



Versioning

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.

Copyright © 2004 ESRI
All rights reserved.
Printed in the United States of America.

The information contained in this document is the exclusive property of ESRI. This work is protected under United States copyright law and other international copyright treaties and conventions. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, or by any information storage or retrieval system, except as expressly permitted in writing by ESRI. All requests should be sent to Attention: Contracts Manager, ESRI, 380 New York Street, Redlands, CA 92373-8100, USA.

The information contained in this document is subject to change without notice.

U.S. GOVERNMENT RESTRICTED/LIMITED RIGHTS

Any software, documentation, and/or data delivered hereunder is subject to the terms of the License Agreement. In no event shall the U.S. Government acquire greater than RESTRICTED/LIMITED RIGHTS. At a minimum, use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in FAR §52.227-14 Alternates I, II, and III (JUN 1987); FAR §52.227-19 (JUN 1987) and/or FAR §12.211/12.212 (Commercial Technical Data/Computer Software); and DFARS §252.227-7015 (NOV 1995) (Technical Data) and/or DFARS §227.7202 (Computer Software), as applicable. Contractor/Manufacturer is ESRI, 380 New York Street, Redlands, CA 92373-8100, USA.

ESRI, the ESRI globe logo, ArcSDE, ArcObjects, ArcMap, SDE, ArcGIS, www.esri.com, and @esri.com are trademarks, registered trademarks, or service marks of ESRI in the United States, the European Community, or certain other jurisdictions. Other companies and products mentioned herein are trademarks or registered trademarks of their respective trademark owners.

Versioning

An ESRI Technical Paper

Contents

The principles of versioning	1
Edit transactions and the geodatabase	1
Versioning and workflows	3
DEFAULT	3
Design	4
Approved	4
Construction and Completed	4
The practice of versioning	4
Versions	5
Database states	8
Delta tables	11
Adds table (A<Registration_ID>)	12
Deletes table (D<Registration_ID>)	13
Editing a versioned geodatabase	15
Reconciling versions	16
Manual reconcile	20
Automatic reconcile	21
Conflict detection	23
Posting versions	26
Special reconcile cases	28
Feature-linked annotation	28
Relationship classes	29
Geometric networks	30
Topologies	36
Versioning and topologies	37
Version administration	57
Database trimming (automatic)	57
Database compression (manual)	58

Versioning

Multiuser geodatabase editing is supported by versioning. This technical paper discusses the practice and principles of versioning, editing a versioned geodatabase, reconciling versions, conflict detection, posting versions, special reconcile cases, and version administration.

The principles of versioning

An enterprise geodatabase must provide support for many users creating and updating large amounts of geographic information. As many of those users may be required to edit the same data at the same time, a geodatabase must provide an editing environment that supports multiuser concurrent editing without creating multiple copies of the data. In providing this functionality, this editing environment must also support edit sessions that typically span a number of days, the facility to undo or redo changes made to the database, the testing and development of data models and alternative application design proposals without affecting the published database, and a facility to monitor how the data and the database have evolved over time. A geodatabase must also be capable of supporting the project workflows implemented in many organizations.

To better understand the particular challenges of managing concurrent multiuser access to geographic information stored in a database management system (DBMS), the following section will describe DBMS edit transactions.

Edit transactions and the geodatabase

A key element in database functionality is the transaction. A transaction represents a logical group of data-based operations that make up a complete operational task. Database transactions can be either short or long.

Short transactions

Short transactions are edit operations that are completed in a matter of seconds. Examples would include automated bank teller transactions, record updates, and so on.



During the course of a transaction, locks are applied to the affected rows. These locks guarantee the correctness of the data by ensuring the changes made by different users are applied in the correct order. Locks are managed internally in the database and are acquired and released based on actions taken by a user, such as commit or rollback operations. When a short transaction completes, the locks are released.

Although short transactions work well in situations where critical applications require immediate access to a consistent view of the data, they are not well suited to the type of editing tasks required when updating geographic data. In situations where many people

are simultaneously editing the data, performing edit tasks that may take more than a few minutes to complete, the short transaction row-locking mechanisms adopted by many DBMSs would be prohibitively restrictive for other database users.

