Revit DevCamp, Russia, Moscow, June 25

О базовой структуре Autodesk® Revit® API

Александра Варсанофьева
Product Manager, Revit Platform Technologies

AUTODESK.

Основополагающие блоки Revit API

Существует много общих правил которым приложение должно следовать независимо от того, как оно взаимодействует с Revit. В этом классе мы рассмотрим наиболее важные правила самых базовых структур Revit API.

Хотя мы не будем вдаваться в глубину каждой структуры, мы постараемся указать на то, что часто неправильно понято или не широко известно.

Цели обучения

К концу этого класса, вы поймете и узнаете как использовать:

- Регенерацию
- Транзакции
- Модифицируемость документа
- Доступность элементов
- Объекты с областью видимости
- Внешние команды
- События, обратные вызовы и модули обновления

Регенерация

- Режимы регенерации больше не существуют
  - Автоматический режим является устаревшым
  - ручной режим ничего не делает

- Нужно иметь в виду:
  - Программисты могут вызывать регенерацию когда нужно
  - Измененная модели должны быть регенерированы перед чтением
  - Регенерация умная
  - Вызов регенерации в одиночку не всегда достаточен
  - Иногда транзакция должна быть совершена для того, чтобы все изменения распространились
  - Совершенная транзакция всегда регенерирует модель
  - Но иногда, очень, очень, очень редко, дополнительные регенерации требуются

Режимы транзакций

- Режим транзакции контролирует как транзакция проводится в внешней команде
- Автоматический
  - Не рекомендуется; поддерживается только для совместимости
- Ручной
  - Рекомендуемый режим
  - Клиент полностью контролирует операцией
  - Можно совершыть несколько транзакций из одной команды
  - Транзакции могут иметь пользовательские имена
- Только для чтения
  - Эффективно не позволяет изменения активного документа (никому!)

Внешние команды

- Команду можно вызвать только тогда, когда:
  - Ни одна другая команда не выполняется, нет режима редактирования, и нет активного редактора
  - Нет открытых транзакций и групп в активном документе
- Revit не гарантирует жизнь команды
  - Есть о чем задуматься при использовании глобальных переменных
- Вызовы API должны быть сделаны только при выполнении команды
  - (и только из того же - основного - потока)
- Изменения, сделанные неудачной командой отбрасываются
- Существует API Firewall вокруг каждого вызова
  - Команды, которые нарушают правила будут отброшены

Объекты с областью видимости

- Объекты, которые имеют явную длительность жизни
  - Обычно имеют begin/end или start/finish методы
  - Примеры - все фазы транзакций, режим редактирования лестниц
- Все объекты с областбю видимости должны быть правильно "закрыты"
  - Revit проверяет это на любом возвращении из вызова API
  - Деструкторы закрывают, но их нужно вызывать
- Лучший способ убедиться что объекты закрыты: “окружать их"
  - переменными стека в C++ / CLI
  - **using** блок в C #
  - или **Using** & **End Using** в VB

Пример окруженной транзакции

```
using(Transaction trans = new Transaction(doc, "задача"))
{
    trans.Start ();
    methodModifyingDocument (doc);
    trans.Commit ();
}
```

- Можно вызвать отмену транзакции, если нужно
- **using** блоки могут быть вложенными
- При желании можно обрабатывать исключения

API Firewall

- Инкапсулирует вызовы API
  - Особенно тех, которым разрешено иметь свои собственные транзакции
- Обеспечивает, что все задачи будут выполнены должным образом
  - включая (но не ограничиваясь) всех объектов с областью видимости которые должны быть закрыты
- Ничего не разрешено оставить открытым в активном документе
  - Если это произойдет, то процедура не удается

API Контекст

- Вызовы API могут быть вложенными, например:
  - Revit вызывает команду …
    - Запуск события для одного приложения …
      - Запуск другого события для другого приложения …
        - Вызов динамического обновления первого приложения
- Вызванные методы могут быть из разных приложений
- Revit держит First-In-Last-Out стек выполнения приложений
- Revit знает самое верхнее запущенное приложение
  - и использует его иногда, чтобы разрешить или отказать некоторые операции (например, удаление Dynamic Updaters)
- За любые ошибки обвиняется приложение первого уровня

Transaction API

«Фазы транзакций» доступные в API:

- Transaction
  - Необходима чтоб изменить документ
  - Разрешается только одна активная транзакция в каждом документе
- TransactionGroup
  - Можно группировать транзакции в блок (и аннулировать их если нужно)
  - Можно запустить только вне транзакции.
- SubTransaction
  - Можно отметить набор изменений в транзакции (и аннулировать их если нужно)
  - Можно запустить только внутри активной транзакции.

Все фазы транзакций должны быть закончены!

