EBOOK SSP-03

A framework for understanding & using the Esri™ versioning system

EVERYTHING YOU NEED TO KNOW ABOUT VERSIONING IN ARCGIS™

Presented by SSP Innovations, LLC | sspinnovations.com

Everything You Need to Know About Versioning in ArcGIS™

Presented by SSP Innovations, LLC

TABLE OF CONTENTS

Foreword

I love working with people who enjoy learning how things work. Exploring, designing, enhancing, and building technical solutions is the main reason I got into this business and nothing is more satisfying than creating something that changes the way business is done. I imagine that you have some of these same qualities if you have downloaded this eBook on Versioning! This topic is one of the most challenging to understand in the Esri world and we've encountered much misinformation over the years. This is why I struck out many years ago to begin the series called "Versioning for Dummies" which provides the initial foundation of content that you are about to read.

At the core, versioning has truly defined the way we have done business in the Esri world, specifically when it comes to utilities and telecoms. Versioning was originally introduced with ArcGIS version 8.0 (circa 2000) and has continued to evolve ever since then. The team of brains behind the original versioning model were challenged to create a method to allow many users to edit the same GIS database concurrently without having to check out a targeted geospatial area that would subsequently lock all other users out of that area. The resulting solution is both complex and pretty amazing as an example of technological innovation.

As you will come to understand while reading this eBook, versioning was primarily implemented at the relational database management system (RDBMS) level through usage of tables, stored procedures, triggers and views. It is one of the most expansive examples of true database-level intelligence that I have seen in my years in the industry and this is one of the main reasons that many users of the technology find it hard to understand. But at the root of the innovation, versioning was implemented within standard RDBMS platforms including Oracle and Microsoft SQL Server. The key here is that these platforms are non-proprietary (open) technologies and while Esri heavily discourages us from interacting with versioning outside of their prescribed methods, we are able to interrogate the RDBMS to understand how versioning works. And that knowledge will shape our decisions around effectively managing an enterprise versioned geodatabase.

While versioning is heavily founded on RDBMS technology, I would be remiss to not touch on the fact that Esri has crafted a rich library of software, API's and functionality around the RDBMS that provides much of the value to us as users of the technology. ArcCatalog, ArcMap, and the surrounding software have long since been our doorway into the world of versioning. These applications truly provide almost every piece of functionality one might need to create, modify and manage the underlying RDBMS. But, this level of abstraction does create a filter through which we view the underlying database by generally encouraging us to view it as a closed black box.

Our goal in publishing this eBook is to lift the veil covering the RDBMS by getting under the hood of the engine that powers versioning. Our hope is that it will both benefit you in your management of your own versioned geodatabases, and that you will find it to be an enjoyable journey of exploration as we try to provide meaningful examples and techniques for interrogating, tuning, and monitoring your system going forward.

My final note would be to thank a few key individuals who have supported the creation of this content over the years. Specifically Esri's Tom Brown who I often refer to as the father of the geodatabase (my assigned title, not his to be fair), Larry Young who has provided leadership within the Esri utility group for as long as I can remember, and finally our own SSP DBA, Jeff Buturff, who is by far the smartest and most talented geospatial DBA I have ever worked with.

So read on, take it slow, and ensure you understand the basic concepts as the building blocks of versioning before tackling the more challenging patterns. I promise the resulting knowledge will be well worth your investment of time. And as always, we look forward to your feedback!

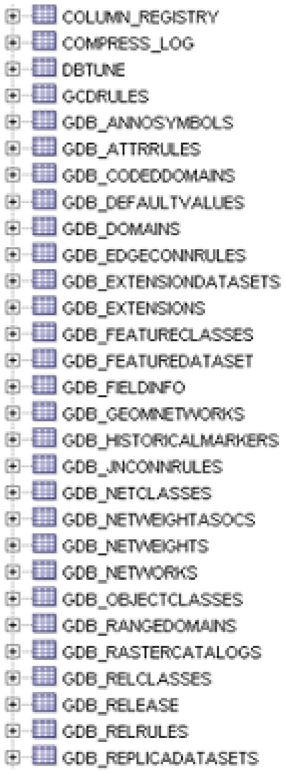
An Intro to Versioning

Versioning is one of the most common topics we are asked about in the consulting field and a good one to unpack here in layman's terms. This section of the book is focused on the basics of versioning as well as providing useful version analysis tips and tools. We then dive into what State IDs are, how they're used, and show why they're the backbone of versioning. Finally, we wrap up with how reconciling and post operations affect your geodatabase and then move into the purpose and usage of multiversion views within your geodatabase.

So maybe you are a GIS manager who wants and/or needs to understand Esri versioning. This is one of the most common topics we are asked about in the consulting field and a good one to unpack here in layman's terms. This eBook will help to understand versioning as well as provide some useful version management tips and tools. There are many articles out there that define what versioning is but we're going to attempt to get behind the scenes to look at *how* versioning works.

In Esri's simple definition, versioning "is the mechanism that enables concurrent multiuser geodatabase editing in ArcSDE geodatabases." This is a good place to start. Versioning is one of the true benefits of enterprise GIS because it allows multiple users to be editing the same geographic area and even the same database record at the same point in time. Each user edits the data within their *own version* in the geodatabase and then ArcSDE provides the tools to merge those edits into the master public version.

Before we get too far into the editing details let's take a look at that *master public version*. This version is what we refer to as SDE.DEFAULT and every enterprise geodatabase has this version of the data. The default version represents the *public* view of the data which will include the most up to date edits that have been *published* for public consumption. When we use the term *public* we mean that all users of the geodatabase can see this data. It should effectively represent the real-life data at any point in time and is often used as the basis for geo-spatial analysis and or interfacing to other systems that consume the public GIS data.



COLUMN_REGISTRY
COMPRESS_LOG
DBTUNE
GCDRULES
GDB_ANNOSYMBOLS
GDB_ATTRRULES
GDB_CODEDDOMAINS
GDB_DEFAULTVALUES
GDB_DOMAINS
GDB_EDGECONNRULES
GDB_EXTENSIONDATASETS
GDB_EXTENSIONS
GDB_FEATURECLASSES
GDB_FEATUREDATASET
GDB_FIELDINFO
GDB_GEOMNETWORKS
GDB_HISTORICALMARKERS
GDB_JNCONNRULES
GDB_NETCLASSES
GDB_NETWEIGHTASOCS
GDB_NETWEIGHTS
GDB_NETWORKS
GDB_OBJECTCLASSES
GDB_RANGEDOMAINS
GDB_RASTERCATALOGS
GDB_RELCLASSES
GDB_RELEASE
GDB_RELRULES
GDB_REPLICADATASETS

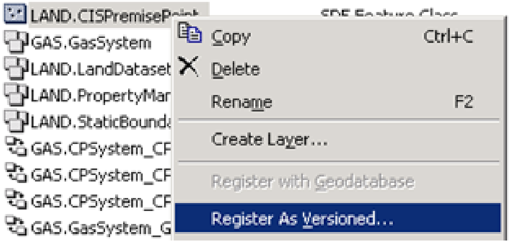
Tables List

The definition of versioning can now be refined to be the mechanism that allows multiple users to concurrently edit and publish updates to the public SDE.DEFAULT version. To utilize Esri versioning you must be using a relational database management system (RDBMS) such as Oracle or Microsoft SQL Server. There are other versioning-enabled databases out there but we'll concentrate on these two popular databases in this eBook.

In addition to the RDBMS software you must be using the Esri ArcSDE (Spatial Database Engine) software. ArcSDE enables the RDBMS to capture, manage, and distribute spatial data in a variety of formats.

When you first create your enterprise geodatabase you must create the SDE repository within the database. In Oracle the repository is made up of the tables and procedures within the SDE *schema*. Within SQL Server the repository is made up of the tables and procedures within the SDE *database*. The repository tables contain information *about* the GIS business data including the business table names, columns, unique sequences, the spatial data (points, lines, and polygons), geometric networks, and of course - you guessed it - *the versions* within the geodatabase.

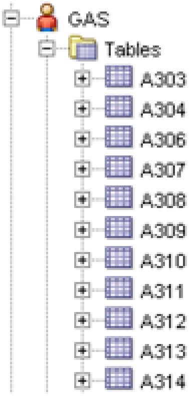
There are also many stored procedures, triggers, and functions which manage the data within the repository. We'll explore some of these tables and procedures in a later section.



Register for Versioned

So now you've got your SDE repository created and you've loaded your GIS business data into the enterprise geodatabase. This alone does not mean you are using versioning - not yet. Next you must explicitly register your GIS business data as versioned. This is easily accomplished within ArcCatalog via a right click menu option at the dataset or individual stand alone table level. You may have even performed this operation in your own geodatabase but what is really happening behind the scenes?

For *each* business table that you register as versioned, ArcSDE creates two *new* tables - the ADD table and the DELETE table. We often refer to these tables as the A&D tables or the *delta* tables because they are used to capture the delta changes to the base business tables.



Delta Tables List

The A&D tables are created directly in the business schema/database right alongside the base business table. They are easy to find because they are named with the letter A or D followed by a unique numeric ID which represents the SDE repository id for the business table. You won't see these tables within ArcCatalog but you can see them within your RDBMS tools - for example *Oracle SQL Developer* or *SQL Server Management Studio*.

To further explore these data structures it can be helpful to know which business tables correspond to which A&D tables. This information can be attained with a simple query against the SDE repository:

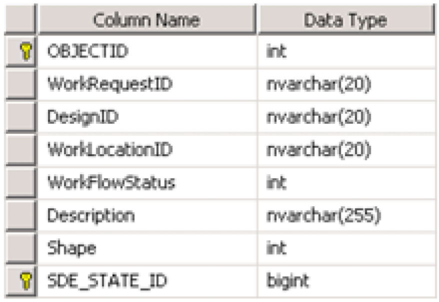
ORACLE:

```
select Registration_ID, Owner, Table_Name from SDE.Table_Registry
order by Owner, Table_Name;
```

SQL SERVER:

```
select Registration_ID, Owner, Table_Name from SDE.SDE_Table_Registry order by Owner, Table_Name;
```

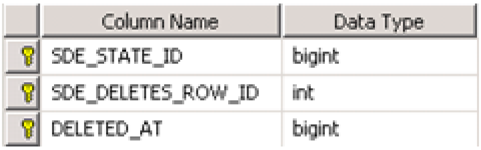
The above queries will return a list of the SDE registration ids for business tables. Combine the registration id with the letter A or D and you now have your versioning tables identified. For example if my PARCEL table has a registration id of 101, the corresponding PARCEL versioning tables are A101 and D101. If you now explore the table schema of the ADD table you



	Column Name	Data Type
🔑	OBJECTID	int
	WorkRequestID	nvarchar(20)
	DesignID	nvarchar(20)
	WorkLocationID	nvarchar(20)
	WorkFlowStatus	int
	Description	nvarchar(255)
	Shape	int
🔑	SDE_STATE_ID	bigint

Add Tables Schema

will see that it contains all of the same columns as the base business table with addition of an SDE_STATE_ID column. As you have probably guessed by now, the ADD table is used to capture new records that are being ADDED to the business table *within a version*. The SDE_STATE_ID is used to determine exactly which version it was added in.



	Column Name	Data Type
🔑	SDE_STATE_ID	bigint
🔑	SDE_DELETES_ROW_ID	int
🔑	DELETED_AT	bigint

Delete Tables Schema

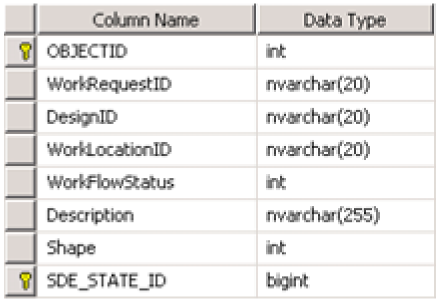
On the flip side, the DELETE table is used to capture all business records that are deleted from the geodatabase within a version. If you explore the table schema of the DELETE table you will see that it only contains three standard columns. Most importantly the SDE_DELETES_ROW_ID contains the unique id (OBJECTID) of the business record that has been deleted and once again the SDE_STATE_ID column is used to determine exactly which version a record was deleted in.

So new records are captured in the ADD table and deleted records are captured in the DELETE table. But what about updates to existing records? Where is the U101 table? An update within the geodatabase can be defined as the *replacement* of an existing record with new values. And in our versioning tables that means an update will contain both an entry in the DELETE table and a matching entry in the ADD table to represent the *replacement* of the existing record. These records will share the same value for the unique id (OBJECTID) and the same SDE_STATE_ID which shows that an *update* took place within the corresponding version.

We've now covered the very basics of how adds, deletes, and updates are managed within the versioned geodatabase. Each version in the geodatabase is *not a copy* of the base business tables but is instead simply a *mechanism for tracking the delta changes* to the base business tables by using the A&D tables as described above. In the next article in this series we will examine how the SDE_STATE_ID values are assigned by ArcSDE and how they can be matched up to specific versions within the geodatabase. We will also provide some useful tools for viewing the A&D edits within the RDBMS.

