Versioning Workflows

An ESRI® Technical Paper • January 2004

This technical paper has been taken from a forthcoming book, *Inside a Geodatabase*, due to be released with the 9.1 release of ArcGIS. The book is aimed primarily at GIS managers and data administrators who are responsible for the installation, design, and day-to-day management of a geodatabase.

In publishing this in advance of the general book release, there's still time to include any feedback on it—content, layout, suitability for the intended audience, and so on. Please send any comments or suggestions to jclark@esriscot.co.uk.

ESRI 380 New York St., Redlands, CA 92373-8100, USA • TEL 909-793-2853 • FAX 909-793-5953 • E-MAIL info@esri.com • WEB www.esri.com

Versioning Workflows

An ESRI Technical Paper

Contents

Versioning Workflows

Versioning supports a variety of workflow processes. This technical paper discusses versioning and the workflow process, reconcile workflow options, and data loading and updating.

Versioning and the workflow process

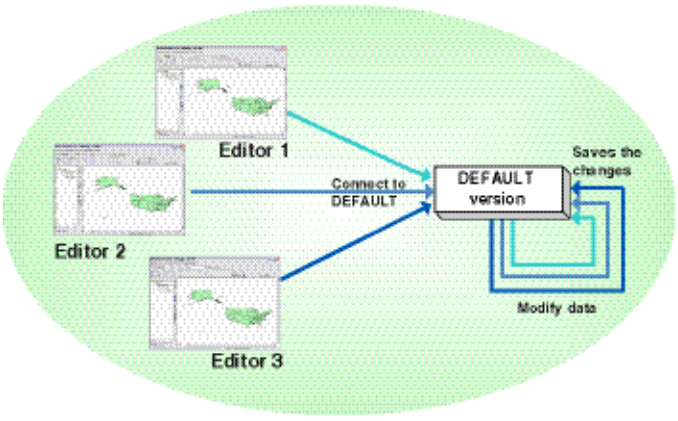
Workflow processes vary greatly from organization to organization and may span days, months, and possibly even years. Irrespective of these organizational differences, a common requirement is the uninterrupted availability of a centrally located, corporate database to support daily operations. This level of availability has to be achieved without duplicating the data and without applying prohibitively restrictive data locks. Versioning offers the type of flexible data management framework that is required to meet these demands and can accommodate the most rudimentary workflow processes as well as the most complex.

Workflows generally progress in discrete stages, with each stage often requiring the allocation of a different set of resources or business rules to be enforced. Typically, each stage in the overall process represents a discrete unit of work, such as a work order. To manage these work units within a common administrative framework, each stage can be associated with a named geodatabase version.

The next section will review the application of versioning—how this technology may be applied within an organization and illustrate the different versioning strategies that are available to GIS project and data managers to manage spatial data in a multiuser environment.

Direct editing of the DEFAULT version

The simplest approach to supporting multiuser access to a versioned geodatabase is for many editors to directly edit the DEFAULT version.

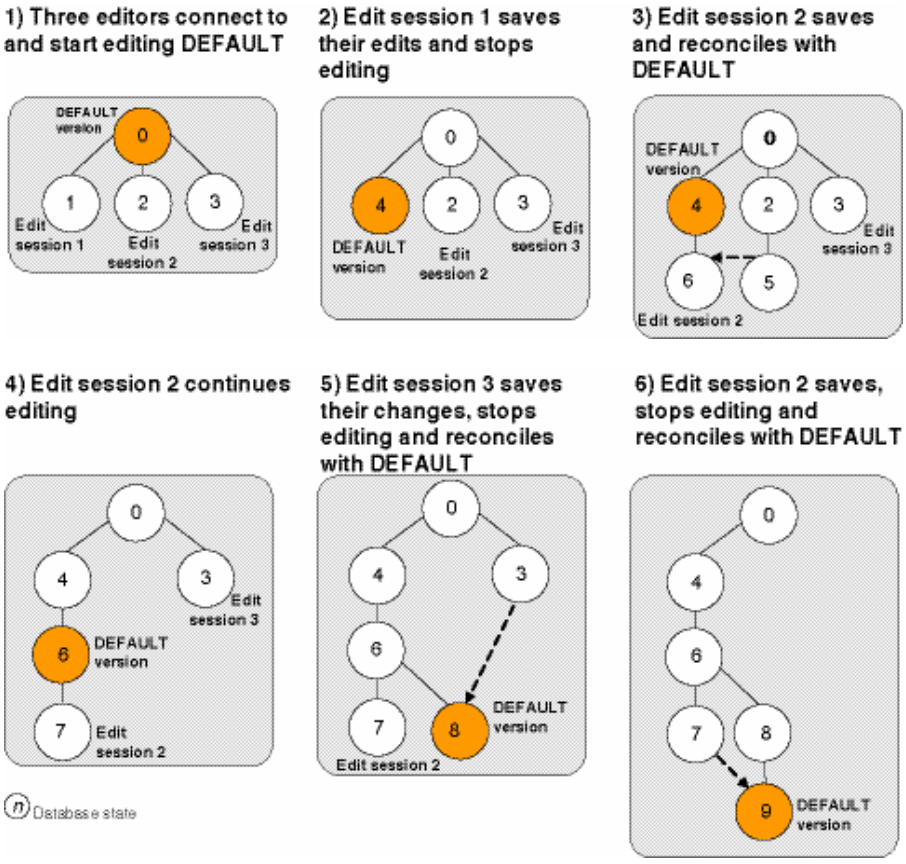


As each editor opens the DEFAULT version for editing, a new, unnamed, temporary version is automatically created—editors do not have to explicitly create a new version.

This temporary version is accessible only to the current editor, and it automatically evolves through the succession of states created by each new edit operation. When the editor saves their work or ends the edit session, this temporary version is automatically reconciled with and posted to the default version.

If another editor has edited DEFAULT since the current editor started editing, when the other editor saves their changes they receive notification that the version has been altered since their edit session began. Their changes must be saved again to accept the results of the automatic reconcile operation that has just taken place. This automatic reconcile notification may be bypassed if required.

The multiple, concurrent edit sessions on the DEFAULT version have the following effect on the state tree.



Pros

- Simplicity: this versioning workflow is probably most applicable to situations where the units of work are fairly small or where persistent design alternatives are not required.
- If no conflicts are detected, the edits are directly posted to the DEFAULT version without user intervention.

Cons

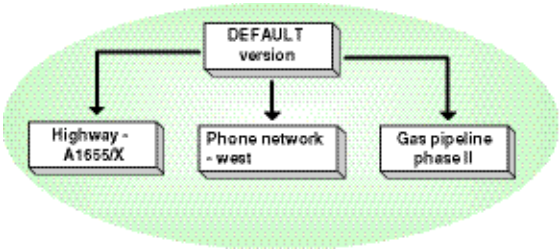
- The DEFAULT version is constantly changing and is vulnerable to inadvertent or malicious modification. Specific action is required on the part of the database administrator to preserve a historical record of the changes made to the DEFAULT version.
- This workflow does not support long transactions, which typically span many edit sessions, or the creation of alternative design versions.
- There are some performance and workflow issues associated with this approach—if a number of editors save and reconcile their changes to the DEFAULT version at the same time, other editors may notice a degradation in response times for their own save requests. The ArcGIS® application software automatically detects changes to the current edit version and will automatically reconcile and post if the version has been modified by another edit session. This is required for data and version consistency.

  As version reconcile is a serial process—that is, only one reconcile operation per geodatabase can be active at any given time—these frequent reconcile and posts in every edit session can have a negative impact on performance and response times. Editors may also receive a warning that the target version has changed since they started their edit session; they must save again to accept the changes made by other editors and may have to wait for any active reconcile and post operations to complete before they will be able to complete their own.

  In a large, multieditor enterprise geodatabase, it is better to avoid situations where many users reconcile and post to a common parent. Reconcile and post exclusively locks the parent version—while this lock is in place, this will prevent other users from completing their tasks.
- As all reconcile operations are undertaken automatically, this approach does not support batch reconcile/post processes. Batch version reconcile is covered in greater detail later in this paper.

Two-tier version tree

The two-tier version hierarchy is a more common implementation of versioning as it supports a more structured approach to workflow management. Discrete work units related to specific projects, work on which may involve many edit sessions typically spanning a number of days, weeks, or in some cases months, can be maintained without affecting the DEFAULT version. Examples of these discrete work units could be a highway improvement scheme, the installation of a new phone service, or an ongoing maintenance project for a gas pipeline.



When a work order or project is initiated, a version is created as a child of the DEFAULT version. One or more editors will work on this version until the project is complete. When all the modifications to each new project version have been completed, the version

is reconciled with and posted to the DEFAULT version, integrating the modifications into the published database. At this point each work order version can be removed or archived as required.

User access permissions to the DEFAULT version may be restricted to enforce this workflow and ensure that the DEFAULT version is not modified. The SDE administrator may set the permission of the DEFAULT version to "protected"; this allows users to continue to view the DEFAULT version but restricts their access level to read-only. Any editor wishing to modify the data must create a new version of their own.

