ChatGPT

Индекс ABC

Каталог содержит документацию всех абстрактных базовых классов (ABC) ETL-архитектуры проекта BioETL. Каждый ABC определяет контракт для определенной сферы ответственности в конвейере обработки данных [1]. Ниже представлена иерархия компонентов ETL по уровням с указанием соответствующих ABC-интерфейсов и их ролей, а также связи между ними.

Уровень оркестрации пайплайна (Pipeline): отвечает за координацию этапов ETL и общую логику выполнения.
- **PipelineBase** – базовый класс конвейера ETL, оркеструющий стадии extract→transform→validate→write [2]. Обеспечивает единый интерфейс запуска пайплайна, логирование, валидацию данных (через Pandera) и атомарную запись результатов, а также управление ресурсами и поддержку режима dry-run без записи результатов [2].
- **StageABC** – интерфейс отдельной стадии конвейера обработки данных [3]. Стадия получает поток входных элементов и возвращает поток выходных, инкапсулируя локальную логику преобразования или обработки. Стадия не знает о других стадиях и управлении пайплайном, что повышает переиспользуемость и тестируемость [4] [3].
- **PipelineHookABC** – интерфейс хуков наблюдения за выполнением конвейера [5]. Хуки получают уведомления о ключевых событиях жизненного цикла пайплайна – запуск и завершение всего конвейера и отдельных стадий, а также ошибки на этапах – для реализации мониторинга, логирования, сбора метрик, уведомлений и т.д., без вмешательства в саму логику пайплайна [5].
- **ErrorPolicyABC** – политика обработки ошибок конвейера [6]. Определяет стратегию реакции на исключения: продолжать выполнение, пропустить проблемную запись или прервать пайплайн [6]. Оркестратор пайплайна вызывает метод политики ошибок при возникновении исключения на стадии, чтобы получить решение, как поступить дальше (продолжить, пропустить запись или остановить конвейер) [7].
- **CLICommandABC** – интерфейс плагинной команды CLI для управления пайплайнами [8]. Определяет имя команды и метод выполнения `run(args)` для запуска пайплайна или сопутствующих операций из командной строки, что позволяет расширять набор CLI-команд без изменения базового кода [8].

Уровень мониторинга и логирования: поддерживает наблюдаемость работы пайплайна.
- **LoggerAdapterABC** – интерфейс адаптера структурированного логирования [9]. Предоставляет унифицированный способ логировать события пайплайна с контекстом (например, идентификатором запуска), позволяя агрегировать логи для мониторинга [9]. Методы `info()/warning()/error()` логируют сообщения с дополнительными полями, а `bind()` создает новый логгер с привязанным контекстом (например, присоединяет `run_id` ко всем сообщениям) [10] [11].
- **ProgressReporterABC** – интерфейс агрегированной отчетности о ходе выполнения конвейера [12]. Предоставляет методы фиксации количества извлеченных записей, валидных записей и отброшенных записей с указанием причин отбрасывания [13]. Реализация может собирать эти показатели для финального отчета или отправлять их во внешние системы, не вмешиваясь в сам процесс обработки данных [12].
- **TracerABC** – интерфейс трассировки выполнения и метрик [14]. Позволяет интегрировать пайплайн с системами распределенной трассировки или собственными механизмами замера производительности. Предоставляет методы для обозначения начала/конца операций и записи

метрик времени и счетчиков событий, которые могут отправляться в системы мониторинга производительности [15] [16].

Уровень доступа к внешним данным (Client): отвечает за извлечение данных из внешних источников (API, базы данных и пр.), контролируя сетевое взаимодействие, пагинацию, ретраи, ограничение частоты и пр.