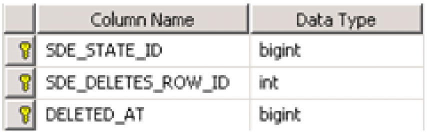
The State ID

Now let's take a closer look at the SDE_STATE_ID (state id) column that is used within the A&D tables. We've previously noted that the state id is a numeric value tracked on both the Add table and the Delete table for each versioned business table:



	Column Name	Data Type
🔑	OBJECTID	int
	WorkRequestID	nvarchar(20)
	DesignID	nvarchar(20)
	WorkLocationID	nvarchar(20)
	WorkFlowStatus	int
	Description	nvarchar(255)
	Shape	int
🔑	SDE_STATE_ID	bigint

Example Add Table



	Column Name	Data Type
🔑	SDE_STATE_ID	bigint
🔑	SDE_DELETES_ROW_ID	int
🔑	DELETED_AT	bigint

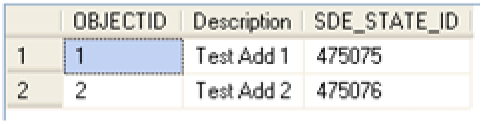
Example Delete Table

Each time an edit is performed within an Esri versioned geodatabase, a state id is assigned to that edit and the edited record is added to the Add and/or Delete table. To provide an example, in ArcMap I have created a version called SDE.TestVersioning that is a child of SDE. Default. Within that version I start an edit session and add two brand new records to the table shown above. The database will assign the next available state ids to my new records and add them to the appropriate Add table. In my test geodatabase the next available state id is 475075. I can query the Add table to check out the records along with their state ids. My example is using SQL Server and I've added description attributes to the records to help identify them:

SQL SERVER:

```
select OBJECTID, Description, SDE_STATE_ID from [DATABASE].[BUSINESS
DATA OWNER].A[REGID];
```

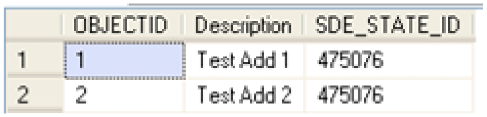
RETURNS:



	OBJECTID	Description	SDE_STATE_ID
1	1	Test Add 1	475075
2	2	Test Add 2	475076

As you can see from the above query, my records were added to the Add table and automatically given sequential state ids - 475075 and 475076. It should be noted that I have not yet saved my edit session in ArcMap. This allows for the undo/redo functionality within the ArcMap edit session because each edit is *uniquely* identified by a state id. If we now SAVE the edit session and run the same query we will see that *ALL* of the edits in the edit session are assigned the *same* state id corresponding to the highest state id used - 475076 in this case:

AFTER SAVE OF EDIT SESSION:



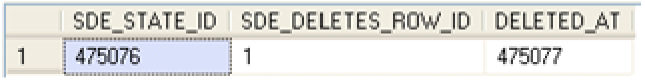
	OBJECTID	Description	SDE_STATE_ID
1	1	Test Add 1	475076
2	2	Test Add 2	475076

After you have saved your edit session the undo/redo for those individual edits *is no longer available* within ArcMap because all of the edits now have the same state id. Next, we will continue this example by *deleting* the first record we created above - "Test Add 1". If we re-query the Add table, nothing changes. However, if we query the Delete table we will now see a new record:

SQL SERVER:

```
select * from [DATABASE].[BUSINESS DATA OWNER].D[REGID]
```

RETURNS:



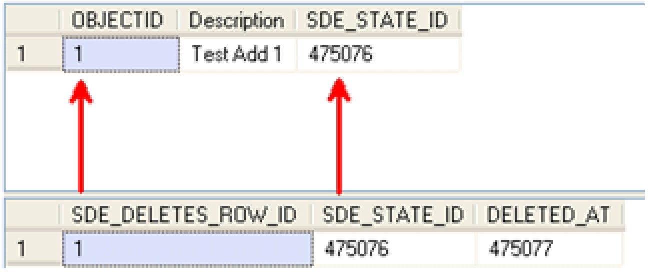
	SDE_STATE_ID	SDE_DELETES_ROW_ID	DELETED_AT
1	475076	1	475077

Let's now compare the Delete table entry to the Add table entry. Three things should be noted here:

1. *The SDE_DELETES_ROW_ID in the DELETE table is the OBJECTID of the record (unique Esri identifier.)*

2. *The SDE_STATE_ID in the DELETE table corresponds to the state the record was **added**. In this case 475076 is the SDE_STATE_ID of the **same** record from our ADD table.*

3. *The DELETED_AT column contains the **new state id** that was assigned to this edit. This is a new unique state id that has not been used before, 475077, which is the next available state id in the system.*

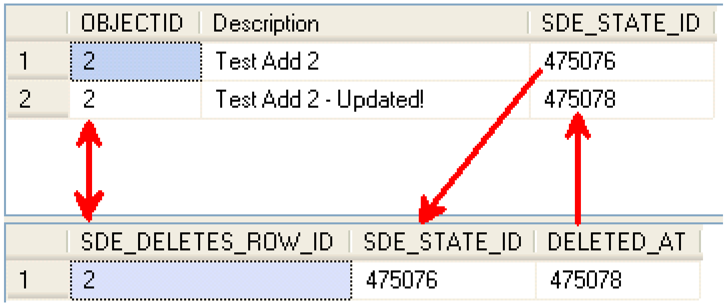


	OBJECTID	Description	SDE_STATE_ID
1	1	Test Add 1	475076

	SDE_DELETES_ROW_ID	SDE_STATE_ID	DELETED_AT
1	1	475076	475077

These matched ids allow the system to show that the record was added at state 475076 and then was subsequently deleted at state 475077. Because these edits are all being tracked by individual states, the system can manage the visibility of each edit. In other words, if I viewed the data at state 475076 I would see the record on the map but if I viewed the data at state 475077 the record would be gone (deleted). Keep that in mind as we continue.

Finally, let's review an update to the second record we created above. In my ArcMap session I have updated the Description value to be "Test Add 2 - Updated!". If we review the corresponding records in the Add and Delete tables we should see a *delete* record corresponding to the original state id plus an *add* record with the new values (see the beginning of ***An Intro to Versioning*** for more information):



	OBJECTID	Description	SDE_STATE_ID
1	2	Test Add 2	475076
2	2	Test Add 2 - Updated!	475078

	SDE_DELETES_ROW_ID	SDE_STATE_ID	DELETED_AT
1	2	475076	475078

The first thing to note above is that the OBJECTID / SDE_DELETES_ROW_ID is the first matched value we can use to tie these records together. Next we will examine the state ids. In the ADD table (top) we now see TWO records with the same OBJECTID but with different state ids. The first state id, 475076, shows the record as it was originally added to the geodatabase with a description of "Test Add 2". We can then match that state id to the same SDE_STATE_ID value in the DELETE table which shows that this record has been deleted (the SDE_STATE_ID in the DELETE table corresponds to the state the record was added). And the DELETED_AT value in the DELETE table, 475078, gives us the state id where the record was deleted. We can then match that value of 475078 back to the ADD table SDE_STATE_ID to find the corresponding *new* record with the *updated* description of "Test Add 2 - Updated!". Therefore if we view the data at state 475078 the geodatabase would show the record on the map with the updated description of "Test Add 2 - Updated!" because it applies the edits by matching up the state ids.

In the above examples we have seen how records are added, deleted, and updated within the A&D tables in the geodatabase. It might be useful to re-read this section a few times to make sure you have a solid grasp on how this works because this is the basis of all versioning.

In these examples we noted that the geodatabase assigned the next available state id to the edits being performed in ArcMap. Next we'll take a look at where those state ids are managed. There is a *states* table in your geodatabase that contains all of the current states which correspond to all of the edits that have occurred in your geodatabase. You can view the records in your states table using the following queries:

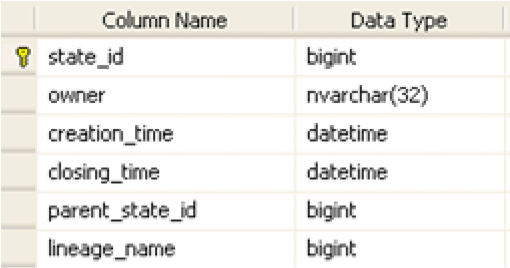
ORACLE:

```
select * from SDE.states order by STATE_ID;
```

SQL SERVER:

```
select * from SDE.SDE.SDE_states order by STATE_ID;
```

The following picture shows the table structure of the states table:

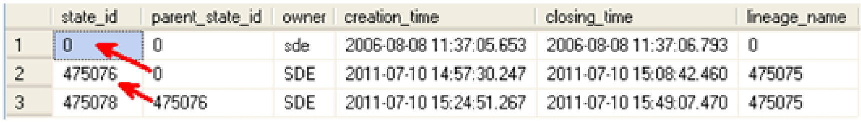


	Column Name	Data Type
🔑	state_id	bigint
	owner	nvarchar(32)
	creation_time	datetime
	closing_time	datetime
	parent_state_id	bigint
	lineage_name	bigint

Here are the uses of each of these columns:

- *The state_id column captures the same state id that we've been reviewing in our above editing examples. It is a unique id that is created for each edit in each edit session.*
- *The owner is the user who performed the edit.*
- *The creation_time is the time the state was created which will match when the edit session was* ***started****.*
- *The closing_time is the time when the state was closed which will match when the edit session was* ***saved****.*
- *The parent_state_id is the preceding state in the state lineage - this is discussed in more detail below.*
- *The lineage_name is a foreign key to another table which defines the full state lineage for any given state.*

Each versioned geodatabase begins with an initial state with a state id of 0. This state always exists in the geodatabase and is the ultimate parent of all of the other state records in the states table. As edits are performed and new states are created the parent_state_id value is always populated with the next logical parent state which will eventually tie back to state 0. If we review the states corresponding to our example edits above we find the following records:



	state_id	parent_state_id	owner	creation_time	closing_time	lineage_name
1	0	0	sde	2006-08-08 11:37:05.653	2006-08-08 11:37:06.793	0
2	475076	0	SDE	2011-07-10 14:57:30.247	2011-07-10 15:08:42.460	475075
3	475078	475076	SDE	2011-07-10 15:24:51.267	2011-07-10 15:49:07.470	475075

Our most recent saved edit session had a state id of 475078 and this is the bottom record above. The parent_state_id of this state is 475076 and we can trace that record back to the logical parent state of 475076 which corresponds to the *first time we saved* in ArcMap. The parent_state_id of this state is 0 and we can trace that record back to the logical parent state of state 0 which is the base state of the geodatabase. To reiterate the point, ALL state records correspond to edit sessions and ALL states can be traced back to state 0. And follow this closely - state 0 corresponds to the records that exist in the *base business tables* (i.e. not the Add and Delete tables but the original business table that existed BEFORE the table was registered as versioned). The path that any state takes back to state 0 is called the **state lineage**. This is just like tracing your family lineage back to your great, great grandfather except that we are tracing parent *states* as opposed to parent *people*.

There can be many different state lineages in a versioned geodatabase. When we query data from the base business table (which correspond to state 0) plus all of the edits tied to a specific *state lineage* we get... you guessed it, drum roll please... an SDE VERSION! We use names like SDE.TestVersioning to describe a version but all these names do for us is give us an *easy to remember text-based name* for a state id/lineage. This can be seen by querying the SDE versions table:

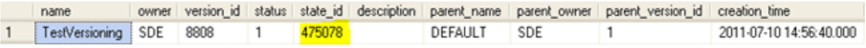
ORACLE:

```
select * from SDE.versions where name='TestVersioning';
```

SQL SERVER:

```
select * from SDE.SDE.SDE_versions where name='TestVersioning';
```

RETURNS:



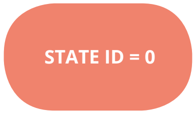
	name	owner	version_id	status	state_id	description	parent_name	parent_owner	parent_version_id	creation_time
1	TestVersioning	SDE	8808	1	475078		DEFAULT	SDE	1	2011-07-10 14:56:40.000

As you can see from the results above, the state_id for the SDE.TestVersioning version is 475078. Hopefully that number rings a bell because it matches the state id from the final save we performed in our example edit session above. When we traverse the state lineage back to state 0 we get all of the edits we performed in our version. And this is exactly what ArcMap is doing when you display data loaded from a specific transactional version.

The topic of the Esri state id is a bit complex but I hope that these examples are helping it to make some sense. The state id is at the core of what versioning is and while we never see these values in ArcMap, they are what makes all the magic happen behind the scenes. In the next bit, we will delve into what happens to the edits in the A&D tables when we post data as well as how the all-important SDE compress operation cleans up the geodatabase.

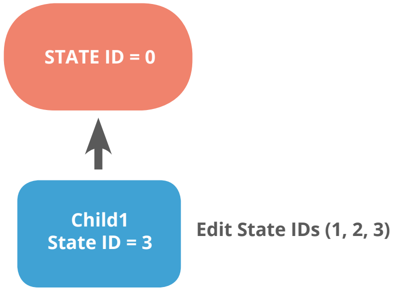
Reconcile, Post, and Compress… Oh My!

Now we want to take a look at what happens to the states in a database when we reconcile, post, and ultimately compress our geodatabase. If you've stuck with me this far you probably have a pretty good idea that there are many, many different state lineages in a geodatabase that has a lot of versioned edits. For the sake of digging into this topic, I want to use the example of a very simple geodatabase. In this geodatabase there is currently only a single version, SDE.Default:

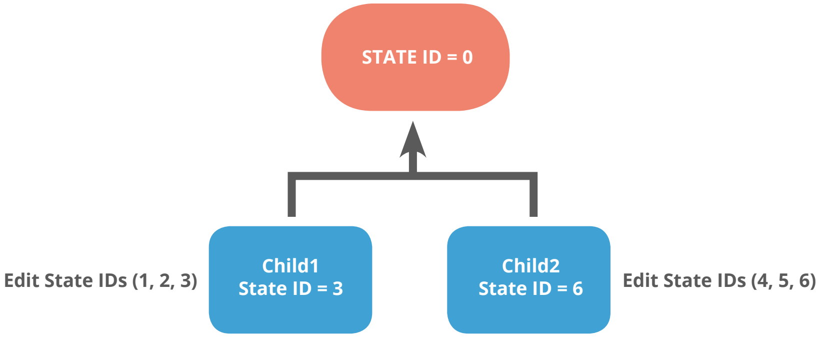


Because there are no edits in the database the current state id of Default is pointing to 0 which indicates that it is rendering data directly from the base business tables and that the A&D tables are currently empty.

Now I am going to create a second version as a child of Default and I am going to add three feature records to this version. As we saw in the last section, each edit is assigned its own state id but when I save the edit session, all of those edits are assigned the same state id corresponding to the max state id used in the version. For simplicity we will assume this version used state ids 1, 2, and 3. So when I save this version, my geodatabase now looks like this:

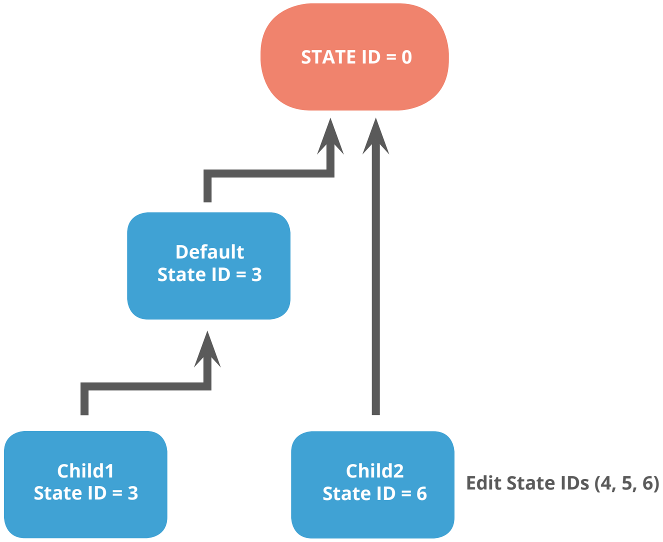


Ok, now let's take it another step further and create a third version with three more new features. These edits will get state ids 4, 5, and 6 and when I save the version the state id points to 6:



If I've already lost you, please go back and read the first two parts of this chapter where we explain what a version is and how the state ids are assigned (no offense intended.) Otherwise let's keep going. Each of the two child versions above will have its own state lineage. Child1 will have a state lineage of 3 → 0 and Child2 will have a state lineage of 6 → 0. Please remember that these are two different lineages that meet back at the common state id of 0.

