Autodesk AEC DevCamp 2012
Core Autodesk Revit API Explained

Arnošt Löbel
Sr. Principal Engineer

Autodesk

Major Revit API frameworks and workflows

- Regeneration modes
- Transaction modes
- Transaction phases
- Document modifiability
- Element validity
- Object lifespan
- API events
- External commands
- API callbacks
- Dynamic updaters
- API Scopes
- API Firewall

Autodesk

Regeneration modes

- Automatic mode is obsolete

- Only the Manual mode is supported
  - It does not have to be declared (and should not be)

- Things to remember
  - Programmers invoke regeneration as they need it
  - A changed model should be regenerated before it is read from
  - Regeneration is smart – it only updates parts that need it
  - End of a transaction always triggers regeneration
  - A regeneration alone is sometimes not enough
    - But sometimes (very, very, very rarely) an extra regeneration is needed before a transaction is committed, even though commitment also regenerates.

Transaction modes

- Automatic mode
  - For compatibility only (possibly going to be phased out)
  - Many limitations, and virtually no advantage

- Manual mode
  - The preferred way if implementing external commands
  - The client fully in control of using transactions as needed

- Read-Only mode
  - Guarantees the active document does not get modified
  - Nobody can make modifications, not even Revit self

Transaction mode applies to external commands only
  - Event handlers invoked during a command are on their own

Autodesk

External commands

- A command can be invoked only when:
  - No other command is being executed
  - No edit mode or editor is in effect
  - No transaction or groups are open in the active document

- Revit does not guarantee lifetime of a command
  - Something to think about when using global variables

- Calls into the API must be made only during the command (and only from the same – main – thread!)

- Changes made by "failed" commands are discarded

- There is an API Firewall around every call out
  - A command that breaks the rules of the firewall will be discarded

Autodesk

Scoped objects

- Object that have explicit duration
  - Usually have begin/end or start/finish and methods alike
  - Examples – all transaction phases, stair edit mode
  - More similar objects in the future

- All scoped objects must be properly "closed"
  - Revit checks this upon any return from API calls
  - Destructors close implicitly, but they must be invoked

- The best way to guarantee scopes are closed – scope them
  - Stack variables in C++/CLI
  - The `using` block in C#
  - or `Using` & `End Using` in VB

An example of scoping a transaction phase

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

- Exceptions can still be handled if desired
- Rollback could still be called if necessary
- *Using* blocks may be nested as needed

Autodesk

The API Firewall

- It encloses API invocations
  - Particularly those allowed to have their own transactions

- It guards tasks to be finished properly
  - Including (but not limited to) all scoped objects to be closed

- Nothing is allowed to be left open in the active document
  - If that happens, the procedure is rendered a failure

- If the active document is allowed to change:
  - The formerly active document must be free of transactions
  - This is currently allowed during commands only (not events)

API Scope

- API Invocations may be nested, for example:
  - Revit may invoke a command…
    - triggering an event to be handled…
      - triggering another event to be handled…
        - causing a dynamic updater to be executed

- The invoked executors may be from different applications

- Revit keeps a FILO stack of the executors

- Revit knows the application running atop
  - And uses it sometimes to grant or refuse certain operations

- The first-level executor is responsible for mistakes

Transaction API

Three are "transaction phases" available in the API:

a. **Transaction**

- Necessary to modify a document

b. **TransactionGroup**

- Can group a set of transactions onto one block

c. **SubTransaction**

- Can mark a block of changes within a transaction

Main differences

- Transactions stay in the document after commit. They can be undone and redone. The end-user can see them in the Undo/Redo menu.

- Groups and sub-transaction vanish after commit. There is no trace of them and no way of getting back to them. The end-user never sees them.

Transaction phases

- Transactions may not be nested!

- Only groups and sub-transactions may be nested (properly)

- Committing may fail – better to check the status!
  - Transaction being committed may actually return the RolledBack status instead of Committed

- API transaction failures are handled modally by default
  - Modeless handling is possible, but requires using the Transaction Finalizer (use cautiously)

- All transaction phases must be completed
  - This is an inherited behavior of all scoped objects

Transaction objects

❑ Transactions represent atomic changes to the model

❑ They are reversible (can be undone and redone)

❑ They mark stages when the model is stable and well defined, both geometrically and logically (as far as Revit is concerned)

  ✓ Entire geometry is regenerated and without errors

  ✓ The UI reflects the latest state of the model (with some exceptions)

