**Background**

Whenever users work with the Cadastral Editor, a description of each edit is appended to the database. These edits are not instantly visible to other users of the system. Instead, they remain visible only to the user who made the edits. The intention is that the user would subsequently publish their work to the wider user community.

This document outlines the planned implementation, which is based on the idea of editing branches. It takes on board the ideas from a technical paper that was produced by Smallworld in the early 1990s (see <http://cfis.savagexi.com/pages/technical_paper_4>).

**What is a Branch?**

A branch has the same meaning as a branch in a revision control system such as Subversion. However, whereas Subversion branches contain a collection of software code, Backsight branches contain a sequence of edits (where each edit is described by an xml document).

The sequence of edits in a branch is strictly append-only. If you need to revise a previously defined edit (e.g. you discover that an observation was incorrectly entered), you can do so using the Cadastral Editor – however, this revision will be represented by an additional edit that is appended to the sequence.[[1]](#footnote-1)

Branches can be arranged in a hierarchy, making it possible to specify an organized data entry regime. For example, you might define branches for workgroups, with child branches for each individual operator. Alternatively, you might want to create a branch for each survey plan that comes into the system.

Backsight tries to be agnostic in terms of branching – it tries to avoid imposition of a branch hierarchy, because that might interfere with the operational goals of people trying to administer the system. The only rule is that each map layer must correspond to a top-level branch. Users can only define sub-branches that are children of these top-level nodes.

**Editing Sessions**

Whenever you start the Cadastral Editor, the software needs to know two things:

1. The branch where the data should be appended.
2. The spatial zone that identifies where the data exists.

As previously noted, all branches are ultimately related to a map layer. The branch as a whole represents the entire coverage of the layer. The zone is a name for a specific region within the layer, (which makes it possible to deal with large databases in a scalable fashion). In normal practice, both these items will be picked up from the CEDX file that is used to launch the Editor (in a situation where the user does not have a CEDX file, they will be prompted).

Session management involves 3 tables:

* *Sessions* lists all editing sessions that have occurred during the lifetime of the database
* *ActiveSessions* lists editing sessions that are currently active
* *BranchSessions* relates branches to sessions

When you start an editing session, a row will be appended to *Sessions* and *ActiveSessions.* The *BranchSessions* is used to obtain the sessions that should be used to construct the editing model, but is not modified when the session starts. As the user works with the Editor, a description of each edit gets appended to the *Edits* table.

When a user concludes an editing session, the row in the *ActiveSessions* table will be removed, and a row will be appended to *BranchSessions*. Thereafter, other users working on the same branch will be able to see the edits that were made during the session.

To get a feel for what is involved, consider the figure below, which shows a simple timeline involving two completed sessions alongside the data that would be recorded in the *Sessions* and *BranchSessions* tables.

|  |  |  |
| --- | --- | --- |
| Session | Branch | Zone |
| S1 | B1 | Z1 |
| S2 | B1 | Z1 |

|  |  |  |
| --- | --- | --- |
| Branch | Revision | Session |
| B1 | 100 | S1 |
| B1 | 101 | S2 |

S1

(100)

(100)

S2

*Sessions*

*BranchSessions*

Figure 1 – A pair of completed sessions

Each session remembers the branch and zone that were originally used to initiate the session. Each branch lists the sessions that have been included in the branch, alongside a revision number that records the sequence of the session within the branch.

Now suppose that Joe starts session S3, while Fred starts session S4, both referring to branch B1, zone Z1. While both sessions remain active, things look like this:

S4

S3

S1

S2

*Sessions*

*BranchSessions*

|  |  |  |
| --- | --- | --- |
| Session | Branch | Zone |
| S1 | B1 | Z1 |
| S2 | B1 | Z1 |
| S3 | B1 | Z1 |
| S4 | B1 | Z1 |

|  |  |  |
| --- | --- | --- |
| Branch | Revision | Session |
| B1 | 100 | S1 |
| B1 | 101 | S2 |

*ActiveSessions*

|  |  |
| --- | --- |
| Session | Revision |
| S3 | 101 |
| S4 | 101 |

Figure 2 – Concurrent Active Sessions

The revision number stored in the *ActiveSessions* table indicates the last branch revision that was considered when the map model was created for the session. Because sessions are invisible until they have been completed, both active sessions see edits up to revision 101.

