PEXL

Синтаксис PEXL

PEXL (Poll Expression Language) — язык выражений для доступа к данным интервью, сравнения значений, логических условий, работы с переменными, параметрами, тегами ответов, подсчётами и псевдослучайными значениями.

Документ описывает синтаксис, реально поддерживаемый грамматикой `pexl.g`, парсером `pexl.js` и тестами проекта.

---

1. Общая модель выражений

Выражение PEXL вычисляется в контексте объекта `interview` и возвращает одно значение:

- `true` / `false` для логических и сравнительных выражений;
- строку;
- число;
- `null`, если значение не найдено;
- иногда объект вопроса или ответа как внутреннее промежуточное значение, но в типовых пользовательских выражениях обычно используются строка, число или булево значение.

Примеры:

```
Q1
Q1 == "Q one"
Q1.A3 == "A three"
Q5.A >= 22
PARAM("PARAM1") == "PAR1"
$X = RAND(1, 10)
```

---

2. Лексические элементы

2.1. Пробелы

Пробельные символы допускаются между токенами и игнорируются.

Примеры:

```
Q1.A3=="A three"
Q1.A3 == "A three"
Q1 .  A3
```

Практически использовать стоит обычную читаемую запись без разрыва внутри ссылок вида `Q1.A3`.

2.2. Числа

Поддерживаются только целые числа:

```
0
1
22
100500
```

Формально: последовательность цифр `\d+`.

Отрицательные, дробные и экспоненциальные числа синтаксисом не предусмотрены.

2.3. Строки

Поддерживаются строковые литералы в двойных кавычках:

```
"Hello"
"A three"
"06bf20de-9658-4afd-9e7b-d712697ccdc7"
```

Также лексер распознаёт литералы в одинарных кавычках, но в грамматике языка они не используются как допустимые значения выражений. Поэтому в корректном PEXL следует

использовать **двойные кавычки**.

2.4. Ключевые слова и служебные токены

Поддерживаются следующие ключевые слова и служебные конструкции:

- `AND`
- `OR`
- `NOT`
- `PARAM`
- `VAR`
- `QUUID`
- `ATUUID`
- `AQUUID`
- `TATUUID`
- `TAQUUID`
- `CQUUID`
- `CRUUID`
- `RAND`
- `COUNT`
- `Q`
- `R`
- `A`
- `T.<TAG_NAME>`

2.5. Операторы и знаки пунктуации

- `==`
- `!=`
- `<`
- `>`
- `<=`
- `>=`
- `=`
- `(`
- `)`
- `.`
- `,`

---

3. Базовая грамматика выражений

На верхнем уровне PEXL поддерживает два больших класса выражений:

1. **вычисление/сравнение значения**;
2. **присваивание переменной**.

Упрощённо:

```
STMT ::= STMT == STMT
       | STMT != IVALUE
       | STMT <  IVALUE
       | STMT >  IVALUE
       | STMT <= IVALUE
       | STMT >= IVALUE
       | STMT AND STMT
       | STMT OR STMT
       | NOT STMT
       | ( STMT )
       | IVALUE
       | ASSIGN_VALUE
```

Где `IVALUE` — это обычное вычисляемое значение, а `ASSIGN_VALUE` — присваивание переменной.

---

4. Приоритет операторов

Согласно грамматике проекта, используются такие приоритеты:

1. `NOT` — самый высокий;
2. сравнения: `==`, `!=`, `<`, `<=`, `>`, `>=`;
3. `OR`;
4. `AND`;
5. `=` (присваивание).

> Важно: приоритеты в `pexl.g` заданы не совсем так, как в большинстве языков программирования. Для сложных выражений рекомендуется **всегда явно ставить скобки**.

Примеры безопасной записи:

```
Q1.A3 AND (NOT Q2.A1)
(Q1.A3 == "A three") AND (Q2.A1 == "A one")
($X = RAND(1, 10)) AND ($X >= 1)
```

---

5. Допустимые виды значений (`IVALUE`)

`IVALUE` может быть одним из следующих типов:

- значение вопроса;
- значение ответа;
- текст ответа;
- значение тега ответа;
- количество ответов;
- значение параметра;
- значение переменной;
- доступ по UUID;
- результат `RAND`;
- числовой литерал;
- строковый литерал.

Подробно — ниже.

