent Context API allows you to interact with your imaginex platform agents in real-time:

💬 **Make agents speak** → Trigger proactive speech to users
🧠 **Add background knowledge** → Enhance agent responses with context
🎯 **Target specific rooms** → Send messages to individual sessions
⚡ **Real-time delivery** → Instant communication with active agents

Perfect for: Live agent control, dynamic content updates, personalized interactions, customer service a
agents

🚀 Quick Start

Prerequisites

- Agent created and deployed on imaginex.bithuman.ai platform
- Agent code identifier from your imaginex dashboard
- Valid API secret from imaginex.bithuman.ai
- Agent actively running in a LiveKit session (not local SDK agents)

Base URL

```
https://your-api-endpoint.com/v1/agent/{agent_code}
```

> **Note**: `{agent_code}` is the unique identifier of your agent from the imaginex platform dashb

📡 API Endpoints

🗣️ Make Agent Speak

Make your agent speak a message proactively to users in the session.

```
POST /v1/agent/{agent_code}/speak
```

Request

```
{
  "message": "Hello! I have an important update for you.",
  "room_id": "room_123"  // Optional: target specific room
}
```

Headers

```
Content-Type: application/json
api-secret: your_api_secret_here
```

Response

```
{
  "success": true,
  "message": "Speech triggered successfully",
  "data": {
    "agent_code": "A12345678",
    "delivered_to_rooms": 1,
    "timestamp": "2024-01-15T10:30:00Z"
  }
}
```

Example Usage

```
import requests

# Make imaginex platform agent announce a promotion
# Note: A12345678 is your agent code from imaginex dashboard
response = requests.post(
    'https://api.example.com/v1/agent/A12345678/speak',
    headers={'api-secret': 'your_secret'},
    json={
        'message': 'Great news! We have a 20% discount available today!',
        'room_id': 'customer_session_1'
    }
)

if response.json()['success']:
    print("Imaginex agent spoke successfully!")
```

```
// JavaScript/Node.js - Control imaginex platform agent
// Note: A12345678 is your agent code from imaginex dashboard
const response = await fetch('/v1/agent/A12345678/speak', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
    'api-secret': 'your_secret'
  },
  body: JSON.stringify({
    message: 'Your order has been confirmed and will arrive tomorrow!',
    room_id: 'order_confirmation_room'
  })
});

const result = await response.json();
console.log('Imaginex agent speech result:', result);
```

🧠 Add Background Context

Add background knowledge to your agent without triggering speech. The agent will use this informatic

```
POST /v1/agent/{agent_code}/add-context
```

Request

```
{
  "context": "User John Smith is a premium customer who prefers email communicat
  "type": "add_context",
  "room_id": "room_456"  // Optional: target specific room
}
```

Headers

```
Content-Type: application/json
api-secret: your_api_secret_here
```

Response

```
{
  "success": true,
  "message": "Context added successfully",
  "data": {
    "agent_code": "A12345678",
    "context_type": "add_context",
    "delivered_to_rooms": 1,
    "timestamp": "2024-01-15T10:35:00Z"
  }
}
```

Example Usage

```python
import requests

# Add customer context to imaginex platform agent
# Note: A12345678 is your agent code from imaginex dashboard
response = requests.post(
    'https://api.example.com/v1/agent/A12345678/add-context',
    headers={'api-secret': 'your_secret'},
    json={
        'context': 'Customer has VIP status and prefers technical explanations',
        'type': 'add_context',
        'room_id': 'vip_customer_session'
    }
)

print("Context added to imaginex agent:", response.json())
```

```javascript
// Add context about user preferences to imaginex platform agent
// Note: A12345678 is your agent code from imaginex dashboard
const response = await fetch('/v1/agent/A12345678/add-context', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
    'api-secret': 'your_secret'
  },
  body: JSON.stringify({
    context: 'User is interested in enterprise features and has a team of 50+ peo
    type: 'add_context'
  })
});
```

🎯 Unified Context Endpoint (Advanced)

For advanced use cases, you can use the unified endpoint that supports both speech and context addit

```
POST /v1/agent/{agent_code}/add-context
```

Request for Speech

```json
{
  "context": "Thank you for your patience. Your issue has been resolved!",
  "type": "speak",
  "room_id": "support_session_1"
}
```

Request for Background Context

```json
{
  "context": "Customer reported billing issue #12345 on January 10th",
  "type": "add_context",
  "room_id": "support_session_1"
}
```

