TR181 Node Comparator API Reference

Overview

The TR181 Node Comparator provides a comprehensive API for extracting, comparing, and validating TR181 data model nodes from various sources. This document covers all public classes, methods, and interfaces.

Core Data Models

TR181Node

The fundamental data structure representing a TR181 parameter or object.

```
@dataclass
class TR181Node:
    path: str                    # Full parameter path (e.g., "Device.WiFi.Radio.1.Chan
    name: str                    # Parameter name
    data_type: str              # string, int, boolean, dateTime, etc.
    access: AccessLevel         # read-only, read-write, write-only
    value: Optional[Any] = None # Current value (if available)
    description: Optional[str] = None  # Parameter description
    parent: Optional[str] = None       # Parent node path
    children: Optional[List[str]] = None  # Child node paths
    is_object: bool = False     # True if this is an object node
    is_custom: bool = False     # True if this is a custom (non-standard) node
    value_range: Optional[ValueRange] = None  # Value constraints and validation rules
    events: Optional[List[TR181Event]] = None  # Associated events
    functions: Optional[List[TR181Function]] = None  # Associated functions
```

Properties: - `path`: The full TR181 path following the standard naming convention - `name`: The parameter name (last component of the path) - `data_type`: Data type as defined in TR181 specification - `access`: Access level determining read/write permissions - `value`: Current parameter value (None if not retrieved) - `description`: Human-readable description of the parameter - `parent`: Path to the parent object (None for root objects) - `children`: List of child parameter paths (for object nodes) - `is_object`: True if this represents an object rather than a parameter - `is_custom`: True if this is a vendor-specific extension - `value_range`: Validation constraints for the parameter value - `events`: List of events associated with this parameter - `functions`: List of functions associated with this parameter

AccessLevel

Enumeration defining parameter access permissions.

```
class AccessLevel(Enum):
    READ_ONLY = "read-only"
    READ_WRITE = "read-write"
    WRITE_ONLY = "write-only"
```

ValueRange

Defines validation constraints for parameter values.

```
@dataclass
class ValueRange:
```

```
    min_value: Optional[Any] = None
    max_value: Optional[Any] = None
    allowed_values: Optional[List[Any]] = None  # For enumerated values
    pattern: Optional[str] = None  # Regex pattern for string validation
    max_length: Optional[int] = None  # For string length validation
```

TR181Event

Represents an event associated with a TR181 parameter.

```
@dataclass
class TR181Event:
    name: str
    path: str
    parameters: List[str]  # Event parameter paths
    description: Optional[str] = None
```

TR181Function

Represents a function associated with a TR181 parameter.

```
@dataclass
class TR181Function:
    name: str
    path: str
    input_parameters: List[str]
    output_parameters: List[str]
    description: Optional[str] = None
```

Extractor Interfaces

NodeExtractor (Abstract Base Class)

Base interface for all TR181 node extractors.

```
class NodeExtractor(ABC):
    @abstractmethod
    async def extract(self) -> List[TR181Node]:
        """Extract TR181 nodes from the source.

        Returns:
            List[TR181Node]: List of extracted TR181 nodes

        Raises:
            ConnectionError: If unable to connect to source
            ValidationError: If source data is invalid
        """
        pass

    @abstractmethod
    async def validate(self) -> bool:
        """Validate the source is accessible and contains valid data.

        Returns:
            bool: True if source is valid and accessible
```

```
        """
        pass

    @abstractmethod
    def get_source_info(self) -> SourceInfo:
        """Get metadata about the data source.

        Returns:
            SourceInfo: Source metadata including type, identifier, and timestamp
        """
        pass
```

CWMPExtractor

Extracts TR181 nodes from CWMP/TR-069 sources.

```
class CWMPExtractor(NodeExtractor):
    def __init__(self, connection_config: Dict[str, Any]):
        """Initialize CWMP extractor.

        Args:
            connection_config: CWMP connection configuration including:
                - endpoint: CWMP endpoint URL
                - username: Authentication username
                - password: Authentication password
                - timeout: Connection timeout in seconds
        """

    async def extract(self) -> List[TR181Node]:
        """Extract all TR181 nodes from CWMP source.

        Uses GetParameterNames and GetParameterValues RPC operations
        to discover and retrieve all available parameters.

        Returns:
            List[TR181Node]: Complete list of TR181 nodes from device

        Raises:
            ConnectionError: If CWMP connection fails
            ValidationError: If CWMP responses are malformed
        """

    async def validate(self) -> bool:
        """Validate CWMP connection and basic functionality."""

    def get_source_info(self) -> SourceInfo:
        """Get CWMP source information."""
```