So now we want go ahead and reconcile Child1 to Default. To do this we open Child1 in ArcMap and click the reconcile button on the versioning toolbar. A reconcile operation syncs any edits that have been posted to Default down into Child1. In our above example, there are no additional edits available in Default at this time. Our next step is to post the edits in Child1 up to Default by clicking the post button on the versioning toolbar. When we do this Default is essentially synchronized with Child1. What this means behind the scenes is that both Default and Child1 point to the same state id of 3:



The other major change we see here is that Default is no longer representative of state 0. It is a common misconception that Default *always* represents state 0. However, in the above example we see that we have now separated out state 0 because Child2 still needs to reference the state 0 data without the posted edits that exist within Child1 *and* Default (i.e., state id 3). All three versions still eventually trace back to state 0, but our state tree has changed.

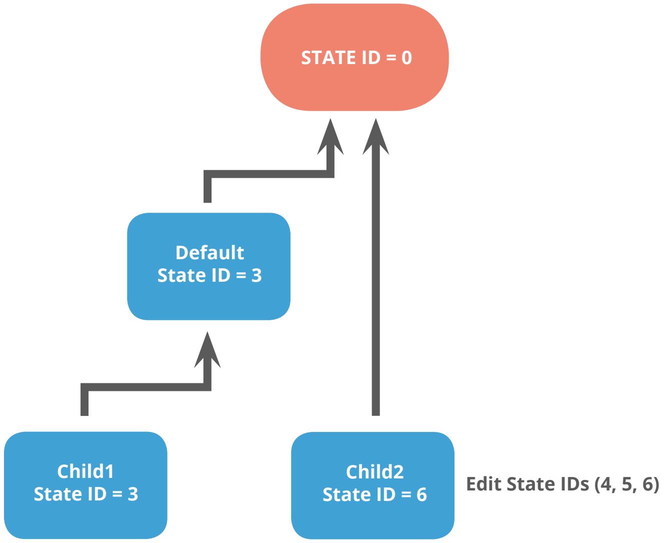
If you are using a versioned geodatabase you are certain to have heard about an SDE compress operation. Most of us know that a compress operation maintains the health of our database by *compressing the state tree*. But do you really know what that means? Many folks believe that the compress moves all posted edits from the A&D tables into the base business tables. In our example above we've posted edits from Child1 into Default. If we ran an SDE compress against the geodatabase in its current state, what would happen? The answer is *absolutely nothing*. In this case, no edits can be moved (compressed) into the base tables because the geodatabase *must* maintain the current state tree to enable Child2 to render state 0 plus the three edits (state ids 4, 5, and 6.) We can run the compress operation over and over again, but it won't do a darned thing.

To put this further into perspective, we could create another 50 versions in the geodatabase, create edits, and then reconcile and post those edits into Default. Each time we post, the state id of Default will be updated to reference a lineage including the new edits. And we can run the compress after each post but it will continue *to do nothing* as long as Child2 is referencing state 0 directly.

Each time we create an edit, records are added to the A&D tables. And as records pile up in the A&D tables, the database has to work harder and harder to render a specific state lineage, because it always has to start with state 0 and then apply all of the edits that exist within the specific lineage. The more edits you have... the slower the database responds. Eventually your system will grind to a halt with performance that is unbearable.

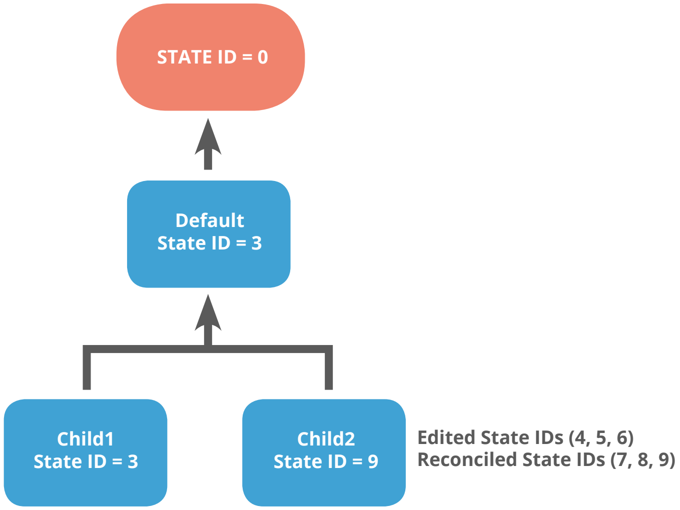
We were called into a small utility a couple of years ago that only had two editors. But they had a case similar to the above scenario with a single version that was directly referencing state 0. After many months, they ended up with 45,000 edits in one of their ADD tables and the system performance was just awful. They couldn't understand how this was possible… because they compressed every day. It's an important lesson that hopefully makes more sense now. The compress is only effective if your state tree has been *fully reconciled*.

So to continue our exploration, let's forget the extra 50 versions and get back to our original example. We have two child versions with edits and have posted Child1 to Default. Here is the same picture as above for reference:



Next, we'll open Child2 and perform a reconcile against Default. This time there are edits that exist within Default (state id = 3) that do not exist within Child2 (state id = 6.) When we reconcile, the software essentially performs a mini edit session and moves the edits 1, 2, and 3 down into Child2.

However, these are considered new edits in Child2 and they are therefore given new state ids of 7, 8, and 9. When we then save Child2, the state id is now 9, corresponding to the final edit that was reconciled:



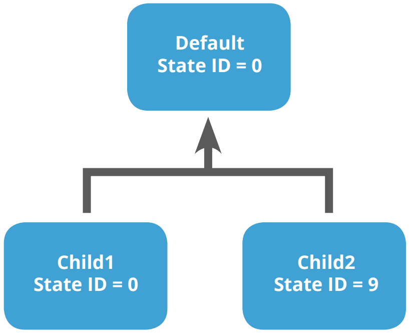
Our state tree has once again been modified because we reconciled down the edits from Default into Child2. Child2 will now have a state lineage of 9 → 3 → 0 whereas both Child1 and Default have a state lineage of 3 → 0. Keep in mind that there are still edits within Child2 that do not exist within Child1 or Default (edits 4, 5, and 6). BUT this reconcile operation has brought the two child versions back into a common lineage. They both traverse *through state 3* to get to state 0.

In the geodatabase, all of the edits still exist in the A&D tables and no modifications have been made to the base business tables... yet.

Now we will run the compress operation once again. This time our state tree has been *fully reconciled* and our child versions share a common lineage from state id 3 to state id 0. This is important because it indicates that state id 3 is no longer needed because it contains edits that are *common to ALL versions in the geodatabase*.

When we run the compress, the software recognizes that state id 3 is obsolete. The compress then moves all of the edits associated with state id 3 (edits 1, 2, and 3) from the A&D tables into the base business tables. They become part of state 0. The compress then deletes the state with id 3 from the states table because it is no longer a referenced state.

Our state tree has now been simplified and looks like this:



The term "compress" is pretty accurate, because it has shortened the state tree by moving common edits into the base business tables and deleting the unreferenced states. Note that Default is now once again referencing state 0. Version Child2 still references state 9 because it has outstanding edits that have not been posted but the state lineage has been shorted back to 9 → 0.

As the number of edits decreases in the A&D tables, database performance goes back up and we can now go to sleep happy because we know our compress has been effective.

The concept remains exactly the same when you have 50 or more versions. As soon as you can *fully reconcile all posted edits down into all child versions*, you will cause intermediate states to become obsolete. Each state corresponds to edits in the A&D tables and when you compress, all of the edits corresponding to obsolete states get moved into the base business tables and all of the obsolete states get deleted from the geodatabase. I may sound like I am saying the same thing over and over again... I am. But it's a very important concept which can make all the difference in the health of your geodatabase.

And now, I want to make a final comment about versioning conflicts. A conflict occurs when we have edited the same record and/or more specifically the same attribute on the same record in two different versions. The conflict appears after we post the first version and attempt to reconcile the second version. The reconcile operation detects the conflict and halts the reconcile operation.

I don't want to get into how to resolve conflicts... there is plenty of existing documentation out there on that topic. The more important point is that it ***halts the reconcile operation*** on that version. And as we just reviewed, when a reconcile is not performed on even a single version it can cause all sorts of performance problems.

The result of this case is exactly the same as our example of not reconciling a version for a long period of time. This is why it is vitally important that you resolve those conflicts on a regular basis, which will allow the reconcile operation to complete and your compresses to be effective.

If you work in a large organization that typically has more than a few versions in your geodatabase at any given time (cough, cough Telvent Designer™ users) it is imperative that you have a process to detect and report your conflicts regularly (daily is great). Ideally this is an automated process so you can use your manpower efficiently. You should attack the oldest outstanding conflicts first because they will represent the state ids that are holding back the compress from doing its job. It's not important to have a geodatabase without any conflicts (in fact it's almost impossible) but it is important to resolve the older conflicts on a regular schedule to keep the system motoring (dare I say speeding) along. If you need direction or assistance with any of this, feel free to give us a shout. We deal with it every day. Off my conflict soapbox...

In summary, we've covered how the reconcile, post, and compress operations affect the states in your geodatabase and why your compress may or may not be working effectively. The state tree in your geodatabase is constantly changing but with a little art and a bit more science you can master the state tree and keep it firmly rooted in your organization. I know that was way too cheesy but it's late at night and I'm up writing about state ids! I often wonder how much separation there is between a versioning geek and a geeky member of The Lonely Island.

Esri Multiversion Views

Now we want to hit on another very useful topic that will enable you to consume and manage your geodatabase directly through SQL access (outside of the Esri software).

As you recall, a version name is simply an easy to remember text-based name for a state id/lineage. The state lineage is used to apply certain edits from our Add & Delete (A&D) tables on top of the corresponding business base tables to create the version of the data that we see in ArcMap or ArcCatalog. It is a common misconception that once we post our edits to SDE.Default that they are moved into the base tables but as we demonstrated previously, this is absolutely not true. If I just lost you, go back and do a quick refresher because we aren't going to dive into it again here.

This scenario often presents a challenge to a business and even a typical IT department because access to the data must go through the Esri tools which can handle the rendering of the versioned data. Users cannot use the same basic SQL skills that they may use in their other standard relational database management systems (RDBMS) such as customer information, accounting, or asset management. When users are not educated about versioning and try to use SQL directly against a geodatabase they can get misleading results from their queries and can even cause data loss/corruption if they execute updates against the geodatabase. There are two typical scenarios we run into on a regular basis:

SCENARIO 1 - QUERYING DATA OUT OF THE GEODATABASE

A user decides to use a SQL query to extract data out of a geodatabase because either it's easier than going through the Esri tools OR they want to do some complicated joins between tables that are not readily available in the Esri tools. So they write a query against the base tables in the geodatabase and extract their data. Sometimes this works but many times the users get misleading results back.

Recently we had a customer call and tell us that either their post operation or their compress operation was not working. They had posted a record to SDE.Default in ArcMap but when they queried the database they were not seeing the record. Why could they see the record in ArcMap when it did not exist in the database table?

The answer lies in the previous section on reconciling and posting. As you post data to SDE.Default, that version is simply updated to point to a new state lineage. The edits are not moved into the base table until the edits have been reconciled down into all other versions and then a compress operation is performed. Therefore querying the base tables in a versioned geodatabase is *almost never an acceptable way to extract data* - even data from SDE.Default.

SCENARIO 2 - UPDATE DATA IN THE GEODATABASE

The second scenario we run into on a regular basis is when a user wants to *update* data within the geodatabase. They write a simple SQL update statement like they would in any other database - “update SomeTable set Column1=X where Column2=Y”. Simple right? Not in geodatabase. When an update is executed against the base table in a geodatabase you are executing against the state 0 data. If there are outstanding edits against that same record via another state, your edit can be lost completely. *Definitely not a sturdy way to make updates*.

So are we left with no way to make SQL updates to our geodatabase? Is all of our knowledge of RDBMS SQL useless within an SDE environment? It’s not quite that bad but it is a bit more complicated than just using SQL.

Esri Multiversion Views were created as the answer to these problems. As you might imagine, behind the scenes in an SDE versioned geodatabase there are many database views, stored procedures, and triggers that manage and expose the versioned data to the Esri applications. A Multiversion View exposes this same database process of managing versioned data directly via SQL. This provides us an easy way to query data using SQL from a specific version in the geodatabase. And when used correctly can even allow us to insert, update, and delete from the versioned data as well.

Esri Multiversion Views are not available in our geodatabase by default. We have to create them using the SDE command line. The SDE command line tools can be installed on any computer - i.e. you can install them on your local laptop or workstation, they don’t have to be installed on the same server where the SDE geodatabase is installed (though they usually are). The installation is available on your Esri ArcGIS Server - SDE disc. Be aware that there are different installations based on the RDMBS you are using.

As usual we will focus on SQL Server and Oracle since these are the most common databases used within utilities. Simply install the correct software based on your RDMBS (yes you can install both SQL Server and Oracle to different directories) and *cancel* the post installation setup if you aren’t actually creating an SDE database or service on your local computer.

Now you are ready to create your Multiversion View. Choose one or more feature or object classes in your geodatabase and use the following SDETABLE syntax to create the views from the command line:

SQL SERVER:

```
sdetable -o create_mv_view -T [ViewName] -t [OWNER.TABLENAME] -i
[SDEService] -s
[SDEServer] -D [SQLDatabaseName] -u [DataOwner] -p [DataOwnerPassword]
```

ORACLE:

```
sdetable -o create_mv_view -T [ViewName] -t [OWNER.TABLENAME] -i
[SDEService] -s
[SDEServer] -u [DataOwner] -p [DataOwnerPassword]
```

A few notes on the parameters which you may already know... but just in case:

- *The SQLDatabaseName parameter is only required for SQL server and should be the name of the SQL Server database where your business data exists (different than the SDE database).*
- *The SDEService parameter can be the service name - esri_sde, the service port - 5151, or a direct connect string such as sde:sqlserver:instanceName or sde:oracle10g:/;LOCAL=TNSName*
- *The DataOwner and DataOwnerPassword should correspond to the owner of the feature/object class that you are creating the view for*
- *The SDEService, SDEServer, and SQLDatabaseName will be the same parameters you use to connect to the geodatabase via ArcCatalog and ArcMap.*
- *The ViewName parameter specifies the name of the view in the database. We often recommend that our clients name their Multiversion Views with the same name as the underlying feature/object class but with a "_MV" attached to the end of the name. This will make your views easy to organize and find in the database.*

Once you have created the MultiVersion View via the SDE command line, log into the geodatabase with ArcCatalog using the data owner and you should see your Multiversion View at the root of the geodatabase. Right-click it and select Privileges. Give permissions on the view to any users or roles that need to access it.

