# Trace replay generator. Руководство пользователя.

## Введение

Назначение данного инструмента – помощь в локализации ошибок, обнаруженных тестовыми сценариями UniTESK. На данный момент реализована только работа со сценариями CTESK.

Инструмент принимает на вход набор трасс CTESK и для каждой генерирует следующие файлы:

- Результат анализа графа трассы.
- Сценарий на языке Си, позволяющий воспроизводить записанную в трассе последовательность вызовов сценарных методов целиком или частично для поиска ошибок.

### *Используемая терминология*

*Инструмент* – данный инструмент, **Trace Replay Generator**.

*Исходная трасса* или просто *трасса* – трасса CTESK, поступившая на вход *инструменту*.

*Исходный сценарий* – сценарий CTESK, который сгенерировал *исходную трассу*.

*Сгенерированный сценарий* или просто *сценарий* – тестовый сценарий, сгенерированный *инструментом*.

*Выходная трасса* – трасса, полученная при запуске *сгенерированного сценария*.

*Состояние* – используемое *исходным сценарием* модельное состояние целевой системы, которое попадает в *трассу*.

*Начальное состояние* – первое *состояние*, встретившееся в *трассе*.

*Конечное состояние – состояние* из *исходной трассы*, в котором было выполнено первое *тестовое воздействие*, обнаружившее ошибку, если в *исходной трассе* была ошибка, или конечное *состояние трассы* в противном случае.

*Тестовое воздействие* или *переход* – вызов сценарного метода. Идентифицируется именем вызванного сценарного метода и значениями его итерационных переменных.

*Тестовая последовательность* – последовательность тестовых воздействий, осуществляемых *сценарием*.

## Инсталляция

Инструмент интегрирован в программный пакет UniTESK Trace Tools.

## Запуск инструмента

**Под Windows:**

```
trg.bat [-noout | -debug] [-nocode] [-pp] имена_трасс
```

**Под Linux и другими ОС:**

```
java [-cp <ClassPath>] -jar [<Path>/]trace-replay-generator.jar
     [-noout | -debug] [-nocode] [-pp] имена_трасс
```

(Значения ClassPath и Path зависит от конкретной системы)

### *Значения параметров*

`[-noout]` – не генерировать файл с анализом *трассы*

`[-debug]` – генерировать дополнительную информацию об анализе *трассы*

`[-nocode]` – не генерировать код *сценария*

`[-pp]` – генерировать в коде *сценария* команды препроцессора (`#if` вместо `if`)

`имена_трасс` – список имен файлов *входных трасс*

# Использование сгенерированного сценария

*Сгенерированный сценарий* можно использовать следующим образом:

1. Заменить в уже имеющемся методе `main()` запуск *исходного сценария* на запуск метода `main_.....()` из *сгенерированного сценария*. Например, в примере из дистрибутива `account` вызов
   ```
   account_scenario(argc, argv);
   ```
   меняется на
   ```
   main_account_scenario_2007_06_15_11_00_01(argc, argv);
   ```
   **или**
   переименовать в *сценарии* метод `main_.....()` в `main()` (не рекомендуется).
2. Собрать (например, в проекте MSVS) *сгенерированный сценарий* вместе с *исходным сценарием*, и со всем, что необходимо для его выполнения (спецификации, медиаторы, тестируемая система, TSBasis, другие внешние системы)
3. Откомпилировать полученный проект.
4. Запустить результат компиляции на исполнение или в отладчике с нужными параметрами командной строки.
5. Проверить полученную в результате повторного прогона трассу на сообщения об ошибках.
6. В зависимости от наличия и типа ошибок – изменить параметры запуска *сценария* и повторить шаги 4-5.

В режиме генерации по умолчанию выбор выполняемых *сценарием тестовых воздействий* осуществляется командами языка Си (`if`), при этом возможно управление *сценарием* через параметры командной строки, в том числе режим автоматической локализации ошибок.

Параметр командной строки инструмента «`-pp`» включает режим генерации, в котором выбор выполняемых *сценарием* тестовых воздействий осуществляется директивами препроцессора (`#if`). При этом влиять на работу сценария можно только путём редактирования параметра `DEFAULT_PATH_NUMBER` в коде сценария с последующей перекомпиляцией. Такой режим более удобен для ручной пошаговой отладки сценария, так как тестовые воздействия, которые не будут выполнены, игнорируются компилятором.

## *Сообщения об ошибках*

Независимо от наличия ошибок в *исходной трассе*, в полученной при прогоне *сценария выходной трассе* возможно появление следующего сообщения:

- **Scenario replay: unexpected failure** – в случае, если при прогоне *сценария* получена ошибка в неожиданном месте. Работа *сценария* на этом прекращается.

