

Agentic Messaging Patterns in MCP

Caitie McCaffrey

- MCP Core Maintainer
- Member of Technical Staff @ Microsoft



August 2026 MCP Roadmap

Agentic Messaging Primitives

Agentic workloads no longer fit the request-and-response pattern

Http Native Transport

Explore unification of Streamable Http & Stdio

Agent Identity & Enterprise Ready Security

Improve Protocol Governance

Improved Primitives

Improving core primitives around tool calls

Messaging Patterns

A **messaging pattern** defines the **shape and direction of communication** between participants in a protocol, independent of the data being exchanged

Sync vs. Async Messaging

The distinction is about whether progress depends on an immediate response, not whether the transport stays open.

Sync

The initiator waits for the interaction to complete before it can proceed.

Coupled in time

The request and result belong to the same active interaction.

Completion signal

A response tells the initiator the operation has finished.

REQUEST → RESPONSE

Async

The initiator can continue while completion or new events arrive later.

Decoupled in time

The producer and consumer do not need to progress together.

Deferred delivery

Results or events arrive through polling, streaming, or callbacks.

START → CONTINUE → RECEIVE LATER

**Base Protocol**

Overview

Versioning and Compatibility

Message Patterns ▾

Overview

Multi Round-Trip Requests

Subscriptions

Cancellation

Progress

Transports >

Authorization >

Client Features

Message Patterns

Overview

📄 Copy page



This page defines the message patterns of the core protocol: the ways a client and server compose JSON-RPC **requests, responses, and notifications** into interactions. Every **transport** carries all of these patterns; transports differ only in how messages are framed and delivered.

Every interaction begins with the client:

- The **client** sends JSON-RPC *requests* and *notifications*.
- The **server** answers each request with a JSON-RPC *response* (a result or error), optionally preceded by *notifications* scoped to that request.

Servers **MUST NOT** initiate JSON-RPC requests, and clients do not send JSON-RPC responses.

**Base Protocol**

Overview

Versioning and Compatibility

Message Patterns ▾

Overview

Multi Round-Trip Requests

Subscriptions

Cancellation

Progress

Transports >

Authorization >

Client Features**Message Patterns**

Overview

📄 Copy page



This page defines the message patterns of the core protocol: the ways a client and server compose JSON-RPC requests, responses, and notifications into interactions. Every transport carries all of these patterns; transports differ only in how messages are framed and delivered.

Every interaction begins with the client:

- The **client** sends JSON-RPC *requests* and *notifications*.
- The **server** answers each request with a JSON-RPC *response* (a result or error), optionally preceded by *notifications* scoped to that request.

Servers **MUST NOT** initiate JSON-RPC requests, and clients do not send JSON-RPC responses.

Request-Response

A client sends a **request** and waits for a **corresponding response** from the server.

Multi Round-Trip Requests

Extends request/response with a structured retry loop **when additional input is required**. The server indicates that it cannot complete the request without additional input. The client collects that input and retries the request, allowing the operation to be reevaluated and completed

Subscriptions

Subscribe & Notify. A **client registers interest** in a resource or event, and the server **sends async notifications** when relevant changes occur.