Your Multiversion View is now ready to use and can be accessed directly within your SQL tools including Oracle SQL Plus, Oracle SQL Developer, TOAD, SQL Server Management Studio, a custom application, etc. If you want to view the schema of the Multiversion Views using one of the above tools, simply browse to the *Views* in the correct database/schema within the database. The Multiversion Views will appear just like any other standard views in the database. Don't be fooled though, they are anything *but* standard views.

Now instead of querying the base business tables, you can query your Multiversion Views and get the *versioned* results. You can validate this by running a simple test. For example let's assume we have a versioned feature class called Utility.Pole with 100 existing pole features and we have just created a new Multiversion View for this feature class. In ArcMap create a new version and place 5 new poles. From our previous articles we know these are entered into the "add" table behind the scenes. Now post the version to SDE.Default. The edits still exist in the "add" table and are not available in the base business table. We can now prove this out using SQL. First run the following query against the base business table:

```
select count(*) from Utility.Pole;
```

This query will return 100 records matching the original count in the feature class. Now run the following query against the Multiversion View:

```
select count(*) from Utility.Pole_MV;
```

This query will now return 105 records because it includes our versioned edits that we posted above. The records exist in the "add" table but the Multiversion View has joined that data to the base business table and given us the compiled result.

Note that we did not explicitly specify a version to run our query against. When we just query against a Multiversion View without specifying a version, it will return records from SDE.Default by default (no pun intended). This can be very useful for running queries against our as-built, energized data.

But we can also run queries against specific versions within our geodatabase. To do this we must first call a stored procedure in the database to tell the Multiversion View *where* to get the data. The syntax varies slightly by RDBMS:

ORACLE:

```
call sde.version_util.set_current_version ('SDE.Working');
```

SQL SERVER:

```
exec sde.sde.set_current_version 'SDE.Working';
```

The parameter above ‘SDE.Working’ can be changed to any other version in the system that your user has access to. Beware the version name is case sensitive in Oracle. The same version rules apply regarding public, private, and protected versions. If your user can’t access the version in ArcCatalog/ArcMap, it will not be able to access it via SQL.

Using the above proc you can now run Multiversion View SQL queries against any version in your geodatabase. Just remember to reference the _MV (or whatever you name it) view instead of the base business table. This can be super helpful when doing analysis on a batch update in a version, data validation on an edit session, and/or to extract any type of information about edits that have not yet been posted.

For example when we integrate our work management system (WMS) to Telvent Designer™ we use the above stored procedures plus some SQL queries to extract additional data about the GIS features that have been created as *part of the design* (within the un-posted edit session). Our WMS is web-based and using the Multiversion Views allows us to get this versioned data directly via the database without requiring Esri ArcObjects or any GIS licenses.

The above versioned query tools are very powerful but we can take it to the next level by actually editing versioned data via SQL as well. Before I give you the details I want to put out a few disclaimers:

- *Editing via Multiversion Views should be implemented very carefully because you are affecting the underlying state ids, state lineage, and the open edit states in the database. I would recommend testing any edit scripts out thoroughly in a test environment before running them in a production database.*
- *I would NEVER recommend editing SDE.Default via a MultiVersion view. The risk is too high and you could really get your geodatabase out of whack. Ideally create a NEW version whenever you want to perform Multiversion View edits. The purpose of the version should be for your SQL edit operations - you typically don’t want to mix Multiversion View edits with your ArcMap edits.*
- *You can’t use a MultiVersion View to create or delete any features that participate in a geometric network. There are a lot of additional network-related edits that occur via ArcMap and the MultiVersion View won’t handle any of them. You can, however, safely update attributes of **existing** geometric network features - but that’s about it.*
- *You aren’t able to create or edit any spatial (shape) data via Multiversion Views so this may limit where you would use them. We have typically used them to create/update object class records which have no spatial component and to possibly update feature class attribution.*
- *As I remember back to the first Multiversion View editing application I wrote in the early 2000’s, I also remember our good friend, Telvent’s Rich Ruh (now VP of Product Development) making the strong point that editing versioned data via Multiversion Views bypasses any and all business logic*

> *you may have implemented via ArcFM™ AutoUpdaters or even Esri Feature Class Extensions.*
>
> *These might include your audit tracking (create/update user and date), any fields set by spatial searches (PLSS grid values), Feeder Manager tracing and updates, and any other custom functions you may have in your geodatabase. This can affect your data quality, integrations with other systems, etc. Esri has officially stated "the only way to **guarantee** the integrity of your geodatabase is to edit it through an ArcObjects application". But then again Esri also **gave us** Multiversion Views along with instructions on using them. We've used them for years in our applications and can safely say that when used appropriately, they can add tremendous value.*

To sum up the above *important* warnings, plan and test your MultiVersion View edits carefully and think through any ramifications *before* you actually make the edits to a production environment. If you have any doubts, call in an experienced and trusted GIS consultant. That's what we are here for… well that and our exceptionally good humor.

OK, now that the warnings are out of the way, let's take a look at how this editing can be accomplished. Just like editing a version in ArcMap, our first step is to start editing. We obviously don't have a handy *start editing* button available to us in SQL but the underlying concept is exactly the same. We must execute a combination of stored procedures to set the current version and then to open the SDE state for editing:

ORACLE:

```
call sde.version_util.set_current_version ('SDE.Working');
call sde.version_user_ddl.edit_version ('SDE.Working', 1);
```

SQL SERVER:

```
exec sde.sde.set_current_version 'SDE.Working';
exec sde.sde.edit_version 'SDE.Working', 1;
```

In the above SQL, the first statement simply sets the connection to point to the specified version and the second statement effectively starts an edit session. The value of "1" causes the database to start editing. In the above example the specified version would already have to exist in the geodatabase. There are some additional stored procedures that can be used to create a new version and/or delete an existing version but I often create and manage my versions in ArcCatalog and ArcMap the old-fashioned way. You can google those topics for some additional reading if interested.

Once you've got an open edit state on the specified version (i.e. started editing) you can now execute insert, update, or delete statements against your Multiversion Views. When issuing insert statements you can't provide a value for the ObjectId since that is managed by SDE just like it is in ArcMap. There will also be an SDE_STATE_ID column - leave that puppy alone as well. It's exactly what you think it is based on our past articles but you need to let SDE do its versioning magic to keep everything in sync. And finally, if you are editing a feature record (with a spatial geometry on the map) the Multiversion View will have a SHAPE column that contains an integer value. You also shouldn't mess with this field at all either. But any other SQL operation on the rest of the business fields works great. For example, if you need to make some mass updates to records you can easily do this via a simple update statement. This quick example would standardize all of your pole heights between 30 and 35 ft to be 35:

```
update Utility.Pole_MV set Height=35 where Height>30 and Height<35;
```

Looks just like regular SQL doesn't it?!? Most typical SQL will work just fine with a few limitations. One major function that does not work is a SQL "update from" statement where you would be updating Table A with some values from Table B based on an inner join between the tables. This is a bit more advanced and is not supported by the Multiversion View. If you need to do this, there are some old-school workarounds I can share if you are interested.

Once you are editing, you can issue multiple SQL statements within the same edit session. Keep in mind that if you lose your connection to the database at any time, you would need to call the set_current_version stored procedure again to get your connection pointed to the right SDE version.

You do not need to call *start editing* on the version more than once unless you have stopped editing (see below). Don't worry though, if you accidentally call it twice, Esri will kick back a pretty informative error message indicating the current state has not been closed (i.e. stop editing has not been called). No harm done. After you have issued your Multiversion View SQL statements within the edit session, you still need to call stop editing to close the state in the database which effectively stops the edit session:

ORACLE:

```
call sde.version_user_ddl.edit_version ('SDE.Working', 2);
```

SQL SERVER:

```
exec sde.sde.edit_version 'SDE.Working', 2;
```

The value of "2" causes the database to *stop editing*. Because we are editing directly in the database via SQL we don't have the concept of Esri edit operations where you can rollback a set of edits. So if you have this requirement when making your SQL edits I'd recommend using a SQL transaction. This is the default when editing in Oracle. You have to either *commit* or *rollback* your edits within the SQL tool (i.e. use the buttons in the UI or SQL statement *commit*;). However, in SQL Server the default is to have your edits committed immediately so you may want to explicitly use the BEGIN TRANSACTION, COMMIT TRANSACTION and/or the ROLLBACK TRANSACTION SQL calls in your script. Alternatively, remember we are editing in an SDE version! You can always stop editing, delete your version, and start all over.

All right, you've now created your SDE version, started editing, made some cool and easy SQL updates, and stopped editing. What's next? Unfortunately there are no reconcile or post operations available in SQL. But this is for good reason. Multiversion Views themselves don't know about conflicts or how to manage the state tree during a reconcile or post. So once you've got your edits saved, you'll need to open the version in good ole ArcMap and reconcile and post it from there. Resolve any conflicts along the way and you'll be in good shape.

Hopefully I haven't totally scared you off the topic of Multiversion Views and you can see the value they might add to your versioned geodatabase. As mentioned earlier, when used carefully and appropriately they are quite safe and powerful. If you have any questions or any corrections (be nice) feel free to send them along. Just use our handy contact page on the website and let us know what you think.

Versioning in Practice

In this next section, we move away from the educational components regarding Versioning and into the more practical everyday elements that will impact how you implement and manage versioning within your geodatabase. We'll first explore your geodatabase performance and touch on everything from reconciling, compressing, managing state lineages and even Oracle tracing for performance validation. Next, we take an in-depth look at removing Oracle KEYSETS and cleaning up your geodatabase for better manageability. Then we'll take a look at a code example on making your ArcFM™ Subtasks version-aware to allow them to fully operate with Esri versions. And finally, we'll review the latest and greatest improvements to versioning from Esri at the utility version of 10.2.1 before closing out with a review of what types of geodatabase changes you can make within a versioned vs. un-versioned geodatabase. These are all topics that you will likely run into if you are a heavy user of Esri versioning.

Life in the Fast Lane

This section describes how to keep your Geodatabase performing well. We will focus on Oracle for this article, but most of the concepts are also applicable to SQL Server. For an overview of Versioning, see the previous chapter, or Esri's article on the same topic. You should be familiar with the basic concepts of versioning before jumping into this deep discussion.

STATES, VERSIONS AND LINEAGES

The first point to make about keeping a geodatabase performing well is that the number of states (and state_lineages) should be kept to a minimum. That is, the number of records in the SDE.STATES table should be as close as possible to the number of records in the SDE.VERSIONS table.

```
SQL> select count(*) from sde.versions;
  COUNT(*)
----------
        23

SQL> select count(*) from sde.states;
  COUNT(*)
----------
        28

SQL> select count(*) from sde.state_lineages;
  COUNT(*)
----------
       108
```

The secret to keeping the number of STATES and STATE_LINEAGES records low is to RECONCILE, RECONCILE, and RECONCILE.

RECOMMENDED RECONCILE ORDER

Reconciling is the process of merging edits from one version (source version) into another (target version). As users edit a Geodatabase, edit change creates a new STATE, and the STATE_LINEAGES defines the list of those changes. In order to keep the number of STATES and STATE_LINEAGES small, you must reconcile any edits posted to Default back into all the other versions.

It's important to note that the versions should be reconciled in the correct order. *See appendix 1 for VBA code to get Recommended Reconcile Order (pre-10.1)*

Any conflicts that arise during the reconcile process should be resolved immediately, before the next version is reconciled.

At ArcGIS 10.1, the ArcPy module has a ReconcileVersions method that can automate the reconcile process while using the Recommended Reconcile Order. *See appendix 2 for Python code to Batch Reconcile Versions (10.1)*

NIGHTLY TUNING (COMPRESS, INDEXES AND STATISTICS)

DBAs are creatures of the night, and they LOVE down time on their servers, so they can get some tuning accomplished while they have exclusive access. Spatial databases are no exception, and there are some basic tuning tasks that your Geodatabase should go thru each night.

1) Compress – A compress is an operation that performs two key tasks. First, it removes unreferenced states (it keeps the SDE.STATE_LINEAGES table small), while also removing records in the Delta tables. Second, it moves any records that it can from the Delta tables up to the Base tables.

There are several ways to run a compress. ArcCatalog has an interactive tool for running the compress, but this is not ideal because it doesn't allow the compress to run as a scheduled task. Therefore, the recommended approach for installations prior to ArcGIS 10.1 is to run the ArcSDE command line:

```
sdeversion -o compress -u sde -p <password> -i esri_sde -N
```

This command can easily be placed in a batch file or shell script file and run as a scheduled task.

2) Indexes – Running that nightly compress helps to keep row counts low in the database. Typically, a database can see 10% or more of their rows moving during a compress operation. This wreaks havoc on the indexes for those tables. So typically after running a compress, a DBA should look for indexes that need to be rebuilt. Most GIS customers have found that rebuilding indexes at least weekly will help keep their database at peak performance.

Note: There has been much speculation by the DBA community as to whether or not indexes should be rebuilt on a schedule, and that debate is beyond the scope of this article. For more detailed insight to the index rebuilding discussion, please visit Oracle's Support site and look for Doc ID: 989093.1.

3) Statistics – Once the compress has been performed, and any possible indexes rebuilt, the most important step in maintaining database performance is to analyze the schemas. When Oracle is asked to perform a query, the Optimizer tries to determine the most efficient way to respond to that query.

In order to accomplish that, the Optimizer needs some information about the tables and indexes involved in that query: row count, row length, index levels and leaf blocks, etc. These are the types of statistics that need to be refreshed after all the nightly tuning operations are complete. As of Oracle 10.2, Oracle has an automatic job (GATHER_STATS_JOB) that will gather statistics for schema objects if those statistics are either missing or stale (out of date).