The inherent connectivity and spatial relationships that define geographic data introduce an element of interoperability that requires a more flexible approach to editing. Suppose there are two editors both working on a utility network; one editor makes changes to one of the feature classes involved in the network, an overhead electricity cable, while another editor updates elements of a related feature class, the electricity poles. Changes made to features in either feature class could have a direct effect on features in the other; for example, moving a pole could reposition a cable. This situation could introduce lock escalation (when row locks become page locks, page locks become whole table locks), and deadlock situations (where two transactions are waiting for each other to unlock data, preventing any further updates to the data until the deadlock is resolved) that can have a negative impact on database performance and scalability.

Once committed to the database, short transactions are also difficult to undo; the database has only one state—the most recently committed transaction.

Long transactions

Editing geographic data often requires more than just a few simple changes—some editing tasks may require hours, days, or even months to complete, such as updating a regional land use dataset or the design and construction of a new section of highway. These extended editing operations, which may involve many short transactions and a number of database editors, are referred to as long transactions.



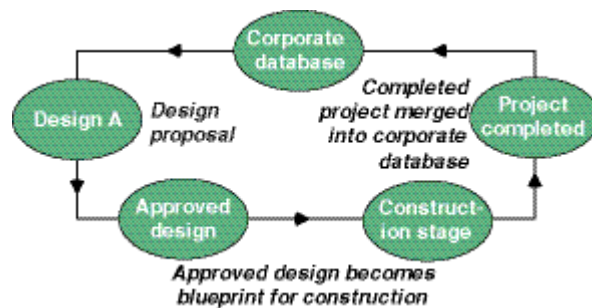
The solution to supporting long transactions had to provide a data management framework that could accommodate multiple editors working on the same data at the same time, often over a long period of time, but without duplicating the data. All these requirements were addressed in the geodatabase with the adoption of an optimistic concurrency data-locking model, which means no locks are applied to the affected features and rows during long transactions, and a data editing environment called 'versioning'.

Versioning involves recording and managing changes to a multiuser geodatabase by creating a 'version' of the database—an alternative, independent, persistent view of the database that does not involve creating a copy of the data and supports multiple concurrent editors. Versioning can only be implemented on ArcSDE® geodatabases hosted on a DBMS platform that supports concurrent multiuser editing. Personal geodatabases, on the other hand, which support only single-user editing, do not support versioning.

Although the absence of row locks introduces the inevitability of editing conflicts, versioning makes it easy to detect and resolve those conflicts. In real-world editing situations, conflicts are the exception rather than the norm. Given the small number of edits in comparison to the volume of data stored in a geographic database, the overhead of resolving these conflicts is relatively minor compared to the restrictions of prohibitive data locks or having to check features out of a central database to some local repository for the duration of a long transaction.

Versioning and workflows

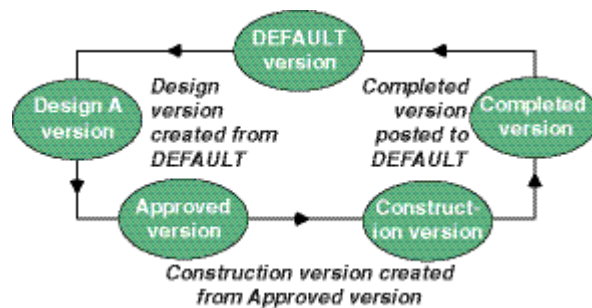
The established workflow processes for many GIS applications are based on a cycle of project design, approval, construction, and maintenance—processes that require many people to simultaneously edit data in an environment that allows them to make those changes visible only to those who have an interest in seeing them.



To accommodate this workflow, discrete versions of the database are required. Versions will typically represent a work order, a design alternative, or a historical snapshot of the database. As the changes made to each version are recorded independently, versions are unaffected by changes occurring in other versions of the database—editors can simultaneously update the features or rows in one version without explicitly applying locks that would prohibit other users from modifying the same data in another version.

In the following simple example, a database has been configured with five versions.

Note: This example shows only one of a number of workflows that versioning supports.



Version workflow cycle

DEFAULT

All versioned geodatabases contain a DEFAULT version, which represents the ‘as-built’ or published view of the geodatabase. This version is created automatically and is the root of the version tree—that is, it does not have a parent, and it is the ultimate ancestor of all other versions in the database.

Design The first stage of a new project is to create an alternative version of the geodatabase from which to develop a design proposal without affecting other database users. If required, this version could be hidden from other database users by making the version 'private'—only the person who created the version or the database administrator could view and edit it. Examples would include an engineer's design or an architect's proposal for a new building project.

