LionWeb Notification System API

This document explains the LionWeb notification system API through some use cases.

Use cases

How to get informed about changes

Every partition node (aka root node) of a model, which supports notification API, triggers a notification when there is a change to the model. In the example below, a partition is connected to a receiver. Receiver will be informed about all the changes to the partition via notifications. In this case, receiver (see `NotificationCounter` class definition below) counts the received notifications in its `Receive` method.

Code below demonstrates how to use the API to get informed about changes to a partition.

Example how to get informed about partition changes

```
var partition = new Geometry("geo");
var receiver = new NotificationCounter();

partition.GetNotificationSender()?.ConnectTo(receiver); ①

partition.Documentation = new Documentation("added"); ②
```

① If notifications are not supported, `partition.GetNotificationSender()` returns null.

② This is a change to the model.

Code below gives an example of API usage demonstrating how to get informed about changes to a forest. A forest is a collection of model trees, represented by each tree’s partition.

Example how to get informed about forest changes

```
var forest = new Forest();
var receiver = new NotificationCounter();

forest.GetNotificationSender()?.ConnectTo(receiver); ①

var partition = new Geometry("geo");
forest.AddPartitions([partition]); ②
```

① If notifications are not supported, `partition.GetNotificationSender()` returns null.

② This is a change to the forest. A partition is added to the forest.

NotificationCounter class

```
private class NotificationCounter: INotificationReceiver
{
    public int Count { get; private set; }

    public void Receive(INotificationSender correspondingSender, INotification
notification) => Count++;
}
```

How to collect multiple changes into one change set

Notifications raised by multiple changes to a model can be collected into one change set. A `NotificationCompositor` composes other forest and/or partition notifications into one `CompositeNotification`. Follow the comments below further explanation.

Example how to capture changes

```
var partition = new Geometry("geo");
var compositor = new NotificationCompositor("compositor");

var sender = partition.GetNotificationSender(); ①
sender?.ConnectTo(compositor); ②

compositor.Push(); ③
UpdateDocumentation(partition); ④
var changes = compositor.Pop(); ⑤

foreach (INotification notification in changes.Parts) ⑥
{
    Console.WriteLine(notification.ToString());
}
```

① If notifications are not supported, `partition.GetNotificationSender()` returns null.

② Connects partition notification sender to compositor.

③ `Push()` creates a new `CompositeNotification` to collect incoming notifications.

④ Updates take place.

⑤ `Pop()` returns the `CompositeNotification`.

⑥ Access the notifications (changes).

`UpdateDocumentation()` method changes the partition.

UpdateDocumentation method

```
private void UpdateDocumentation(Geometry partition)
{
    partition.Documentation = new Documentation("documentation"); ①
    partition.Documentation.Text = "hello"; ②
}
```

① First change to the partition.

② Second change to the partition.

How to replicate changes

Partition replicator

`PartitionReplicator` replicates received changes (via notifications) on a local equivalent partition. Follow the comments below for further explanation.

Example set-up

```
var localPartition = new Geometry("geo"); ①
ReplicateChangesOn(localPartition, changes); ②
```

① Changes will be applied to this local partition.

② `ReplicateChangesOn()` replicates the received changes on local partition.

Example how to replicate changes

```
private void ReplicateChangesOn(Geometry localPartition, IEnumerable<INotification>
changes)
{
    var sharedNodeMap = new SharedNodeMap(); ①
    var replicator = PartitionReplicator.Create(localPartition, sharedNodeMap,
"partition replicator"); ②

    var creator = new Creator(); ③
    creator.ConnectTo(replicator); ④

    foreach (var notification in changes)
    {
        creator.ProduceNotification(notification); ⑤
    }
}
```

① The `SharedNodeMap` maps all locally known node ids to node instances. It is shared between all notification pipes in one client or repository.

② Creates a `PartitionReplicator`.

③ `Creator` simulates a notification producer.

④ `replicator` will receive changes from `creator`.

⑤ `creator` sends changes to `replicator`.

Creator class

```
private class Creator() : NotificationPipeBase(null), INotificationProducer
{
    public void ProduceNotification(INotification notification) => Send(notification);
}
```

Forest replicator

`ForestReplicator` replicates notifications for a local forest and all its partitions. It works exactly the same way as for one partition. Instead of `PartitionReplicator.Create()`, we use `ForestReplicator.Create()` helper method.