🎨 Use Cases & Examples

📢 Announcements & Notifications

```python
# System maintenance announcement to imaginex platform agents
# Note: SUPPORT_AGENT is your agent code from imaginex dashboard
def notify_maintenance():
    requests.post('/v1/agent/SUPPORT_AGENT/speak',
        headers={'api-secret': API_SECRET},
        json={
            'message': 'We will have scheduled maintenance tonight from 2-4 AM ES
        }
    )

# Flash sale notification to sales agent on imaginex platform
def announce_sale():
    requests.post('/v1/agent/SALES_AGENT/speak',
        headers={'api-secret': API_SECRET},
        json={
            'message': 'Flash Sale Alert! 50% off all premium plans for the next
        }
    )
```

🎯 Personalized Customer Service

```python
# Add customer context when they join - for imaginex platform agents
# Note: agent_code should be from your imaginex dashboard (e.g., A12345678)
def setup_customer_context(agent_code, customer_data, room_id):
    context = f"""
    Customer: {customer_data['name']}
    Account Type: {customer_data['tier']}
    Last Purchase: {customer_data['last_order']}
    Preferred Contact: {customer_data['contact_method']}
    Previous Issues: {customer_data['support_history']}
    """

    requests.post(f'/v1/agent/{agent_code}/add-context',
        headers={'api-secret': API_SECRET},
        json={
            'context': context,
            'type': 'add_context',
            'room_id': room_id
        }
    )

# Proactive issue resolution with imaginex platform agent
def proactive_support_followup(agent_code, issue_id):
    requests.post(f'/v1/agent/{agent_code}/speak',
        headers={'api-secret': API_SECRET},
        json={
            'message': f'I see you had issue #{issue_id} last week. Is everything
        }
    )
```

🔄 Dynamic Content Updates

```
# Update product information for imaginex platform agents
# Note: agent_code should be from your imaginex dashboard
def update_product_knowledge(agent_code, product_updates):
    for update in product_updates:
        requests.post(f'/v1/agent/{agent_code}/add-context',
            headers={'api-secret': API_SECRET},
            json={
                'context': f"Product Update: {update['product']} now has {update
                'type': 'add_context'
            }
        )

# Live event updates via imaginex platform agent
def broadcast_event_update(agent_code, event_info):
    requests.post(f'/v1/agent/{agent_code}/speak',
        headers={'api-secret': API_SECRET},
        json={
            'message': f"Event Update: {event_info['title']} starts in {event_in
        }
    )
```

🔧 Integration Patterns

⚡ Real-time Webhooks + Agent Context

```
from flask import Flask, request
import requests

app = Flask(__name__)

@app.route('/webhook/order-confirmed', methods=['POST'])
def handle_order_confirmation():
    order_data = request.json

    # Add order context to agent
    requests.post(f'/v1/agent/{order_data["agent_code"]}/add-context',
        headers={'api-secret': API_SECRET},
        json={
            'context': f"Customer just placed order #{order_data['order_id']} fo
            'type': 'add_context',
            'room_id': order_data['session_id']
        }
    )

    # Make agent speak confirmation
    requests.post(f'/v1/agent/{order_data["agent_code"]}/speak',
        headers={'api-secret': API_SECRET},
        json={
            'message': f"Great! Your order #{order_data['order_id']} has been co
            'room_id': order_data['session_id']
        }
    )

    return {'status': 'success'}
```

🎯 CRM Integration

```
# Sync customer data with agent context
def sync_crm_data(customer_id, agent_code, room_id):
    # Fetch from CRM
    customer = crm_client.get_customer(customer_id)

    # Format context for agent
    context = f"""
    Customer Profile:
    - Name: {customer.name}
    - Tier: {customer.tier}
    - Lifetime Value: ${customer.ltv}
    - Satisfaction Score: {customer.satisfaction}/10
    - Recent Activity: {customer.recent_activity}
    - Preferences: {customer.preferences}
    """

    # Send to agent
    requests.post(f'/v1/agent/{agent_code}/add-context',
        headers={'api-secret': API_SECRET},
        json={
            'context': context,
            'type': 'add_context',
            'room_id': room_id
        }
    )
```