When an editor has finished modifying the data, they or the SDE administrator can reconcile and post the version to the DEFAULT version. If conflicts are detected, they must be resolved in the usual way and the changes saved again during the edit session. The editor's version can then be deleted as required.

Pros

- Simplicity: each work unit is logically segregated in the geodatabase.
- Supports long transactions, spanning many edit sessions, and the creation of alternative designs. This allows editors to develop proposals without affecting the production database.
- Creating a new version from the DEFAULT version protects the production view of the database from unintentional modification—individual work projects are integrated with the production database when completed.
- Supports batch reconcile/post processes.

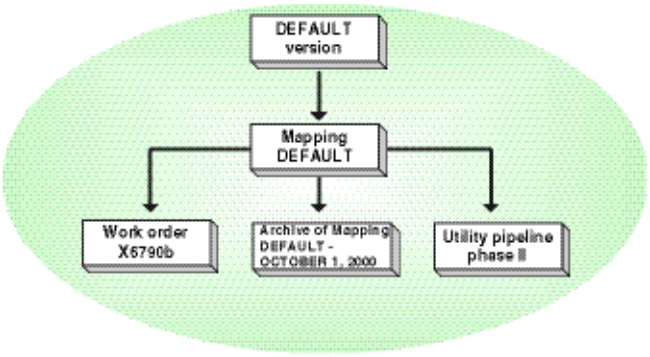
Cons

- As with any multitier version configuration, the more rows that are maintained in the version delta tables, the greater the potential impact on version query performance. This overhead can be minimized by compressing the database regularly and updating the database management system (DBMS) statistics.

Surrogate DEFAULT version

A variation on the two-tier approach is to use a surrogate DEFAULT version as a reconcile and post target for all other versions. This is another way of protecting the DEFAULT version from direct editing and unauthorized or unintentional modification. A new named version, in this case called Mapping_DEFAULT, is created as a child of the DEFAULT Version; all other versions are then created from this new version as required.

This isolates the changes made to the published version of the geodatabase until such times as it is considered appropriate to integrate these changes. A database administrator or project supervisor can reconcile, resolve any conflicts, and post the Mapping_DEFAULT version to the DEFAULT version without affecting other database users.



Pros

- Simplicity: easy to set up and supports persistent design alternatives and/or historical snapshots of the database as required.
- Safe: prevents accidental or unauthorized modifications to the published database.

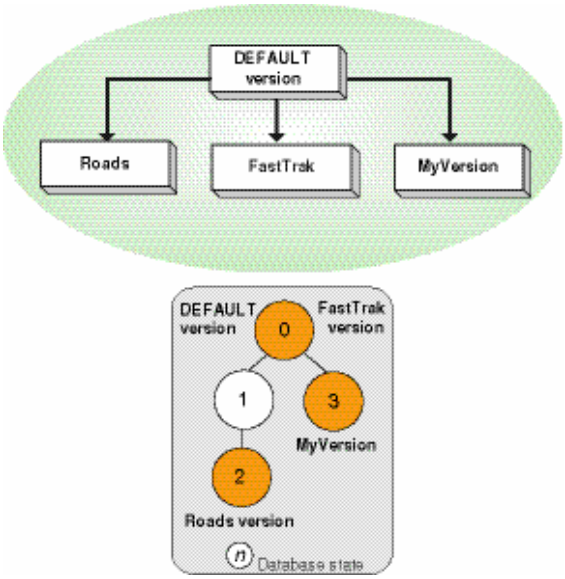
Cons

- Only suitable for smaller applications where the number of edits will be limited for the following reasons:
  - If there are many versions created from this Mapping_DEFAULT version, the reconcile and post of all these versions may take time to complete.
  - If a large number of edits is posted to Mapping_DEFAULT, the overhead of auditing all the changes in that version may prove impractical.
  - Reconciling a large number of changes between the Mapping_DEFAULT and DEFAULT version may also take time to complete.

Fast-track versions

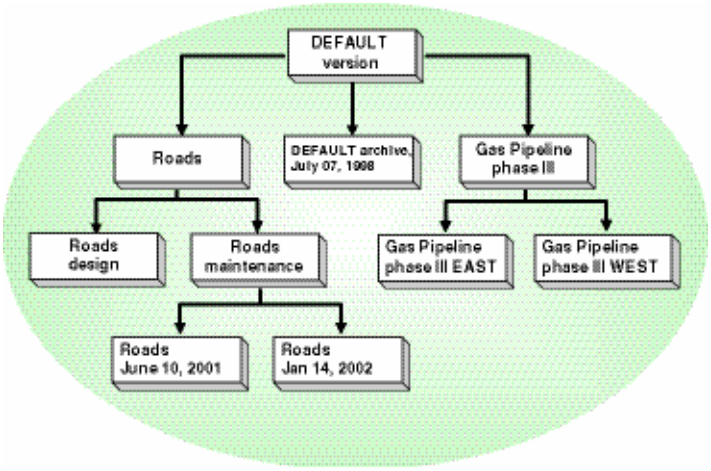
A further adaptation of the two-tier approach is to create a static, fast-track version to support rapid querying and reporting operations in situations where the most recent changes are not required. This version should be created from a compressed database where the state tree had been returned to 0. As this read-only version would continue to point to state 0, all the rows required to represent that version would be stored in the base table. As no version difference queries have to be performed to access this version, this ensures the best performance for query operations.

After each scheduled database compress, this version would be re-created as required.



Multiple-tier version tree

More complex projects will require a more elaborate workflow structure than that provided by either the direct editing or the two-tier approach. These projects may be further divided into multiple functional or geographic units from which a more complex versioning hierarchy will develop. For example, a project to design and construct a new shopping mall might have distinct construction phases—subdivided into east and west sections or subdivided by construction activities, such as building, gas, water or electricity installations, or landscaping.



For larger projects that will involve many editors, often working on different teams and on numerous discrete units of work, a multiple-tier version tree is an effective way to organize the workflow. The teams working on different aspects of the same project create their own version to maintain a private view of their updates. Once the project has been completed, the versions can be reconciled and posted back to the DEFAULT version and become an integral part of the published database.

Pros

- Supports complex projects.
- Supports long transactions, which span many edit sessions.
- Supports automated batch reconcile and post processes.

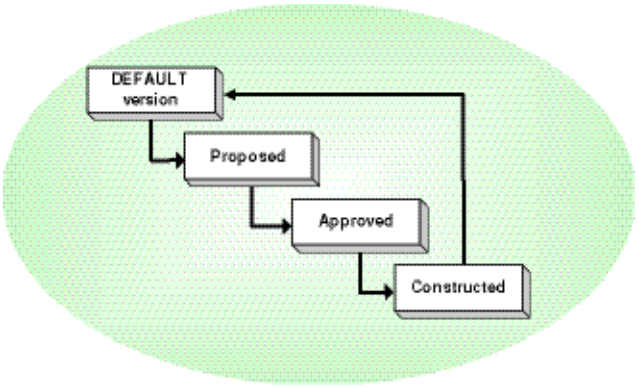
Cons

- First-level versions cannot be posted to the DEFAULT version until the second- and third-level versions have been posted to their respective parents.
- Reconciling and posting can only take place between versions in the direct path—it is not possible to reconcile and post across version paths.
- Maintaining a complex version tree has some associated performance costs—the more rows in the version delta tables, the greater the potential impact on query performance.

Cyclical version tree

Many projects evolve through a prescribed or regulated group of stages that require engineering, administrative, or legal approval before proceeding to the next stage. For example, within the utility domain common project stages include working, proposed, accepted, construction, and as built. This particular process is essentially cyclical—a work order is initially assigned to an engineer and modified over time as the project evolves through the various stages before full integration with the production database.

In this approach, a version is created to represent each stage of this process—initial design or proposed version, an approved version, and a version for the construction phase. As the project advances through the various project milestones, each stage is reviewed and approved, then superseded by the next version until the last stage is reached and completed. The older versions may be maintained for historical reference or deleted as required.



Once the project is complete, the constructed version can be reconcile with and posted directly to the DEFAULT version, without having to reconcile and post with the previous versions in the lineage.

Pros

- Suitable for projects that evolve through a series of stages, where each stage must be isolated as a distinct unit of work.
- As with all other multiple-tier configurations, this workflow allows editors to develop proposals and design alternatives without affecting the production database.

- Changes can be posted directly to DEFAULT, which eliminates the overhead of progressively posting changes up the version tree to the DEFAULT version.

Cons

- Not suitable for batch reconcile and post processes as any implementation would require a great deal of application logic to determine which versions to reconcile and post.

Managing historical versions

A key requirement for many projects is the preservation of a version that reflects some historic state of the project at a given point in time. A versioned geodatabase can be used to manage history at both the database and the individual feature level as all the information required to represent the state of the database at any given point in time is available and can be archived as required.