- **BaseClient** – базовый класс клиентов внешних источников (новая архитектура) [17] [18]. Определяет унифицированные методы `fetch_one()`, `iter_records()`, `iter_pages()`, `metadata()` и `close()` для получения данных по запросу, итерации по записям и страницам, получения метаданных и освобождения ресурсов [19] [18]. Используется в связке с `ConfiguredHttpClient` для автоматической загрузки настроек из YAML и выполнения запросов через абстракцию HTTP-бэкенда [20].
- **SourceClientABC** – ABC-контракт клиента внешнего источника данных (устаревшая схема) [21] [22]. Определяет базовые методы взаимодействия с API: `send(request)` для отправки запроса и получения ответа, а также высокоуровневые методы `fetch_one()` и `fetch_many()` для получения одной записи или потокового получения записей с разбиением на страницы [23] [24]. Реализации могут использовать `RateLimiterABC` для ограничения запросов, `RetryPolicyABC` для повторных попыток, `CacheABC` для кэширования ответов и `SecretProviderABC` для аутентификации [21] [25]. *(Примечание: старая реализация* `BaseExternalDataClient` *реализует этот интерфейс и управляется через YAML-конфигурации; в новой архитектуре рекомендуется использовать* `BaseClient` *и фабрику клиентов [26] [27].)*
- **RequestBuilderABC** – интерфейс построения транспортных запросов к внешним системам [28]. Отвечает за формирование объекта запроса (например, URL, параметры, тело) на основе входных данных пайплайна, абстрагируя логику формирования HTTP/DB запросов от конкретной реализации.
- **ResponseParserABC** – интерфейс разбора ответов внешнего источника [28]. Определяет, как преобразовать сырой ответ (JSON, XML, CSV и т.п.) от внешнего API в внутренние записи (Record), пригодные для дальнейшей обработки конвейером.
- **PaginatorABC** – интерфейс стратегии постраничного обхода результатов [29]. Определяет, как осуществлять пагинацию при извлечении данных: например, через параметры page/limit (offset-based) или через маркеры продолжения (cursor-based). Реализации инкапсулируют логику получения следующей страницы данных.
- **RateLimiterABC** – интерфейс ограничителя частоты запросов и параллелизма [28]. Предоставляет методы `try_acquire()` и `acquire()` для контроля отправки запросов с заданной скоростью (например, не более N запросов в секунду), предотвращая перегрузку внешних API. Реализация обычно основана на токен-бакет алгоритме (см. `TokenBucketRateLimiterImpl`) [30] [31].
- **RetryPolicyABC** – интерфейс политики повторных попыток при ошибках [32]. Определяет, следует ли повторить запрос при возникновении ошибки (метод `should_retry(attempt, error)`) и вычисляет паузу перед новой попыткой (метод `get_backoff_seconds(attempt)`), например реализуя экспоненциальный backoff [33]. Политика может учитывать тип ошибки и номер попытки; стандартная реализация – экспоненциальная задержка с джиттером (`ExponentialBackoffRetryImpl`) [34].
- **CacheABC** – интерфейс кэша [35]. Определяет контракт для кэширования ответов внешних источников, чтобы снизить количество повторных запросов. Может предоставлять методы получения/сохранения данных по ключу запроса. Реализации могут быть в памяти или persistent (например, на диске); в проекте предусмотрено использование кэша для HTTP-запросов (HTTP cache) [36].
- **SecretProviderABC** – интерфейс доступа к секретным данным (например, токенам API) [37]. Позволяет абстрагировать источник секретов – хранение и подстановку ключей API, паролей и пр. – чтобы не хардкодить их в коде. Реализация может читать секреты из защищенных

хранилищ или окружения и предоставлять их клиентам по запросу.
- **SideInputProviderABC** – интерфейс доступа к побочным входным данным для обогащения [37]. Предоставляет пайплайну доступ к дополнительным данным (например, справочникам или предыдущим вычислениям), которые могут потребоваться на стадиях трансформации или валидации. Обеспечивает унифицированный способ получения таких данных, будь то чтение из файла, БД или вызов другого сервиса.