При наличии в *исходной трассе* ошибок, в конце *выходной трассы* обязательно встречается или сообщение о неожиданной ошибке, или одно из следующих сообщений:

- **Scenario replay: repeatable failure** – если при воспроизведении последовательности вызовов *исходной трассы* удалось воспроизвести ожидаемую ошибку
- **Scenario replay: could not repeat failure** – если ожидаемую ошибку воспроизвести не удалось

Ошибкой считается возникновение одного из следующих событий:

1. Ошибка оракула (сообщения «**Postcondition failed**», «**Precondition failed**» и т.д.) При этом ошибки, возникшие внутри блока **serialization**, игнорируются, если сериализация в целом завершилась успешно.
2. Ошибка сценарного метода (сообщение «**Scenario function failed**»)
3. При выполнении *сгенерированного сценария* система пришла в состояние, отличное от ожидаемого.

Ошибки 1 и 2 типа могут как встретиться в исходной трассе, так и возникнуть при выполнении *сгенерированного сценария*. Возникшая при выполнении *сгенерированного сценария* ошибка считается *ожидаемой*, если она произошла в том же *тестовом воздействии*, что и в исходной трассе; совпадение типа ошибки и каких-либо других её характеристик не проверяется. Ошибка 3 типа (попадание в другое состояние) может возникнуть только при выполнении *сценария*, и потому не может быть ожидаемой.

## *Управление выполнением сценария*

*Сценарий*, сгенерированный в режиме по умолчанию, может принимать на вход параметр командной строки, управляющий полнотой воспроизведения *исходной трассы*.

Формат командной строки:

**имя_исполняемого_файла [параметры test engine] [-path <path_number>] [прочие параметры исходного сценария]**

При значении параметра **path_number = 1** сценарий проходит по кратчайшему пути до *конечного состояния* и в случае наличия в *исходной трассе* ошибки повторяет вызов, который вызвал эту ошибку в *исходном сценарии*. При каждом увеличении значения параметра на 1 (вплоть до максимально возможного), к выполняемой *сценарием тестовой последовательности* добавляется ровно 1 цикл из *исходной трассы*, согласно разбиению, которое можно увидеть в файле анализа.

В коде *сценария* содержится также параметр **DEFAULT_PATH_NUMBER**, задающий значение параметра **path_number** по умолчанию, при отсутствии параметра **-path** в командной строке. Этот параметр можно редактировать. В режиме управления препроцессором *сценарий* всегда работает со значением параметра `path_number` по умолчанию.

В начале работы *сценарий* сбрасывает в выходную трассу сообщение вида:

- **Scenario replay: trying path <*path_number*>** , содержащее текущее значение параметра

В режиме автоматической локализации ошибок (см. ниже), когда перебираются различные пути по графу состояний сценария, такое сообщение сбрасывается каждый раз перед воспроизведением очередного пути.

## *Стратегия ручной локализации ошибок.*

Если ошибка воспроизводится при значении параметра **path_number=1**, то её причину следует искать на кратчайшем пути (заголовок «**Path #1**» в анализе трассы). Ошибка может возникать непосредственно в том тестовом вызове, который её обнаруживает, а может возникнуть раньше, но проявиться не сразу.

Если при каком-то значении параметра **path_number** ошибка не воспроизводится, его следует увеличить (не обязательно на 1) и повторить попытку.

Если при каком-то значении параметра **path_number** ошибка воспроизводится – можно попробовать его уменьшить (не обязательно на 1) и повторить попытку.

Если обнаружено минимальное значение параметра **path_number**, при котором ошибка стабильно воспроизводится, следует в первую очередь исследовать тот цикл, которая впервые добавляется при данном значении параметра. Например, если при **path_number=4** ошибка еще нет, а при **path_number=5** она уже возникает, то такой цикл в анализе трассы помечена заголовком «**Path #5**». С большой вероятностью ошибка в тестируемой системе возникает в именно этом цикле, даже если она проявляется позже

### *Автоматическая локализация ошибок.*

Если *сценарий* был сгенерирован в режиме по умолчанию, то при значении параметра **path_number=0** он запускается в режиме автоматического поиска в графе состояний минимального пути, обнаруживающего ошибку. При этом *сценарий* последовательно перебирает все значения данного параметра, начиная с 1 и заканчивая максимально возможным для построенного разбиения *исходной трассы*, и останавливается при первом обнаружении ошибки. При этом в выходную трассу дополнительно сбрасываются сообщения следующего вида:

- **Scenario replay: failure found at path <*path_number*>** – если ошибка впервые возникла при указанном значении параметра.
- **Scenario replay: could not repeat failure at any path** – если ошибку не удалось воспроизвести ни при каком значении параметра.