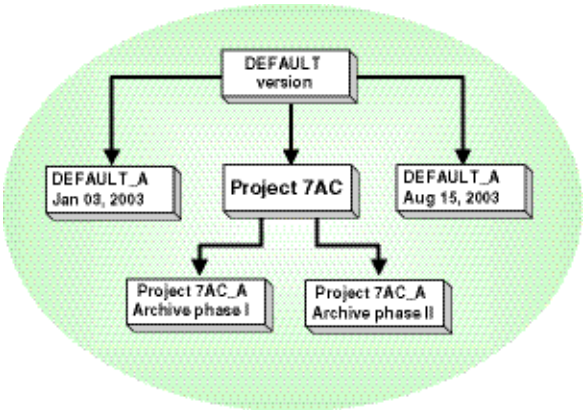
These historical snapshots of the database are based on change events—a change event is any database action that moves the database from one state to another; adding a new feature, dropping a table, or modifying an attribute are all examples of change events. The temporal granularity of a geodatabase, or the frequency of modifications made to the database, is determined by how often these change events occur and are recorded.

In a geodatabase, changes to the database can occur as a result of a long transaction, a collection of individual database actions or short transactions. The frequency of recording these transactions will be determined by the individual requirements of each organization or application. For some, every change to the database must be preserved in a historical record; for others, a less frequent archiving regime will suffice.

Some of the typical historical queries that a versioned geodatabase may have to support include:

- What was the state of the database at a given time? How frequently historical versions are captured will determine how accurate the information returned by these queries will be.
- How has a particular feature changed over time? This is generally referred to as a lineage search. As with the previous query, the level of information that will be returned by the query will be determined by how frequently historical records were captured.
- Given that an object or feature was removed from the database at a certain date, what features currently exist where the deleted feature used to be? This type of query forms the basis of a comparison between historical and current feature configurations.

A common requirement for maintaining a historical record is to preserve an archive of the DEFAULT version, although historical versions can be created from any versions. As an example, to preserve a record of a project at any stage in the project life cycle, a new historical version could be created from the project version itself. When the project version was reconciled and posted to its parent version, the project archive would remain as a record of the project at a particular stage.



Pros

- As with any other version, when the historical version is captured, all aspects of the data model at that point in time are recorded, including network connectivity, relationships, topology errors, and so on.
- No additional data modeling or application customization is required to support historical versions—this functionality is intrinsic to the versioning model.

Cons

- This version configuration may affect performance in the long term—as with the other more complex version structures, the more information that is maintained in the delta tables, the longer each version difference query will take. To maintain a good level of performance in the geodatabase, at some point in the future, this historical information should be archived offline.

Version-based distributed data management

Disconnected editing

With disconnected editing as part of the workflow, organizations can exchange data between a central office and field-workers or work around some of the limitations of reduced bandwidth connections from a central server to geographically dispersed offices. Checking out or extracting data[1] is also a good opportunity to prototype designs, configurations, and alternate data models without affecting the format of the production geodatabase or other database users—for example, altering and assessing the impact of changes to the rules of a topology, altering network weights, and experimenting with schema changes.

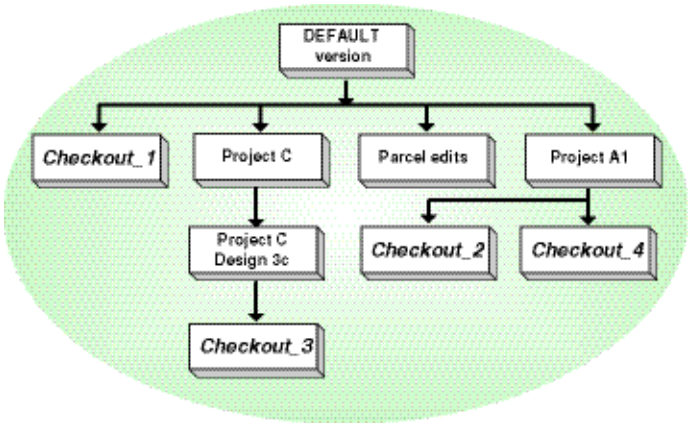
The next section will describe the disconnected editing solution for managing spatial data in a distributed environment and illustrate how versioning supports this distribution of data.

[1] Extracting data is very similar to checking out data only without any means to check in the data. This provides the capability to selectively copy one, some, or all of the records in any given dataset to another geodatabase. It also provides a mechanism for transferring geodatabase models to other sites.

Checking out data

As the process of checking out data is based on geodatabase versioning technology, data must first be registered as versioned before it can be checked out. When a check-out is created, a new version of the data, the master check-out version, will be created in the master (or source) geodatabase as a child of the version to which the editor is currently connected. For example, if connected to the DEFAULT version, a new version with the same name as the check-out will be created as a child of DEFAULT.

This check-out version reflects the state of the data, as represented by the parent version, at the time the check-out was created and will receive the changes from a check-out geodatabase when the data is checked back in. The master geodatabase may host multiple check-outs; each check-out will have its own check-out version.

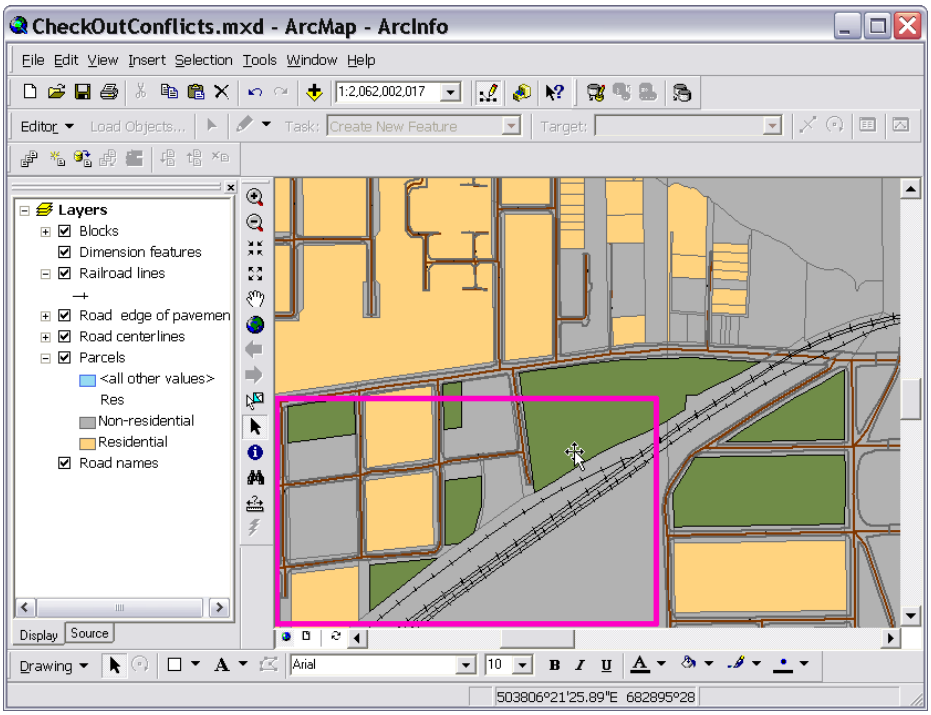


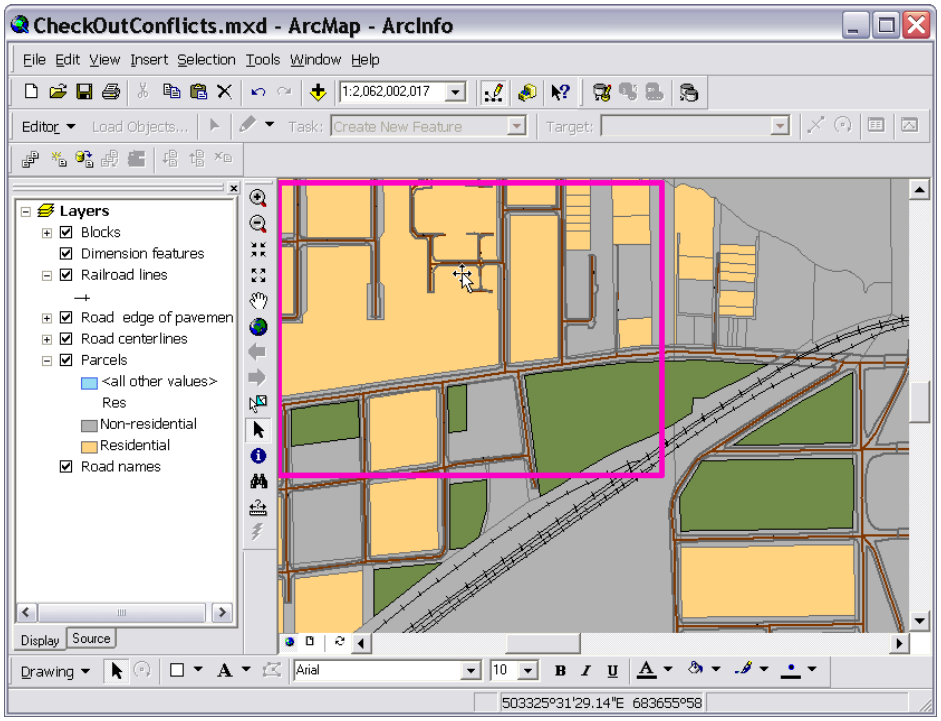
Project managers and version administrators should be aware that these versions created in the master geodatabase are created like any other version in the geodatabase—they are not hidden from other geodatabase users, and they have the same three levels of user access as other versions (private, public, and protected). By default, these check-out versions are created with public access so other database users could directly edit or create new child versions from these versions. However, they should not be edited while the check-outs they represent are still active. Any changes made to these versions may be overwritten during the check-in phase.

