**Object API — Подробная документация (с нюансами)**

Сервисы: ObjectService, ObjectArchetypeService, ObjectTypeService; Контроллер: ObjectController

Дата: 2025-09-05

## 1. Назначение и обзор

API обеспечивает интеграцию с MidPoint (создание/изменение/удаление объектов) и доступ к внутренним таблицам/представлениям для получения структуры полей и конкретных значений по archetype или objectType. Контроллер требует Basic Auth и использует Spring Pageable для пагинации/сортировки.

## 2. Глоссарий

• objectType — тип объекта MidPoint (например: user/role/org).

• archetype — внешний архетип (extArchetype) для группировки полей.

• ext-таблица — таблица/представление, имя которой содержит 'ext' (хранит расширенные атрибуты).

• Lookup — поле со списком допустимых значений (берутся из EnumValueRepository).

• vDisplayName — человекочитаемое имя объекта; берётся из getVDisplayName() или через RefValueRepository для ссылок.

• FilterRequest — фильтры, список включаемых/исключаемых полей и т.п.; применяется при поиске/чтении.

## 3. Аутентификация и безопасность

• Все эндпоинты помечены @SecurityRequirement(name = "basicAuth"). Требуется заголовок Authorization: Basic ...

• Креденшелы не логируются. В логах фиксируются типы операций и OID.

• Внимание к ext\_whereclause: RepositoryResolver строит Specification из where-части (должно быть безопасно парсингом, без прямой подстановки).

• Рекомендуется ограничить список допустимых objectType и проверять входные данные (валидация).

## 4. Пагинация и сортировка

• Везде, где возвращаются коллекции, используется Pageable (page, size, sort).

• Сортировка проксируется в RepositoryResolver.executeQueryPaged(...).

## 5. Эндпоинты

|  |  |  |
| --- | --- | --- |
| Метод | Путь | Назначение |
| POST | /api/object/{objectType} | Создать объект в MidPoint |
| PUT | /api/object/{objectType}/{oid} | Частичное обновление объекта (itemDelta) |
| DELETE | /api/object/{objectType}/{oid} | Удалить объект в MidPoint |
| GET | /api/object/getObjectFields/{archetype} | Справочник полей по archetype |
| POST | /api/object/getObjectFields/{archetype} | Справочник полей по archetype (фильтры) |
| POST | /api/object/getObjectFields/{archetype}/{oid} | Значения полей по archetype и OID |
| POST | /api/object/getObject/{archetype}/field/{fieldName} | Данные по ext\_object+ext\_whereclause для поля |
| GET | /api/object/getObjects/{objectType} | Список объектов по objectType (без фильтров) |
| POST | /api/object/getObjects/{objectType} | Список объектов по objectType (с фильтрами) |
| POST | /api/object/getObjectFields/{objectType}/{archetype}/{oid} | Поля объекта по objectType+archetype+OID |

## 6. Подробности по ключевым операциям

### 6.1 Создание объекта (POST /api/object/{objectType})

Вход: массив объектов {"name","value"}. Сервис конвертирует в Map<path,value> и вызывает MidPointClient.createObject(...).

Ответ: {"oid": "..."}.

Пример body:

[  
 {"name":"name/familyName","value":"U"},  
 {"name":"name/givenName","value":"X"},  
 {"name":"activation/administrativeStatus","value":"enabled"}  
]

### 6.2 Частичное обновление (PUT /api/object/{objectType}/{oid})

Вход: список AttributeDelta (name, modification, value). name с нижними подчёркиваниями преобразуется в path с '/'. value упаковывается в соответствии с типом (строка/список/Map).

Пример запроса (сервис сформирует body MidPoint с objectModification/itemDelta):

[  
 {"name":"activation\_administrativeStatus","modification":"REPLACE","value":"disabled"},  
 {"name":"assignment\_targetRef","modification":"ADD","value":{"oid":"<role-oid>","type":"RoleType"}}  
]

Замечание: метод возвращает Mono<String>, контроллер оборачивает в 200 OK.

### 6.3 Удаление (DELETE /api/object/{objectType}/{oid})

Вызывает MidPointClient.deleteObject(...). При успехе возвращается 204 No Content.

### 6.4 Поля по archetype (GET/POST /getObjectFields/{archetype})

GET: просто список полей по extArchetype с send=true; если пусто — fallback на objectType.

POST: дополнительно применяются фильтры FilterRequest (включаемые/исключаемые поля, условия).

d\_get\_m\_user – все данные о пользователях

d\_get\_user\_by\_oid

d\_get\_m\_user(oid=’{uuid}’)

d\_get\_m\_user\_with\_ext(oid=’{uuid}’) – все данные о пользователе с учетом ext-полей

### 6.5 Значения по archetype и OID (POST /getObjectFields/{archetype}/{oid})

POST /getObjectFields/{objectType} – получение всех объектов типа {objectType}

Формат ответа:

{

“oid”: null,

“vdisplayname”: null,

“columns”: [

“tableName”

]

}

Атрибуты мы получаем из таблицы d\_object\_fields

POST /getObjectFields/{objectType}/{archetype} – тоже самое, но сначала ищем по {archetype}, если не нашли, то ищем по {objectType}

POST /getObjectFields/{objectType}/oid/{oid} – получение заполненного списка полей

“oid”: “uuid”,

“vdisplayname”: “string”,

“columns”: [

“fieldName”: “string”

]

}

POST /getObjectFields/{objectType}/{archetype}/oid/{oid} – тоже самое, но ищем по {archetype}, если не нашли, то по {objectType}

POST /getObjects/{objectType}/{archetype} – получение всех объектов типа {archetype}, а если нет, то по {objectType} – возвращать результат выполнения функции

Эта функция должна пойти в d\_config\_param и посмотреть маппинг objectType - function

### Данные по полю (POST /getObject/{archetype}/field/{fieldName})

Находит запись поля (сначала по archetype, затем по objectType). Использует ext\_object как таблицу и ext\_whereclause как базовое условие. Комбинирует спецификацию whereSpec с filterSpec и buildWhereClause(...). Возвращает Page<Map<String,Object>>.