Существенные отличия

Transactions

- Остаются в документе после совершения. Их потом можно отменить или повторить.
- Конечный пользователь видит их в Отмена / Повтор меню.
- Не могут быть вложенными. Только одна активная транзакция разрешается в любой момент.

Groups and SubTransactions

- Исчезают после совершения (не их содержание!) без следов и нет способа их обратно получить.
- Конечный пользователь никогда их увидет.
- Могут быть вложенными, но должным образом (то есть один полностью внутри другого)

TransactionStatus

- Свойство TransactionStatus определяет этап Transaction фазы
- Оно также возвращается из большинства методов
  - **Uninitialized** - после создания объекта
  - **Started** - после начала
  - **Committed** - после успешного совершения
  - **RolledBack** - после отката или неудачного совершения
  - **Pending** - в процессе совершения немодальной обработки ошибок

Специальное примечание № 1: RolledBack может быть возвращен из совершения

- Зависит от различных решений обработки ошибок (ввод пользователя, обратные вызовы API и т.д.)

Специальное примечание № 2: В ожидании может возвращаться из всех методов

- Означает, что обработка ошибок не завершена
- Только возвращается, если обработка ошибок установлена немодальной
- Вызывающая функция не может вносить изменения, не может создать другие операции, не может кончить TransactionGroup

Транзакции - самые распространенные «фазы»

- Транзакции - атомные изменения в модели
- Они реверсивны (можно отменить или повторить)
- Транзакции отмечают стадии, когда модель стабильная и четко определеная, геометрически и логически
  - Вся геометрия отрегенерирована без ошибок
  - UI отражает последнее состояние модели
- Только одна транзакция может быть активной
- Все транзакции должны быть закончены!!
  - Это унаследованное поведение всех объектов с областью видимости

3 типа методов API

- Методы которым не нужна транзакция
  - Как правило, методы чтения данных из модели
  - Они также могут работать в рамках транзакции,
  - но регенерация может потребоваться для получения корректных данных
- Методы которым нужна активная транзакция
  - В основном все методы, которые изменяют модель
  - Некоторые из них не очевидны - они не похожи на методы, которые изменяют модель, и часто они составляют лишь временные изменения (наприме экспорт) - такие случаи описаны в документации
- Методы которые не должны вызываться внутри транзакции
  - Примеры: Save, SaveAs и т.д.
  - Пользователи должны закончить свою транзакцию до вызова этих методов
  - Автоматический режим Транзакции обрабатывает эту ситуацию автоматически

TransactionGroup – Ассимиляция и Совершения

Оба метода кончяют и совершают группу. Отличие состоит в:

- Совершение не имеет прямого влияния на уже совершенных транзакций в группе. Группа практически исчезает.

- Ассимиляция сливает закрытые транзакции
  - После процесса объединения остается только одна транзакция
  - Процесс слияния "упрощает" список элементов в объединенной транзакции
  - Конечный пользователь будет видеть только одну эту транзакцию в отмена / повтор меню.
  - Транзакция берет на себя название группы
    (Если не было только одной транзакции в группе, в этом случае слияния не происходит и оригинальная транзакция остается.)
  - Когда ассимиляция оканчивается, группа исчезает.

Pending Transactions

“Pending Transaction" – транзакция на которой было вызванно Commit или RollBack, но из-за нерешенных ошибок она не может завершится. Конечный пользователь (или приложение API) должен сначала решить, о том, как ошибки следует решить.

Модальная и Немодальная обработка ошибок

- Модальный режим включен по умолчанию
- В модальном режиме, обработка ошибок должна быть завершена до возвращения из транзакции. Поэтому гарантированно что транзакция не может стать Pending.
- В немодальном режиме, если обработка ошибок должна исправить ошибки, Revit вернет управление обратно вызывающему немедленно, но состояние транзакции останется Pending, пока ошибки все не решенны.
- Состояние транзакции автоматически обновляется когда транзакция, наконец, завершена

Transaction Finalizer

- Для того, чтобы получать уведомления, когда отложенная (немодальная) транзакция, наконец, завершена, владелец объекта транзакции может добавить ITransactionFinalizer
- Очень простой интерфейс:
  - OnCommitted (String transactionName)
  - OnRolledBack (String transactionName)
- Когда вызван, транзакция уже завершена, а статус определен. В теории, интерфейс может перезапустить транзакцию.
- Типичное применение:
  - Закрытие TransactionGroup когда текущая транзакция, наконец, завершена

Модифицируемость документа

- Документ - Modifiable
  - Если он имеет открытую транзакцию и не находится в режиме только для чтения
- Документ - ReadOnly
  - Он постоянно или временно в режиме только для чтения
- Документ - ReadOnlyFile
  - Это свойство относится только к файлу на диске
- Для того, чтобы начать транзакцию:
  - Документ не должен быть ReadOnly
  - и документ не должен быть Modifiable