❑ There can only be one transaction active at a time

Three kinds of API methods

1. Methods that **do not need** a transaction
   - Typically, methods just reading data from the model
   - They can also run within a transaction, but regeneration may be needed to obtain correct data

2. Methods that **need** a transaction to be open
   - Basically all methods that modify the model
     - Some of them are not obvious – they do not look like methods that modify the model, and often they make only temporary changes (e.g. IFC export, DWF export, etc.) - such cases are explained in the documentation

3. Methods that **must not** be called inside a transaction
   - Example: Save, SaveAs, Close, etc.
   - Users must finish their transaction prior calling these methods
     - Automatic Transaction mode handles this situation automatically

Autodesk

Group Assimilation vs. Committing

Both methods end and commit a group. The difference is:

A. **Committing** has no direct effect on the transaction (already committed) in the group. The groups practically vanishes like it never existed.

B. When **assimilating**, enclosed transactions are merged together
   - Meaning there will be only one transaction after the merging process is completed.
   - The end user will only see this one transaction in the undo/redo menu.
   - The transaction's name will be the name of the group
     (Unless there was only one transaction in the group, in which case no merging happens and the transaction get to keep its name.)
   - When assimilation is over, the group is forgotten. There is no way to rejuvenate it.

Autodesk

Transaction Status

The TransactionStatus property specifies a stage in which a phase currently is. Beside existing as a property, most methods returns it too (the values may not agree!)

The important status values are:

- Unitialized – after an object is constructed
- Started – after phase started
- Committed – after successful commitment
- RolledBack – after roll-back or unsuccessful commitment
- Pending – during commitment process with a modeless failure handling

Special note #1: RolledBack can be returned from Commit

⇨ It depends on various error-handling decisions (user's input, API callbacks, etc.)

Special note #2: Pending can be returned from all methods

⇨ It means that error handling has not been resolved yet
⇨ It can only be returned if error handling is set to be modeless (false by default)
⇨ The caller cannot make changes, cannot start other transactions, cannot end a transaction group

Autodesk

Transaction Pending

A pending transaction is such a transaction that was requested to be committed or rolled back, but due to unresolved failures it could not be completed immediately. The end user (or an API application) must first make decision about how the failures should be resolved.

Pending transaction can only happen in modeless mode!

Modal vs. Modeless failure handling

- Modal mode is by default and API clients are expected to use it primarily.
- In modal mode, error handling must be completed before returning from finishing a transaction, therefore it is guaranteed a transaction cannot end up pending.
- In modeless mode, if error handling must resolve failures, it will return the control back to the caller immediately, but the status of the transaction will be pending until failures are all resolved.
- The transaction status (in the transaction object if it still exists) is updated automatically when pending is finally completed.

Autodesk

Transaction Finalizer

In order to be notified when a pending (modeless) transaction is finally completed, the owner of the transaction object can pass in an instance of **ITransactionFinalizer** interface when committing or rolling back a transaction.

A very simple Interface:
a) OnCommitted (String transactionName)
b) OnRolledBack (String transactionName)

When the callback is invoked, the transaction is already finished and its status set. In theory, the Interface can restart the transaction again.

Typical use:
⇨ Closing a transaction group after a pending transaction is finally completed
  ▪ The owner of the group can decide whether to commit or roll back the group depending on the outcome of completing the transaction.

Document modifiability

- Document is Modifiable
  - If it has an open transaction and is not in read-only state

- If a document is ReadOnly
  - It is permanently or temporary in a read-only state

- Document is ReadOnlyFile
  - This property only refers to the actual file on disk

To be able to start a transaction:

- Document must not be read-only

- and Document must not be modifiable yet

Autodesk

Testing transaction availability

A. To check if changing the model is currently possible

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

B. To check if a transaction or transaction group can be started

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

Autodesk

Element validity

An element ceases to exist when:

- The element is deleted
- Creation of the element is undone
- Deletion of the element is redone
- The element-creating transaction was rolled back
- The native object is physically destroyed

The element comes back to life when:

- The deletion of the element was undone
- The creation of the element was redone

  (This resurrecting only works for elements!)

Revit will throw an `InvalidObjectException` if an attempt is made to access an element that ceased to exist (temporarily or permanently).