Applying an appropriate level of user access and adopting an intuitive naming convention for all check-out versions, if the default name is unsuitable, will help avoid any contention between the many editors and workflow processes a versioned geodatabase will support.

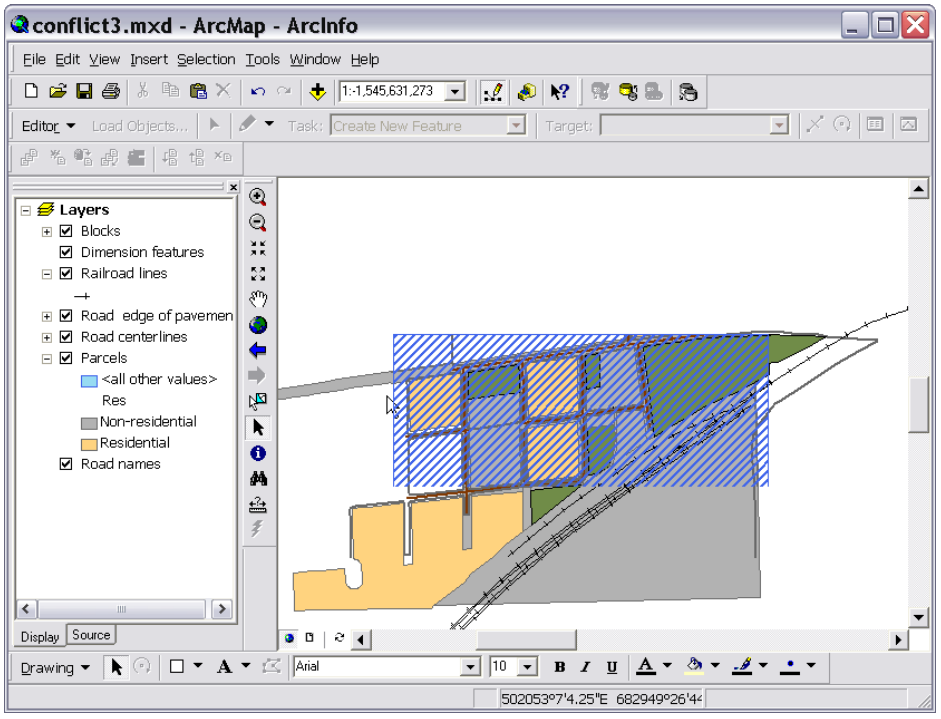
Another important pre-check-out consideration for project managers is the physical segregation of work units that will be distributed to field or remote users. At the time the check-out is created, there are no warnings to advise if features from one version have already been checked out. If overlapping areas are subsequently checked out, version edit conflicts may be introduced on check in.

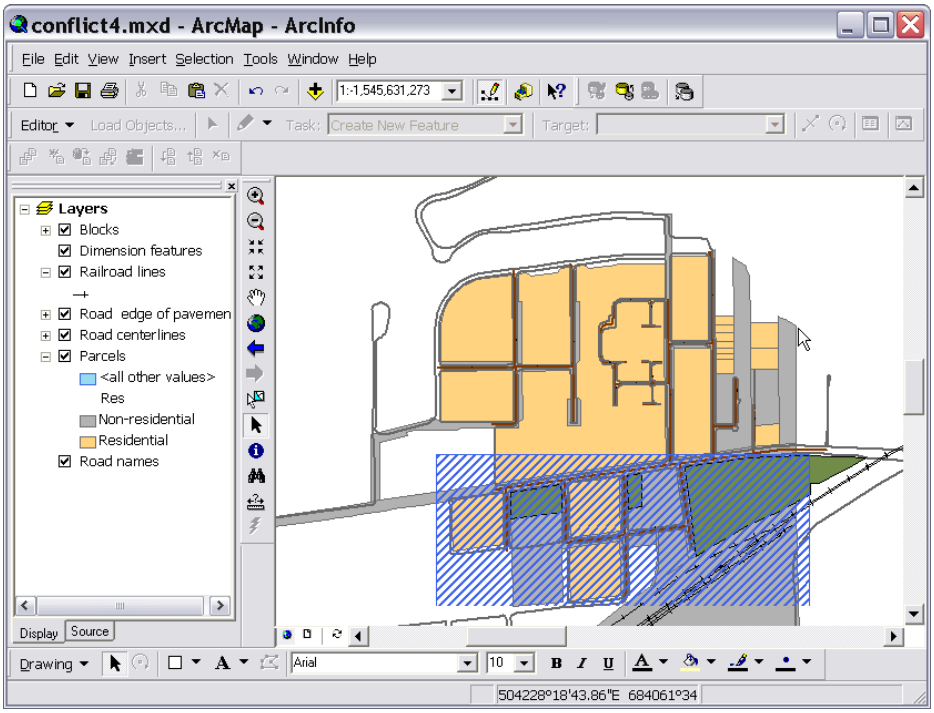
The two check-outs below are created from the same map document. For each check-out, a different but overlapping check-out extent has been used to define the check-out boundary.



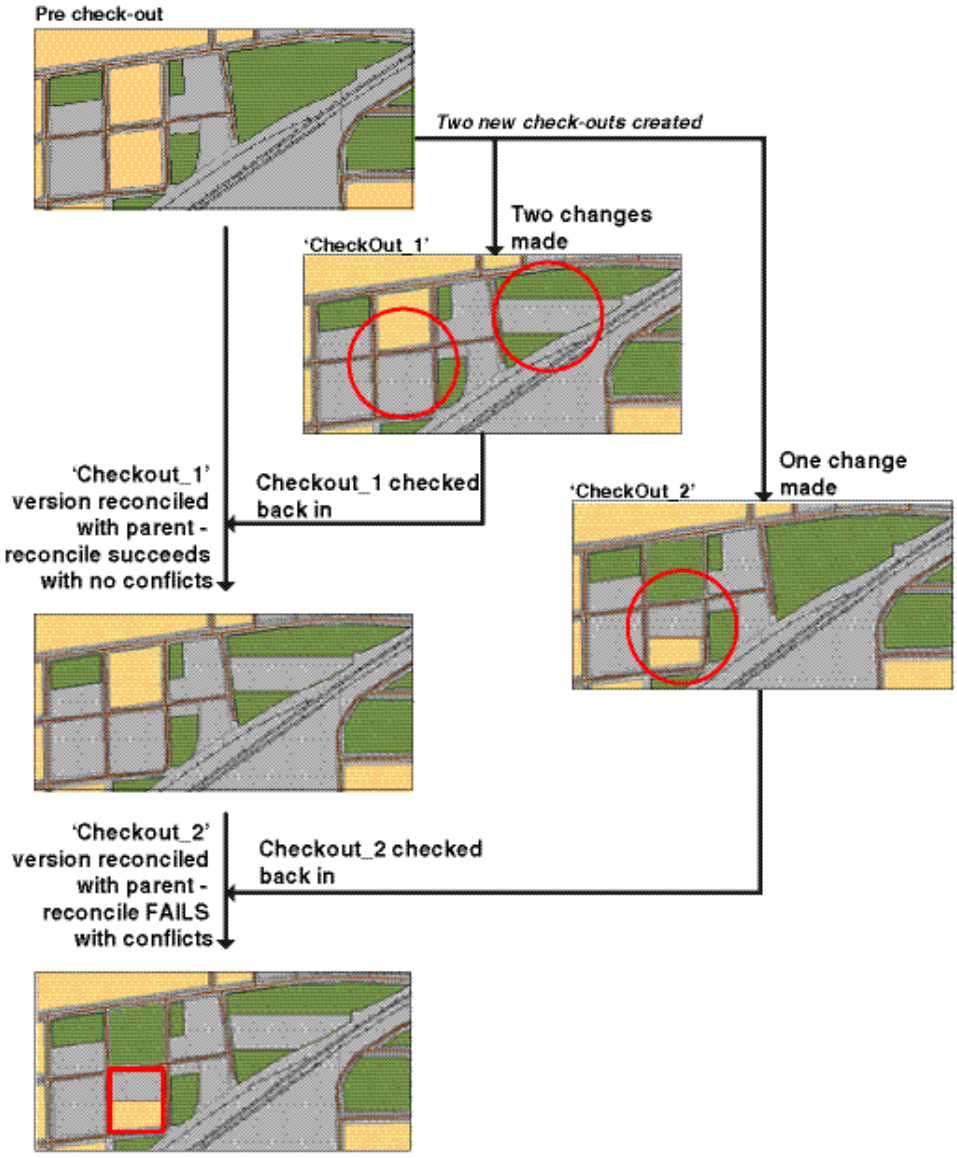


This creates the following two check-outs:



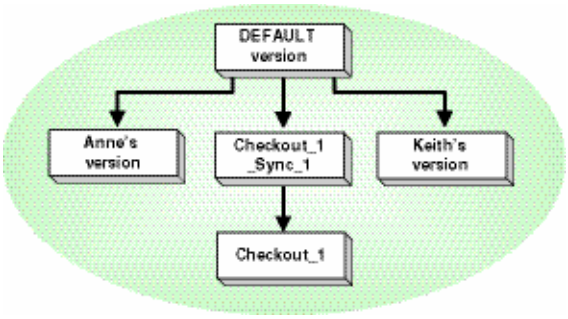


Any changes made to the features within the hatched blue overlap zone could potentially result in a version edit conflict on check in.