Доступность транзакций

A. Проверить разрешены ли изменения модели

```
if( document.IsModifiable )
{
    MakeChanges();
}
```

B. Проверить можно ли запустить Transaction или TransactionGroup

```
if( !(document.IsModifiable || document.IsReadOnly) )
    {
    using(Transaction trans = new Transaction(document))
    {
        trans.Start( "doing something" );
        MakeChanges();
        trans.Commit();
    }
}
```

Доступность элементов

- Элемент перестает существовать, когда:
  - Элемент удаляется
  - Создание элемента отменено
  - Удаление элемента переделано
  - Транзакция создавщая элемент была отменена
  - Native объект физически уничтожен (например, при закрытии документа)
- Элемент возвращается к жизни, когда:
  - удаления элемента было отменено
  - создание элемента было переделано
- Revit бросит InvalidObjectException при попытке доступа к элементу который перестал существовать (временно или постоянно).

Пример тестирования доступность элемента

```
1.private void ElementTest(Document document, Element element)
2.{
3.    string name = element.Name; // OK, только для чтения вызовам не нужны
Транзакции
4.    using(Transaction trans = new Transaction (document, "удаление") )
5.    {
6.       trans.Start(); // позволяет изменения в модели
7.       name = element.Name; // OK, только для чтения метод можно вызвать
8.       document.Delete(element); // удаляет Revit элемент, а не managed object
9.       try
10.      {
11.          name = element.Name; // ошибка! – Элемент больше не доступен;
12.      }
13.      catch( InvalidObjectException ex )
14.      {
15.         TaskDialog.Show("Revit", "Попытка доступа к недопуступному элементу.");
16.      }
17.      trans.RollBack(); // Это эффективно отменяет удаление
18.    }
19.    name = element.Name; // OK; элемент обратно в модели
20.}
```

Жизнь managed объектов

- Большинство managed объектов толко обертки для native объектов
  - Несколько исключений: XYZ, UV, ElementId
- Некоторые обертки содержат native объекты, некоторые нет (большинство нет)
- Контролируйте продолжительность жизни с помощью **using** блока
- Сбор мусора заботится о managed объектах
- Revit не удаляет собранные native ресурсы немедленно
  - Он может удалять только когда придет время
  - Можно очистить их с помощью вызова PurgeReleasedAPIObjects

API события - ускоренный курс

- Обработчики вызываются на первый пришел-первым обслужен основе
- Pre-events возникают перед post-events
- Цикл вызовов может быть остановлен отменой события
- Post-events всегда вызываются
- Document-level обработчиков больше не называют первыми
- Обработчикам не разрешается использовать немодальную обработку ошибок
- Существует API Firewall вокруг каждого из вызовов

Обработчик, который нарушает правила Firewall будет отброшен!

Обратные вызовы API (а.к.а. Интерфейсы API)

- Делегаты которые будут вызваны позже
- Имена начинаются с “I"
  - Динамическое обновление - IUpdater
  - Финалайзер Транзакций – ITransactionFinalizer
- Обратный вызов не только managed делегат
  - Каждый обратный вызов имеет native часть, которую вызывают первой и она обходит вызовы на managed объект
  - Продолжительность жизни обратного вызова в большинстве случаев контролируется Revit, но не всегда
    - Если не уверен, владелец объекта должен сохранить срок службы объекта

Dynamic Updaters

- Расширяют регенерацию модели за пределами Revit
- Вызываются в конце совершенной транзакции
- Updaters не вызываются для отмены или повтора транзакций
- Регистрация и отмена регистрации не допускается во время обновления
- Updaters могут быть необязательными (все являются обязательными по умолчанию)
- Revit удаляет противоречивые Updaters
- Дает доступ к «изменившымся» элементам:
  - добавленные, удаленные, обновленные

Запрещенные методы во время динамических обновлений

Запрещенные вызовы и изменения:

- Методы, которые нуждаются в транзакции (определенные экспорт / импорт), или методы, которые не должны быть в транзакции (Save & SaveAs)
- UI методы (сбор, выборы и т.д.)
- TransactionGroups не могут быть использованы, SubTransaction можно использовать если нужно
- Взаимозависимость элементов (изменение, которое вводит или изменяет взаимоотношения между двумя элементами – это ограничение связано с возможным несоответствием worksets)

Другие вещи которые не надо делать во время выполнения обновления

- Register/Unregister Updaters
- Добавлять или удалять триггеры, или изменять приоритет
- Совершать транзакции или вызывать методы, которые начинают транзакции
- Совершать транзакции (в других документах) или вызывать методы, которые используют транзакции

Dynamic Model Update vs. Document Changed Event

DMU

- Важная часть транзакции и регенерации
- Работает только когда транзакция фактически совершена.
- Не выполняется, когда транзакция открывается
- Пользователь может внести изменения
- Вызывается только для желаемых изменений

DCE

- Практически не часть транзакции и происходит после завершения всех регенераций
- Работает когда транзакция либо совершена или открывается
- Пользователь не имеет права вносить изменения в модели
- Вызывается для всех изменений

Оба DMU и DCE выполняются до завершения транзакции!