Approved Once the design has been reviewed and approved, a new child version is created from the Design version to support the next stages of the project. By setting the appropriate permissions, other database users may view, but not edit, this Approved version.

Any further modifications made to the Design version will not be visible to users working with the Approved version unless an editor merges, or reconciles, the changes. (Version reconciliation and posting will be covered in greater detail later in this paper.)

Construction and Completed At the beginning of the construction phase a new version, Construction, is created from the Approved version. Any design modifications or implementation changes that arise during the construction phase will be recorded in this version. This prevents any accidental or unintentional alterations to the Approved version, which may be required as a permanent record of the state of the data when the project design was approved.

Once construction is over, a new version is required to record the final state of the data when the work was complete. A new version, Completed, is created from the Construction version. Again, any subsequent changes to the Construction version will not be visible to users working with the Completed version unless the changes are reconciled.

Once all work on the project has been completed, the Completed version, and all the modifications it represents, can now be integrated with the published version of the database, the DEFAULT version, and made available to all database users. Each of the versions created throughout the course of the project may be retained as a historical record of the project or deleted as required.

Versioning provides database editors, administrators, and project managers with the following functionality:

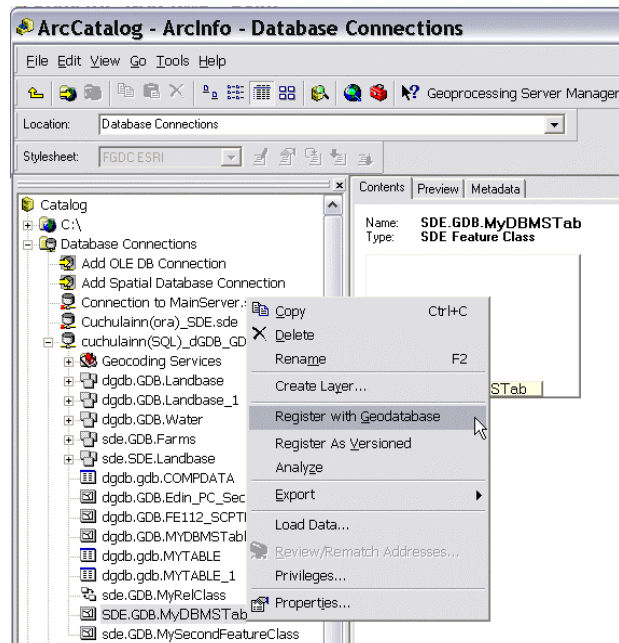
- Support for multiple editors who may be required to edit the same data concurrently.
- Multieditor support is achieved without creating multiple copies of the data.
- Support for the creation of alternative database designs without affecting the published geodatabase.
- Support for the creation of historical versions of the database, which may be queried at some point in the future to determine the representation of the database at any given point in time.

The practice of versioning

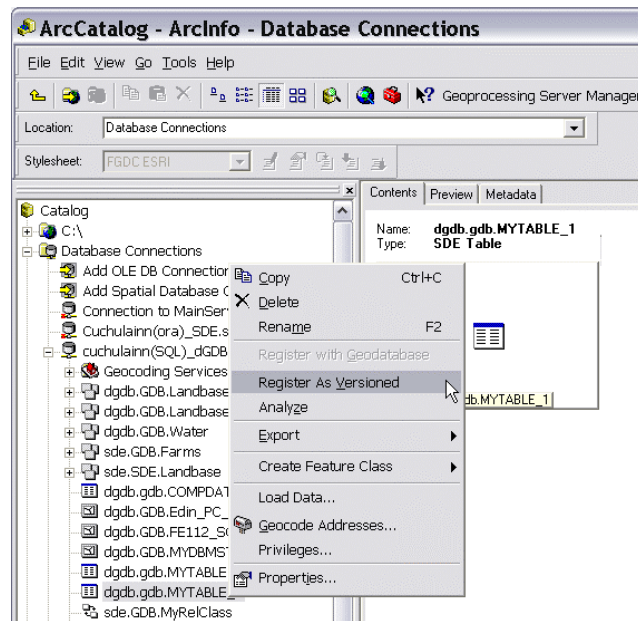
This next section will review how versioning is implemented in the geodatabase and managed internally in the underlying DBMS. This will include a discussion of the important role of database states and version delta tables—understanding how these two elements manage the changes made to a versioned geodatabase will help GIS project/data managers optimize the performance of their geodatabase implementation.