An example of testing Element availability

```
1.  private void ElementTest(Document document, Element element)
2.  {
3.      string name = element.Name;      // OK, read-only calls do not need a Transaction
4.      using(Transaction trans = new Transaction (document, "deleting") )
5.      {
6.          trans.Start();                // allowing modifications to the model
7.          name = element.Name;          // still OK, read-only method can still be called
8.          document.Delete(element);     // deletes Revit element, not the managed object
9.          try
10.         {
11.             name = element.Name;      // error! – the element is not available anymore;
12.         }
13.         catch( InvalidObjectException ex )
14.         {
15.             TaskDialog.Show( "Revit", "Attempting to access an invalid element." );
16.         }
17.         trans.RollBack();             // this effectively undoes the deletion
18.         name = element.Name;          // OK; the element's back in the model
19.     }
20. }
```

Autodesk

Object lifespan

- Most objects are wrappers around native instances

- Some wrappers own native objects, some don't

- Easily control lifespan with the `using` block

- Garbage collector takes care of managed objects

- Revit does not delete collected native resources immediately
  - It can only delete when the right time comes
  - You can purge them explicitly by calling `PurgeReleasedAPIObjects`

Autodesk

API Event crash course

- Handlers are called on first-come-first-served basis

- Pre-events are raised before post-events

- Invocation loop stops by cancelling the event

- Post-events are always raised

- Document-level handlers are no longer called first

- Handlers are not permitted to use modeless failure handling

- Managed events depend on event observers in Revit

- There is an API Firewall around every call out

  - A handler that breaks the rules of the firewall will be discarded!

Autodesk

API Events for the curious

- Managed events depend on native event observers in Revit

  - All managed handlers of a particular event are grouped in one internal observer of that event

- Application vs. the Controlled Application events

  - These subscriptions are virtually identical
  - Subscribes are held in one common repository

- Unsubscribing from events during events is possible

  - One can unsubscribe from any other event
  - Even from the one currently being handled

API callbacks

- Delegates given to Revit to be invoked later

- Callback classes' names start with an '**I**'
  - Dynamic updater - **IUpdater**
  - Transaction Finalizer - **ITransactionFinalizer**

- A Callback is not just a managed delegate
  - Every callback has a native part, which is called first and it bypasses the calls to the managed object
  - Callbacks' "lifespan" is often controlled by Revit, but not always
    - The owner of the object should maintain the object's lifespan

Dynamic updaters

- They technically extend model regeneration outside of Revit

- They are invoked at the end of a committed transaction

- Updaters <u>are not</u> invoked for undone and redone transactions

- Registering and unregistering is not allowed during updates

- Updaters may be optional (all are mandatory by default)

- Revit will resolve "fighting" updaters by removing one

- Updates provide access to "changed" elements:
  - Added and deleted rule each other out. Changed are neither added nor deleted.

Forbidden operations during Dynamic Update

Forbidden calls and changes

1. Methods that need a transaction (certain Export/Import), or methods that must be out of a transaction (Save & SaveAs)
2. UI methods (picking, selections, etc.)
3. Transaction groups cannot be used either (obviously), but sub-transaction can if needed
4. Interdependency of elements (a change that introduces or changes mutual relation between two elements – this restriction is due to possible inconsistency of work-sets.)

Other things not to do while executing an updater

1. Do not register/unregister updaters (self or others)
2. Do not add or remove triggers
3. Do not change priority
4. Do not commit transaction (in the updated document) or call methods that use transactions
   ⇨ This is actually prohibited on the transaction level
5. Do not commit transaction (in other documents) or call methods that use transactions
   ⇨ This is not prohibited, but Revit currently does not handle reentrancy of dynamic updates

Note: Memory footprint and execution time is not limited, but it is wise to keep an updater light - do not invoke methods that take a long time (opening document, exporting, iterating through all elements in the document, etc.)

Autodesk

Dynamic Model Update **vs.** DocumentChanged event

DMU

- ✓ This is an essential part of the transaction and regeneration

- ✓ It only runs when a transaction is actually committed. It is not executed when a transaction is undone / redone

- ✓ User can make changes

- ✓ Invoked for desired changes only, controlled by filters

DCH

- ✓ Is not (practically) part of transaction and happens after complete regenerations

- ✓ It runs when transaction is either committed or rolled back, and also when it is undone or redone

- ✓ User may not make changes to the model

- ✓ Invoked for <u>all</u> changes (even for non-element instances)

Both DMU and DC are executed <u>before</u> a transaction is completed!

Autodesk

DMU and DCH usage patterns

Use DMU:

When you need to react to changes in the model by making other changes to the same model.

In other words – you are adding rules about what it means for a model to be logically correct.

Use DCH:

When you need to react to changes in the model by modifying external data or another model.

In other words – you need to keep a Revit model in sync with something on the outside (of the model). It could even be the UI.

Autodesk

Questions?

Autodesk

Autodesk