SubsetManager

Manages custom TR181 subsets and node definitions.

```
class SubsetManager(NodeExtractor):
    def __init__(self, subset_path: str):
```

```
        """Initialize subset manager.

        Args:
            subset_path: Path to subset definition file (JSON/YAML)
        """

    async def extract(self) -> List[TR181Node]:
        """Load TR181 nodes from subset definition.

        Returns:
            List[TR181Node]: Nodes defined in the subset

        Raises:
            FileNotFoundError: If subset file doesn't exist
            ValidationError: If subset format is invalid
        """

    async def save_subset(self, nodes: List[TR181Node], path: str = None) -> None:
        """Save TR181 nodes to subset file.

        Args:
            nodes: List of TR181 nodes to save
            path: Optional path to save to (uses instance path if None)

        Raises:
            ValidationError: If nodes contain invalid definitions
            IOError: If unable to write to file
        """

    async def add_custom_node(self, node: TR181Node) -> None:
        """Add a custom node definition to the subset.

        Args:
            node: Custom TR181 node to add

        Raises:
            ValidationError: If node definition is invalid
        """

    async def validate_subset(self) -> ValidationResult:
        """Validate all nodes in the subset follow TR181 conventions."""
```

HookBasedDeviceExtractor

Extracts TR181 nodes from devices using pluggable communication hooks.

```
class HookBasedDeviceExtractor(NodeExtractor):
    def __init__(self, device_config: DeviceConfig, hook: DeviceConnectionHook):
        """Initialize device extractor with communication hook.

        Args:
            device_config: Device connection configuration
            hook: Communication hook implementation (REST, CWMP, etc.)
```

```
        """

    async def extract(self) -> List[TR181Node]:
        """Extract TR181 nodes from device using the configured hook.

        Returns:
            List[TR181Node]: All TR181 nodes available on the device

        Raises:
            ConnectionError: If device connection fails
            ValidationError: If device responses are invalid
        """

    async def validate(self) -> bool:
        """Test device connectivity and basic functionality."""

    def get_source_info(self) -> SourceInfo:
        """Get device source information."""
```

Comparison Engine

ComparisonEngine

Core comparison functionality for TR181 nodes.

```
class ComparisonEngine:
    async def compare(self, source1: List[TR181Node], source2: List[TR181Node]) -> Comp
        """Compare two sets of TR181 nodes.

        Args:
            source1: First set of TR181 nodes
            source2: Second set of TR181 nodes

        Returns:
            ComparisonResult: Detailed comparison results including:
                - Nodes only in source1
                - Nodes only in source2
                - Common nodes with differences
                - Summary statistics
        """
```

EnhancedComparisonEngine

Extended comparison with validation and testing capabilities.

```
class EnhancedComparisonEngine(ComparisonEngine):
    async def compare_with_validation(self,
                                      subset_nodes: List[TR181Node],
                                      device_nodes: List[TR181Node],
                                      device_extractor: DeviceExtractor = None) -> Enhanc
        """Perform enhanced comparison with validation and testing.

        Args:
            subset_nodes: Expected TR181 nodes from subset
```

```
            device_nodes: Actual TR181 nodes from device
            device_extractor: Optional device extractor for event/function testing

        Returns:
            EnhancedComparisonResult: Comprehensive results including:
                - Basic comparison results
                - Validation results for each node
                - Event testing results
                - Function testing results
        """
```

Validation

TR181Validator

Comprehensive validation for TR181 nodes and values.

```
class TR181Validator:
    def validate_node(self, node: TR181Node, actual_value: Any = None) -> ValidationRes
        """Validate a TR181 node definition and optionally its value.

        Args:
            node: TR181 node to validate
            actual_value: Optional actual value to validate against node constraints

        Returns:
            ValidationResult: Validation results with errors and warnings
        """

    def validate_data_type(self, expected_type: str, value: Any) -> bool:
        """Validate that a value matches the expected TR181 data type."""

    def validate_path_format(self, path: str) -> bool:
        """Validate that a path follows TR181 naming conventions."""

    def validate_range(self, value: Any, range_spec: ValueRange) -> ValidationResult:
        """Validate that a value falls within specified constraints."""
```