Versions Each version represents a unique, seamless view of the database, distinguished from other versions by a particular set of edits made to that version since it was created. Versions differ only in the number of rows that represent each database object (table), taking into account any new, modified, or deleted features or rows. The schema of each versioned database object is identical across all the versions; any supported schema changes made to a dataset—for example, adding a new field to a table—will appear in all versions of the geodatabase.

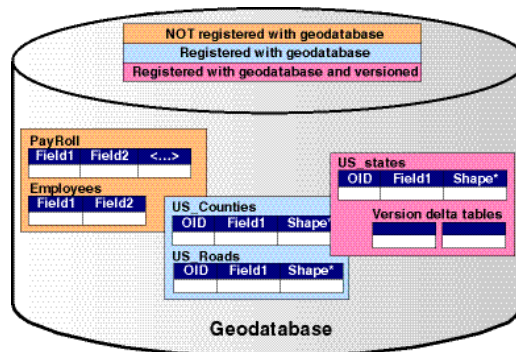
To enable versioning on a dataset, it must first be registered with the geodatabase. When a new feature class or table is created using the desktop applications or some custom ArcObjects™ application code, this registration happens automatically. Non-ArcGIS® tables, those created by third party application software, must be registered with the geodatabase manually.



This process registers the dataset in the geodatabase system tables and adds a unique integer field, the ObjectID (OID) field, to the dataset: versioning relies on the presence of this field. Once a dataset has been registered with the geodatabase, it can then be registered as versioned.



Once registered as versioned, the dataset will participate in all versions of the database. If the dataset is edited in a number of versions, each version will represent an alternate view of the same dataset. A geodatabase can contain both versioned and unversioned data.



Internally, versioning is managed by a number of DBMS tables. The tables are an addition to the ArcSDE data storage schema objects, the business or base table, feature and spatial index tables, and so on.

System Table	Role
VERSIONS	Maintains the list of all database versions. Each version has a name, owner, description, creation date, parent version and associated database state.
STATES	Maintains the list of all database states. Each state has a creation time, closing time, parent state, and owner.
STATE_LINEAGES	Maintains a collection of states organized by lineage name.
MVTABLES-MODIFIED	Maintains the list of all tables that are modified in each state of the database.

A versioned database will typically contain a number of versions, in addition to the DEFAULT version, that might represent a work order, a design alternative, a disconnected editing session, a historical snapshot, and so on. The VERSIONS table contains a descriptive list of these versions with each version identified by a unique name and ID (IDs are automatically generated by ArcSDE). In addition, each version has an owner, description, parent version, associated database state (states will be discussed in more detail later in this paper), and level of user access. The three levels of version access are:

- Private: only the owner can view and edit
- Protected: all users can view, but only the owner can edit
- Public: all users can view and edit

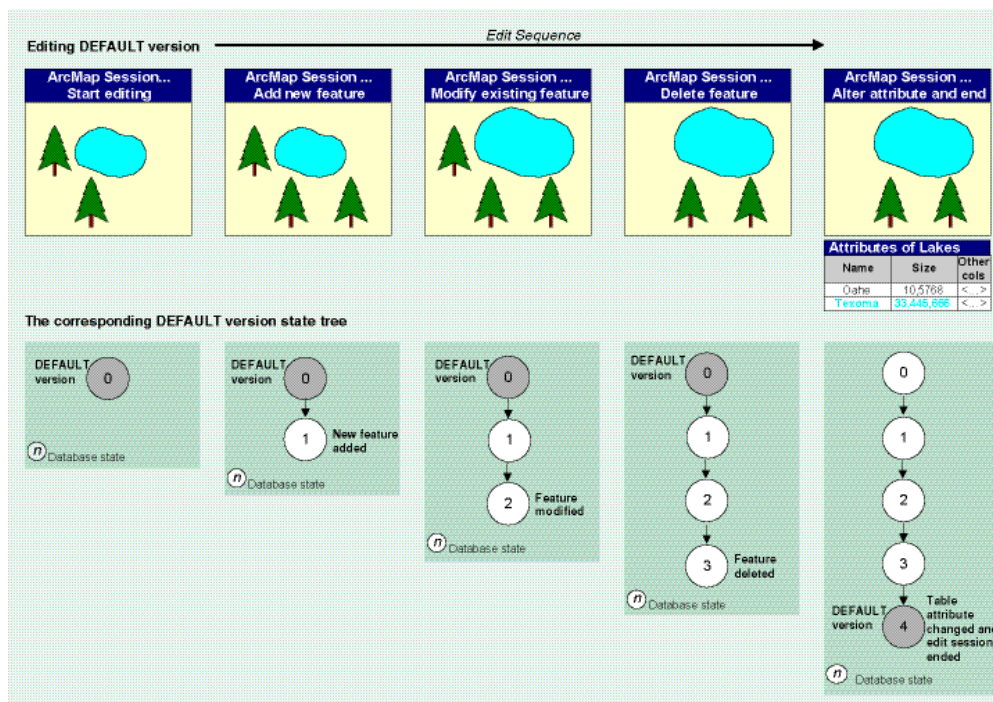
When the VERSIONS table is first created, the details of the DEFAULT version are automatically recorded in this table. The SDE administrator owns the DEFAULT version, and the initial ID of the corresponding database state will be set to 0. The description string will read "Instance Default Version" and as the DEFAULT version has been granted PUBLIC access, any user can modify it. To support general connections to the geodatabase, the level of access to the DEFAULT version must remain either Public or Protected; if the access level were set to Private, only the SDE administrator could connect to the database.