This can be avoided by ensuring, wherever possible, that the overlap areas are kept to a minimum.

If the target check-out geodatabase is an ArcSDE® geodatabase, then two new versions are created in the check-out geodatabase. The first version, the synchronization version, will be created as a child of the DEFAULT version. This synchronization version will reflect the state of the data at the time the check-out was created and again should not be edited. The second version, the check-out version, will be created as a child of this synchronization version. Only the edits made to this check-out version can be checked back into the master geodatabase. Once data has been checked out to an ArcSDE geodatabase, it is automatically registered as versioned.



Checking in data

Checking in the data involves transferring only the changes made to the data while it was checked out back in to the master geodatabase. These changes are identified as follows:

- ArcSDE check-out geodatabases—a comparison is made between the edited check-out version and the static synchronization version. If the synchronization version had been edited after the check-out was created, the comparison would return incorrect results and could result in the wrong data being checked back in or some data may not be checked in at all.
- Personal geodatabases—the edits are logged in a separate geodatabase system table.

The changes are transferred directly to the check-out version in the master geodatabase; there is no version reconciliation with the check-out version. If the check-out version has been modified since the data was checked out, these changes may be overwritten. By default, a check-in operation will terminate at this point—there would be minimal impact on other active database users. Once the data has been checked back in to the master geodatabase, any versions created in an ArcSDE check-out geodatabase will also be removed automatically.

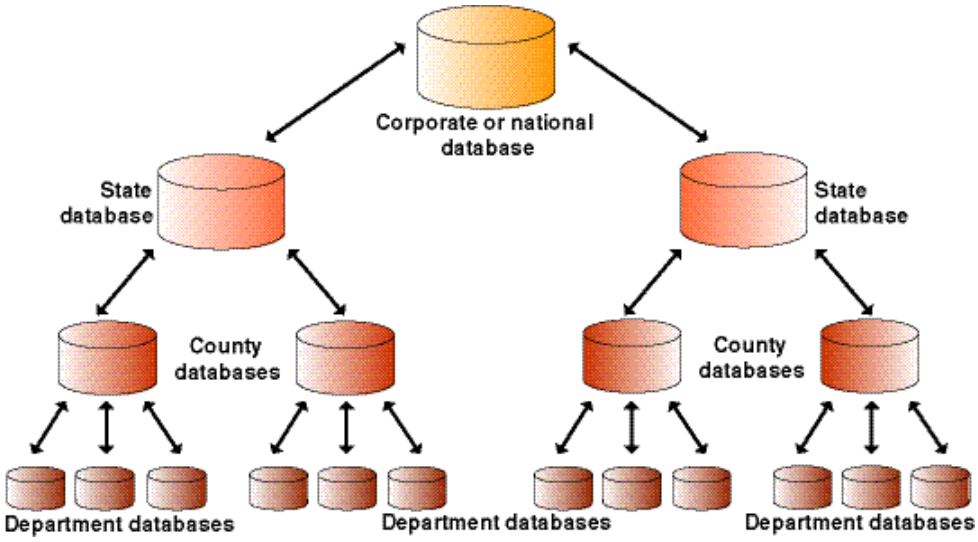
Organizational quality control measures may require that the changes be audited before the check-out version is reconciled with and the changes posted to its parent version—these checks could be implemented, as part of the general data management workflow, at this stage in the process.

If no audit is required, the check-in process may optionally involve an additional step of automatically reconciling and posting the changes in the check-out version to its parent version in the master geodatabase. If the reconcile and post operation completes with no conflicts, the check-out version is then deleted. If conflicts are detected, the process will terminate at this point. Any conflicts must be addressed in the usual manner using the conflict detection and resolution tools in ArcMap™.

Versioned geodatabase replication workflows

Versioned geodatabase replication takes disconnected editing to the next stage and adds support for bidirectional, multigeneration data synchronization between two enterprise geodatabases. Geodatabase replication provides a framework for organizations to distribute and refresh copies of data, exchanging multiple generations of changes, between geographically dispersed sites. Versioned geodatabase replication will be available with the 9.1 release of ArcGIS.

In the distributed environment, a hierarchical relationship often exists between the many offices that represent a particular organization—for example, a government institution with national, state, county, and departmental offices.



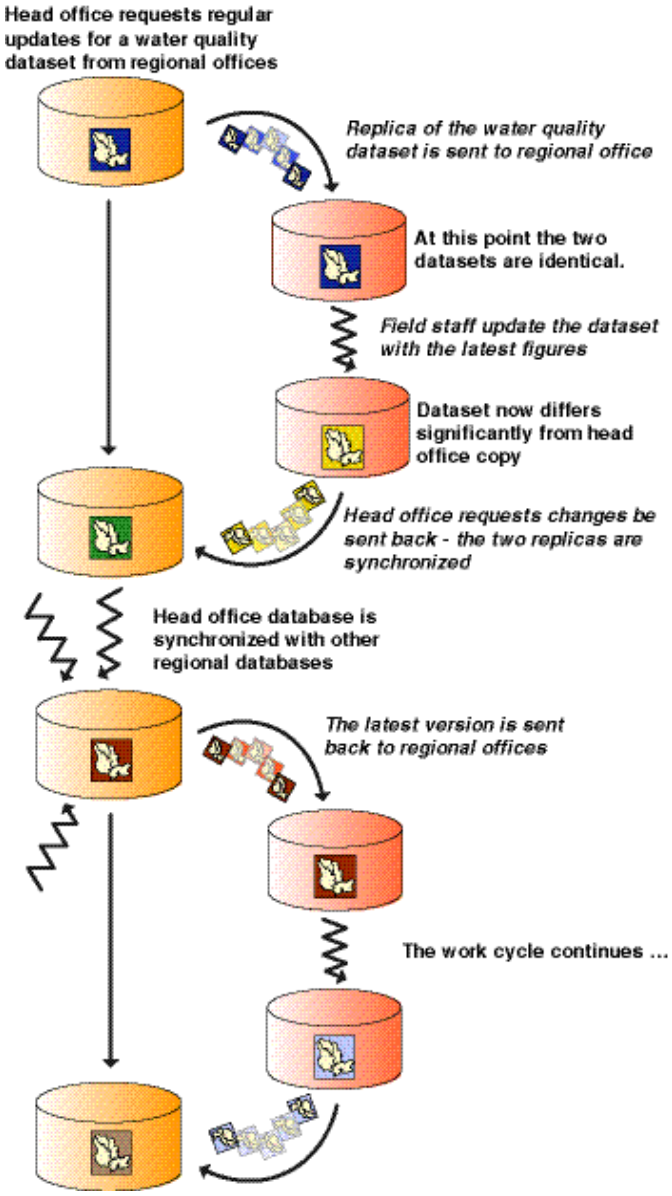
In the structure illustrated above the corporate or national database represents the root replica database, and its child replicas are represented by the county databases. The county database in turn represents a root replica for the child city databases, and so on, throughout the hierarchy.

Some of the key functional requirements for replication are:

- Each individual office needs to update its copy of the data and to synchronize their updates with the different databases in the distributed environment.
- Replicating all or part of a single centralized data repository across the different offices.
- The central office maintains all the data covering the entire geographical extent of the organization. Each regional office needs only the data that pertains to their particular geographical area or region. For example, the central office receives regular updates to one of their key data holdings from an external agency. The relevant updates must then be replicated at the appropriate regional or local office level.
- Data maintained by one of the regional or local offices may be edited at either site or the central office but cannot be modified by other regional or local offices.
- As one of the primary roles of the central office is to undertake analysis of the data, it must have access to the most current version of the data. To support this, the regional and local databases must be synchronized with the central server to integrate all changes made to the data at the regional or local offices.

Versioned geodatabase replication allows organizations to replicate geographic data across multiple versioned geodatabases. Once the replication framework is in place, each

geodatabase can then be updated independently and then synchronized with other geodatabases.



Synchronization involves exchanging only the differences between the participating geodatabases and may be carried out over multiple generations of changes.

Supporting replication relies on the creation of a number of versions, which are managed automatically by the replication process itself. These versions track the updates made to a geodatabase and identify the differences between two geodatabases that must be

synchronized. Some of the versions are created when the replica is first created; others are created and deleted as the changes are synchronized between the replica pairs.