ValidationResult

Container for validation results.

```
class ValidationResult:
    def __init__(self):
        self.is_valid: bool = True
        self.errors: List[str] = []
        self.warnings: List[str] = []

    def add_error(self, message: str):
        """Add a validation error."""

    def add_warning(self, message: str):
        """Add a validation warning."""
```

Device Communication Hooks

DeviceConnectionHook (Abstract Base Class)

Base interface for device communication protocols.

```python
class DeviceConnectionHook(ABC):
    @abstractmethod
    async def connect(self, config: DeviceConfig) -> bool:
        """Establish connection to device."""

    @abstractmethod
    async def disconnect(self) -> None:
        """Close device connection."""

    @abstractmethod
    async def get_parameter_names(self, path_prefix: str = "Device.") -> List[str]:
        """Get all parameter names under the specified path."""

    @abstractmethod
    async def get_parameter_values(self, paths: List[str]) -> Dict[str, Any]:
        """Get current values for specified parameter paths."""

    @abstractmethod
    async def get_parameter_attributes(self, paths: List[str]) -> Dict[str, Dict[str, A
        """Get parameter attributes (type, access, etc.) for specified paths."""
```

RESTAPIHook

REST API implementation for device communication.

```python
class RESTAPIHook(DeviceConnectionHook):
    def __init__(self):
        """Initialize REST API hook."""

    async def connect(self, config: DeviceConfig) -> bool:
        """Connect to device REST API."""

    # ... other methods implement REST-specific communication
```

CWMPHook

CWMP/TR-069 implementation for device communication.

```python
class CWMPHook(DeviceConnectionHook):
    def __init__(self):
        """Initialize CWMP hook."""

    async def connect(self, config: DeviceConfig) -> bool:
        """Connect to device via CWMP."""

    # ... other methods implement CWMP-specific communication
```

Configuration Management

SystemConfig

Main system configuration container.

```
@dataclass
class SystemConfig:
    devices: List[DeviceConfig]
    subsets: List[SubsetConfig]
    export: ExportConfig
    logging: Dict[str, Any]
```

DeviceConfig

Device connection configuration.

```
@dataclass
class DeviceConfig:
    name: str
    type: str  # 'cwmp', 'rest', etc.
    endpoint: str
    authentication: Dict[str, Any]
    timeout: int = 30
    retry_count: int = 3
    hook_config: Optional[HookConfig] = None
```

Error Handling

Custom Exceptions

```
class TR181Error(Exception):
    """Base exception for TR181 comparator errors."""

class ConnectionError(TR181Error):
    """Raised when device connection fails."""

class ValidationError(TR181Error):
    """Raised when data validation fails."""

class ConfigurationError(TR181Error):
    """Raised when configuration is invalid."""
```

Usage Examples

Basic Node Extraction

```
from tr181_comparator import CWMPExtractor, SubsetManager

# Extract from CWMP source
cwmp_config = {
    'endpoint': 'http://device.local:7547/cwmp',
    'username': 'admin',
    'password': 'password'
}
```

```
cwmp_extractor = CWMPExtractor(cwmp_config)
cwmp_nodes = await cwmp_extractor.extract()

# Load from subset
subset_manager = SubsetManager('my_subset.json')
subset_nodes = await subset_manager.extract()
```

Basic Comparison

```
from tr181_comparator import ComparisonEngine

engine = ComparisonEngine()
result = await engine.compare(cwmp_nodes, subset_nodes)

print(f"Nodes only in CWMP: {len(result.only_in_source1)}")
print(f"Nodes only in subset: {len(result.only_in_source2)}")
print(f"Differences found: {len(result.differences)}")
```

Enhanced Comparison with Validation

```
from tr181_comparator import EnhancedComparisonEngine, HookBasedDeviceExtractor, RESTAP

# Set up device extractor
device_config = DeviceConfig(
    name="test_device",
    type="rest",
    endpoint="http://device.local/api"
)
hook = RESTAPIHook()
device_extractor = HookBasedDeviceExtractor(device_config, hook)

# Perform enhanced comparison
enhanced_engine = EnhancedComparisonEngine()
result = await enhanced_engine.compare_with_validation(
    subset_nodes,
    device_nodes,
    device_extractor
)

# Get comprehensive summary
summary = result.get_summary()
print(f"Validation errors: {summary['validation']['nodes_with_errors']}")
print(f"Event test failures: {summary['events']['failed_events']}")
```