VERSIONS	
Column Name	Description
Name	The unique name of the version
Owner	The version owner
Version_ID	The unique ID of the version
Status	Determines the access level - whether the version is available to all users or private to the user who created it
State_ID	The ID of the database state that this version points to.
Description	An optional text description of the version
Parent_Name	The name of the parent version
Parent_Owner	The owner of the parent version
Parent_Version_ID	The ID of the parent version
Creation_Time	The date/time the version was created

Database states

To manage the edits made to the data, a versioned geodatabase maintains a collection of database *states*, or units of change to the database. A state represents a discrete snapshot of the database whenever a change is made: every edit operation creates a new database state. (An edit operation is any task or set of tasks [additions, deletions, or modifications] undertaken on features and rows.) All geodatabase versions reference one of these database states and evolve over time through a series of states.

In the following illustration, the DEFAULT version initially points to database state zero. As a series of edits are made to that version in an ArcMap™ edit session, the version evolves to state four.



All geodatabase states have the same schema and differ only in the number of rows that represent each modified table or feature class. To identify conflicts, which can occur when the same feature is edited in either the same or different versions, the version state lineages are compared for differences or row conflicts during version reconcile.

All the information relating to states is managed in the STATES table.

STATES	
Column Name	Description
State_ID	A unique, SDE-assigned integer ID
Owner	The user who created the state
Creation_Time	The date/time the state was created
Closing_Time	The date/time the state was closed
Parent_State_ID	The state's parent state ID
Lineage_Name	References the state's lineage stored in the STATE_LINEAGES table

The VERSIONS and STATES tables are queried to identify which database state each version references.

States are maintained in a tree structure, where the parent/child relationships can be derived from the state lineage. Information about the state lineage of each version is

maintained in a separate table, STATES_LINEAGES. This table stores a multiple entry index for traversing state parent/child relationships and is used for all version queries.

STATE LINEAGES	
Column Name	Description
Lineage_name	Name that describes a state
Lineage_ID	Unique ID of individual states

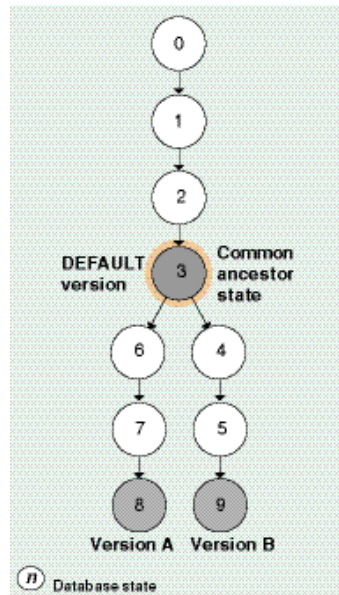
To return the correct view of a version, its states lineage is queried to identify all the states that recorded each change made to that version. From this list of states, the table rows that correctly represent the version can be determined. In the previous example, to represent the DEFAULT version after it has been edited, the rows from the base table, plus the changes in states one, two, three, and four, are returned by the query. As the geodatabase is edited and the versions change over time, the state tree will become more complex.