---

6. Ссылки на вопросы

6.1. Вопрос `Qx`

Форма:

```
Q<number>
```

Примеры:

```
Q1
Q2
Q10
```

Семантика:

- возвращает вопрос по `questionCode`;
- в сравнении используется `question.question` — текст вопроса;
- в булевом контексте истинно, если вопрос найден.

Примеры:

```
Q1
Q1 == "Q one"
NOT Q123
```

6.2. Строка табличного вопроса `Qx.Ry`

Форма:

```
Q<number>.R<number>
```

Примеры:

```
Q3.R1
Q3.R2
```

Семантика:

- возвращает вопрос-строку таблицы;
- в сравнении используется текст вопроса этой строки.

Примеры:

```
Q3.R1 == "Q three R one"
Q3.R2
```

---

7. Ссылки на ответы

7.1. Конкретный ответ `Qx. Ay`

Форма:

```
Q<number>. A<number>
```

Пример:

```
Q1. A3
```

Семантика:

- ищет ответ с кодом `Ay` у вопроса `Qx`;
- возвращает объект ответа;
- при сравнении используется `answerText`;
- если `answerText` выглядит как целое число, при некоторых сравнениях он будет приведён к числу.

Примеры:

```
Q1. A3 == "A three"
Q7. A3 == 22
```

7.2. Конкретный ответ в строке таблицы `Qx. Ry. Az`

Форма:

```
Q<number>. R<number>. A<number>
```

Пример:

```
Q3. R1. A3
```

Примеры использования:

```
Q3.R1.A3 == "A three"
Q3.R2.A1
```

---

8. Текст ответа

8.1. Первый/основной текст ответа вопроса `Qx.A`

Форма:

```
Q<number>.A
```

Пример:

```
Q5.A
```

Семантика:

- возвращает `answerText` первого подходящего ответа вопроса;
- если текст числовой, в ряде операций используется как число.

Примеры:

```
Q1.A == "A three"
Q5.A == 22
Q5.A >= 22
```

8.2. Текст ответа строки таблицы `Qx.Ry.A`

Форма:

```
Q<number>.R<number>.A
```

Пример:

```
Q3.R2.A
```

Примеры:

```
Q3.R2.A == "33"
Q3.R1.A == "A three"
```

---

9. Теги ответов

Тег кодируется как специальный токен вида:

```
T.<TAG_NAME>
```

Допустимые символы имени тега: буквы, цифры, `_`, `-`.

Примеры:

```
T.A_TAG_INT
T.A_TAG_STR
T.score-1
```

9.1. Тег конкретного ответа `Qx.Ay.T.TAG`

Форма:

```
Q<number>.A<number>.T.<TAG>
Q<number>.R<number>.A<number>.T.<TAG>
```

Примеры:

```
Q1.A3.T.A_TAG_INT
Q3.R1.A3.T.A_TAG_STR
```

Семантика:

- возвращает значение тега из `answerTags[tag]`;
- если значение тега числовое, оно приводится к числу;
- если тега нет, возвращается `null`.

Примеры:

```
Q1.A3.T.A_TAG_INT == 55
Q1.A3.T.A_TAG_STR == "Tag value"
```

9.2. Тег первого/основного ответа вопроса `Qx.A.T.TAG`

Форма:

```
Q<number>.A.T.<TAG>
Q<number>.R<number>.A.T.<TAG>
```

Примеры:

```
Q1.A.T.A_TAG_INT
Q3.R1.A.T.A_TAG_STR
```

Семантика:

- перебирает ответы вопроса и возвращает первый найденный тег с таким именем;
- если тег нигде не найден, возвращает `null`.

Примеры:

```
Q1.A.T.A_TAG_INT == 55
Q1.A.T.A_TAG_STR == "Tag value"
```

---

10. Подсчёт количества ответов

10.1. Количество ответов вопроса `Qx.COUNT`

Форма:

```
Q<number>.COUNT
```

Пример:

```
Q2.COUNT
```

Семантика:

- возвращает число ответов вопроса;
- в циклическом контексте результат зависит от `loopType`.

Примеры:

```
Q2.COUNT == 4
Q6.COUNT == 6
```

10.2. Количество ответов строки таблицы `Qx.Ry.COUNT`

Форма:

```
Q<number>.R<number>.COUNT
```

Пример:

```
Q3.R1.COUNT
```

10.3. Количество заполненных строк таблицы `Qx.R.COUNT`

Форма:

```
Q<number>.R.COUNT
```

Пример:

```
Q3.R.COUNT == 3
```

Семантика:

- считает количество дочерних вопросов, чьи коды начинаются с `Qx.R`.

