

API Reference

Table of Contents

Core Services

IServiceContainer

IConfigurationManager

IHealthMonitor

Agent Services

IAgentRegistry

AgentDeploymentService

AgentManagementService

Communication Services

SocketIOService

Event Types

Project Services

ProjectAnalyzer

ProjectRegistry

Infrastructure Services

LoggingService

MonitoringService

CLI API

Main CLI

Agent Commands

Auto-Configuration

Local Deployment

Hook API

HookContext

HookResult

Hook Registration

API Reference

Complete API documentation for Claude MPM services.

Table of Contents

- [Core Services](#)
- [Agent Services](#)
- [Communication Services](#)
- [Project Services](#)
- [Infrastructure Services](#)
- [CLI API](#)
- [Hook API](#)

Core Services

IServiceContainer

Dependency injection container.

```
from claudemppm.services.core.interfaces import IServiceContainer

class IServiceContainer:
    def register(
        self,
        service_type: type,
        implementation: type,
        singleton: bool = True
    ) -> None:
        """Register service implementation."""

    def resolve(self, service_type: type) -> Any:
        """Resolve service instance."""

    def is_registered(self, service_type: type) -> bool:
        """Check if service is registered."""
```

IConfigurationManager

Configuration management.

```
from claudemppm.services.core.interfaces import
    IConfigurationManager

class IConfigurationManager:
    def get(self, key: str, default: Any = None) -> Any:
        """Get configuration value."""

    def set(self, key: str, value: Any) -> None:
        """Set configuration value."""

    def get_section(self, section: str) -> Dict[str, Any]:
        """Get configuration section."""

    def reload(self) -> None:
        """Reload configuration from file."""
```

IHealthMonitor

Service health monitoring.

```
from claudemppm.services.core.interfaces import IHealthMonitor

class IHealthMonitor:
```

```

async def check_health(self, service_name: str) ->
    HealthStatus:
        """Check health of specific service."""

async def get_system_health(self) -> HealthStatus:
        """Get overall system health."""

def register_health_check(
    self,
    service_name: str,
    check_func: Callable
) -> None:
        """Register custom health check."""

```

Agent Services

IAgentRegistry

Agent discovery and management.

```

from claudemppm.services.agents.interfaces import IAgentRegistry

class IAgentRegistry:
    async def discover_agents(
        self,
        force_refresh: bool = False
    ) -> Dict[str, AgentMetadata]:
        """Discover all available agents."""

    async def get_agent(self, agent_id: str) ->
        Optional[AgentMetadata]:
        """Get specific agent metadata."""

    async def search_by_capability(
        self,
        capability: str
    ) -> List[AgentMetadata]:
        """Search agents by capability."""

    async def get_specialized_agents(
        self,
        agent_type: str
    ) -> List[AgentMetadata]:
        """Get agents of specific type."""

```

AgentDeploymentService

Agent deployment operations.

```

from claudemppm.services.agents import AgentDeploymentService

```

```

class AgentDeploymentService:
    def deploy_agents(
        self,
        force: bool = False,
        include_all: bool = False
    ) -> Dict[str, Any]:
        """Deploy agents to environment."""

    def validate_agent(
        self,
        agent_path: Path
    ) -> Tuple[bool, List[str]]:
        """Validate agent configuration."""

    def clean_deployment(
        self,
        preserve_user_agents: bool = True
    ) -> bool:
        """Clean deployed agents."""

```

AgentManagementService

Agent lifecycle management.

```

from claudempm.services.agents import AgentManagementService

```

```

class AgentManagementService:
    async def create_agent(
        self,
        agent_config: Dict[str, Any]
    ) -> str:
        """Create new agent."""

    async def update_agent(
        self,
        agent_id: str,
        updates: Dict[str, Any]
    ) -> bool:
        """Update agent configuration."""

    async def delete_agent(self, agent_id: str) -> bool:
        """Delete agent."""

    async def list_agents(
        self,
        tier: Optional[str] = None
    ) -> List[AgentInfo]:
        """List available agents."""