Уровень трансформации данных (Transform): отвечает за преобразование, обогащение и очистку сырых данных во внутренний формат.
- **TransformerABC** – интерфейс трансформации записи в нормализованную форму [38]. Определяет метод для преобразования сырых данных (например, из API) в структуру, соответствующую целевой схеме. Реализации инкапсулируют бизнес-логику нормализации полей. В проекте конкретные трансформеры реализованы в пайплайнах – например, `ActivityTransformer` для нормализации полей активности ChEMBL. Общие преобразования (типизация, форматирование дат, вычисление хешей) вынесены в базовые классы (например, `BaseChemblNormalizer`) [39] [40].
- **BusinessKeyDeriverABC** – интерфейс вычисления бизнес-ключа записи [41]. Предназначен для получения уникального *бизнес-ключа* по записи, устойчивого к незначительным изменениям данных и отражающего бизнес-идентичность сущности [42]. Используется для дедупликации и слияния записей: например, реализация для ChEMBL может возвращать комбинацию полей, однозначно идентифицирующую сущность (таких как составной ключ из основных идентификаторов) вместо технического первичного ключа.
- **DeduplicatorABC** – интерфейс удаления дубликатов из потока записей [43]. Определяет метод `deduplicate(records, key_fn)` для фильтрации повторяющихся записей на основе бизнес-ключа, гарантируя уникальность выходного набора [44]. Реализация может просто пропускать повторные записи или использовать стратегию слияния через `MergeStrategyABC` [45]. Например, стандартная реализация может оставлять только первую встреченную запись каждого бизнес-ключа.
- **MergeStrategyABC** – интерфейс стратегии слияния дубликатов [46]. Определяет метод `merge(records, business_key)` для объединения нескольких записей с одним бизнес-ключом в одну итоговую запись [47]. Разные реализации могут выбирать, как объединять поля: например, отдавать приоритет самым свежим данным, либо мерджить поля из разных версий. Данный интерфейс дополняет Deduplicator: Deduplicator может собирать все дубликаты, а MergeStrategy – объединять их в одну сущность [48].
- **LookupEnricherABC** – интерфейс обогащения записей на основе внешнего словаря [49]. Определяет метод для дополнения записи информацией из стороннего источника (например, мэппинг ID на имя из справочника). Реализации могут кэшировать справочные данные и добавлять недостающие поля к записям.
- **HasherABC** – интерфейс хеширования записей и ключей [50]. Предоставляет функции для вычисления хеш-значений (например, SHA-256 или MD5) для целых записей или их частей. В конвейере может использоваться для генерации контрольных сумм строк (например, `hash_row`) или бизнес-ключей (`hash_business_key`) для отслеживания изменений данных [51] [52]. В проекте, например, `hash_row` вычисляется для каждой записи для контроля версионности и детерминизма результатов.

Уровень валидации и контроля качества (Validate & Schema): отвечает за проверку соответствия данных ожиданиям и бизнес-правилам.
- **ValidatorABC** – интерфейс валидатора записи по схеме [53]. Определяет метод `validate(record, schema)`, возвращающий структурированный результат валидации [54]. Реализация проверяет запись на соответствие заданной схеме: типы данных, диапазоны значений, обязательные поля, форматы и т.д. [55]. В проекте дефолтная реализация –

`PanderaValidatorImpl`, использующая Pandera DataFrameModel для проверки DataFrame целиком [56]. Валидатор **не** изменяет данные, он только проверяет и сигнализирует об ошибках, которые затем могут быть обработаны (например, собраны через DQRuleABC или вызовут исключение, обрабатываемое ErrorPolicyABC).
- **SchemaProviderABC** – интерфейс поставщика схемы данных [57]. Определяет контракт получения объекта схемы (например, Pandera DataFrameModel) для данного набора данных. Используется пайплайном для загрузки/вывода нужной схемы при валидации. Реализация в проекте – **Schema Registry**: централизованный реестр Pandera-схем, откуда по имени сущности можно получить актуальную схему для валидатора [58] [59].
- **DQRuleABC** – интерфейс правила контроля качества данных [60]. Каждое правило анализирует набор записей и выявляет проблемы качества (возвращает объекты DQIssue) – например, пропуски, некорректные форматы, нарушения бизнес-правил [60]. Правила могут выполняться после основных стадий ETL для формирования отчета о качестве данных. Реализации могут включать, например, проверку на уникальность ключей, на непротиворечивость связанных полей и т.п. Результаты правил могут фиксироваться через ProgressReporterABC для итоговой статистики качества [61].

Уровень записи данных (Write & Storage): отвечает за сохранение выходных данных конвейера и сопутствующих артефактов.
- **WriterABC** – интерфейс записи коллекции записей в хранилище [62]. Определяет контракт на сохранение результатов пайплайна, будь то файл, база данных или облачное хранилище. Должен поддерживать атомарность операции записи (например, запись во временный файл с последующей заменой) для обеспечения целостности данных [63]. В проекте реализован унифицированный writer – `UnifiedOutputWriter`, осуществляющий атомарную запись DataFrame в CSV/Parquet, вычисление контрольных сумм и запись метаданных [2] [64].
- **PathStrategyABC** – интерфейс стратегии построения путей вывода данных [65]. Определяет, как сформировать путь (директорию/имя файла) для сохранения результатов, исходя из контекста (например, именования сущности, даты, профиля и др.) [66]. Реализации позволяют гибко менять структуру хранения результатов (например, раскладывать по папкам по дате или по типу сущностей) без изменения кода writer'ов. По умолчанию, проект использует стандартную стратегию: результаты пайплайна сохраняются в директорию, сформированную из кода пайплайна и метки запуска (при необходимости эту стратегию можно расширить).
- **MetadataWriterABC** – интерфейс сохранения служебных метаданных результатов [67]. Определяет методы записи метаданных о выполнении пайплайна – например, параметров запуска, версий схем, количества обработанных записей, хешей артефактов – в отдельный артефакт (JSON/YAML файл). В реализации BioETL метаданные о каждом запуске сохраняются в файл (например, `run_meta.yaml`), содержащий код пайплайна, версию, хеш конфигурации, число записей и контрольные суммы файлов [68] [69]. Это позволяет отслеживать происхождение и качество данных каждого запуска.

