Command Menu System Guide

Document Version: 1.0
Last Updated: February 19, 2026
Application Version: UDP Server Manager v2.0+

Overview

The Command Menu System provides an organized, hierarchical interface for accessing device commands. Commands are grouped by functional category into flyout submenus, reducing clutter and improving usability compared to flat dropdown lists.

Key Benefits

- **Organized Structure:** Commands grouped by function (Control, Status, Error Handling, System Admin)
- **Scalability:** Easily accommodates dozens of commands without overwhelming users
- **Discovery:** Menu categories help users find relevant commands quickly
- **Extensibility:** New commands and categories added without UI redesign
- **Consistency::** Same menu structure across all device types

Menu Architecture

Hierarchical Structure

```
[Select Command ▼]
    │
    ├── Device Control
    │     ├── ENCODER
    │     ├── HPL
    │     ├── LED
    │     └── STEPPER
    │
    ├── Device Status
    │     ├── GET_ALL
    │     ├── GET_ENCODER
    │     ├── GET_HPL
    │     ├── GET_LED
    │     └── GET_STEPPER
    │
    ├── Error Handling
    │     ├── CLEAR_DEBUG_LOG
    │     ├── CLEAR_ERROR_LOG
    │     ├── CLR_WARN
    │     ├── GET_DEBUG_LOG
    │     └── GET_ERROR_LOG
    │
    └── System Administration
```

```
            ├── GET_TIME_STATUS
            └── SET_TIME
```

Category Definitions

Category	Purpose	Typical Commands
Device Control	Active device operations	LED, STEPPER, MOTOR, HPL, ENCODER
Device Status	Query device state	GET_LED, GET_STEPPER, GET_ALL
Error Handling	Error management	CLR_WARN, GET_ERROR_LOG, CLEAR_ERROR_LOG
System Administration	System-level operations	SET_TIME, GET_TIME_STATUS, REBOOT

Command Dictionary Integration

JSON Structure

Commands are defined in JSON files located in `shared_dictionaries/command_dictionaries/`. The category system is integrated into these definitions.

Basic Structure

```
{
  "categories": [
    "Device Control",
    "Device Status",
    "Error Handling",
    "System Administration"
  ],
  "commands": {
    "COMMAND_NAME": {
      "category": "Device Control",
      "description": "Command description",
      "param_count": 2,
      "parameters": [ ... ],
      "function": "handler_function"
    }
  }
}
```

Category Assignment

Each command must have a `category` field matching one of the defined categories:

```
"LED": {
  "category": "Device Control",
  "description": "Controls the state of an LED.",
```

```
  "param_count": 3,
  ...
}
```

Rules:

- Category must be listed in `categories` array at top of dictionary
- Category is case-sensitive
- Commands without category appear in "Uncategorized" menu
- Category determines menu placement

Adding New Commands

Step 1: Define Category (if new)

Add category to `categories` array if it doesn't exist:

```
{
  "categories": [
    "Device Control",
    "Device Status",
    "Error Handling",
    "System Administration",
    "Diagnostics"  ← New category
  ],
  ...
}
```

Step 2: Add Command Definition

Define command with category assignment:

```
"commands": {
  "NEW_COMMAND": {
    "category": "Diagnostics",
    "description": "Performs diagnostic test",
    "param_count": 1,
    "parameters": [
      {
        "name": "test_type",
        "type": "enum",
        "param_number": 1,
        "options": ["quick", "full", "network"],
        "units": "",
        "notes": "Type of diagnostic to run"
      }
    ],
    "function": "handle_new_command"
```

```
    }
}
```

Step 3: Implement Handler

Create handler function in device handler file:

```python
def handle_new_command(self, params):
    """Execute NEW_COMMAND with validation."""
    test_type = params[0]

    # Implementation here

    return f"OK: {test_type} test started"
```

Step 4: Test Integration

1. Restart application
2. Select device
3. Click command menu button
4. Verify new category appears (if added)
5. Verify command appears in correct category
6. Test command execution

Menu Behavior

Menu Population

Process: (On device selection) ↓ Load command dictionary ↓ Parse categories and commands ↓ Group commands by category ↓ Sort categories (predefined order) ↓ Sort commands alphabetically within category ↓ Build Qt menu structure ↓ Display menu button

Category Sort Order:

1. Device Control
2. Device Status
3. Error Handling
4. System Administration
5. (Other categories alphabetically)
6. Uncategorized (if any)

Selection Behavior

User Flow:

1. User clicks "Select Command..." button
2. Flyout menu displays with category submenus

3. User hovers over category → submenu expands
4. User clicks command → menu closes
5. Button label updates to command name
6. Message Creator Panel updates to show parameters

State Management:

- Selected command stored in `_current_command` variable
- Command persists until new selection or device change
- Button label reflects current selection
- "Select Command..." shown when no command selected

Customization

Modifying Category Order

Edit `build_command_menu()` in `message_creator_panel.py`:

```
def build_command_menu(self):
    ...
    # Define preferred category order
    category_order = [
        "Device Control",
        "Device Status",
        "Error Handling",
        "System Administration",
        "Diagnostics",  ← Add new category here
        "Maintenance"
    ]
    ...
```