Важно request может быть null — требуется проверка перед whereNode.addAll(request.getFilters()).

Алгоритм

1) Получить список полей по archetype (или fallback на objectType).

2) Сгруппировать по tableName и для каждой таблицы:

• resolveEntityClass(tableName) через RepositoryResolver.

• Если ext-таблица: читать значение по oid+extAttrName (getExtValueByOidAndField).

• Иначе: читать из основной entity через геттеры/поля.

3) Определить vDisplayName: getVDisplayName() или RefValueRepository (для референсных полей).

4) Для Lookup полей подгрузить variables из EnumValueRepository.

5) Вернуть объект с oid, vDisplayName и columns[].

### 6.6 Данные по полю (POST /getObject/{archetype}/field/{fieldName})

Находит запись поля (сначала по archetype, затем по objectType). Использует ext\_object как таблицу и ext\_whereclause как базовое условие. Комбинирует спецификацию whereSpec с filterSpec и buildWhereClause(...). Возвращает Page<Map<String,Object>>.

Важно request может быть null — требуется проверка перед whereNode.addAll(request.getFilters()).

### 6.7 Списки по objectType (GET/POST /getObjects/{objectType})

GET: находит первую не-ext таблицу, возвращает страницу данных без фильтра.

POST: то же, но применяется FilterRequest через FilterSpecificationBuilder.

### 6.8 Поля по objectType+archetype+OID (POST /getObjectFields/{objectType}/{archetype}/{oid})

Поля берутся по archetype, при отсутствии — fallback к objectType (маппинг в archetype DTO), после чего значения читаются по основной таблице (первая в списке DTO).

## 7. Спецификации входных структур

### 7.1 AttributeDelta

• name: String — путь атрибута; символ '\_' будет заменён на '/'.

• modification: String — тип модификации (ADD/DELETE/REPLACE).

• value: Object — допускается String, Map, List<...>. В тело MidPoint кладётся в массив, если это не строка/список Map.

### 7.2 FilterRequest

• filters: List<FilterNode> (field, op, value).

• fields: List<String> — включить только указанные поля в ответе.

• excludeFields: List<String> — исключить указанные поля из ответа.

• Pageable: через параметры запроса page/size/sort.

Замечание: в некоторых местах request используется без проверки на null — см. раздел 'Известные нюансы'.

## 8. Внутренние алгоритмы и хелперы

• isExtTable(name): определяет ext-таблицу по подстроке 'ext'.

• getExtValueByOidAndField: строит Specification по oid+extAttrName и читает extAttrValue через рефлексию.

• trimDto: применяет DtoFieldTrimmer к DTO/Map.

• buildSpec: делает JPA Specification через FilterSpecificationBuilder.

• resolveEntityClass: RepositoryResolver.resolveEntityClass(tableName).

• executeQueryPaged/ForViews/UntypedQuery: унифицированные вызовы для запроса данных.

## 9. Коды ответов и ошибки

• 200 OK — успешные чтения/обновления.

• 204 No Content — успешное удаление.

• 400/404 — при IllegalArgumentException/EntityNotFoundException (зависит от глобального обработчика ошибок).

• Логи: информативные, уровни INFO/WARN/ERROR, OID/таблица/поле фиксируются.

## 10. Производительность и поведение

• updateObject использует реактивный клиент, но .block() вызывается в create/delete; смешанный стиль (reactive+blocking).

• resolveEntityClass вызывается для каждой таблицы; при множестве таблиц целесообразно кешировать.

• При большом количестве полей возможен N+1-паттерн (много мелких выборок по ext). Нужно попробовать батч-запросы.

• Пагинация и сортировка делаются на уровне базы через RepositoryResolver.

• Валидация входных данных уменьшит количество пустых/лишних запросов.

## 12. Нужно учесть при разработке новой версии

• Добавить валидацию входных параметров (objectType, archetype, fieldName).

• Централизованный обработчик ошибок (ControllerAdvice) с нормализованными кодами и телами ответов.

• Кешировать resolveEntityClass(tableName) и конфигурацию полей для archetype/objectType.

• Добавить метрики (время запросов, количество ext-вызовов) и трейсинг.

• Чёткое разделение реактивных/блокирующих вызовов (избегать .block() в WebFlux-контексте).

• Явный признак ext-таблиц в конфигурации вместо эвристики по названию.

• Для vDisplayName — единый контракт: где возможно, хранить в основной таблице, чтобы не бегать по ссылкам.

NPE-риски при request == null:

– buildResultMapFromEntity(...) вызывает request.getFilters без проверки.

– getByFieldNameAndArchetypeWithExtQuery(...) делает whereNode.addAll(request.getFilters()) без проверки.

Нужно использовать Optional.ofNullable(request).map(FilterRequest::getFilters).orElse(List.of()).

• Пустой список таблиц: getObjectsByObjectType(...) берёт fields.get(0) без проверки на пустоту. Нужно возвращать 400 с понятным сообщением или выполнять fallback.

• Определение ext-таблицы по подстроке 'ext' — эвристика. Лучше хранить явный флаг/тип.

• В updateObject упаковка value: проверьте ожидаемый формат MidPoint для одиночных/множественных атрибутов.

• buildWhereClauseSpec(buildWhereClause) — нужно убедиться парсер whereClause не допускает инъекций и корректно экранирует значения.