11. Доступ по UUID

11.1. Вопрос по UUID: `QUUID("...")`

Форма:

```
QUUID("<question_uuid>")
```

Семантика:

- ищет вопрос по `questionId`;
- в сравнении используется текст вопроса.

Примеры:

```
QUUID("06bf20de-9658-4afd-9e7b-d712697ccdc7")
QUUID("06bf20de-9658-4afd-9e7b-d712697ccdc7") == "Q one"
```

11.2. Конкретный ответ по UUID: `ATUUID("...")`

Форма:

```
ATUUID("<answer_template_uuid>")
```

Семантика:

- ищет ответ по `templateAnswerId`;
- в сравнении используется `answerText`.

Примеры:

```
ATUUID("a2101b08-b18e-4137-abb0-5eb2e44d751e")
ATUUID("a2101b08-b18e-4137-abb0-5eb2e44d751e") == "A three"
```

11.3. Текст ответа по UUID: `AQUUID("...")`

Форма:

```
AQUUID("<question_uuid>")
```

Семантика:

- возвращает текст первого/основного ответа вопроса по UUID вопроса;
- для числовых строк поддерживаются числовые сравнения.

Примеры:

```
AQUUID("06bf20de-9658-4afd-9e7b-d712697ccdc7") == "A three"
AQUUID("c82d8380-cae4-43c1-93be-3cf5a06449f3") >= 22
```

11.4. Тег конкретного ответа по UUID: `TATUUID("answerUuid", "TAG")`

Форма:

```
TATUUID("<answer_template_uuid>", "<TAG>")
```

Примеры:

```
TATUUID("a2101b08-b18e-4137-abb0-5eb2e44d751e", "A_TAG_INT") == 55
TATUUID("a2101b08-b18e-4137-abb0-5eb2e44d751e", "A_TAG_STR") == "Tag value"
```

11.5. Тег первого/основного ответа вопроса по UUID: `TAQUUID("questionUuid", "TAG")`

Форма:

```
TAQUUID("<question_uuid>", "<TAG>")
```

Примеры:

```
TAQUUID("06bf20de-9658-4afd-9e7b-d712697ccdc7", "A_TAG_INT") == 55
TAQUUID("06bf20de-9658-4afd-9e7b-d712697ccdc7", "A_TAG_STR") == "Tag value"
```

11.6. Количество ответов вопроса по UUID: `CQUUID("...")`

Форма:

```
CQUUID("<question_uuid>")
```

Примеры:

```
CQUUID("06bfcccc-9658-4afd-9e7b-d712697ccdc7") == 6
```

11.7. Количество строк таблицы по UUID: `CRUUID("...")`

Форма:

```
CRUUID("<table_uuid>")
```

Примеры:

```
CRUUID("7a33bdab-d53b-4958-a293-60e3fded8aa8") == 3
```

---

12. Параметры интервью

Форма:

```
PARAM("<name>")
```

Примеры:

```
PARAM("PARAM1")
PARAM("PARAM1") == "PAR1"
PARAM("PARAM1") == PARAM("param2")
```

Семантика:

- читает `interview.params[name]`;
- регистр имени зависит от того, как значения реально хранятся в объекте `params`.

---

13. Переменные

PEXL поддерживает две формы обращения к переменным.

13.1. Переменная вида `$NAME`

Форма:

```
$<name>
```

Допустимые символы имени: буквы, цифры, `_`, `-`.

Примеры:

```
$VAR1
$var2
$MY-VAR
```

Чтение:

```
$VAR1 == "VAL1"
```

Присваивание:

```
$VAR3 = "Test3"
$N = 22
$X = Q1
$Y = Q3.R1.A3
$R = RAND(1, 10)
```

13.2. Переменная вида `VAR("NAME")`

Форма:

```
VAR("<name>")
```

Примеры:

```
VAR("VAR1")
VAR("var2") == "val2"
VAR("VAR4") = "Test4"
VAR("VAR_QQ") = Q1
```

Семантика у `$NAME` и `VAR("NAME")` общая:

- чтение идёт из `interview.vars[name]`;
- запись идёт в `interview.vars[name]`.