```

Communication Services

SocketIOService

Socket.IO server management.

```
from claudempm.services.communication import SocketIOService
```

```
class SocketIOService:
    async def start(self, port: int = 8765) -> None:
        """Start Socket.IO server."""

    async def stop(self) -> None:
        """Stop Socket.IO server."""

    def emit(
        self,
        event: str,
        data: Dict[str, Any],
        room: Optional[str] = None
    ) -> None:
        """Emit event to clients."""

    def is_running(self) -> bool:
        """Check if server is running."""
```

Event Types

```
# Agent events
"agent_started"      # Agent begins task
"agent_completed"    # Agent finishes task
"agent_error"        # Agent encounters error

# Task events
"task_delegated"     # Task delegated to agent
"task_completed"     # Task completed

# Memory events
"memory_updated"     # New memory stored

# System events
"error_occurred"     # System error
"session_started"    # Session initiated
"session_ended"      # Session terminated
```

Project Services

ProjectAnalyzer

Project structure and stack analysis.

```
from claudempm.services.project import ProjectAnalyzer

class ProjectAnalyzer:
    async def analyze_stack(
        self,
        project_dir: Path
    ) -> Dict[str, Any]:
        """Analyze project technology stack."""

    async def detect_frameworks(
        self,
        project_dir: Path
    ) -> List[str]:
        """Detect frameworks used."""

    async def analyze_structure(
        self,
        project_dir: Path
    ) -> Dict[str, Any]:
        """Analyze project structure."""
```

ProjectRegistry

Project configuration management.

```
from claudempm.services.project import ProjectRegistry

class ProjectRegistry:
    def get_project_config(
        self,
        project_dir: Path
    ) -> Dict[str, Any]:
        """Get project configuration."""

    def save_project_config(
        self,
        project_dir: Path,
        config: Dict[str, Any]
    ) -> None:
        """Save project configuration."""

    def get_project_agents(
        self,
        project_dir: Path
```

```
) -> List[str]:  
    """Get project-specific agents."""
```

Infrastructure Services

LoggingService

Structured logging.

```
from claudempm.services.infrastructure import LoggingService
```

```
class LoggingService:  
    def log(  
        self,  
        level: str,  
        message: str,  
        **kwargs  
    ) -> None:  
        """Log structured message."""  
  
    def get_session_logs(  
        self,  
        session_id: str  
    ) -> List[Dict[str, Any]]:  
        """Get logs for session."""  
  
    def configure(self, config: Dict[str, Any]) -> None:  
        """Configure logging."""
```

MonitoringService

Performance monitoring.

```
from claudempm.services.infrastructure import MonitoringService
```

```
class MonitoringService:  
    def record_metric(  
        self,  
        name: str,  
        value: float,  
        tags: Optional[Dict[str, str]] = None  
    ) -> None:  
        """Record metric."""  
  
    def get_metrics(  
        self,  
        name: Optional[str] = None,  
        start_time: Optional[datetime] = None,  
        end_time: Optional[datetime] = None  
    ) -> List[Metric]:  
        """Get recorded metrics."""
```

```
def start_timer(self, name: str) -> Timer:
    """Start timing operation."""
```

CLI API

Main CLI

```
import click
from claude_mpm.cli import main

@main.command()
@click.option('--option', help='Option description')
def my_command(option: str):
    """Command description."""
    pass
```

Agent Commands

```
# List agents
claude-mpm agents list [--by-tier] [--capabilities]

# Create agent
claude-mpm agents create <name> [--tier project|user]

# Deploy agents
claude-mpm agents deploy [--force] [--all]

# Validate agents
claude-mpm agents validate [--agent <name>]
```

Auto-Configuration

```
# Auto-configure project
claude-mpm auto-configure [--preview] [--threshold <0-100>]
```

Local Deployment

```
# Start deployment
claude-mpm local-deploy start --command <cmd> [--name <name>]

# List deployments
claude-mpm local-deploy list

# Monitor deployment
claude-mpm local-deploy monitor <id>

# Stop deployment
claude-mpm local-deploy stop <id>
```


Hook API

HookContext

Context passed to hooks.

```
from claude_mpm.hooks import HookContext

class HookContext:
    task_id: str           # Unique task ID
    agent_id: str          # Agent identifier
    input: str             # User input
    metadata: Dict[str, Any] # Additional data
    timestamp: datetime    # Execution time
    session_id: str        # Session identifier
```

HookResult

Result returned by hooks.

```
from claude_mpm.hooks import HookResult

class HookResult:
    success: bool          # Hook succeeded
    abort: bool = False    # Abort execution
    reason: str = ""        # Error/abort reason
    modified_context: Optional[HookContext] = None
```

Hook Registration

```
from claude_mpm.hooks import HookRegistry

@HookRegistry.register("pre_execution")
async def my_hook(context: HookContext) -> HookResult:
    """Pre-execution hook."""
    return HookResult(success=True)
```

Examples: See [extending.md](#) for usage examples **Architecture:** See [architecture.md](#) for system design