In conjunction with the STATES table, the MVTABLES_MODIFIED table maintains the list of all tables that have been modified at each state of the database.

MVTABLES_MODIFIED	
Column Name	Description
State_ID	The state in which the table was modified
Registration_ID	The Registration ID of the table that was modified in the state

Any time a feature class or table is modified in a state, a new entry is created in the MVTABLES_MODIFIED table (the table registration_id and state_id). When two versions are reconciled, the first step in the process is to identify the states these two versions reference—the current edit version's state and the target version's state. From these states, a common ancestor state is identified by tracing back through the state lineage of these two versions. For example, if the state lineage of version A (the current edit version) is one, two, three, six, seven, and eight and the state lineage of versions B (the target version) is one, two, three, four, five, and nine, the common ancestor would be state three.

The MVTABLES_MODIFIED table is then queried to identify all the tables that were modified between the common ancestor state and the target version state. For example, for version A, this would mean the tables modified at states six, seven, and eight: for version B, it would be all the tables modified at states four, five, and nine. From this list of modified tables, a second list of tables common to both state lineages is generated. For all common tables in this second list, a number of version difference queries are executed—insert, update, delete, update_update, update_delete. For tables only modified in the lineage of version A, the current edit version, only the insert, update, and delete difference queries are executed as no editing conflicts will exist. (Conflict detection will be covered in more detail later in this paper.)



Delta tables

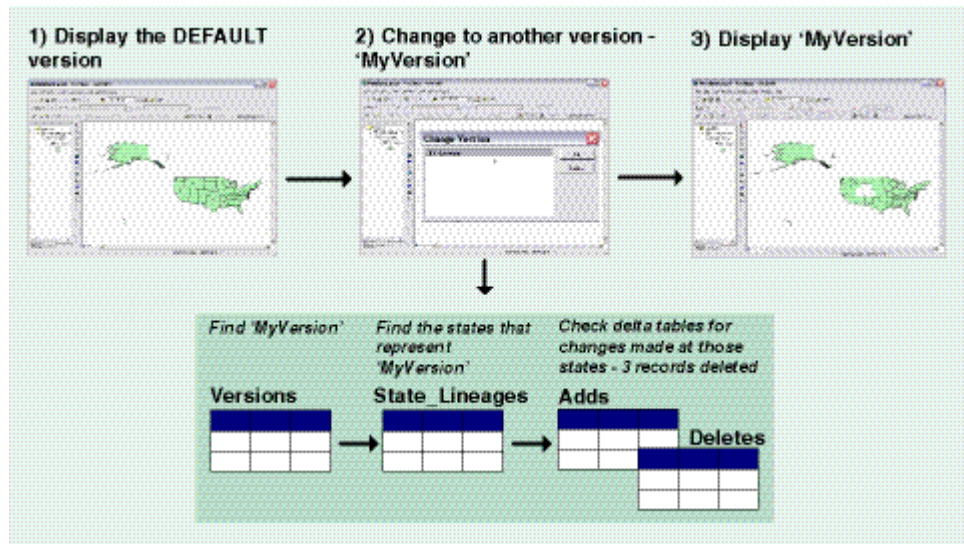
To a client application, versioned data appears much the same as unversioned data: versioned tables and feature classes look like the unversioned tables and feature classes. A table or feature class viewed in one version will contain a certain number of rows, and the same table or feature class in another version may contain a different number of rows, giving the impression that each version is a separate copy of the data.

However, instead of creating a new copy of, or modifying the original data, the geodatabase leaves the versioned table or feature class in its original form and stores any changes to that data in separate geodatabase system tables. The geodatabase system tables that record version changes are referred to as the delta tables. For each table or feature class that has been versioned, two new delta tables, an Adds and a Deletes table, are created.

User Table	Role
A<Registration_ID>	The Adds table stores all additions and modifications to the base table. This table has the same schema as the base table, with an additional SDE_state_ID column.
D<Registration_ID>	The Deletes table stores all deleted and modified row references.
Version base table name	The base (business) table points to state 0 of the data.
Versioned view name	An optional versioned view of the base, adds, and deletes table that reconstructs the correct representation of a table for a version.

These delta tables record any changes—new, modified, or deleted records—made to that table or feature class at each state of the database. To correctly represent each database

version, these tables are queried, in conjunction with the Versions and State_Lineages tables, to identify which change was made at which database state. A version will then return a seamless view of the data that takes into account the original state of the data plus any changes.