13.3. Что можно присваивать переменной

Правая часть присваивания может быть:

- строкой;
- числом;
- вопросом `Q...` / `QUUID(...)` — в переменную попадёт текст вопроса;
- ответом `Q...A...` / `ATUUID(...)` — в переменную попадёт `answerText`;
- `COUNT`-значением;
- tag-значением;
- `Qx.A` / `Qx.Ry.A` / `AQUUID(...)`;
- другой переменной;
- `RAND(...)`.

Примеры:

```
$VAR3 = "Test3"
$VAR_Q = Q1
$VAR_Q = Q3.R1.A3
$MY = $VAR3
VAR("VAR4") = "Test4"
```

```
VAR("VAR_QQ") = Q1
```

---

14. Псевдослучайные значения

14.1. Короткая форма `RAND(n)`

Форма:

```
RAND(<max>)
```

Семантика:

- возвращает целое число в диапазоне от `1` до `n` включительно.

Пример:

```
RAND(5)
```

14.2. Полная форма `RAND(min, max)`

Форма:

```
RAND(<min>, <max>)
```

Семантика:

- возвращает целое число в диапазоне `[min, max]` включительно;
- если аргументы переданы в обратном порядке, фактически используются `Math.min` и `Math.max`.

Примеры:

```
RAND(3, 7)
RAND(42, 42)
$X = RAND(1, 100)
RAND(10, 20) >= 10 AND RAND(10, 20) <= 20
```

Особенность реализации:

- генератор детерминированно инициализируется от `interview.id`;
- для одного и того же `interview.id` результат воспроизводим между запусками;
- если `interview.id` отсутствует, используется резервный seed.

Совместимость с конвертерами:

- `RAND(...)` корректно проходит через `convert2UUID` (выражение → UUID-форма для хранения в БД) и `convert2QA` (UUID-форма → читаемая форма для отображения);
- внутри `RAND(...)` нет ничего, что нужно конвертировать в UUID, поэтому литерал просто пробрасывается без изменений;
- допускается использование `RAND` в любых выражениях, которые сохраняются в poll-объекте: условиях ветвления, присваиваниях переменным и т.д.

---

15. Операторы сравнения

Поддерживаются:

- `==`
- `!=`
- `<`
- `>`
- `<=`
- `>=`

Примеры:

```
Q1 == "Q one"
Q1.A3 == "A three"
Q5.A != 20
Q5.A < 123
Q5.A <= 22
Q5.A > 21
Q5.A >= 22
```

15.1. Правила сравнения

Реализация сравнения устроена так:

1. Левая часть должна быть truthy, иначе результат выражения будет ложным.
2. Если **и левое, и правое** значения распознаются как целые числа, сравнение выполняется как числовое.

3. Иначе сравнение выполняется как обычное JS-сравнение значений/строк.

Примеры числового сравнения:

```
Q7.A3 == 22
Q7.A3 < Q7.A4
Q5.A >= 22
```

Примеры строкового сравнения:

```
Q7.A1 == "AAAA"
Q7.A1 < Q7.A2
```

15.2. Сравнение значения слева и справа

С обеих сторон могут стоять не только литералы, но и выражения:

```
Q7.A1 == Q7.A1
"BBBB" == Q7.A2
22 == Q7.A3
PARAM("PARAM1") == PARAM("param2")
```

---

16. Логические операторы

16.1. `AND`

Форма:

```
<expr> AND <expr>
```

Примеры:

```
Q1.A3 == "A three" AND Q2.A1 == "A one"
Q1 == "Q one" AND Q3.R1 == "Q three R one"
```

Семантика: логическое И.

16.2. `OR`

Форма:

```
<expr> OR <expr>
```

Пример:

```
Q1 == "Q one" OR Q1 == "Wrong Q"
```

16.3. `NOT`

Форма:

```
NOT <expr>
```

Примеры:

```
NOT Q1
NOT Q123
NOT Q1.A3 == "A three"
Q1.A3 AND NOT Q2222.A1 AND Q5555.A2
```

16.4. Скобки

Форма:

```
( <expr> )
```

Примеры:

```
Q1 AND (Q1 == "Q one" OR Q1 == "Wrong Q")
Q1.A3 AND (Q2.A1 == "A one" OR Q1111.A111 == "Wrong Q")
Q1.A3 AND (NOT Q2222.A1) AND Q5555.A2
```

---

17. Поведение при отсутствии данных

Если вопрос, ответ, тег или переменная не найдены, обычно возвращается `null`.