The following PL/SQL code block is used to perform steps 2 and 3 above (rebuilding all indexes and updating DBMS statistics):

```
declare
    CURSOR owner_cur IS
        SELECT DISTINCT(OWNER) OWNER from SDE.TABLE_REGISTRY ORDER
BY OWNER;

    CURSOR index_cur IS
        SELECT owner, index_name FROM dba_indexes
```

```
            WHERE owner in (select distinct(owner) from sde.table_
registry)
        and INDEX_TYPE = ‘NORMAL’ ORDER BY owner, index_name;

    SQL_STMT VARCHAR2(200);

begin
    FOR IndexRec in index_cur LOOP
        SQL_STMT := ‘alter index ‘ || IndexRec.owner || ‘.’ ||
                   IndexRec.index_name || ‘ rebuild nologging’;
        EXECUTE IMMEDIATE SQL_STMT;
    END LOOP;

    FOR OwnerRec in owner_cur LOOP
        dbms_stats.gather_schema_stats (OwnerRec.owner,
CASCADE=>TRUE);
    END LOOP;
end;
/
```

ORACLE TRACING

Even when the above performance tuning recommendations are followed, there are still occasional user complaints that surface. When they do, the DBA should be prepared to troubleshoot!

Oracle provides the ability to capture all the SQL queries, how long it took to respond to those queries, and how it answered the query. This is called an Oracle Trace (or more specifically, an Oracle 10046 trace event). This is the single most useful tool a DBA has when trying to solve performance issues.

There are several methods of capturing an Oracle trace file. We are going to focus this article from a DBA perspective, so we will use only Oracle tools to capture and analyze the trace.

Note: It is VERY important that the trace file be properly scoped. We don’t want to trace EVERYTHING the user does, we only want to trace a small portion of their activity that they consider “slow”. For example, users typically claim that the following types of activities are slow: “opening a specific stored display”, or “adding a transformer”. So, before we start tracing the user activity, the user should have their session ready up to the point of starting that activity. This is a coordinated effort between user and DBA.

Once the user is ready to start their activity, they alert the DBA. The user should not start their activity until the DBA gives them a green light. The DBA needs time to find their session and begin the trace. So in our example, user “Jeff” has complained that drawing service points is slow. The user opens his stored display, and gets ready to draw Service Points, then tells the DBA to trace his session. The DBA can use the following to start the Oracle trace:

Step 1) Find the Process ID (or PID) of our session:

```
SQL> select pid from v$process where addr = (select paddr from
v$session where username = ‘JEFF’);

       PID
----------
        70
```

Step 2) Use OraDebug to attach to the session:

```
SQL> oradebug setorapid 70
Oracle pid: 70, Windows thread id: 9784, image: ORACLE.EXE (SHAD)
```

Step 3) Use OraDebug to start tracing:

```
SQL> oradebug event 10046 trace name context forever, level 12;
Statement processed.
```

Step 4) The user performs the activity, and immediately tells the DBA when it's complete. Upon completion of the activity, the DBA will use the following commands to stop the Oracle trace:

```
SQL> oradebug event 10046 trace name context off;
Statement processed.
```

Step 5) Locating the trace file

```
SQL> oradebug tracefile_name
C:\ORACLE\diag\rdbms\gisprod\gisprod\trace\gisprod_ora_9784_JEFF.trc
```

Once we know the trace file name, we can use Oracle's TKPROF utility to format the trace file into more human-readable text. We can also use TKPROF to sort the output, and/or generate an explain plan. Here is one example of using TKPROF on the tracefile:

```
tkprof gisprod_ora_9784_JEFF.trc jeff.txt sort=fchela explain=sde/
sde@gisprod
```

This will create an ouptut textfile called "jeff.txt". The output is sorted by Elapsed Time to Fetch each statement, and will generate an explain plan to show us how Oracle is solving each query. Continuing with our example, here is the first SQL statement in our output file:

```
SELECT  /*+  USE_NL(V__2108)  INDEX(SHAPE F1001_UK1) */  SHAPE,  Sub-
typeCD  ,
  V__2108.eminx,V__2108.eminy,V__2108.emaxx,V__2108.emaxy ,SHAPE.fid,
  SHAPE.numofpts,SHAPE.entity,SHAPE.points,SHAPE.rowid
FROM
 (SELECT /*+ LEADING */ b.SHAPE,b.SUBTYPECD  ,S_.sp_fid,S_.eminx,S_.
eminy,
  S_.emaxx,S_.emaxy  FROM (SELECT  /*+ INDEX(SP_ S1001_IX1) */ DIS-
TINCT
  sp_fid, eminx, eminy, emaxx, emaxy FROM ARCFM.S1001 SP_  WHERE SP_.
gx >= :1
  AND SP_.gx <= :2 AND SP_.gy >= :3 AND SP_.gy <= :4 AND SP_.eminx
<= :5 AND
  SP_.eminy <= :6 AND SP_.emaxx >= :7 AND SP_.emaxy >= :8)S_,
  ARCFM.ServicePoint b WHERE S_.SP_FID = b.SHAPE AND b.OBJECTID NOT
IN
  (SELECT /*+ HASH_AJ */ SDE_DELETES_ROW_ID FROM ARCFM.D2108 WHERE
DELETED_AT
  IN (SELECT l.lineage_id FROM SDE.state_lineages l WHERE l.lineage_
name =
  :source_lineage_name AND l.lineage_id <= :source_state_id) AND
SDE_STATE_ID
  = :”SYS_B_0”) UNION ALL SELECT /*+ LEADING */ a.SHAPE,a.SUBTYPECD
,
  S_.sp_fid,S_.eminx,S_.eminy,S_.emaxx,S_.emaxy  FROM (SELECT  /*+
INDEX(SP_
  S1001_IX1) */ DISTINCT sp_fid, eminx, eminy, emaxx, emaxy FROM AR-
CFM.S1001
  SP_  WHERE SP_.gx >= :1 AND SP_.gx <= :2 AND SP_.gy >= :3 AND SP_.
gy <= :4
  AND SP_.eminx <= :5 AND SP_.eminy <= :6 AND SP_.emaxx >= :7 AND
SP_.emaxy >=
   :8)S_, ARCFM.A2108 a,SDE.state_lineages SL WHERE S_.SP_FID =
a.SHAPE AND
  (a.OBJECTID, a.SDE_STATE_ID) NOT IN (SELECT /*+ HASH_AJ */
  SDE_DELETES_ROW_ID,SDE_STATE_ID  FROM ARCFM.D2108 WHERE DELETED_AT
IN
  (SELECT l.lineage_id FROM SDE.state_lineages l WHERE l.lineage_name
=
  :source_lineage_name AND l.lineage_id <= :source_state_id) AND
SDE_STATE_ID
  > :"SYS_B_1") AND a.SDE_STATE_ID = SL.lineage_id AND SL.lineage_
name =
  :source_lineage_name AND SL.lineage_id <= :source_state_id )
V__2108,
  ARCFM.F1001 SHAPE  WHERE V__2108.SHAPE = SHAPE.fid
```

```
call    count  cpu     elapsed disk    query   current rows
------- ------ ------- ------- ------- ------- ------- -------
Parse   0      0.00    0.00    0       0       0       0
Execute 3      0.00    0.00    0       0       0       0
Fetch   148    0.35    0.39    0       79932   0       14622
------  ------ ------  ------  ------  ------  ------  ------
total   151    0.35    0.40    0       79932   0       14622
```

There's some UGLY looking SQL in there, but the end result is that our database is fetching 14,622 rows (in this case, from ARCFM.ServicePoint feature class) in under ½ a second. This is an example of a database performing well! The only problem with this user operation is that the user is probably drawing too many ServicePoint records, and should set a proper scale dependency on the layer.

SCRIPT 1: VBA CODE TO GENERATE THE RECOMMENDED RECONCILE ORDER

```
Public Sub RecOrder()
On Error GoTo EH:
    ‘ Get our Application object
    Dim pApp As IGxApplication
    Set pApp = ThisDocument.Parent

    ‘ Get our selected object.
    If Not TypeOf pApp.SelectedObject Is IGxObject Then
        MsgBox “Error 1.  Exiting.”
        Exit Sub
    End If

    Dim pObj As IGxObject
    Set pObj = pApp.SelectedObject

    ‘ Check for a database connection.
    Dim pDB As IGxDatabase
    If Not TypeOf pObj Is IGxDatabase Then
        ‘ Check for a database feature class/dataset
        If Not TypeOf pObj.Parent Is IGxDatabase Then
            MsgBox “Please select an Oracle database connection and
retry.”
            Exit Sub
        Else
```

```
            Set pDB = pObj.Parent
        End If
    Else
        Set pDB = pObj
    End If

    ‘ QI for the workspace
    Dim pWorkSpace As IWorkspace
    Set pWorkSpace = pDB.Workspace

    ‘ Make sure it’s a remote database.
    If pWorkSpace.Type <> esriRemoteDatabaseWorkspace Then
        MsgBox “Please select an Oracle spatial database connection
and retry.”
        Exit Sub
    End If

    Dim pVWS As IVersionedWorkspace2
    Set pVWS = pWorkSpace

    Dim pEnumVersion As IEnumVersionInfo
    Set pEnumVersion = pVWS.RecommendedReconcileOrder

    Dim pVI As IVersionInfo
    pEnumVersion.Reset
    Set pVI = pEnumVersion.Next

    Do Until pVI Is Nothing
        Debug.Print pVI.VersionName
        Set pVI = pEnumVersion.Next
    Loop

    Exit Sub
EH:
    MsgBox Err.Description
End Sub
```

SCRIPT 2: PYTHON CODE FOR BATCH RECONCILING VERSIONS (REQUIRES ARCGIS 10.1 OR LATER)

```
import arcpy
try:
    # set a variable for the workspace
    workspace = ‘c:\\connections\\gisprod_sde.sde’

    # get a list of connected users.
    userList = arcpy.ListUsers(workspace)
    print userList

    #disconnect all users from the database.
    arcpy.DisconnectUser(workspace, “ALL”)

    # set the workspace
    arcpy.env.workspace = workspace

    #block new connections to the database.
    arcpy.AcceptConnections(workspace, False)

    # Get a list of versions to pass into the ReconcileVersions tool.
    versionList = arcpy.ListVersions(workspace)

    # Execute the ReconcileVersions tool.
    arcpy.ReconcileVersions_management(workspace, “ALL_VERSIONS”,
“sde.DEFAULT”, versionList, “LOCK_ACQUIRED”, “NO_ABORT”, “BY_OB-
JECT”, “FAVOR_TARGET_VERSION”, “POST”, “DELETE_VERSION”, “c:/temp/
reconcilelog.txt")

    # Run the compress tool.
    arcpy.Compress_management(workspace)

    #Allow the database to begin accepting connections again
    arcpy.AcceptConnections(workspace, True)

except Exception, e:
    print e
    print arcpy.getmessages()
    arcpy.AcceptConnections(workspace, True)
```

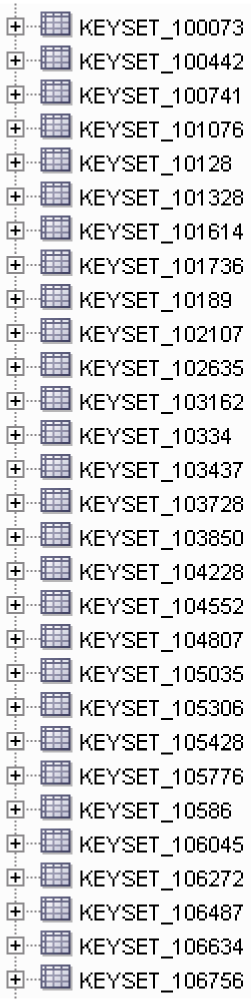
Cleaning the Gunk Out of Your Oracle GDB

Have you ever been browsing through your Oracle geodatabase via TOAD or SQL Query Analyzer and noticed a certain calcification of KEYSET tables? These might exist in your SDE schema and/or other user schemas in the database. If you're like me, you've noticed them but never paid them much attention.

But what are they? And how are they affecting the geodatabase storage, performance, and organization?

ArcGIS KEYSET tables are used by the geodatabase for quickly traversing relationship classes where the input set exceeds 100 objects. As a reminder, when you create an Esri Relationship Class in the geodatabase, it is not creating a *physical* relationship in the underlying database. That would be far too cumbersome for the geodatabase, especially when you factor in the additional ADD tables created by versioning your data.

But in the ArcMap application, the relationship class acts like a real database relationship and is able to show all related records for a selected object (i.e. all transformer units for a transformer bank OR all gas mains related to a cathodic protection system).



KEYSET_100073
KEYSET_100442
KEYSET_100741
KEYSET_101076
KEYSET_10128
KEYSET_101328
KEYSET_101614
KEYSET_101736
KEYSET_10189
KEYSET_102107
KEYSET_102635
KEYSET_103162
KEYSET_10334
KEYSET_103437
KEYSET_103728
KEYSET_103850
KEYSET_104228
KEYSET_104552
KEYSET_104807
KEYSET_105035
KEYSET_105306
KEYSET_105428
KEYSET_105776
KEYSET_10586
KEYSET_106045
KEYSET_106272
KEYSET_106487
KEYSET_106634
KEYSET_106756

Oracle KEYSET Tables

To achieve effective performance in rendering these relationships in the user interface, Esri creates a temporary KEYSET table for the current session that provides tabular linkage for large relationships. This allows it to act like there is a physical relationship in the database by emulating it via a join table.

The owner of the table (SDE or a user account) is the user who was connected and used the relationship in ArcMap. OR in some cases - where individual users don't have the Oracle privileges to create KEYSET tables - they may *ALL* end up under the SDE user schema. The numeric value at the end of the table name corresponds to a specific connected user session within the GIS.

So if these are temporary join tables, why are they sticking around?

In SQL Server databases, the keyset table is indeed created as a temporary table. In Oracle, however, the KEYSET tables are created within user schemas as described above. And when a session terminates abnormally, the KEYSET table may not be dropped from the database, thus orphaning it and beginning the stalagmite growth that you may now find in the bowels of your geodatabase.