Adds table
(A<Registration_ID>)

The Adds table maintains information on each inserted or updated row in a versioned table and is queried to identify which rows have been added or modified at a particular database state. Each column in the base DBMS table is duplicated in the Adds table, which also includes an additional SDE_state_ID column. The SDE_state_ID column indicates the state the row was added at and the ObjectID (or Row_ID) column, already defined in the base table, represents the unique identifier for the row.

A<Registration_ID> (Adds table)	
Column Name	Role
ObjectID	Unique identifier for the new feature
<other columns>	Attributes of the new feature
SDE_State_ID	The state the new feature belongs to


Pre edit session



Adds Table (A<Registration_ID>)

ObjectID	Other Columns	SDE_State_ID

Add new parcel



Adds Table (A<Registration_ID>)

ObjectID	Other Columns	SDE_State_ID
21	<....>	12


Deletes table (D<Registration_ID>)

The second delta table created when a table is registered as versioned is the Deletes table. This table maintains information on all rows that were deleted or updated in a versioned table and is queried to identify which rows have been deleted or modified at a particular state. When a row is deleted, the record isn't physically removed: it is flagged as deleted and never returned in subsequent database queries.

The Deleted_At column indicates the state of the database at which the change was made; the SDE_Deletes_Row_ID column represents the Row ID of the deleted/updated row, and the SDE_State_ID column again identifies the state from which the row is being deleted.

D<Registration_ID> (Deletes table)	
Column Name	Role
Deleted_at	The state in which the modification is made
SDE_Deletes_row_ID	The unique ID of the updated/ deleted row
SDE_State_ID	The state that the row is being deleted from


Pre edit session



Deletes Table (D<Registration_ID>)

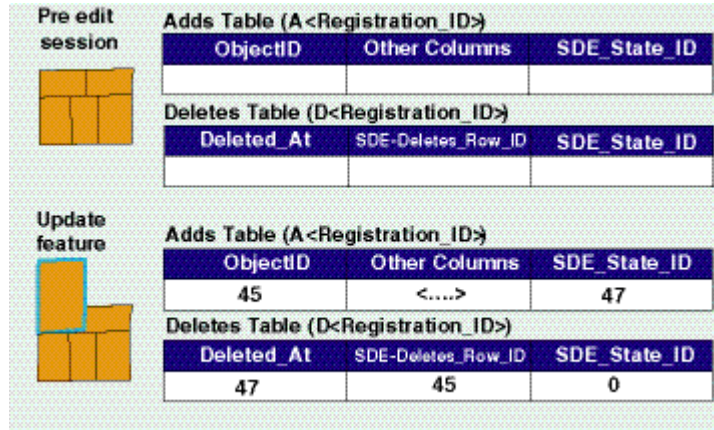
Deleted_At	SDE-Deletes_Row_ID	SDE_State_ID

Delete two parcels

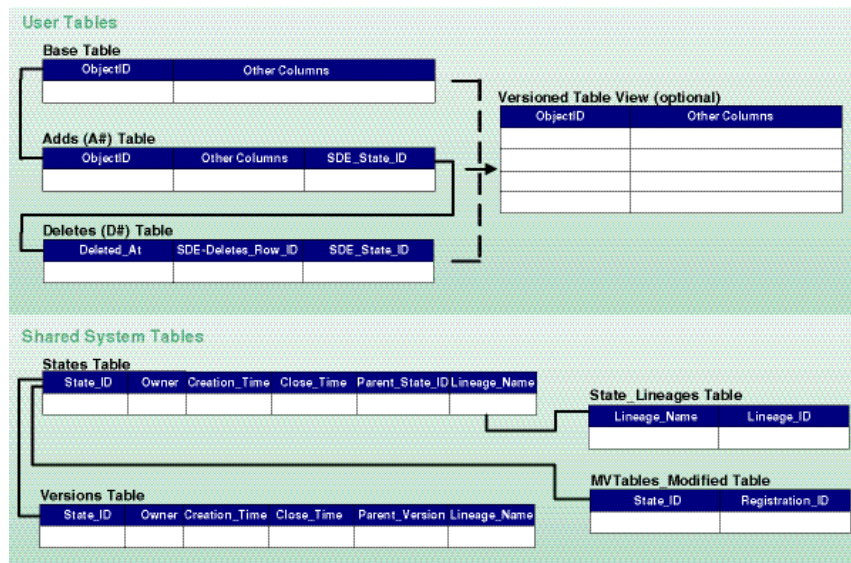
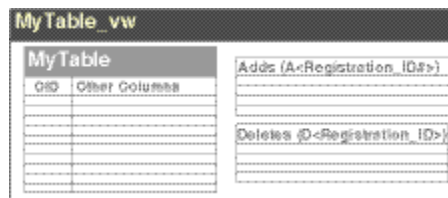