Правила и соглашения проекта

Именование и структура: Все интерфейсы-ABC имеют суффикс *ABC* в названии класса (например, `RetryPolicyABC`), а их стандартные реализации – суффикс *Impl* или особое имя. Компоненты организованы по слоям: базовые абстракции находятся в пакете `bioetl.core` (pipeline, io, logging, etc.), реализации клиентов – в `bioetl.clients` (с подразделами по источникам), специфичные пайплайны – в `bioetl.pipelines.<source>`, схемы – в `bioetl.schemas`. Такая структура отражает разделение ответственности: PipelineBase и Core-сервисы предоставляют каркас, pipeline-пакеты содержат бизнес-логику конкретных источников, а слой clients инкапсулирует детали извлечения данных [70] [71]. Конфигурации хранятся отдельно

в `configs/` (профили, YAML источников) и загружаются через `ConfigResolverABC` и фабрики, обеспечивая разделение кода и конфигурации.

Валидация и маршрутизация данных: Проверка данных осуществляется на этапе `validate()` с помощью Pandera-схем: при несоответствии записи схеме валидатор может бросить исключение или отметить запись как ошибочную в результирующем объекте `ValidationResult`. Пайплайн настроен детерминированно – при включенной строгой валидации pipeline прерывается при первой же ошибке, либо (если настроено иначе) невалидные записи пропускаются дальше, а информация о них накапливается через DQRuleABC и ProgressReporterABC (метод `record_rejected()` с указанием причины) [72]. При возникновении исключений на этапах (сетевые ошибки, сбои преобразования) оркестратор вызывает `ErrorPolicyABC.decide()` для определения дальнейших действий – например, *skip* для пропуска текущей записи или *abort* для остановки пайплайна [7]. Таким образом, маршрутизация потока данных управляется политиками: успешные данные проходят все стадии и сохраняются, ошибочные – либо останавливают процесс, либо отходят в отчеты/логи качества. Дубликаты записей в потоке объединяются DeduplicatorABC: повторные экземпляры с тем же бизнес-ключом не проходят в финальный набор, либо сливаются MergeStrategyABC в одну запись. Все отброшенные или объединенные данные фиксируются в счетчиках (ProgressReporterABC) и могут сохраняться для анализа качества (например, через QC-артефакты) [73].

Тестирование: Проект строго следует принципам детерминизма и покрывается тестами. Каждый новый пайплайн сопровождается модульными тестами (`tests/bioetl/pipelines/...`) и, при необходимости, интеграционными тестами [74]. Существует практика *golden tests* – проверок, удостоверяющихся, что при неизменных входных данных pipeline выдает идентичный результат (детерминизм по хешам, порядку строк и т.д.), даже после изменений в коде. Для общих компонентов (ABC и их реализаций) поддерживается более высокая покрытие тестами, чем для специфичных, чтобы гарантировать стабильность каркаса [75]. Перед добавлением нового компонента разработчики проверяют, нет ли уже существующего ABC для этой задачи [76]. Если приходится создавать новый ABC, его документация добавляется в `docs/abcs/`, а соответствующий default/impl класс регистрируется в реестрах `abc_registry.yaml` и `abc_impls.yaml` [77]. Такой подход и покрытие тестами обеспечивают качество и облегчает отладку: при сбоях детальные структурированные логи (LoggerAdapterABC) и метрики (TracerABC) позволяют быстро идентифицировать проблему, а воспроизводимость запусков (через фиксированные профили и dry-run) упрощает диагностику и исправление.