As with other version structures, the more complex the version tree, and the greater volume of data maintained in the delta tables, the greater the potential impact on query performance. Dynamic geodatabases that support a number of disconnected editors and replication should be compressed and reanalyzed regularly to maintain an optimum state tree configuration.

Reconcile workflow options

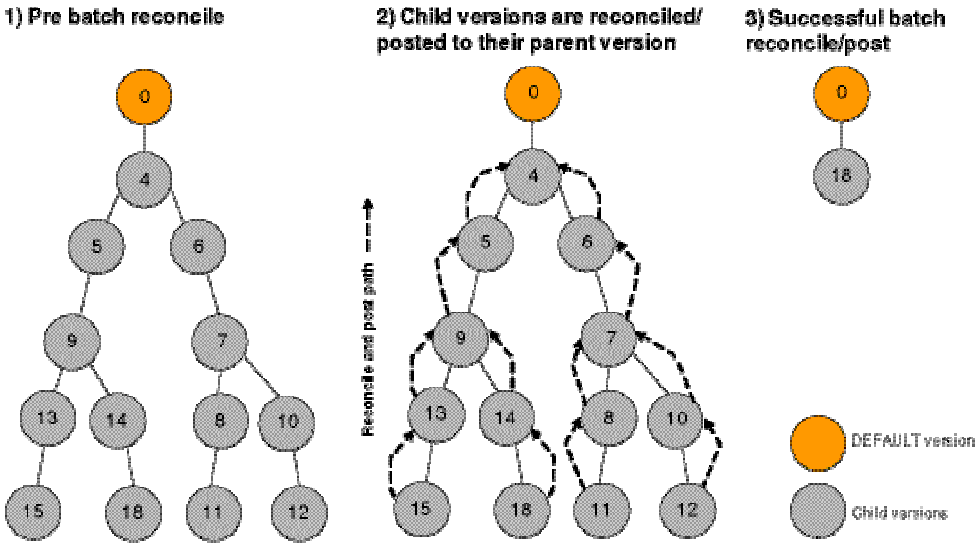
Batch version reconcile and version reconcile services

Automating the reconcile and post processes will help streamline edit operations, reduce the version administration overhead, and ensure all the changes made to versions are automatically propagated through the version tree as required. These two processes may be automated by using a batch version reconciliation tool or implementing a version reconcile service.

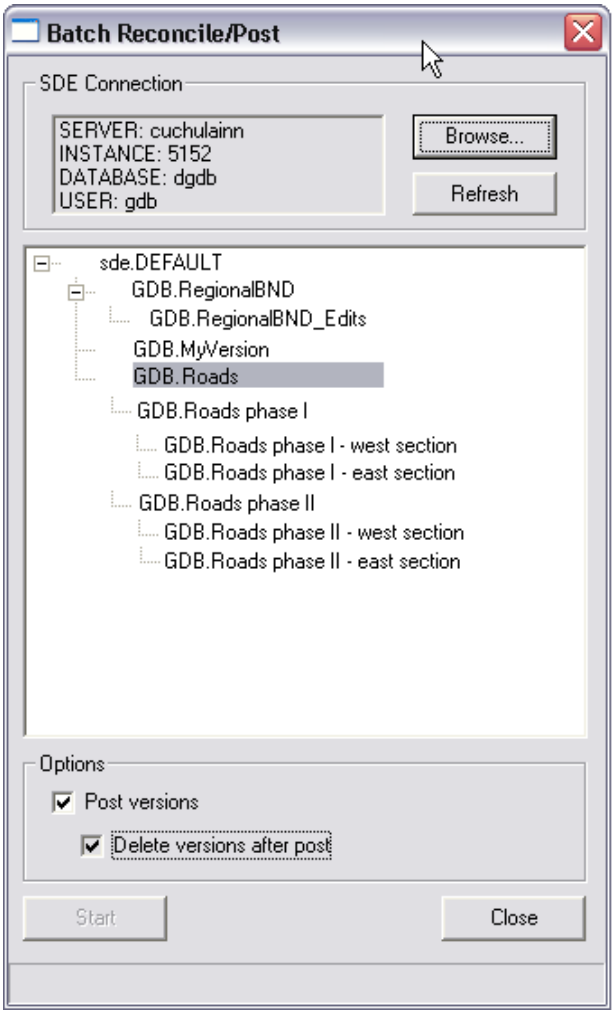
Note that both the batch reconcile and version reconcile service utilities are available as ArcGIS developer samples.

Batch reconcile

Batch version reconciliation can be implemented to recursively reconcile, and optionally post and delete, all versions in a versioned geodatabase. If the reconcile and post are successful—that is, no conflicts were detected—the child versions are deleted.



The batch reconcile utility allows an editor or project manager to connect to a versioned geodatabase, select a version, reconcile, and optionally post and delete every version beneath that version. This administrative task could be performed at the end of every day or week or at whatever time interval is appropriate to each project or application.



Any conflicts detected would halt the reconcile process; conflicts must be addressed in the usual manner using the tools available in ArcMap.

The batch reconcile utility could also be customized to start automatically at a predetermined time with some preconfigured operating parameters. Always posting when reconcile completes successfully and always deleting the version once the post has completed helps keep user intervention to a minimum.

Batch reconcile and post processes work well for organizations with work orders to process at the end of each day or week. The versioning workflows that would benefit from a batch reconcile and post program include:

- Two-tier version tree
- Surrogate DEFAULT version tree
- Multitier version tree

- Distributed data workflows; disconnected editing and replication

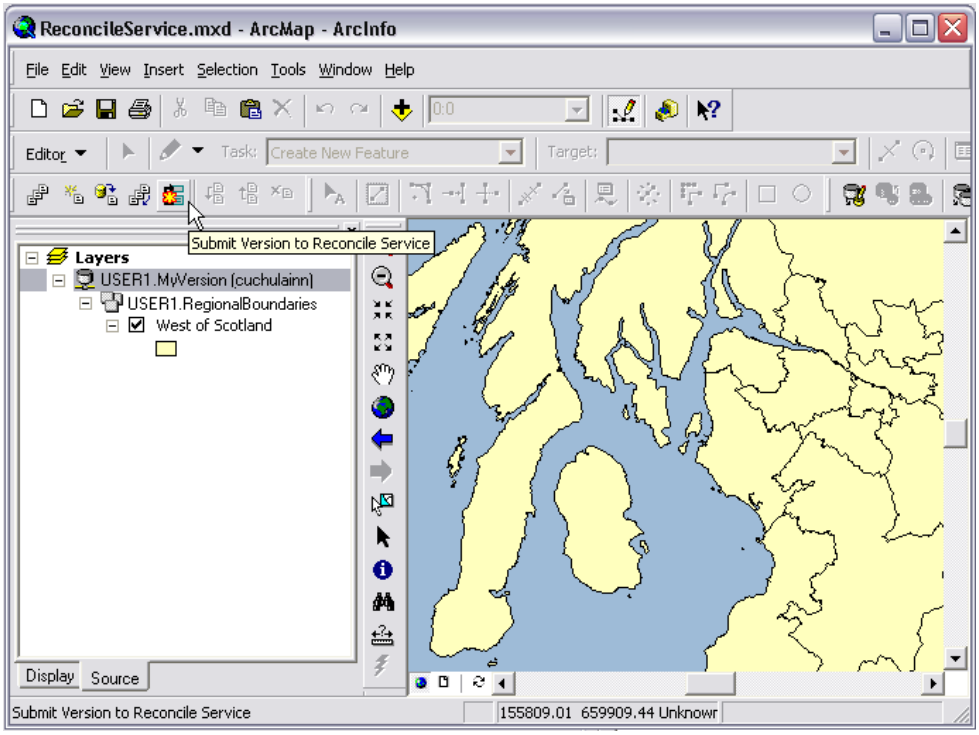
Version workflows that are unsuitable for batch reconcile and post include:

- Directly editing the DEFAULT version—as all the reconcile operations are automatic; this does not support batch reconcile and post.
- Cyclical version workflows—as this would require some specific application logic to determine which versions to reconcile and post with which parent, it is generally unsuitable for batch reconciles.