Async

Tasks Extension

Async Request-Response with lifecycle management for Long Running Operations



Async

Events Extension

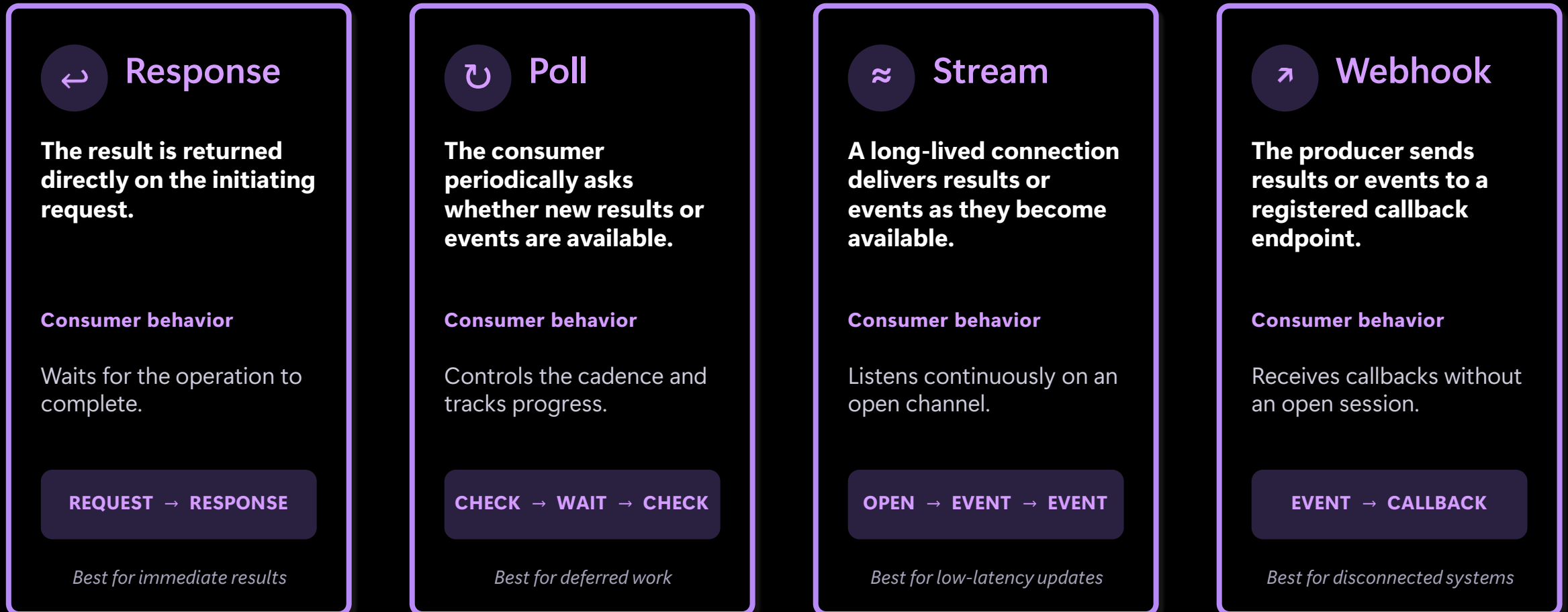
Clients Subscribe to events from an MCP Server, the **Server Publishes** events. Events are delivered via **Poll, Stream or Webhook**.



Async

How Messages Are Delivered

The messaging pattern defines the interaction; the delivery mode defines how the result or event reaches the consumer.



Delivery modes are orthogonal to messaging patterns: the same asynchronous pattern may support polling, streaming, or webhook delivery.

MCP Message Patterns

Messaging pattern	Sync / Async	Shape	Event / Result delivery	MCP examples
Request / Response	Sync	Request → Response	Response	Tools, Resources, Prompts
Multi-Round-Trip Request	Sync, multi-turn	Request → Input Required → Request + Input → Response	Response	Elicitation, Sampling, Roots
Tasks <i>(Long Running Operations)</i>	Async	Request → Task → Check Status → Result	Poll + notifications	Tasks Extension
Subscriptions	Async	Subscribe → Notifications	Stream	Resource updates, list changes
Events <i>(Publish/Subscribe)</i>	Async	Subscribe → Event(s)	Poll / Stream / Webhook	Events Extension

MCP is evolving from sync
request/response protocol into a
protocol supporting async long-running,
streaming, and event-driven interactions

August 2026 MCP Roadmap

Agentic Messaging Primitives

Modern agentic workloads no longer fit the standard request-and-response pattern. Loops can run for longer, servers can push streamed results, and there is a clear need to steer work mid-flight. MCP has been growing to meet these requirements, introducing Tasks, subscriptions/listen, progress notifications and Events. **We want to make sure that we not only offer the right primitives for the job, but also that they work well together.**

Agentic Messaging Patterns

01

Long-running work

Start work now and receive the outcome later, manage lifecycle, without holding a single request open.

Tasks

02

Progress & intermediary results

Expose checkpoints, partial outputs, and evidence so consumers can understand what is happening.

Extend Tasks + Events +
Progress Notifications

03

Steer & Adjust

Accept new context, corrections, priorities, or cancellation while execution is still active.

Extend Tasks

Get Involved

- Tasks & Events Working Group
- Join the community & Discord
- Share your feedback & scenarios.