Devices Engine Reference

Understand the devices engine architecture.

Reference **Modules** **Devices** UI | Engine

Devices Engine (Reference) orchestrates communication between FrameworX and field devices through protocol drivers, managing execution threads, communication groups, and diagnostics.

The Devices Engine provides:

- Protocol driver orchestration
- Communication group optimization
- Thread pool management
- Event queue processing
- Performance diagnostics
- Connection lifecycle management

The engine standardizes device communication regardless of protocol, ensuring consistent initialization, execution, and monitoring.

Operating Architecture
Execution Model
Communication Flow
Module Responsibilities
Diagnostics and Monitoring
Configuration Guidelines
Error Handling
Performance Tuning
Best Practices Checklist
Troubleshooting

Operating Architecture

Module Initialization

At startup, the Device Engine:

1. Identifies configured protocol drivers
2. Creates optimized communication groups based on:
    - **Operand Type** - Bits, words, registers, blocks
    - **AccessType** - Read, Write, or ReadWrite
    - **Scan Intervals** - Polling time configuration
    - **Protocol Limits** - Size and quantity constraints

This grouping optimizes performance and ensures protocol compliance.

Communication Groups

Groups are organized by:

- Common data types
- Similar polling rates
- Access patterns
- Protocol specifications

Execution Model

Communication Triggers

Trigger Type	Description	Use Case
Periodic	Fixed interval polling	Regular data updates
Event-Driven	On-demand execution	Tag writes, user requests
Mixed	Both periodic and event	Critical data with backup polling

Thread Architecture

Event Queue Processing:

1. Triggers place events in execution queue
2. Central thread scans queue every 20ms
3. Events assigned to independent threads
4. Parallel execution without blocking

Benefits:

- Low latency response
- Fault isolation between groups
- Efficient multicore utilization
- Non-blocking operations

Communication Flow

Execution Sequence

1. **Group Activation** - Thread receives communication task
2. **Message Generation** - TX packet created per protocol
3. **Connection Management** - Open/reuse TCP, UDP, or Serial
4. **Data Exchange** - Send request, await response
5. **Value Mapping** - Update tags with received data
6. **Error Handling** - Retry logic and diagnostics

Read Operations

```
Device  Protocol Driver  Tags
1. Build read request
2. Send to device
3. Receive response
4. Parse data
5. Update tag values
```

Write Operations

```
Tags  Protocol Driver  Device
1. Collect tag values
2. Build write request
3. Send to device
4. Await confirmation
5. Update status
```

Module Responsibilities

The Device Engine manages:

- **Protocol Management** - Driver lifecycle and configuration
- **Group Creation** - Optimal communication grouping
- **Trigger Definition** - Time and event-based execution
- **Thread Scheduling** - Independent thread allocation
- **Data Routing** - Tag-to-device value mapping
- **Diagnostics** - Performance and status monitoring

Diagnostics and Monitoring

Performance Metrics

Metric	Description	Use
Execution Time	Thread processing duration	Performance analysis
Cycle Time	Interval between executions	Throughput monitoring
Success Rate	Communication reliability	Quality assessment
Queue Depth	Pending operations	Load evaluation

Status Indicators

- Connection state (Active/Failed/Reconnecting)
- Last communication timestamp
- Error counts and types
- Retry attempts
- Data quality

Configuration Guidelines

Group Optimization

```
Optimal Group Size:
- Modbus: 100-125 registers
- OPC UA: 500-1000 items
- Ethernet/IP: 400-500 bytes
```

Thread Pool Sizing

```
Threads = min(CPU_Cores * 2, Active_Groups)
Queue Size = Active_Groups * 2
Scan Rate = 20ms (default)
```

Error Handling

Retry Strategies

Error Type	Retry Logic	Max Attempts
Timeout	Exponential backoff	3
Connection Lost	Immediate retry	5
Protocol Error	Delay retry	2
Device Busy	Queue and retry	10

Fault Isolation

- Thread failures don't affect other groups
- Connection issues isolated per channel
- Protocol errors logged per driver
- Graceful degradation on partial failures

Performance Tuning

Reducing Latency

- Decrease scan interval
- Optimize group sizes
- Use event-driven triggers
- Enable connection pooling

Improving Throughput

- Increase thread pool
- Combine read operations
- Use block transfers
- Enable compression

Resource Optimization

- Share connections
- Cache static data
- Filter unchanged values
- Implement deadbands

Best Practices Checklist

- [ ] **Group Similar Points** - Same data type and polling rate
- [ ] **Balance Group Sizes** - Respect protocol limits
- [ ] **Stagger Polling** - Distribute communication load

- [ ] **Monitor Cycle Times** - Identify bottlenecks
- [ ] **Use Event Triggers** - For write operations
- [ ] **Enable Diagnostics** - Track performance
- [ ] **Configure Timeouts** - Match network conditions

Troubleshooting

High Cycle Times:

- Check network latency
- Reduce group size
- Increase thread pool
- Review timeout settings

Communication Failures:

- Verify device connectivity
- Check protocol settings
- Review error logs
- Test with smaller groups

Queue Overflow:

- Increase thread count
- Reduce polling rates
- Optimize group sizes
- Check for blocking operations

In this section...