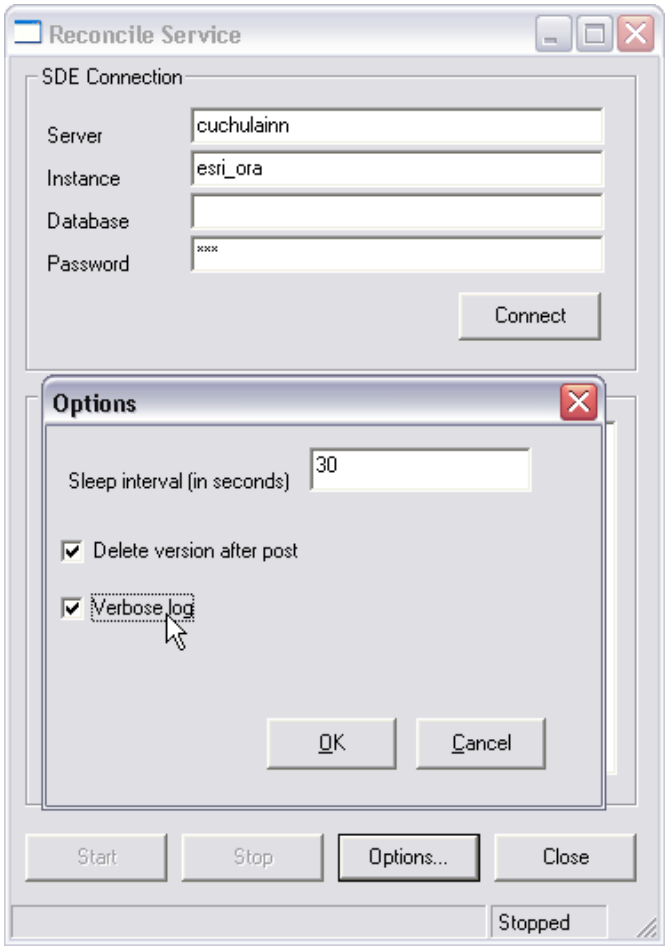
Version reconcile services

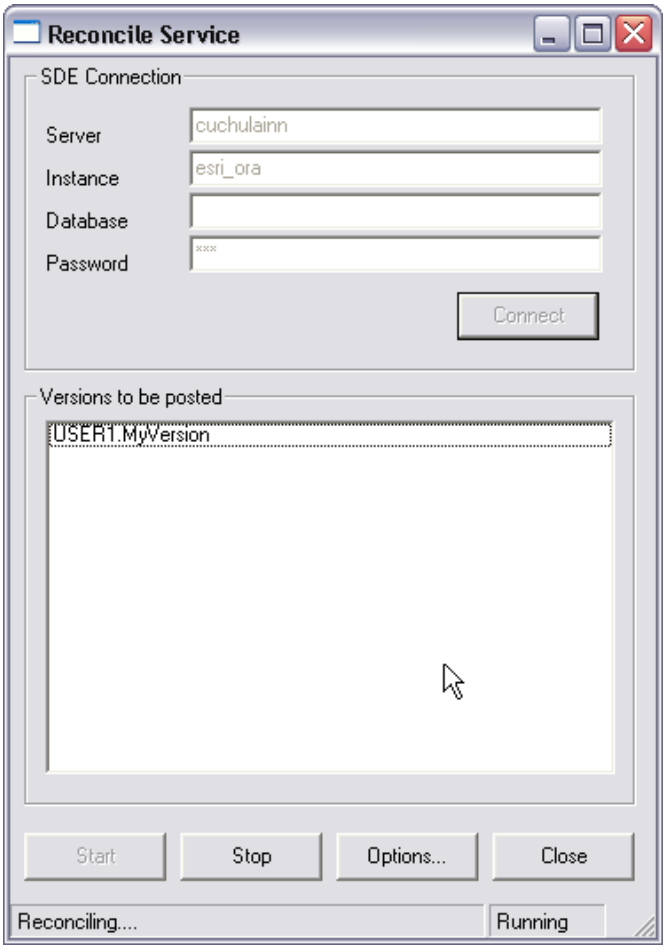
A reconcile service can be configured to periodically check to see if versions have been flagged as 'ready to reconcile'. If any such versions are detected, the service will then automatically reconcile and post these to their parent versions. Automating the version reconcile process by implementing a reconcile service helps manage the workflow more efficiently by delegating the responsibility for remembering to reconcile a version to the reconcile service. Data editors are then free to continue with other tasks. By streamlining the workflow in this manner, the geodatabase system will scale much better to support additional data and editors, performing a variety of operations.

To use a reconcile service, once an editor has completed the required modifications to the current edit version, the version is submitted to the reconcile service. This service runs as a background process on the local client machine.



The reconcile service utility can be customized to alter how often the service checks for new versions to reconcile and post and if verbose logging is required.





Versioning workflow options: choosing the right one

The final choice of which versioning workflow model to adopt will most likely be either the direct editing workflow, with many editors modifying the DEFAULT version, or some combination of the others. An understanding of organizational and business requirements and an appreciation of the pros and cons of each versioning option will influence the choice of versioning strategies to adopt. This will, in turn, have a significant impact on the overall performance and scalability of the geodatabase implementation.

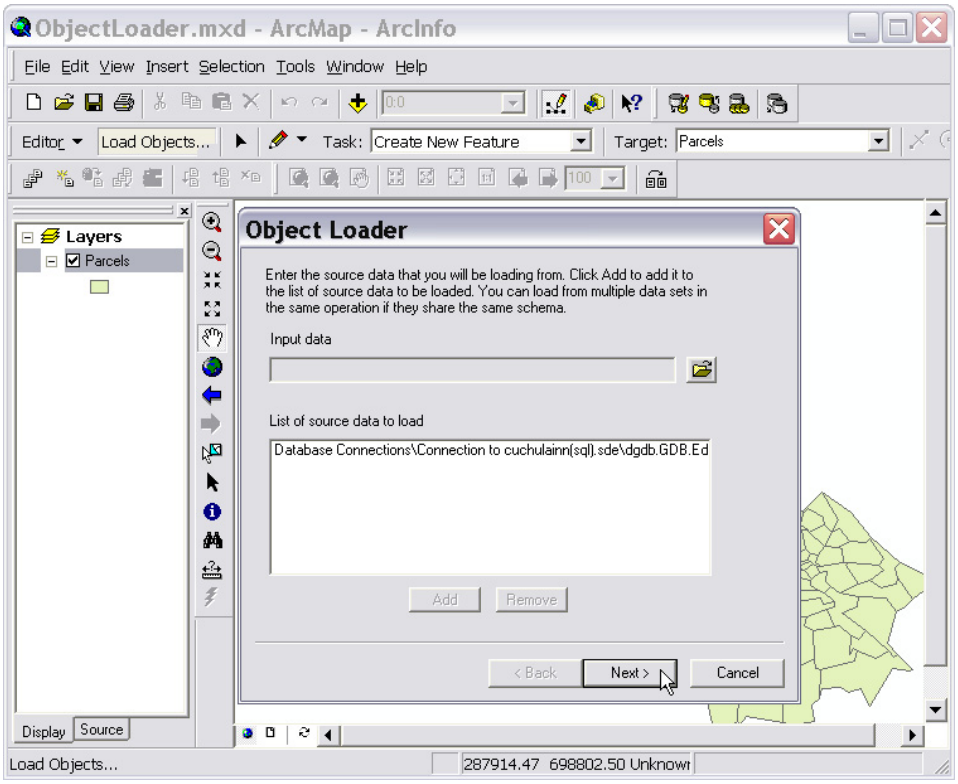
Data loading and updating

One of the key tasks for any geodatabase administrator is to populate the database with data. This is generally undertaken prior to the database becoming operational, but there will inevitably be times when the data holdings must be updated or replaced.

During the setup and initialization of a geodatabase, the most efficient way to populate a geodatabase is to use the simple data loader in ArcCatalog™ and load the data *prior* to creating any versions. Once the simple data has been loaded—that is, no networks, topologies, feature-linked annotation and so on—networks can be re-created and connectivity re-established, and topological relationships and feature dependencies can be reconstructed. The data can then be registered as versioned as required.

By loading data into an unversioned geodatabase, all the data is loaded directly into the base tables—this obviates the requirement to compress and reanalyze the geodatabase once the data has been loaded. If the data were already versioned, all the new or updated rows would be loaded into the version delta tables—the Adds and Deletes table—and as has been illustrated previously, there are some performance costs associated with maintaining a large number of rows in these delta tables.

However, although this is the recommended workflow, there will inevitably be occasions when bulk data loading or appending will be required to augment or replace the existing data holdings. For any datasets that do not form part of a network or topology, appending data is relatively straightforward—the data is first loaded using the object loader in ArcMap, the geodatabase is then compressed and the modified tables reanalyzed to update the DBMS statistics.



However, although this approach is reliable and works well for simple features, it can be time consuming to load objects such as networks and topologies, as it ensures all geodatabase object behavior is executed as the data is loaded. For every network or topology element that is added or modified, specific behavior is triggered to ensure the integrity to the data, such as moving geometrically coincident network edges, and so on. It also requires some post data loading version management to maintain acceptable performance levels.

A faster alternative for loading objects with specific geodatabase behavior would be to roll back the objects and the geodatabase to a simpler, unversioned state and then bulk load or update the data. To achieve this without losing any version edits, all child versions should be saved, reconciled, and posted to their parent version and the database compressed. At this point the version state tree has been returned to state 0, so all the child versions can be deleted to leave only the DEFAULT version, the data can be unregistered as versioned, and any networks, topologies, or relationship classes deleted. The data could now be loaded using the simple data loader in ArcCatalog. Once data loading is complete, any networks, topologies, and relationships classes can be rebuilt and the data versioned once more.

While this approach would improve the speed of data loading operations, in situations where the target geodatabase is established and operational, with version structures already in place, it is generally not feasible to delete all the versions in the version tree. It would be impractical, given constraints imposed by organizational requirements, to reconcile, post, and delete versions that may be an integral part of an active project.