Типичные последствия:

- просто выражение `Q123` в булевом контексте считается ложным;
- `NOT Q123` даёт `true`;
- сравнение вида `Q111.A == "..."` не обязано возвращать строго `false`, потому что промежуточно может участвовать `null`.

Из тестов следует, что при проектировании выражений лучше явно учитывать возможность отсутствия значения.

---

18. Контекст циклов

Синтаксис выражения не меняется, но семантика поиска вопроса/ответа может зависеть от `options`, переданных в `execute(...)`.

Поддерживаются режимы:

- без цикла;
- `loopType: "DOWHILE"`;
- `loopType: "FOREACH"`;
- `loopType: "NONE"`.

18.1. `DOWHILE`

Требует:

```
{ loopType: "DOWHILE", loopPass: <number> }
```

В этом режиме конструкции вроде:

- `Qx`
- `Qx.Ay`
- `Qx.A`

- `QUUID(...)`
- `ATUUID(...)`
- `AQUUID(...)`
- `CQUUID(...)`

ищут данные внутри соответствующей итерации `loopPass`, а при необходимости могут использовать значения вне цикла как fallback.

Примеры выражений:

```
Q2 == "Q two 2"
Q2.A1 == "A one. LoopPass2"
Q2.A == "A one. LoopPass2"
QUUID("10264445-3949-41b3-8b87-4cc20bf6c5c0") == "Q two 2"
ATUUID("4a8e4bd6-7500-4195-a731-158780787d21") == "A two. LoopPass2"
```

18.2. `FOREACH`

Требует:

```
{ loopType: "FOREACH", iterationTemplateAnswerId: "<uuid>" }
```

В этом режиме поиск значения выполняется в контексте конкретной итерации `iterationTemplateAnswerId`.

Примеры:

```
Q6 == "Q Loop"
Q6.A1 == "A six one"
Q6.A == "A six one 2"
AQUUID("06bfcccc-9658-4afd-9e7b-d712697ccdc7") == "A six one 2"
Q6.COUNT == 3
```

18.3. Ошибки конфигурации цикла

Реализация `execute()` выбрасывает ошибку, если:

- указан `loopPass` или `iterationTemplateAnswerId`, но не указан `loopType`;
- `loopType: "FOREACH"` без `iterationTemplateAnswerId`;
- `loopType: "DOWHILE"` без `loopPass`;
- указан неизвестный `loopType`.

19. Полный каталог синтаксических форм

Ниже перечислены все поддерживаемые формы пользовательских выражений.

19.1. Литералы

```
123
"text"
```

19.2. Вопросы и ответы

```
Q1
Q3. R1
Q1. A3
Q3. R1. A3
Q1. A
Q3. R2. A
```

19.3. Теги

```
Q1. A3. T. A_TAG_INT
Q3. R1. A3. T. A_TAG_STR
Q1. A. T. A_TAG_INT
Q3. R1. A. T. A_TAG_STR
```

19.4. Подсчёты

```
Q2. COUNT
Q3. R1. COUNT
Q3. R. COUNT
```

19.5. UUID-функции

```
QUUID("question-uuid")
ATUUID("answer-uuid")
AQUUID("question-uuid")
TATUUID("answer-uuid", "TAG")
TAQUUID("question-uuid", "TAG")
CQUUID("question-uuid")
CRUUID("table-uuid")
```

19.6. Параметры и переменные

```
PARAM("PARAM1")
$VAR1
VAR("VAR1")
```

19.7. Присваивания

```
$X = "abc"
$X = 10
$X = Q1
$X = Q1.A3
$X = Q1.A
$X = Q1.A3.T.A_TAG_INT
$X = Q2.COUNT
$X = AQUUID("question-uuid")
$X = TAQUUID("question-uuid", "TAG")
$X = RAND(1, 10)

VAR("X") = "abc"
VAR("X") = Q1
VAR("X") = Q1.A3
```

19.8. RAND

```
RAND(5)
RAND(1, 10)
```

19.9. Логика и сравнения

```
Q1 == "Q one"
Q1.A3 != "Wrong"
Q5.A < 100
Q5.A <= 22
Q5.A > 10
Q5.A >= 22
NOT Q123
Q1 AND Q2
Q1 OR Q2
(Q1 AND Q2) OR Q3
```