Now suppose Fred concludes S4. At that time, the edits performed as part of S4 become an official part of the branch, as shown below:

S1

S2

S3

S4

*Sessions*

*BranchSessions*

|  |  |  |
| --- | --- | --- |
| Session | Branch | Zone |
| S1 | B1 | Z1 |
| S2 | B1 | Z1 |
| S3 | B1 | Z1 |
| S4 | B1 | Z1 |

|  |  |  |
| --- | --- | --- |
| Branch | Revision | Session |
| B1 | 100 | S1 |
| B1 | 101 | S2 |
| B1 | 102 | S4 |

|  |  |
| --- | --- |
| Session | Revision |
| S3 | 101 |

*ActiveSessions*

Figure 3 – First To Finish Appended to Branch

When Joe finally concludes S3, it is clear that it is based on an older revision (the revision number stored in the *ActiveSessions* table does not match the last revision number in *BranchSessions*). To be able to close the session, the work that Joe has performed in S3 must be reconciled with the work that Fred has already appended to the branch.

To do this, the Cadastral Editor needs to reload the map model to the end of the branch (up to revision 102), then “replay” the edits that have been made by session S3 (this is the same sort of logic that occurs whenever old edits are updated). If these edits can be re-calculated without error, the work from S3 can be safely included in the branch, leading to the situation shown in Figure 4.

S2

S1

S3

S4

*BranchSessions*

*Sessions*

|  |  |  |
| --- | --- | --- |
| Session | Branch | Zone |
| S1 | B1 | Z1 |
| S2 | B1 | Z1 |
| S3 | B1 | Z1 |
| S4 | B1 | Z1 |

|  |  |  |
| --- | --- | --- |
| Branch | Revision | Session |
| B1 | 100 | S1 |
| B1 | 101 | S2 |
| B1 | 102 | S4 |
| B1 | 103 | S3 |

Figure 4 – Second To Finish Reconciled

Now suppose that replaying the work from S3 reveals conflicts (i.e. the edits cannot be re-calculated without error). If the conflicts cannot be immediately resolved (perhaps the user is in a rush to go home), two options could be considered:

1. The user could decide to temporarily abandon the session. This would effectively simulate something like a power failure, by leaving the entry in the ActiveSessions table. When the user re-starts the Editor, they would be reminded that their previous session is still active, and given the option to resume the session.
2. The user could decide to move the session to a child branch. This would be a rather radical choice, indicating a complicated conflict that is more difficult to resolve.

The first case is covered by the structure previously illustrated in Figure 3. Creating a brand new branch would require the structure shown below.

Branch B1

S1

S2

S4

Branch B2

S3

*Sessions*

*BranchSessions*

|  |  |  |
| --- | --- | --- |
| Session | Branch | Zone |
| S1 | B1 | Z1 |
| S2 | B1 | Z1 |
| S3 | B2 | Z1 |
| S4 | B1 | Z1 |

|  |  |  |
| --- | --- | --- |
| Branch | Revision | Session |
| B1 | 100 | S1 |
| B1 | 101 | S2 |
| B1 | 102 | S4 |
| B2 | 103 | S3 |

*Branches*

|  |  |  |
| --- | --- | --- |
| Branch | Parent | LastPost |
| B1 |  |  |
| B2 | B1 |  |

Figure 5 – Second To Finish Moved to Child Branch

The required data structure is very similar to what we had in figure 4 (the shaded cells hold the differences). The only new element is the *Branches* table, which is used to keep track of branch structure. The *LastPost* column remains blank at this stage. It is used when a child branch sends editing sessions to its parent, as described in the next section.

**Posting and Receiving**

Editing sessions in one branch can be transferred to a parent branch by *posting* the branch.

Stated another way, this logic means that edits made while a session is active are strictly private. The reason is because the user has the opportunity to discard edits on completion of the session. As such, it is important to ensure that other users do not gain premature access to the edits within the session.

**Map Layers, Themes, and Branches**

Backsight makes it possible to relate map layers in a hierarchy called a *theme*. For example, the *Property* theme (defined as part of the sample Manitoba environment) consists of three layers called *Survey*, *Ownership*, and *Assessment*.

Data on the Survey layer is regarded as base material for the Ownership layer. Most of the spatial data is exactly the same, except for a few small differences that reflect a different perspective on the data. For example, an organization dealing with ownership issues might combine adjacent survey lots if they are owned by the same person. An organization dealing with tax assessment might also have a slightly different view that is based on ownership.

Given that a theme represents a hierarchy of map layers, it is tempting to think of these layers as a hierarchy of branches (Survey 🡪 Ownership 🡪 Assessment).

**Database Structure**

1. I believe that the git version control system (see <http://git-scm.com>) also takes an append-only approach. [↑](#footnote-ref-1)