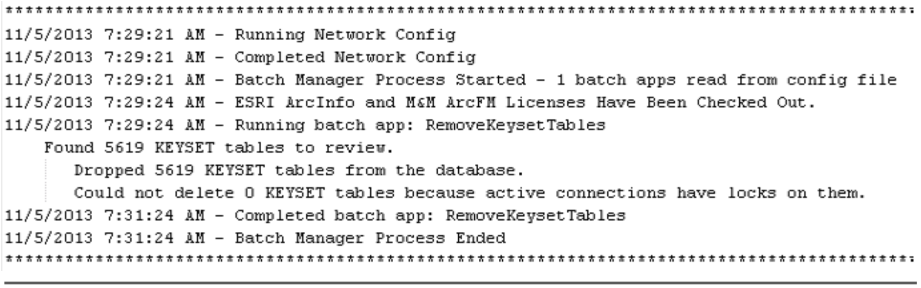
We work with a lot of Oracle customers and have regularly seen a TON of these orphaned tables. To check out your system, run the following SQL query:

```
select count(*) from all_tables where table_name like 'KEYSET_%';
```

In one of my customer test databases from a mid-sized client, there were ***6,668 orphaned KEYSET tables!*** Any thoughts about ignoring these little nuggets of joy should go out the window. Once orphaned, they are never needed again and can be safely removed from the database.

To make this an automated process, we have created a new [SSP Nightly Batch Suite]() (NBS) application that can clean up these tables on a weekly or even nightly basis. The application validates that each KEYSET table is indeed orphaned by comparing the owner and session id against the active SDE process table.

If there's no reference to the KEYSET table then the app whacks it, mafia style. It then kicks out a nice report into our batch log file:



```
*******************************************************************************************:
11/5/2013 7:29:21 AM - Running Network Config
11/5/2013 7:29:21 AM - Completed Network Config
11/5/2013 7:29:21 AM - Batch Manager Process Started - 1 batch apps read from config file
11/5/2013 7:29:24 AM - ESRI ArcInfo and M&M ArcFM Licenses Have Been Checked Out.
11/5/2013 7:29:24 AM - Running batch app: RemoveKeysetTables
    Found 5619 KEYSET tables to review.
       Dropped 5619 KEYSET tables from the database.
       Could not delete 0 KEYSET tables because active connections have locks on them.
11/5/2013 7:31:24 AM - Completed batch app: RemoveKeysetTables
11/5/2013 7:31:24 AM - Batch Manager Process Ended
*******************************************************************************************:
```

Batch Log File

The end result? A clean and efficient database. If you combine this cleanup app with our other [Oracle indexing and statistics NBS apps]() you are sure to provide your geodatabase with a much-needed performance boost. Not to mention you won't spend all day scrolling through those pesky tables when you are exploring your SDE schema!

So say goodbye to the KEYSETS and say hello to… *I've run out of silly puns in this article, but you get the idea!*

A special thanks to our resident DBA, Jeff Buturff, for coming up with the idea for this app. I love working with people who are always striving to improve the systems we work on just for the fun of it.

Happy geodatabasing.

Making your Subtask Version-Aware

There are many workarounds a developer must endure when working with the intricacies of Esri and ArcFM™.

One such workaround encapsulates posting a session via ArcMap and firing a customized subtask.

In a technical sense, posting a session will move the version of the database back to Default so iterating through an ITable might cause a few unwanted headaches.

Like many facets of programming and life, there are many ways to accomplish the same result. This article is a biased perspective on how to work around posting a session that loses all your changes utilizing static classes and refiring a subtask.

SCENARIO

So let's create a simple fictitious scenario that will encapsulate our need to utilize a static class and refire of a subtask.

Create a customized application that will edit or create new users in the database.

While editing a session within ArcMap, you launch a customized application in ArcMap that edits or updates users.

These user changes will be saved in the ITable for users. Create a subtask that will grab the user changes in the User ITable and send an email called "Get Users".

Have the subtask (Get Users) fire after posting the session and you'll notice that none of your users will get emailed, because posting a session will place the pointer back to default or the unversioned database.

SOLUTION

Now let's look at a solution for the dilemma described prior. Basically, launch the subtask prior to the Post Session subtask and after the Post Session subtask.

The first launch of "Get Users" will look to see if a list of static classes that stores properties for the user is null or not. If the static class is null, then the subtask of "Get Users" will store all the changed users in the static User class.

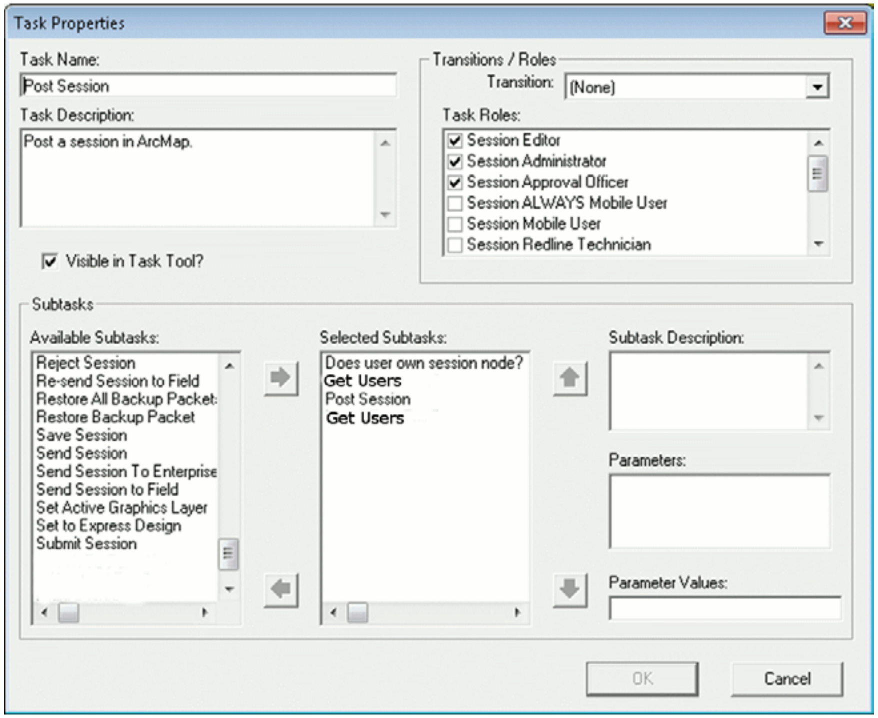
If the static class is not null, then the subtask of "Get Users" will iterate through the user class and send each user an email.

PROCESS FRAMEWORK ADMINISTRATION TOOL SETUP

1. *Have the "Get Users" subtask fire before the Post Session to grab all the updated users in the ITable.*

2. *Launch the Post Session subtask after "Get Users".*

3. *Launch "Get Users" again*



Task Properties

Task Name: Post Session

Task Description: Post a session in ArcMap.

☑ Visible in Task Tool?

Transitions / Roles

Transition: (None)

Task Roles:

☑ Session Editor
☑ Session Administrator
☑ Session Approval Officer
□ Session ALWAYS Mobile User
□ Session Mobile User
□ Session Redline Technician

Subtasks

Available Subtasks:

Reject Session
Re-send Session to Field
Restore All Backup Packet:
Restore Backup Packet
Save Session
Send Session
Send Session To Enterprise
Send Session to Field
Set Active Graphics Layer
Set to Express Design
Submit Session

Selected Subtasks:

Does user own session node?
Get Users
Post Session
Get Users

Subtask Description:

Parameters:

Parameter Values:

STATIC CLASS IMPLEMENTATION

Static classes allow the software engineer to store and reuse information without expensive hits to the database or other data storages.

1) Static class to store user information

```
    /// <summary>
    ///    User Class
    /// </summary>
         public class UserClass
         {
            public string UserId { get; set; }
            public string FirstName { get; set; }
            public string LastName { get; set; }
            public string Email { get; set; }
          }
```

2. Service Factory or Class that implements the static UserClass. The REALLY important aspect is the last method of ResetValues() because it will null out the UserClass.

```
    public sealed class UserContainer
    {
        private static volatile UserContainer _instance = new User-
Container ();
        private static volatile List< UserClass > _getUserClasses;
        private static readonly object SyncRoot = new object();
        /// <summary>
        ///    Static Instance
        /// </summary>
        public static UserContainer Instance
        {
            get
    {
        if (_instance == null)
        {
            lock (SyncRoot)
            {
                if (_instance == null)
                    _instance = new UserContainer ();
            }
        }
        return _instance;
    }
}
/// <summary>
/// Get and Set  List of UserClass
/// </summary>
public static List< UserClass > UserClasses
{
     get
    {
        return _ getUserClasses;
    }
    set
    {
        _ getUserClasses = new List< UserClass >();
        _ getUserClasses = value;
    }
 }
/// <summary>
/// Reset
/// </summary>
public static void ResetValues()
{
    _ getUserClasses = null;
```

```
        }
    }
}
```

SUBTASK IMPLEMENTATION

The subtask will launch with the Execute method below and verify that the static class is null or hydrated to then verify the logical path of execution.

```
/// <summary>
       ///     Execute the subtask
       /// </summary>
       /// <param name=”pPxNode”>The IMMPxNode</param>
       /// <returns>Boolean</returns>
       public bool Execute(IMMPxNode pPxNode)
       {
                     if (UserContainer.UserClasses == null)
                    {
                        List< UserClass> listUserClass  = new
List<UserClass>();
                         //Query the iTable of Users and store in
the listUserClass
                    }
                    else
                    {
 //Do logic to iterate through each User in listUserClass
//IMPORANT TO RESET VALUES
                        UserContainer.ResetValues();
                    }
                    return true;
        }
```

CONCLUSION

Though this section was a little bit on the technical side, I feel that sharing the knowledge of how to work around issues that are not completely obvious is a good thing to share.

The above example code was taken from actual code with variables and classes renamed to keep things a little more generic.

I hope you found this helpful, insightful, and beneficial in your journey as an ArcGIS guru.

Conflict Management Improvements at 10.2.1

Esri's version 10.2.1 was released right around the end of last year and Schneider Electric's 10.2.1 has been released and patched over the course of Q1 2014. At this point we have a pretty solid release with a lot of performance improvements from both Esri and Schneider.

I want to focus on another area of major improvement in 10.2.1 that will impact every single one of our customers - *conflict management*. Before I jump in, if you are interested in getting more familiar with versioned conflicts and how they work, see the previous section titled ***Reconcile, Post, and Compress... Oh My!***.

While we have the occasional customer that only manages a small number of versions on a daily basis, many of our customers are Schneider Electric Designer™ users, which almost guarantees a significant amount of long transaction versions that are prone to being impacted by conflicts. In past versions of the software some of the most troublesome areas with conflicts were caused by:

- ***Mass data updates*** *- if you needed to initialize or update a field across an entire feature class it would definitely have the potential to cause significant conflicts. Especially if you didn't turn off your Schneider update user/date AU's!*

- ***Feeder Manager Updates*** *- most electric utilities use Feeder Manager to manage their electric networks and when a permanent switching order, phase swap or other similar update was made, it had the potential to trigger updates to hundreds of additional downstream features.*

- ***Geometric Network Updates*** *- the trouble with updating records in your geometric network (gas, electric, water, etc.) is that there are edits occurring to the network tables (N_ in the database) that the user isn't even aware of. Network edits can cause additional conflicts because of conflicting edits to the logical network records for the same record and often even affect changes to the adjacent records in the network at the logical level. This just increases your exposure to conflicts because of the additional edits occurring behind the scenes.*

- ***Relationship Updates*** *- when you create relationship classes in your geodatabase you can specify notification directions. This can allow an update to one feature in the relationship to notify related records of the update. This can be helpful functionality for keeping your system updated but that additional edit notification can also generate conflicts in the system based on those related records.*

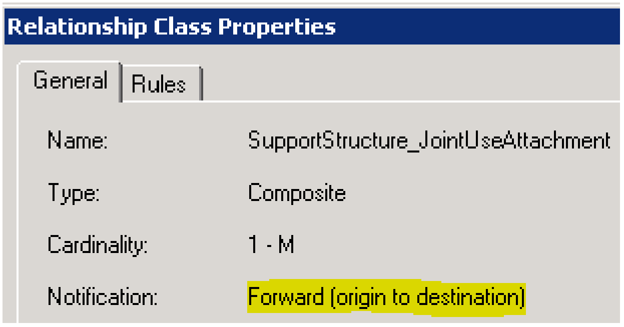
When you combine these updates with a Designer™ system that has hundreds or even thousands of existing edit versions, you will likely have many conflicts in a given day and occasionally even end up with a HUGE list of conflicts resulting from single large data update. Just about every GIS administrator I know is familiar with receiving that (hopefully automated) *dreaded* notification that they have a large percentage of the system in conflict.

Over the years many utilities have implemented automated systems to help with the reconcile process. Schneider's Geodatabase Manager™ is a great tool that manages version posting queues during the day and can also reconcile all of your versions. And SSP's Nightly Batch Suite will do a full state tree reconcile of the entire system with *a single conflict report each night* along with your SDE geodatabase compress, statistics, indexing, and performance analysis.

These tools give a GIS administrator everything they need to have a good handle on their system health, including where the conflicts are and how the system is performing. *BUT they don't directly help to reduce conflicts...*

And that's where the new Esri 10.2.1 release comes into play. So what did Esri do to help with conflicts?

- ***Reduced propagation due to relationship classes*** *- as mentioned above relationship classes can cause additional edits and therefore additional conflicts. Esri has added specific logic to better review these changes to reduce propagating edits that don't need to occur.*



Relationship Class Properties

Name: SupportStructure_JointUseAttachment

Type: Composite

Cardinality: 1 - M

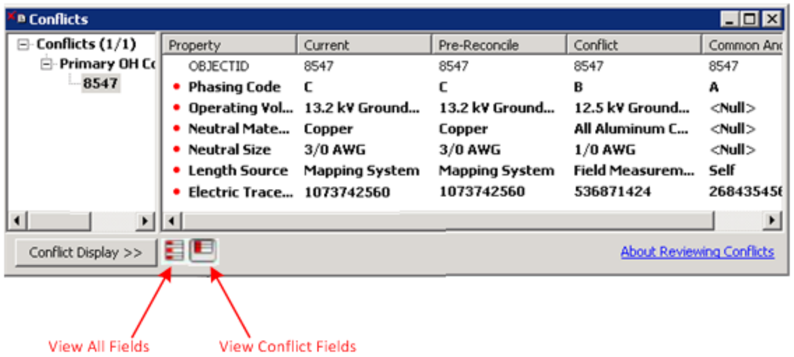
Notification: Forward (origin to destination)

Relationship Class Notification

- ***Filtering out false update-update conflicts*** *- if you've been working with conflicts for any length of time you have inevitably run into the case where you review the conflicts in the conflict dialog only to say, "What the heck, there's not even a conflict on these records?!?" or perhaps some other choice language. Well Esri has heard your plea and they have added addi-*

tional checks so that many of those cases won't even be flagged as conflicts in the first place.