---

20. Практические рекомендации

1. Используйте **двойные кавычки** для строк.
2. Для сложной логики всегда ставьте **скобки**.
3. Если значение может отсутствовать, учитывайте возможность `null`.
4. Для числовых сравнений используйте числовые литералы:

```
Q5.A >= 22
```

5. Для читаемости разделяйте длинные выражения пробелами:

```
Q1.A3 == "A three" AND Q2.A1 == "A one"
```

---

21. Примеры полных выражений

Простые проверки

```
Q1
Q1.A3
NOT Q123
```

Сравнение вопросов и ответов

```
Q1 == "Q one"
Q1.A3 == "A three"
Q3.R1 == "Q three R one"
Q3.R1.A3 == "A three"
```

Работа с числовыми ответами

```
Q5.A == 22
Q5.A != 20
Q5.A < 123
Q5.A >= 22
```

Теги

```
Q1.A3.T.A_TAG_INT == 55
Q1.A.T.A_TAG_STR == "Tag value"
TATUUID("a2101b08-b18e-4137-abb0-5eb2e44d751e", "A_TAG_INT") == 55
TAQUUID("06bf20de-9658-4afd-9e7b-d712697ccdc7", "A_TAG_STR") == "Tag value"
```

Параметры и переменные

```
PARAM("PARAM1") == "PAR1"
$VAR1 == "VAL1"
VAR("VAR1") == "VAL1"
$TMP = Q1.A
VAR("N") = RAND(1, 100)
```

Сложная логика

```
Q1.A3 AND (Q2.A1 == "A one" OR Q1111.A111 == "Wrong Q")
Q1 AND (Q1 == "Q one" OR Q1 == "Wrong Q")
Q1.A3 AND NOT Q2222.A1 AND Q5555.A2
```

---

22. Краткая формальная сводка

```
Вопрос:
  Qn
  Qn.Rm

Ответ:
  Qn.Am
  Qn.Rm.Ak

Текст ответа:
  Qn.A
  Qn.Rm.A

Теги:
  Qn.Am.T.TAG
  Qn.Rm.Ak.T.TAG
  Qn.A.T.TAG
  Qn.Rm.A.T.TAG

Подсчёты:
  Qn.COUNT
  Qn.Rm.COUNT
  Qn.R.COUNT

UUID:
  QUUID("uuid")
  ATUUID("uuid")
  AQUUID("uuid")
  TATUUID("uuid", "TAG")
  TAQUUID("uuid", "TAG")
  CQUUID("uuid")
  CRUUID("uuid")
```

```
Параметры:
  PARAM("name")

Переменные:
  $NAME
  VAR("NAME")

Присваивание:
  $NAME = <value>
  VAR("NAME") = <value>

RAND:
  RAND(n)
  RAND(min, max)

Логика:
  expr AND expr
  expr OR expr
  NOT expr
  (expr)

Сравнения:
  expr == expr
  expr != expr
  expr < expr
  expr > expr
  expr <= expr
  expr >= expr
```

---

23. Источник истины

Этот документ составлен по фактической реализации проекта:

- `pexl.g` — исходная грамматика основного парсера;
- `pexl.js` — сгенерированный парсер и лексер;
- `test_pexl.js` — набор тестов, фиксирующий поддерживаемое поведение основного парсера;

- `convert2UUID.g` / `convert2UUID.js` — грамматика и парсер преобразования выражений с алиасами (`Q1.A1`) в UUID-форму для хранения в БД;
- `convert2QA.g` / `convert2QA.js` — грамматика и парсер обратного преобразования (UUID-форма → читаемые алиасы) для отображения;
- `test_convert2UUID.js`, `test_convert2QA.js` — тесты конвертеров.

Грамматики `pexl.g`, `convert2UUID.g`, `convert2QA.g` должны держаться в синхроне по набору поддерживаемых токенов и продакшнов: если в одной из них появляется новый оператор (например, `RAND`), его обязательно нужно добавить и в две другие, иначе сохранение/отображение выражений ломается.

Если реализация и документ когда-либо разойдутся, источником истины следует считать код и тесты.

Revision #3
Created 28 April 2026 15:30:04 by Admin
Updated 29 April 2026 05:53:23 by Admin