## Известные ограничения.

### *Ограничения на входную трассу.*

Инструмент корректно работает, если для *входной трассы* выполняются **все** условия из следующего списка:

- *Исходная трасса* сгенерирована с помощью CTESK.
- *Исходная трасса* содержит запуск ровно одного тестового сценария.
- *Исходный сценарий* имеет тип **DFSM**.
- *Исходная трасса* содержит сценарные сообщения из единственного потока выполнения (при этом оракулы и медиаторы могут сбрасывать в трассу сообщения из различных потоков).
- *Исходная трасса* не была оборвана в результате ненормального завершения *исходного сценария*.
- *Исходная трасса* не содержит ошибок вне **transition** (допускаются ошибки **test engine**, следующие за ошибками внутри **transition** и вызванные ими).
- *Исходная трасса* содержит **state** между каждыми последовательными **transition**, а также перед первым **transition** и после последнего **transition**.
- Если какой-то сценарный метод встречается в *исходной трассе*, то он хотя бы раз входит в самый глубокий из своих циклов **iterate**.

При нарушении некоторых из этих условий *инструмент* выдает соответствующее сообщение об ошибке, при нарушении других – *сгенерированный сценарий* может не компилироваться или быть неработоспособным. Например, при подаче на вход *инструменту* трассы, сгенерированной с помощью JavaTESK, на выходе получается корректный анализ трассы и не имеющий смысла *сценарий* на Си.

### *Ограничения алгоритма.*

*Инструмент* строит разбиение записанного в *трассе* пути по графу модельных состояний на набор подпутей, удовлетворяющий следующим условиям:

1. Каждый подпуть является простым путем или простым циклом в графе *исходной трассы*. При этом если *начальное* и *конечное* состояния *трассы* совпадают, то все подпути являются циклами, иначе ациклический путь в разбиении ровно один, и он ведет из начального состояния в конечное.
2. В каждом подпути порядок дуг (тестовых воздействий) соответствует их порядку в *исходной трассе*.

В общем случае такое разбиение не единственно. *Инструмент* строит ровно одно такое разбиение по простейшему алгоритму и не предлагает никаких других разбиений.

Используемая *инструментом* тестовая гипотеза говорит, что на возникновение ошибки с большей вероятностью влияют последние перед её возникновением тестовые вызовы, поэтому при каждом увеличении параметра `PATH_NUMBER` на единицу в сценарий добавляется по одному циклу из конца трассы (циклы упорядочены по порядковому номеру в *исходной трассе* последней дуги, входящей в цикл). Других стратегий наращивания тестовой последовательности не предусмотрено.

## *Дополнительные ограничения.*

- Инструмент предназначен для поиска детерминированных ошибок. Если на одной и той же *тестовой последовательности* ошибка возникнет недетерминированно, то полезность данного инструмента для ее локализации снижается.
- Поскольку *инструмент* пользуется Trace Analyzer'ом, он наследует все его ограничения по разбору трасс. В частности, могут некорректно обрабатываться трассы, содержащие не-ASCII данные внутри блока CDATA.
- Поддерживается режим обработки *входной трассы* только до первой ошибки. При этом анализируется путь до этой ошибки и игнорируется все, что произошло после неё.
- При автоматическом поиске минимального пути, воспроизводящего ошибку, все прогоны *исходного сценария* выполняются в рамках одного процесса ОС, а для сброса состояния между прогонами используется вызов методов `finish()` и `init()` *исходного сценария*. Если этого недостаточно для полноценного сброса состояния, то результат автоматической локализации может отличаться от результата ручной.
- Эквивалентные состояния сценарного КА, в том числе полученные из разных запусков сценария, должны иметь одинаковое строковое представление. В случае недетерминизма функций преобразования состояния в строку, для применения *инструмента* необходима соответствующая доработка исходного сценария (написание детерминированного преобразования в строковое представление) или сгенерированного сценария (например, с помощью пользовательских функций, восстанавливающих значения соответствующих типов из строкового представления)
- Реализовано корректное воспроизведение значений итерационных переменных только для следующих типов:
  - Примитивные числовые: `bool, int, unsigned int, short, unsigned short, long, unsigned long, long long, unsigned long long, __int64, unsigned __int64, float, double, long double`
  - Символьные: `Char*, char, unsigned char`
  - Строковые: `String*, char*, unsigned char*` (ввиду неоднозначности последних двух типов в языке Си, их применение не рекомендуется)
  - Спецификационные, соответствующие примитивным типам: `Double*, Float*, Integer*, UInteger*, Long*, ULong*, Short*, UShort*` (в используемой схеме генерации не реализовано воспроизведение для этих типов значения `NULL`)

    Для прочих типов генерируются заглушки в виде декларации внешних пользовательских функций, которые должны преобразовывать строковые значения, полученные из трассы, в значения соответствующих типов.