Category-Specific Styling

Apply custom styles to category submenus:

```
# In build_command_menu()
for category in category_order:
    if category in categories:
        submenu = menu.addMenu(category)

        # Apply custom styling
        if category == "Error Handling":
            submenu.setStyleSheet("background-color: #fff3cd;")
        elif category == "Device Control":
            submenu.setStyleSheet("background-color: #d4edda;")

        # Add commands to submenu
        ...
```

Adding Separators

Insert separators between command groups:

```
# Add separator after control commands
if category == "Device Control":
    submenu.addSeparator()
    # Add special commands after separator
    ...
```

Command Icons

Add icons to menu items:

```
from PySide6.QtGui import QIcon

# When creating action
action = submenu.addAction(cmd_name)
action.setIcon(QIcon("path/to/icon.png"))
```

Advanced Features

Dynamic Menu Updates

Reload menu when device capabilities change:

```
class MessageCreatorPanel(QWidget):
    def update_device_capabilities(self, new_commands):
        """Rebuild menu when device reports new capabilities."""
        self._command_dict['commands'].update(new_commands)
        self.build_command_menu()  # Rebuild menu
```

Context-Sensitive Menus

Enable/disable commands based on device state:

```
def build_command_menu(self):
    ...
    for cmd_name in sorted(categories[category]):
        action = submenu.addAction(cmd_name)

        # Disable if device not ready
        if not self.is_device_ready():
```

```
            action.setEnabled(False)

        action.triggered.connect(...)
```

Search/Filter

Add search functionality to menu:

```
def filter_menu(self, search_text):
    """Show only commands matching search text."""
    menu = self.command_button.menu()
    for action in menu.actions():
        visible = search_text.lower() in action.text().lower()
        action.setVisible(visible)
```

Troubleshooting

Issue: Commands appear in wrong category

Cause: Category field mismatch **Solution:**

1. Check `category` value in command definition
2. Verify category exists in `categories` array
3. Check for typos (case-sensitive)
4. Restart application to reload dictionary

Issue: New category not appearing

Cause: Category not in predefined order **Solution:**

1. Add category to `category_order` list in `build_command_menu()`
2. Or: Category will appear alphabetically after predefined ones
3. Restart application

Issue: Menu button shows "Select Command..." after selection

Cause: `_current_command` not set properly **Solution:**

1. Check `on_command_selected()` method
2. Verify lambda function passes command name correctly
3. Review button text update logic

Issue: Commands not sorted correctly

Cause: Sorting applied before grouping **Solution:**

1. Ensure `sorted()` called on command list within each category
2. Check that sorting happens after category grouping

3. Verify Python version supports stable sorts

Best Practices

Command Naming

- **Consistent Prefixes:** Use GET_ for queries, SET_ for writes, CLR_ for clears
- **Descriptive:** Name should indicate function without needing documentation
- **Uppercase:** Convention for command names in most protocols
- **Underscores:** Separate words (LED_ON, not LEDON or led-on)

Category Design

- **Functional Grouping:** Group by what user wants to accomplish
- **Balanced Size:** Aim for 3-8 commands per category
- **Clear Names:** Category name should be instantly understood
- **Consistent Level:** Keep categories at same abstraction level

Menu Depth

- **Two Levels Maximum:** Category → Command (no deeper nesting)
- **Avoid Single-Item Categories:** Merge into related category
- **Limit Total Categories:** 4-7 categories ideal for usability

Documentation

- **Category Purpose:** Document in command dictionary comments
- **Command Descriptions:** Include in description field
- **Parameter Notes:** Detail usage in notes field
- **Examples:** Provide in handler documentation

Migration Guide

From Flat Dropdown to Hierarchical Menu

Legacy Structure:

```
self.command_dropdown = QComboBox()
self.command_dropdown.addItems(command_list)
```

New Structure:

```
self.command_button = QPushButton("Select Command...")
self.command_button.setMenu(hierarchical_menu)
```

Migration Steps:

1. **Backup:** Save current command dictionary
2. **Add Categories:** Insert `categories` array at top
3. **Assign Categories:** Add `category` field to each command
4. **Update UI Code:** Replace QComboBox with QPushButton + QMenu
5. **Update References:** Change `command_dropdown.currentText()` to `_current_command`
6. **Test Thoroughly:** Verify all commands accessible and functional

Compatibility: Old command dictionaries without categories still work (commands appear in "Uncategorized" menu)

API Reference

Core Methods

```
def build_command_menu(self):
    """Construct hierarchical menu from command dictionary."""

def on_command_selected(self, command):
    """Handle command selection from menu."""

def update_command_related_dropdowns(self, command):
    """Update instance and parameter display for selected command."""
```

Properties

```
self._current_command  # Currently selected command name (or None)
self.command_button    # QPushButton displaying menu
self._command_dict     # Loaded command dictionary
```

Related Documentation

- **UI Components:** See `ui_components_guide.md` for full UI details
- **Command Dictionary Tutorial:** See `command_dictionary_tutorial.md`
- **Architecture:** See `architecture.md` for system design
- **Message Creator Panel:** See `message_creator_panel.md` for panel details

For PDF Generation: This document is formatted for professional PDF output. See `pdf_generation_guide.md` for conversion instructions.