FNT
// simplify complexity

Use, enrich, and configure events for integration scenarios

Filter and enrich base events

Send and receive enriched events to and from external systems

Implement customer-specific rules for FNT Command

FNT EventEngine

A powerful integration tool for complete event processing

As part of the FNT IntegrationCenter, the FNT EventEngine specializes in managing all types of event-based interfaces. It can be used to monitor events from FNT Command to trigger specific actions either within FNT Command or an external application. These events can be filtered and enriched with additional data from FNT Command. The enriched events can then be sent to other external systems or reconciled with data from FNT Command using the FNT ReconEngine.

EVENT-DRIVEN USE CASES

Event-Driven Updates of the FNT Command Core System

When an external system sends an event, the data is extracted using Groovy scripts. The required information is forwarded to the FNT ReconEngine, which either automatically aligns the data or generates a delta report based on predefined rules. Users can then approve changes in the delta report and view the updated data directly in the GUI.

FUNCTIONS

- Receive FNT Command Core events and events from other sources
- Filter incoming events
- Route incoming events
- Process incoming events with custom logic
- Enrich incoming events with FNT Command Core data
- Forward incoming events

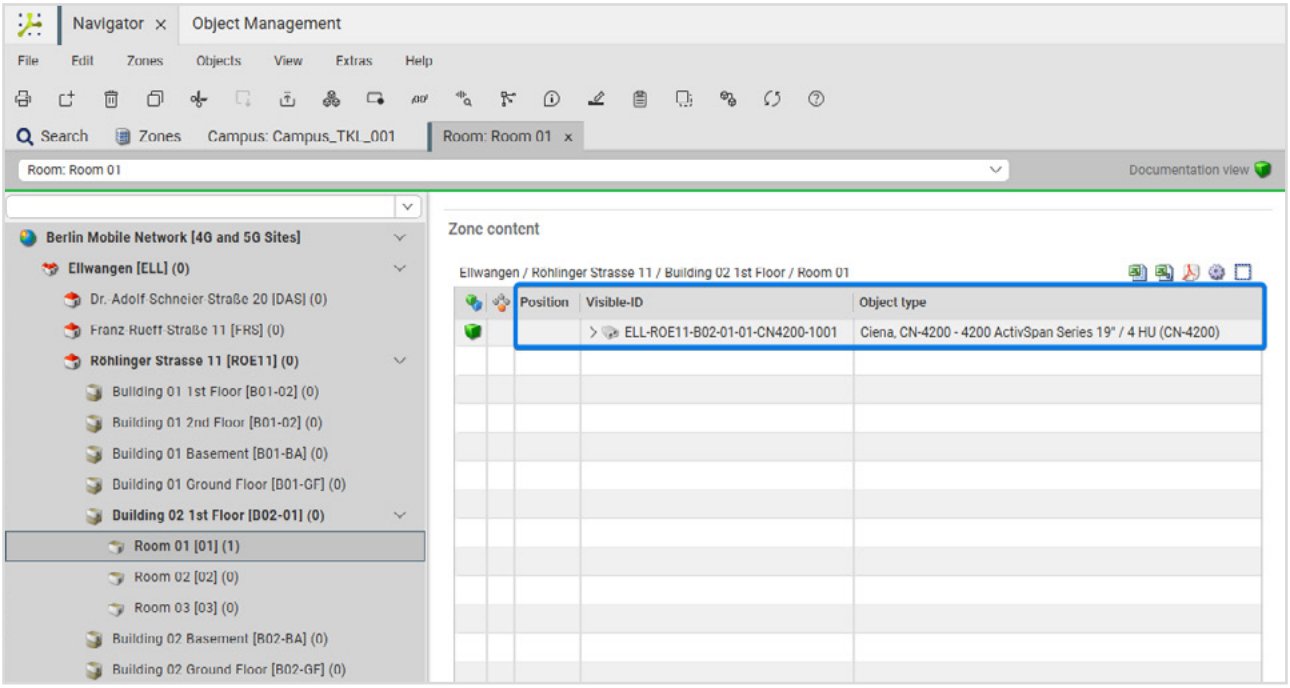

Figure 1: Example of Automatic Generation of the Device VisibleID with the EventEngine

Data Enrichment of Events

When an external system sends an event, such as an alarm notification from a monitoring tool, the data is extracted using Groovy scripts and additional logic, such as filters and mappings, is applied. Supplementary information, such as location address, cable overview, affected customers, and protection status, can be added from FNT Command. The enriched data can then be forwarded to a ticketing system.

```groovy
public class GenerateDeviceVisibleId {

    // Wire beans
    static Logger LOG = LoggerFactory.getLogger(GenerateDeviceVisibleId.class)
    static ZoneService zoneService = ApplicationContextProvider.getBean(ZoneService.class)
    static CampusRepository campusRepo = ApplicationContextProvider.getBean(CampusRepository.class)
    static BuildingRepository buildingRepo = ApplicationContextProvider.getBean(BuildingRepository.class)
    static FloorRepository floorRepo = ApplicationContextProvider.getBean(FloorRepository.class)
    static RoomRepository roomRepo = ApplicationContextProvider.getBean(RoomRepository.class)

    static EventMessageBody apply(Exchange exchange) {
        EventMessageBody event = exchange.getMessage().getMandatoryBody(EventMessageBody.class)
        LOG.info("{}", event)
        IntegrationObjectMessageBody integrationObject = event.getIntegrationObjects().get(0)
        DeviceBase dto = integrationObject.getObject(DeviceBase.class)

        List<Zone> zoneList = zoneService.findZoneByChildElement(dto.getElid())
        LOG.info("Found {} zones for device: {}", zoneList.size(), zoneList)
        if (zoneList.size() != 1) {
            return event
        }
        Zone zone = zoneList.get(0)
```

Figure 1: Example Groovy Script

User-Specific Implementation of Rules

When a user creates an object in FNT Command, such as a new service, the system generates a basic event. Filters can be applied using Groovy scripts, allowing the process to request additional data from FNT Command when needed. User-specific logic and rules can also be implemented via Groovy scripts (see Figure 2), and the resulting data is saved back into FNT Command (see Figure 1).

User-Specific, Event-Driven Updates

When a user places a card on a shelf, a basic event is generated in FNT Command. Groovy scripts can be used to apply filters and logic to enrich the event with additional data from FNT Command. Based on predefined routing rules, the event is forwarded to the specified system.

Interface Customization (Wrapper)

When an external API call is made, such as a query for data in an externally defined format, this call is transformed into one or more FNT Command API calls. The requested data is retrieved from FNT Command, transformed back into the externally defined format, and then sent as a response to the external system.

KEY BENEFITS OF FNT EventEngine

- **Define Custom Logic:** Customers and partners can define their own logic and forward it to the relevant areas within the EventEngine.
- **Easy Interaction with Data Stored in FNT Command:** Adapters make integration with the FNT Command platform possible, enabling seamless interaction with data stored in FNT Command and the ReconEngine. Events triggered by Command can be received and processed. The Command BGE component allows for querying and editing data, as well as access to the ReconEngine NMS cache and synchronization tasks.
- **Efficient Event Management:** Manages all event-based interfaces to handle the data and event flow between FNT Command and external systems seamlessly.
- **Real-Time Data Updates:** Enables synchronization and updates to data across multiple systems in near real-time, which is useful in scenarios where a daily full data sync is not sufficient.
- **Scalability for Complex Use Cases:** Ideal for scenarios where data needs to be dynamically provided or updated across multiple systems, making it suitable for more complex use cases that go beyond traditional batch updates.