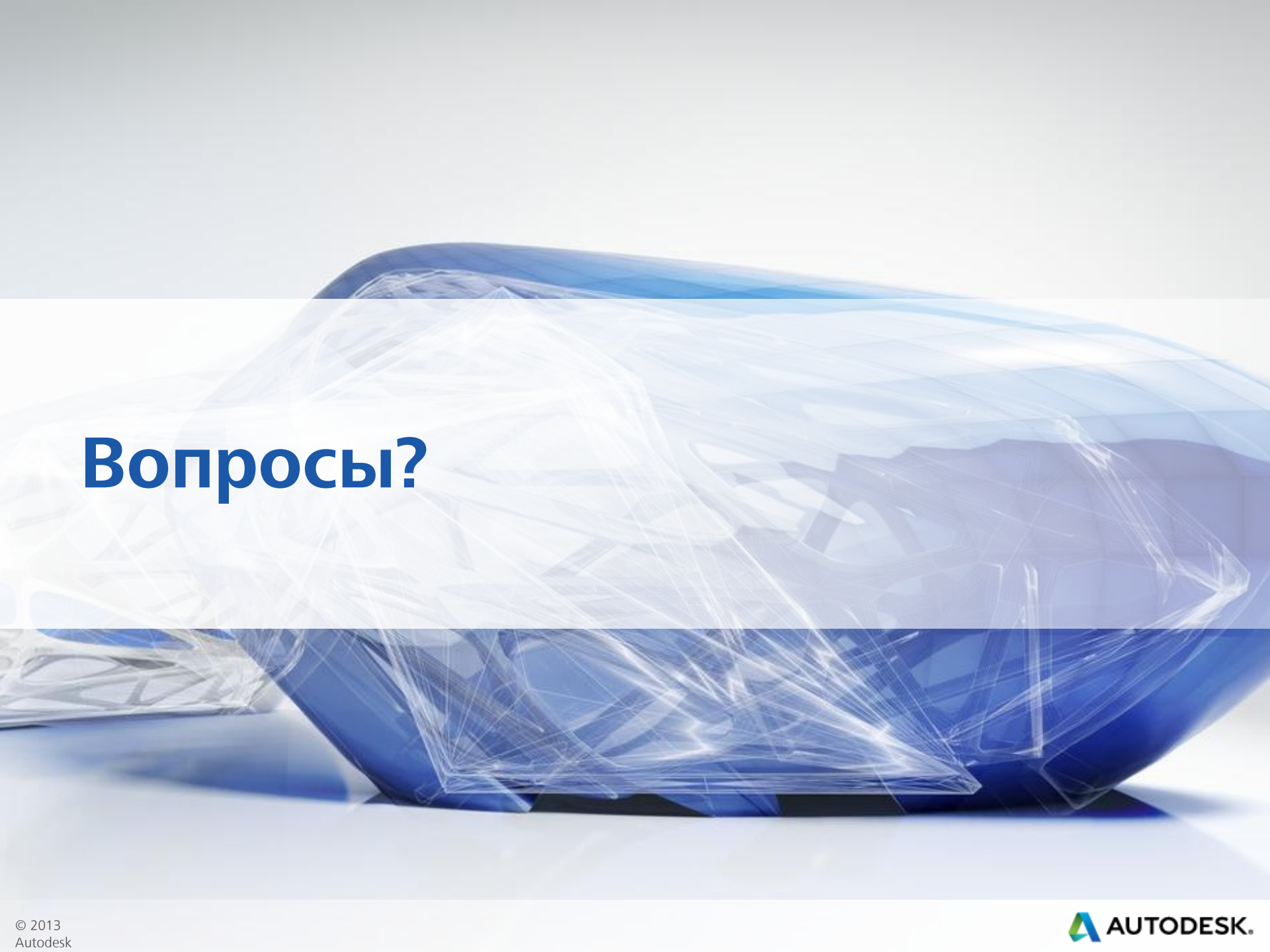
Модели использования DMU и DCE

Используйте DMU:

- Когда вам нужно реагировать на изменения в модели, и сделать другие изменения в той же модели.
- Другими словами - вы добавляете правила о том, что требуется для модели чтобы быть корректной.

Используйте DCE:

- Когда вам нужно реагировать на изменения в модели, и изменять внешние данные или другую модель.
- Другими словами - вы должны держать Revit модель в синхронизации с чем-то снаружи модели. Это может быть даже пользовательский интерфейс.



AUTODESK.