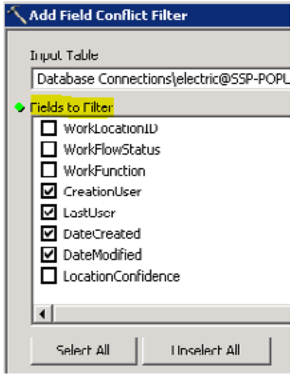
- ***Allow users to only see fields that are in conflict in the conflict dialog*** *- does anyone have classes with a TON of attributes? Have you ever spent too much time searching for the conflict at the attribute level? Well now you can click a filter button that will ONLY show the fields that are in conflict. This is a simple but powerful change that should make your conflict review process faster.*



	Property	Current	Pre-Reconcile	Conflict	Common An
	OBJECTID	8547	8547	8547	8547
●	Phasing Code	C	C	B	A
●	Operating Vol...	13.2 kV Ground...	13.2 kV Ground...	12.5 kV Ground...	<Null>
●	Neutral Mate...	Copper	Copper	All Aluminum C...	<Null>
●	Neutral Size	3/0 AWG	3/0 AWG	1/0 AWG	<Null>
●	Length Source	Mapping System	Mapping System	Field Measurem...	Self
●	Electric Trace...	1073742560	1073742560	536871424	26843545

Filter Fields in Conflict Dialog

- ***Conflict FilterConflict filters available in the product!*** *- This is definitely one of the most exciting items for me because we have built custom conflict filters for customers in the past. This is a great add to the product… basically you will be able to configure specific fields within your data model to NOT cause conflicts. For example, those pesky update user/date fields that are updated every time you make an edit.*



Input Table

Database Connections\electric@SSP-POPL

Fields to Filter

- □ WorkLocationID
- □ WorkFlowStatus
- □ WorkFunction
- ☑ CreationUser
- ☑ LastUser
- ☑ DateCreated
- ☑ DateModified
- □ LocationConfidence

Conflict Filters

You can now add a conflict filter to those fields so they NEVER cause a conflict again. Very useful stuff… conflict filters can also be applied temporarily when you have to make a mass update to a specific field. Just post your mass update, reconcile all the versions, and finally remove the conflict filter.

The combination of these conflict management changes will be HUGE for the utility industry. The result will be less time spent on managing conflicts which will drive a healthier and better performing geodatabase. This doesn't change the need for tools like Geodatabase Manager™ or the SSP Nightly Batch Suite but it will result in those processes operating more effectively and efficiently.

One final major benefit comes from the Schneider Electric release of

Feeder Manager 2.0. If you switch to only using FM 2.0, the application will no longer update ALL of those downstream features during circuit initialization, switching orders, etc. This is because the new feeder id and feeder info fields are now tracked by the application in memory as opposed to being stored as attributes on the individual features. This results in a huge reduction in feature edits being applied to your versions which directly reduces the potential for conflicts.

There have been a few concerns around moving to Feeder Manager 2.0 due to challenges supporting existing business processes & integrations but Schneider is doing a nice job of listening to those concerns and will continue to add functionality to Feeder Manager 2.0 like the *Feeder Synchronization* tool in the next release this summer. But that's a topic for another article! We have our first Feeder Manager 2.0 systems in production and so far the benefit of the new functionality has heavily outweighed the challenges.

All in all these changes are a huge win for users of Esri versioning. It's taken a while for them to be added to the product but it's been worth the wait - a huge shout out to Esri's Larry Young who spearheaded the effort to get these enhancements included for utilities! It should also be noted that some of these enhancements *have* and/or *are being* ported back to previous releases at 10.1, 10.0, and perhaps even 9.3.1. If you need more info on that let us know... otherwise best of luck pushing toward 10.2.1!

Debunking the Myth about Applying Data Model or Schema Changes to a Versioned Geodatabase

A question we are often asked is: what data model or schema changes are allowed on a versioned geodatabase, if any? The answer may surprise a few of you, as there are a lot more modifications allowed than you might think.

In ArcGIS, data model changes can be separated into two categories: **minor** - such as adding or deleting fields and **major** - such as removing a feature class from a geometric network.

For the **minor** data model changes identified below, the participating feature dataset, standalone feature class or table in the geodatabase DOES NOT have to be unversioned. Meaning these changes can be applied with versions remaining in the geodatabase with no impact.

MINOR DATA MODEL CHANGES - NOT REQUIRED TO UNVERSION

- *Adding or deleting fields*
- *Adding or deleting simple feature classes (those that do not participate in a topology or geometric network)*
- *Adding or deleting tables*
- *Creating or deleting relationship classes*
- *Creating or deleting attribute domains*
- *Creating or deleting subtypes*
- *Changing the default value for an attribute of a given class or subtype*

- *Changing the attribute domain for an attribute of a given class or subtype*
- *Creating or deleting indexes for an attribute of a given class*
- *Altering the spatial index for a given feature class*
- *Adding an empty feature class to a geometric network*
- *Adding or removing relationship class rules*
- *Adding or removing geometric network connectivity rules*
- *Altering the alias name for classes or fields*
- *Adding weights to a geometric network (beginning with ArcGIS 9.2)*
- *Deleting a geometric network*
- *Renaming or deleting a topology*

For the **major** data model changes identified below, the participating feature dataset, stand-alone feature class or table in the geodatabase MUST be unversioned. In other words, these changes would have considerable impacts on versions and would affect geodatabase performance.

Unversioning the geodatabase requires that (1) all versions to be reconciled and posted to DEFAULT, (2) all versions deleted, and (3) the geodatabase compressed, ensuring all edits in the delta tables are moved to the base tables. Perform these three steps prior to *unregistering as versioned* the participating feature dataset, standalone feature class or table.

MAJOR DATA MODEL CHANGES - REQUIRED TO UNVERSION

- *Removing weights from a geometric network*
- *Removing a feature class from a geometric network*
- *Adding or removing a feature class to a topology*
- *Adding or removing topology rules*
- *Altering a topology's cluster tolerance*
- *Altering the rank for a feature class in a topology*

There are a lot of incorrect assumptions out there about this issue. After reading this article, you should have a better understanding of what data model or schema changes can be applied to a versioned geodatabase, hopefully saving you time when you come across these situations in the future.

A Versioning Case Study

So you've read about the basics of Versioning. Learned about tips and techniques to keep your geodatabase humming at peak performance. Now it's time to bring this all together and talk about the possibilities of the elusive State 0. As noted in the section on debunking the myths about applying data model or schema changes to a versioned geodatabase, there are certain operations that require state 0. In addition to this list we've found many real life cases where state 0 was a necessity to enable various business processes or other advanced geoprocessing updates like data re-projection, conflation, or even moving data between geodatabases.

Here, we close out our Versioning book with a real-life case study on how it's possible to obtain State 0 using everything we've covered so far… plus a little product we've created of course! We'll walk you through, step by step, on how this was achieved for a fearless Colorado utility and then expand on the use of the product for replaying data between geodatabases.

The Elusive State 0 - Can We Get There?

Over the years we've constantly run into a challenge with Esri utility customers who need to get to 'State 0' to perform some geodatabase operation. In a past article we documented the geodatabase operations that can be done with versions in place and those that require you to get to State 0 with an un-versioned geodatabase.

For those readers who aren't familiar with the concept of State 0, this basically means getting rid of all of your Esri versions, compressing the geodatabase to move your edits from the Add & Delete tables to the base tables, and then most often un-versioning your geodatabase which completely removes the Add & Delete tables. If you're asking yourself what the Add & Delete tables are in the first place, please read the ***An Intro to Versioning*** chapter in this eBook.

The main pain-point for utilities arises because we are so focused on geometric networks and long transactions.

Our geometric networks exist to manage our electric, gas, water, or telecom connectivity and we need those long transactions to allow us to create design versions for our proposed network changes - this is common at all utilities but is especially prevalent if you are a Schneider Electric (SE) Designer™ user where versions might be outstanding for months or even years in some cases.

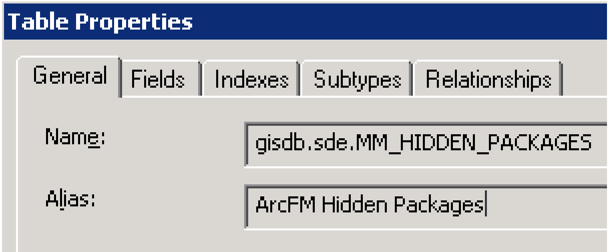
So when you occasionally get to a point where you have to un-version your geodatabase most utilities are in the *'oh crap that's not happening anytime soon'* mindset. Some of our largest customers have more than 6,000 versions at any given time while even most smaller Designer™ sites usually have over 100 versions they can't get rid of.

So therein lies our challenge.

Is there a way for us to get to State 0 with an un-versioned geodatabase without losing all those versions?

If you follow our newsletter/blog you may have read Dennise Ramirez's post called Show Me the Edits, where she covers enhancements to our SSP All Edits tool which allows users to save all of a version's edits into our own storage tables within your geodatabase. That alone is pretty cool but that got us to thinking about how we might utilize that infrastructure to get to State 0 without losing our versioned edits.

The goal would be to run the All Edits export against ALL of the outstanding design and edit versions to cache the edits into the SSP table structure. Our tool already reconstructs a version's edits into Esri graphics with full attribution even after the original version has been deleted. So it's not too far a stretch to say we could reconstruct them back into actual Esri versions in the geodatabase.



Name: gisdb.sde.MM_HIDDEN_PACKAGES

Alias: ArcFM Hidden Packages

Design XML

The twist here for SE Designer™ users is that SE maintains quite a bit of information about the Design version within a binary XML field in the database. Most importantly, this XML contains object ids that point to the specific edited records within a design version. So in addition to recreating the versioned edits we also need to handle re-synching the XML with the ***new*** object ids.

With that in mind, here's our approach:

1. *Run a full system reconcile on an existing geodatabase, identify any version conflicts, and resolve them in their respective versions. The starting point should be a geodatabase without any conflicts, which is fully reconciled. Obviously we should remove any versions that don't need to be kept around.*

2. *Create a [Nightly Batch Suite] application to run the SSP All Edits extraction against ALL of the outstanding Esri versions to cache their edits into the SSP table structures, including all attribution and geometries.*

3. *Delete all Esri versions, compress the geodatabase, and un-version the GIS data.* ***I know… a scary thought for most of us…***

4. *Perform whatever un-versioned, State 0 activities that caused us to attempt this crazy idea in the first place (geometric network changes, re-projection, topology changes, etc).*

5. *Re-version the geodatabase.*

6. ***Here comes the magic…*** *Create a second [Nightly Batch Suite] application that will programmatically recreate all of the edit versions, replay the edits from the SSP table structures as* ***actual Esri edits*** *within those versions, and resynchronize the Designer™ XML to point to the* ***new*** *object ids.*

And *voila!*, we open up our SE Sessions, Designs, and Esri versions like nothing ever happened! Impossible you say? Well we're set to find out and have found a [customer that believes we can do it at Intermountain REA] right here in beautiful south Denver.

We have received a lot of interest from many, many utilities who want to track our progress and results in this effort. So far we have completed the first Nightly Batch Suite app detailed in #1 above and are starting work on #6. We will keep you all updated and will even host the celebration party here in Denver when we make it all work!

The Big State 0 Theory

Up until now State 0 has been nothing but a theory, or simply a subject that has been avoided most of the time due to its complexity. SSP is very happy to announce that we have successfully proven the State 0 theory, which was explained in the previous section.

We would like to thank Duane Holt – Director of GIS (and general rock star) at Intermountain Rural Electric Association (IREA) who believed in our solution when it was just a theory, and Schneider Electric for their constant support throughout this process.

WHAT DROVE US TO GET INTO THIS JOURNEY?

A simple request from a client can make a huge difference in creating innovative solutions. Don't ever be afraid to request something, even if it sounds crazy. There is always a chance you will find crazy-idea lovers like us who are willing to give it a shot!

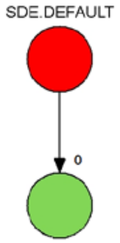
The request from IREA was to re-project their data from NAD 1927 to NAD 1983 without losing their versions. As you can see it's a very simple request, but the complexity was challenging to say the least.

A combination of the SSP Nightly Batch Suite, the SSP All Edits Tool, several white-board sessions, and a creative vision made it all possible.

HOW DOES THE PROCESS WORK?