---

[1] [28] [29] [35] [37] [38] [49] [50] [57] [62] [67] GitHub
https://github.com/SatoryKono/BioactivityDataAcquisition/blob/47444b81a28f5b8397ab197f5bc608866d84a7d5/docs/reference/abc/INDEX.md

[2] [63] [64] GitHub
https://github.com/SatoryKono/BioactivityDataAcquisition/blob/47444b81a28f5b8397ab197f5bc608866d84a7d5/docs/02-pipelines/00-pipeline-base.md

[3] [4] GitHub
https://github.com/SatoryKono/BioactivityDataAcquisition/blob/47444b81a28f5b8397ab197f5bc608866d84a7d5/docs/reference/abc/24-stage-abc.md

[5] GitHub
https://github.com/SatoryKono/BioactivityDataAcquisition/blob/47444b81a28f5b8397ab197f5bc608866d84a7d5/docs/reference/abc/14-pipeline-hook-abc.md

6 7 GitHub
https://github.com/SatoryKono/BioactivityDataAcquisition/blob/47444b81a28f5b8397ab197f5bc608866d84a7d5/docs/reference/abc/06-error-policy-abc.md

8 GitHub
https://github.com/SatoryKono/BioactivityDataAcquisition/blob/47444b81a28f5b8397ab197f5bc608866d84a7d5/docs/reference/abc/02-cli-command-abc.md

9 10 11 GitHub
https://github.com/SatoryKono/BioactivityDataAcquisition/blob/47444b81a28f5b8397ab197f5bc608866d84a7d5/docs/reference/abc/08-logger-adapter-abc.md

12 13 72 73 GitHub
https://github.com/SatoryKono/BioactivityDataAcquisition/blob/47444b81a28f5b8397ab197f5bc608866d84a7d5/docs/reference/abc/15-progress-reporter-abc.md

14 15 16 GitHub
https://github.com/SatoryKono/BioactivityDataAcquisition/blob/47444b81a28f5b8397ab197f5bc608866d84a7d5/docs/reference/abc/25-tracer-abc.md

17 18 19 20 22 26 27 70 71 GitHub
https://github.com/SatoryKono/BioactivityDataAcquisition/blob/47444b81a28f5b8397ab197f5bc608866d84a7d5/docs/02-pipelines/clients/06-clients-overview.md

21 23 24 25 GitHub
https://github.com/SatoryKono/BioactivityDataAcquisition/blob/47444b81a28f5b8397ab197f5bc608866d84a7d5/docs/reference/abc/23-source-client-abc.md

30 31 GitHub
https://github.com/SatoryKono/bioactivity_data_acquisition/blob/98f17f432532cddd56d0a07fd4796b37c54677ec/src/bioetl/core/http/rate_limiter.py

32 33 GitHub
https://github.com/SatoryKono/BioactivityDataAcquisition/blob/47444b81a28f5b8397ab197f5bc608866d84a7d5/docs/reference/abc/19-retry-policy-abc.md

34 39 40 51 52 56 75 76 77 GitHub
https://github.com/SatoryKono/BioactivityDataAcquisition/blob/47444b81a28f5b8397ab197f5bc608866d84a7d5/docs/project/04-architecture-and-duplication-reduction.md

36 58 59 GitHub
https://github.com/SatoryKono/BioactivityDataAcquisition/blob/47444b81a28f5b8397ab197f5bc608866d84a7d5/README.md

41 42 GitHub
https://github.com/SatoryKono/BioactivityDataAcquisition/blob/47444b81a28f5b8397ab197f5bc608866d84a7d5/docs/reference/abc/00-business-key-deriver-abc.md

43 44 45 GitHub
https://github.com/SatoryKono/BioactivityDataAcquisition/blob/47444b81a28f5b8397ab197f5bc608866d84a7d5/docs/reference/abc/04-deduplicator-abc.md

46 47 48 GitHub
https://github.com/SatoryKono/BioactivityDataAcquisition/blob/47444b81a28f5b8397ab197f5bc608866d84a7d5/docs/reference/abc/10-merge-strategy-abc.md

53 54 55 GitHub
https://github.com/SatoryKono/BioactivityDataAcquisition/blob/47444b81a28f5b8397ab197f5bc608866d84a7d5/docs/reference/abc/27-validator-abc.md

60 61 GitHub
https://github.com/SatoryKono/BioactivityDataAcquisition/blob/47444b81a28f5b8397ab197f5bc608866d84a7d5/docs/reference/abc/05-dq-rule-abc.md

65 66 GitHub
https://github.com/SatoryKono/BioactivityDataAcquisition/blob/47444b81a28f5b8397ab197f5bc608866d84a7d5/docs/reference/abc/13-path-strategy-abc.md

68 69 GitHub
https://github.com/SatoryKono/bioactivity_data_acquisition/blob/98f17f432532cddd56d0a07fd4796b37c54677ec/src/bioetl/core/io/output.py

74 GitHub
https://github.com/SatoryKono/BioactivityDataAcquisition/blob/47444b81a28f5b8397ab197f5bc608866d84a7d5/docs/guides/adding-new-pipeline.md