📊 Analytics-Driven Interactions

```
# Trigger proactive engagement based on analytics
def analytics_driven_engagement():
    # Check user behavior analytics
    users_about_to_churn = analytics.get_churn_risk_users()

    for user in users_about_to_churn:
        # Add context about user's situation
        requests.post(f'/v1/agent/{user["assigned_agent"]}/add-context',
            headers={'api-secret': API_SECRET},
            json={
                'context': f"User {user['name']} has {user['churn_risk']}% churn
                'type': 'add_context',
                'room_id': user['session_id']
            }
        )

        # Proactive outreach
        requests.post(f'/v1/agent/{user["assigned_agent"]}/speak',
            headers={'api-secret': API_SECRET},
            json={
                'message': f"Hi {user['name']}! I noticed you haven't been active
                'room_id': user['session_id']
            }
        )
```

🛡️ Error Handling

Common Error Responses

```
// Agent not found
{
  "success": false,
  "error": "AGENT_NOT_FOUND",
  "message": "Agent with code 'A12345678' not found"
}

// Invalid API secret
{
  "success": false,
  "error": "UNAUTHORIZED",
  "message": "Invalid api-secret"
}

// No active sessions
{
  "success": false,
  "error": "NO_ACTIVE_ROOMS",
  "message": "No active sessions found for agent"
}

// Invalid context type
{
  "success": false,
  "error": "VALIDATION_ERROR",
  "message": "Invalid type. Must be one of: speak, add_context"
}
```

Error Handling Best Practices

```
def safe_agent_speak(agent_code, message, room_id=None):
    try:
        response = requests.post(
            f'/v1/agent/{agent_code}/speak',
            headers={'api-secret': API_SECRET},
            json={'message': message, 'room_id': room_id},
            timeout=10
        )

        if response.status_code == 200:
            return response.json()
        elif response.status_code == 404:
            print(f"Agent {agent_code} not found or no active sessions")
        elif response.status_code == 401:
            print("Invalid API credentials")
        else:
            print(f"Unexpected error: {response.status_code}")

    except requests.exceptions.Timeout:
        print("Request timed out")
    except requests.exceptions.RequestException as e:
        print(f"Request failed: {e}")

    return None
```

📈 Best Practices

🎯 Context Management

- **Be specific**: Provide clear, actionable context information
- **Stay relevant**: Only add context that affects current interactions
- **Update regularly**: Refresh context as situations change
- **Organize data**: Structure context for easy agent comprehension

💬 Speech Optimization

- **Natural language**: Write messages as if the agent is speaking directly
- **Appropriate timing**: Don't interrupt ongoing conversations
- **User value**: Ensure proactive messages provide real value
- **Frequency control**: Avoid overwhelming users with too many messages

🔧 Technical Best Practices

- **Retry logic**: Implement retries for network failures
- **Rate limiting**: Don't exceed API rate limits
- **Monitoring**: Track delivery success rates
- **Security**: Secure your API secrets properly

🚀 Advanced Features

🎛️ Room Targeting

Target specific rooms when agents handle multiple concurrent sessions:

```
# Send different messages to different rooms
rooms = ['room_1', 'room_2', 'room_3']
messages = ['VIP customer message', 'Standard message', 'Trial user message']

for room, message in zip(rooms, messages):
    requests.post(f'/v1/agent/{AGENT_CODE}/speak',
        headers={'api-secret': API_SECRET},
        json={'message': message, 'room_id': room}
    )
```

📊 Delivery Tracking

Monitor message delivery across your agent fleet:

```
def track_delivery_success():
    results = []
    for agent_code in ACTIVE_AGENTS:
        response = safe_agent_speak(agent_code, "System check message")
        results.append({
            'agent': agent_code,
            'success': response is not None,
            'timestamp': datetime.now()
        })
    return results
```

🔄 Batch Operations

Send context to multiple agents efficiently:

```
def batch_context_update(agent_codes, context_data):
    """Update multiple agents with new context"""
    for agent_code in agent_codes:
        requests.post(f'/v1/agent/{agent_code}/add-context',
            headers={'api-secret': API_SECRET},
            json={
                'context': context_data,
                'type': 'add_context'
            }
        )
```

🎉 Ready to Get Started?

1. 🔑 **Get your API secret** - Visit imaginex.bithuman.ai
2. 🤖 **Create and deploy an agent** - Create your agent on the imaginex platform
3. 📋 **Get your agent code** - Find the agent code (e.g., A12345678) in your imaginex dashboard
4. 🧪 **Test the APIs** - Try the examples above with your imaginex agent
5. 🚀 **Build integrations** - Connect your systems for real-time interaction with platform agents

Need Help?

- 💬 **Community support:** Discord

Start building real-time interactions with your imaginex platform agents today! 🎯

‹ Previous
Agent Generation API
☁️ CLOUD SERVICES