1. *We use a new configurable SSP Nightly Batch Suite (NBS) application to extract all versions into the SSP All Edits tables and feature classes. These are effectively the same tables and feature classes used by the SSP All Edits Report when extracting an individual version for re-visualizing it at a later point in time.*

2. *Delete all versions and compress the database. Your versioning lineage should look something like this:*



3. *If required, drop all relationships, all geometric networks, and un-version the GIS Data.*

4. *Do whatever you need to do once you are at State 0 (In IREA's case we had to re-project their data from NAD 1927 to NAD 1983).*

5. *Use a second configurable SSP NBS application to recreate all versions extracted in step 1. This application creates the versions using the exact same name and then replays all of the edits including inserts, updates, and deletes.*

6. *If you are a Schneider Electric Designer™ shop, then there is one additional SSP NBS application that is used to re-synchronize each design's XML which includes references to the versioned object ids.*

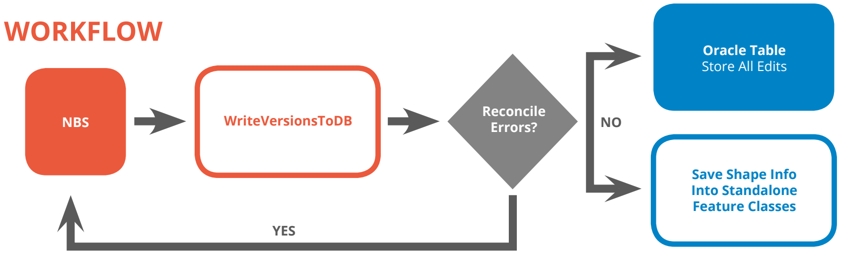
7. *Celebrate!*

The following diagrams will help you understand the flow a little better:

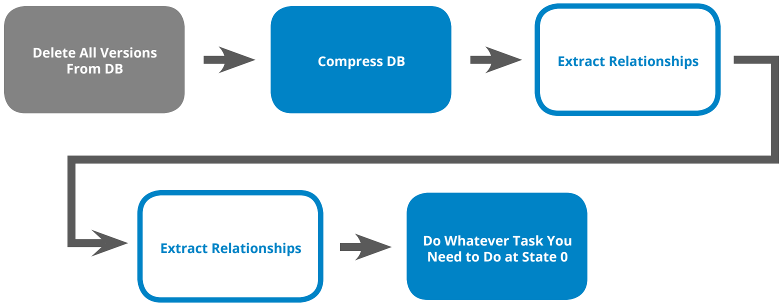
LEGEND

- NBS
- ALL EDITS
- MANUAL PROCESS
- DATABASE
- GIS

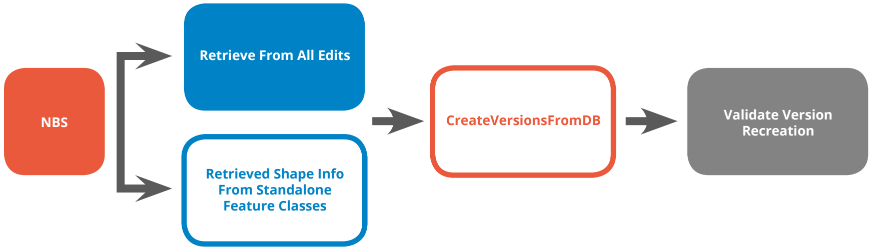
WORKFLOW



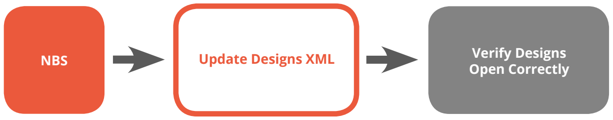
WORKFLOW


Delete All Versions From DB
Compress DB
Extract Relationships
Extract Relationships
Do Whatever Task You Need to Do at State 0

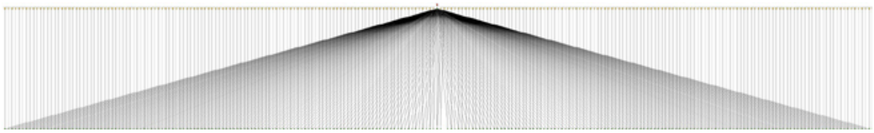
WORKFLOW


NBS
Retrieve From All Edits
Retrieved Shape Info From Standalone Feature Classes
CreateVersionsFromDB
Validate Version Recreation

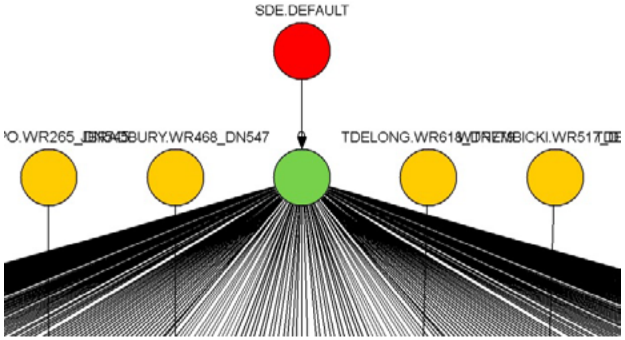
IF SCHNEIDER ELECTRIC DESIGNER™ SHOP



After recreating the versions and verifying data, IREA’s versioning lineage looked like this:



Close up shot:



It truly is a nice thing to have your versioning lineage as displayed above; normally you can't even find where SDE.Default is within this view.

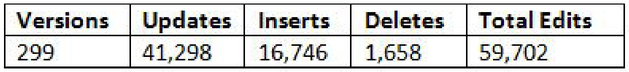
Now that you have a general idea of how the whole process works, there are probably a few questions popping up in your head: How long did it take to run? Was it a few hours, or days? How long was production shut down? etc., etc.

Production was shut down only for a single day (Friday) and the process took us 2 ½ days to get the whole project completed - including the re-projection which was time-consuming as well:

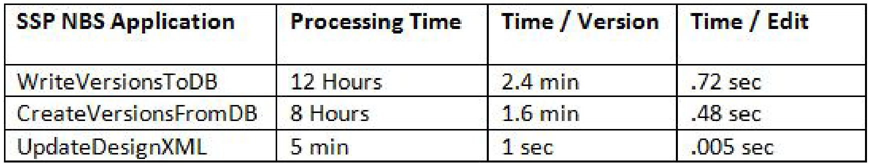
- *Thursday afternoon – Preparation time. Installing code, updating config files and kicking off the first SSP NBS application "WriteVersionsToDB".*
- *Friday – Deleting versions, relationships, dropping network, un-versioning GIS Data, re-projecting Electric and Land Dataset, rebuilding the database (recreate relationships, network, version GIS Data) and kicking off the second SSP NBS application "CreateVersionsFromDB".*
- *Saturday – Validating all versions were recreated successfully, running the final SSP NBS application "UpdateDesignXML" to update each design's XML with the new object ids and full system testing.*

The following statistics were gathered from running the test process in our office with a total 299 versions and from running the production process at IREA's site with a total 242 versions.

SSP'S OFFICE

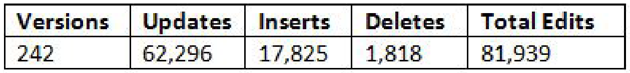


Versions	Updates	Inserts	Deletes	Total Edits
299	41,298	16,746	1,658	59,702

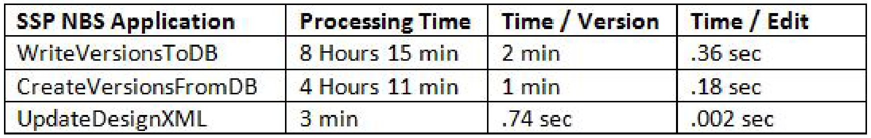


SSP NBS Application	Processing Time	Time / Version	Time / Edit
WriteVersionsToDB	12 Hours	2.4 min	.72 sec
CreateVersionsFromDB	8 Hours	1.6 min	.48 sec
UpdateDesignXML	5 min	1 sec	.005 sec

IREA'S OFFICE



Versions	Updates	Inserts	Deletes	Total Edits
242	62,296	17,825	1,818	81,939



SSP NBS Application	Processing Time	Time / Version	Time / Edit
WriteVersionsToDB	8 Hours 15 min	2 min	.36 sec
CreateVersionsFromDB	4 Hours 11 min	1 min	.18 sec
UpdateDesignXML	3 min	.74 sec	.002 sec

The difference in timing between our office and IREA are mainly due to system infrastructure (i.e. memory, processors, etc.) but I hope it gives you an idea as to what to expect if/when you consider getting to State 0 without losing any of your versions. You should be able to use your version- and/or edit-count from your geodatabase to come up with some approximate processing times within your environment.

The processing time at IREA took about 12 hours and 29 minutes (give or take) for 242 versions, but what if you have thousands of versions? Would you need to shut down production for more than couple of days? The answer is no - our solution is ***scalable***. We recently made some changes so that you can process a range of versions within a multi-threaded implementation of the SSP NBS.

FOR EXAMPLE:
Let's say you have 3,000 versions. We would set up 3 SSP NBS processes to run concurrently, each processing 1000 versions at a time.

- *NBS1 will process versions 1-1,000.*
- *NBS2 will process versions 1,001-2,000.*
- *NBS3 will process versions 2,001-3,000.*

This will decrease the time it takes to process the entire geodatabase by splitting up the workload. The processing time may go up slightly per each edit/version because there is only a single geodatabase supporting the multiple processes, but this approach will tremendously reduce the overall down time within your production environment.

Additional NBS processes can be added as long as you have the hardware and database resources to support running the processes concurrently (i.e., additional application servers and a well-performing geodatabase).

WHAT DID THIS PROJECT ACCOMPLISH FOR IREA, AND HOW DO THEY FEEL ABOUT THE PROJECT?

We asked IREA's Duane Holt for some feedback for this article and here is what he sent us:

The AEZERO project provided IREA the ability to reach State 0 and re-project the geodatabase to a NAD83 coordinate system. Without going through this process, the antiquated NAD27 geodatabase would be restricted by "on-the-fly" projection limitations that are present in Web Mapping Services.

Completing the AEZERO project allows IREA to progress into the future of GIS and deploy more web GIS technologies such as ArcGIS Online. Successful completion of the project instills confidence that the same process can be repeated if necessary to reach State 0 again.

The AEZERO project employed a process that utilized two of SSP's out-of-the-box products with a little customization and specific configuration. The thorough development and testing completed by SSP and IREA resulted in a succinct step-by-step process. Much like following a recipe to bake a cake, the AEZERO project at IREA stepped through exactly as planned in shorter time than expected.

Working side by side with SSP developer Dennise Ramirez, IREA's Duane Holt was able to work through the project and at the same time gain the knowledge and confidence that the two companies had created a scalable, reusable method to reach State 0 in a versioned database without losing the versions; a far-reaching idea for any GIS with a versioned SDE geodatabase. I am continually satisfied and surprised at how easy project rollouts have been working with SSP Innovations.

CONCLUSION

There you have it, reaching State 0 is no longer just a theory; the issue is now a thing of the past. *You can get to State 0 without losing your versions*; just leave the magic to us.

If you are wondering whether or not we hosted the celebration party after the project was completed... we sure did!

An All Edits and State 0 Combo Platter

At a recent event, GIS colleagues were discussing their latest work activities and project challenges. During the conversation, someone asked, "How will you do that?", referring to a recent SSP project venture that entailed a significantly large production geodatabase plus a massive multi-year data improvement program... soon enough, SSP came up with a solution that would combine the functionality of two of our most successful solutions: All Edits and State 0. This writeup describes the opportunity.

SSP is part of a talented multi-faceted project team brought together by a valued utility client. The over-arching mission is to carve out a process and solution that allows for a significant, multi-year utility GIS data improvement program to take place, all the while having the utility's production geodatabase keep humming along by executing and posting data transactions over the course of the data improvement program.

You could be wondering..."how can a third party company be making edits to the production database while the client is making edits at the same time?"

On the surface of it all, the requisite component elements of the program were defined, quantified and procedurally planned:

- *Source data for the utility data improvement program identified, gathered and prepped – check*
- *Established vendor to apply source data to existing production GIS database – check*
- *Data processing and technical specifications for data improvement program – check*
- *Project administration and organization established – check*
- *Enthusiasm to elevate utility data to the next order of completeness and quality – check*

A crucial remaining element to be understood, defined and fulfilled was the need for a solution that would gather the data packages provided by the data processing vendor and incorporate those into the production geodatabase with minimal disruption to existing data management operations. SSP's role is to provide for that "copy" from working database and "paste" to production database.

Would it be as easy as a ctrl+c and crtl+p? You must be wondering... The answer is Yes and No. There are some conditions that we must adhere to and also some core assumptions:

The data improvement program will process massive amounts of geospatial and attribute data over a wide and far-ranging service area; a solution will need to operate in a batch-type mode and be repeatable, able to handle hundreds of discrete data package deliverables over multiple years given the quantity of work.

For the program, the utility's Service Area will be divided into "Working Areas" to localize, manage and contain data processing for the scheduled cycles. Project coordination will be used to separate work among the data vendor and the utility's day-to-day data maintenance operations based on geography to minimize overlap and conflicts.

The utility GIS data model will need to be frozen over the course of the program.

The processed data, at some point in its journey, will need to account for and ensure that existing automated business rules such as ArcFM™ auto-updaters will be executed within the process lifecycle.

Existing production data interfaces among internal business systems must be accommodated for when bringing the processed data back into the production environment.

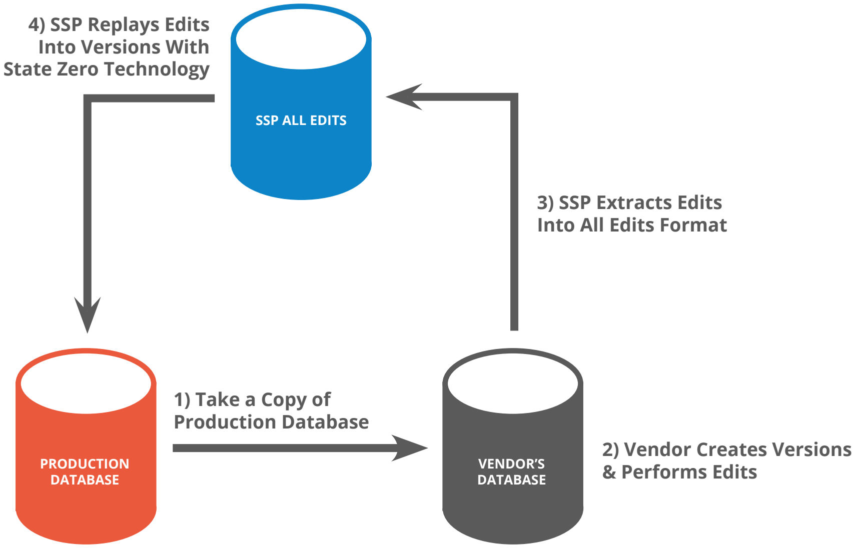
Enough of the background information! Here is how we will enable the crtl+c and crtl+p functionality:

The approach will leverage core software stack capabilities such as the Esri versioned geodatabase framework – the data packages will be delivered, transferred and applied to the target geodatabase via geodatabase versions.

Also key to the solution will be existing tools and applications built by SSP overtime and further extended. Primary among these applications will be the SSP All Edits Reporting and QA Tool, the NBS application framework (Nightly Batch Suite) and functionality from State 0.

The All Edits Reporting / QA tool has been evolving over a number of years and based on this particular effort, will now have the ability to replay edits from one geodatabase (the data vendors working database) and apply those edits into another geodatabase (the utility's production database).

THE FINAL ALL EDITS / STATE 0 COMBINATION


4) SSP Replays Edits Into Versions With State Zero Technology
SSP ALL EDITS
3) SSP Extracts Edits Into All Edits Format
PRODUCTION DATABASE
1) Take a Copy of Production Database
VENDOR'S DATABASE
2) Vendor Creates Versions & Performs Edits

SSP will utilize the All Edits extraction engine to systematically extract all of the edits from specified versions as provided by the data processing vendor. The extracted data will be written to the All Edits schema for both spatial and attribute data, and from there will be replayed into versions created in the utility production geodatabase.

The SSP NBS application framework will provide for the requirements pertaining to version creation, the batch execution of the extraction engine schedule, providing notifications of status and completion, and other functions.

You see Business Challenges. We See GIS Solutions.

Call us to discuss your situation, and we will help you think through your needs and uncover some surprising possibilities. It's simple, informal, and may even reframe your thinking.

Talk to Dean Perry at 720.229.0227