This workflow would not be compatible with capturing and maintaining history in a geodatabase or supporting many distributed users, each of whom require a certain version structure to be maintained to facilitate checking out and in or replicating data. To remove all check-out and replication versions to accommodate a bulk data loading exercise while there were still active check-outs from that geodatabase would require the reconstruction of the versioning framework to support the continued distribution and synchronization of data at some point in the future.

There are alternatives to unversioning the data—for example, archiving the historical versions to an offline storage medium or scheduling the data loading activities for such times in the operational cycle of the geodatabase when the version tree is relatively flat with few changes to be posted to the DEFAULT version. Once the data has been loaded, and the versions reconciled and posted to DEFAULT, the database can be compressed to push all those recently added rows into the base tables.

The implications of loading data into a versioned geodatabase, given inherent geodatabase and version constraints, must be weighed against the operational requirements to maintain an established versioning structure. For those responsible for the design and implementation of a geodatabase, the challenge is to anticipate the data that will be required, and the usage of that data, before the system becomes operational.

If it becomes unavoidable, and bulk data loading and updating operations must be performed on a versioned geodatabase, what are the likely implications and effects on other database users? The following scenarios describe the potential impact of such operations.

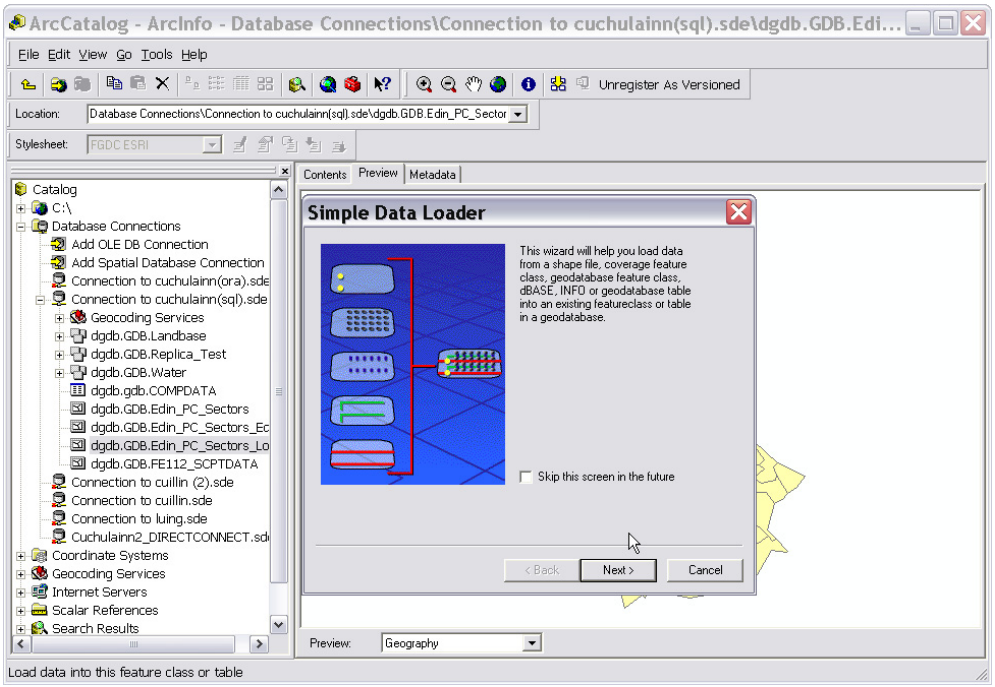
Using the object data loader in ArcMap

One editor, connected to the DEFAULT version, starts using the object data loader in an ArcMap session to load a large number of records into a dataset. Once the load completes, the changes are saved. This request to save triggers a full version reconcile. Depending on the volume of changes being applied to the geodatabase, other editors, also connected to and working with the DEFAULT, may experience a degradation in performance while this reconcile process is running. Their own requests to save and reconcile will be scheduled to complete once the data load reconcile completes.

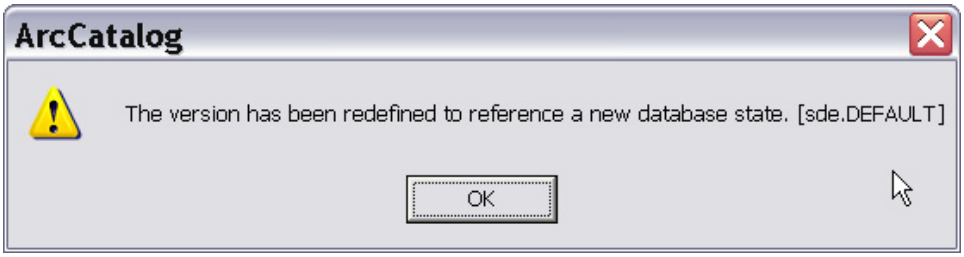
In this case, the creation of new named versions for each editor and the implementation of a version reconcile service would help avoid any lengthy interruptions to the workflow process—editors would simply submit a request to reconcile their version to an automated reconcile service and the service would handle the scheduling of these requests, leaving the editors free to continue with other tasks.

Using the simple data loader in ArcCatalog

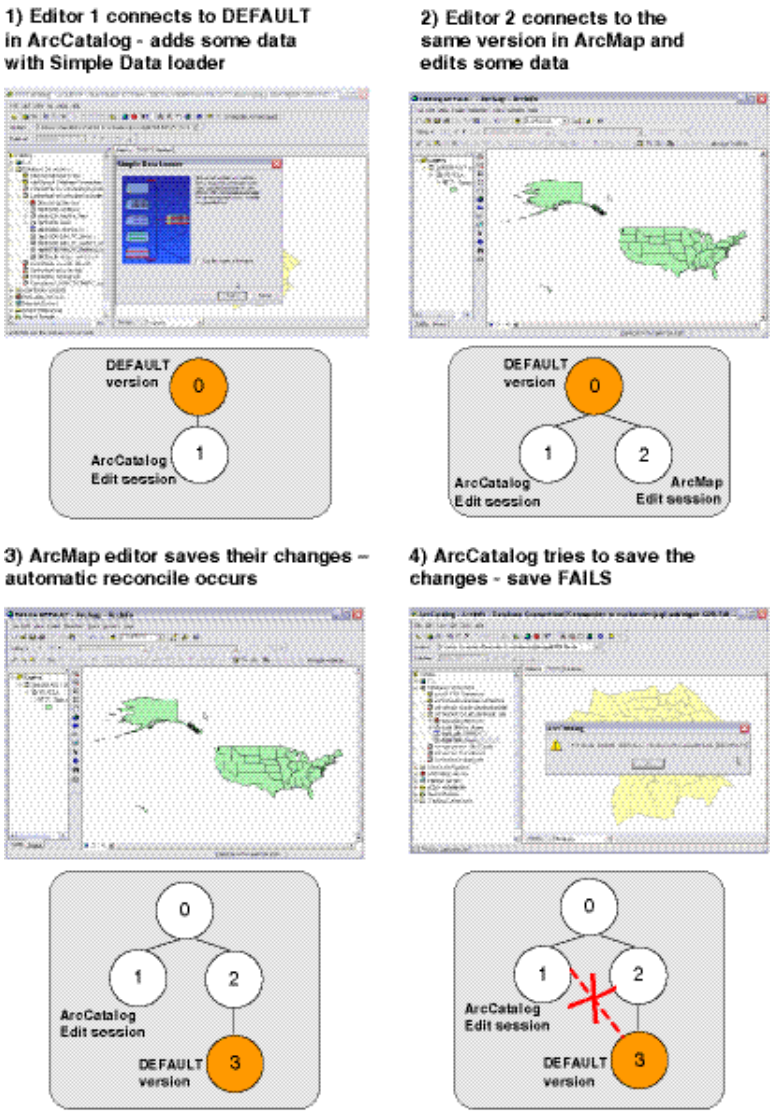
One editor connects to the DEFAULT version of a geodatabase in ArcCatalog and begins a data loading operation using the simple data loader. This actions starts an edit session—the data is loaded into a temporary internal version.



At the same time, another editor connects to the same database in ArcMap and makes some changes to the same DEFAULT version. The ArcMap editor saves their changes and their internal temporary version is automatically reconciled with the DEFAULT version. In ArcCatalog, when the data loading operation reaches the stage when the reconcile between the temporary internal version and the DEFAULT version must take place, the following error is returned:



This situation has arisen because of the edits made during the ArcMap edit session—they have moved the DEFAULT version to a new database state. The reconcile operation following the data loading operation in ArcCatalog is trying to reconcile the temporary version with a state that is no longer referenced by the DEFAULT version. All the data that had been loaded has now been lost and must be reloaded again.



Database state

Inevitably some data loading will still be required once a geodatabase system has been implemented: if rolling back the geodatabase to a simple, unversioned state is not feasible, try to perform these operations when the database is ‘quiet’, during off-peak hours or overnight, and always remember to compress and reanalyze the database once the data loading is complete. Schedule usage of the simple data loader in ArcCatalog for such times when there are no other editors currently modifying a particular version of the database.