Deletes Table (D<Registration_ID>)

Deleted_At	SDE-Deletes_Row_ID	SDE_State_ID
34	15	0
34	16	0



An optional DBMS multiversioned view can be created as a substitute for the base table. This view reconstructs the correct representation for a specified version of the database of a table or feature class, taking into account all the changes that have been made and logged in the delta tables. These views provide support for third party client applications that have a requirement to connect to a geodatabase and work with a seamless, versioned view of a particular table.



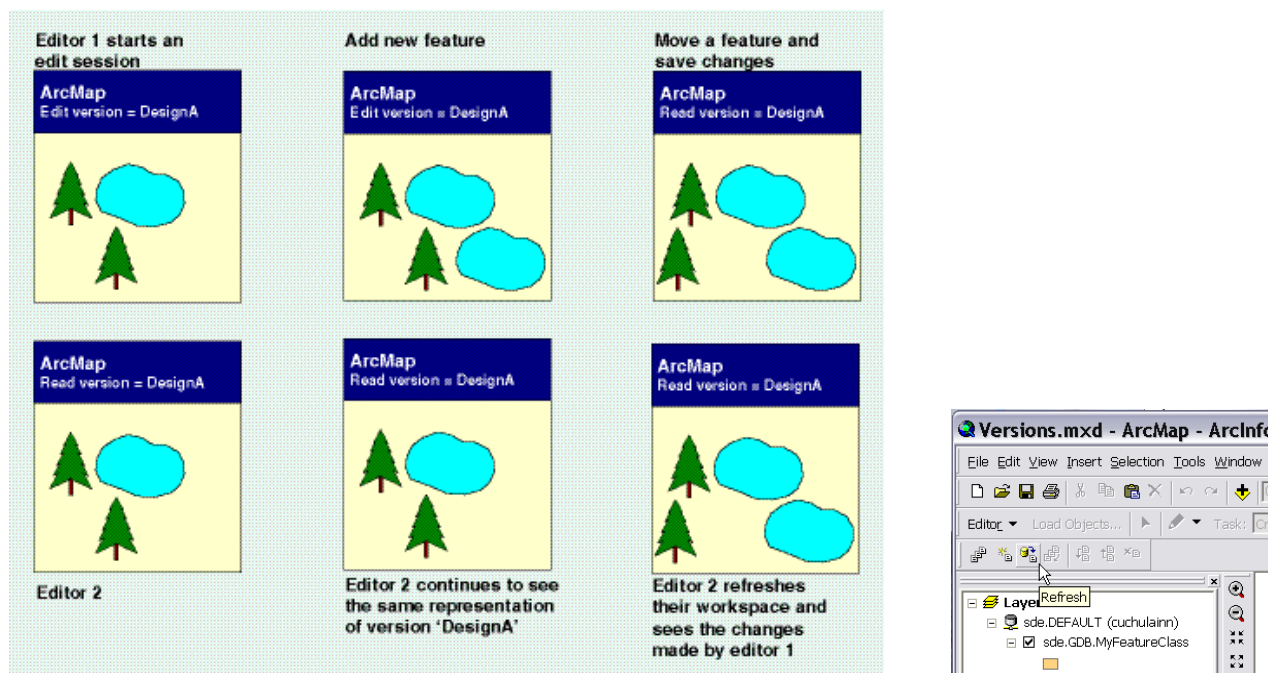
Geodatabase/ArcSDE Versioned Tables

Editing a versioned geodatabase

Editing versioned data in ArcMap takes place within an edit session. These edit sessions represent a long transaction that could potentially result in a number of changes made to the data. Every edit operation executed within an edit session creates a new geodatabase state. These edit operations provide the capability to perform undo/redo operations.

Internally, the edit session on the current edit version works with its own representation of the version. This isolates any saved changes made to that version from other database users who may be connected to that version at the same time. These other users will continue to view the version as it was before the edit session began by referencing the initial state for read consistency.

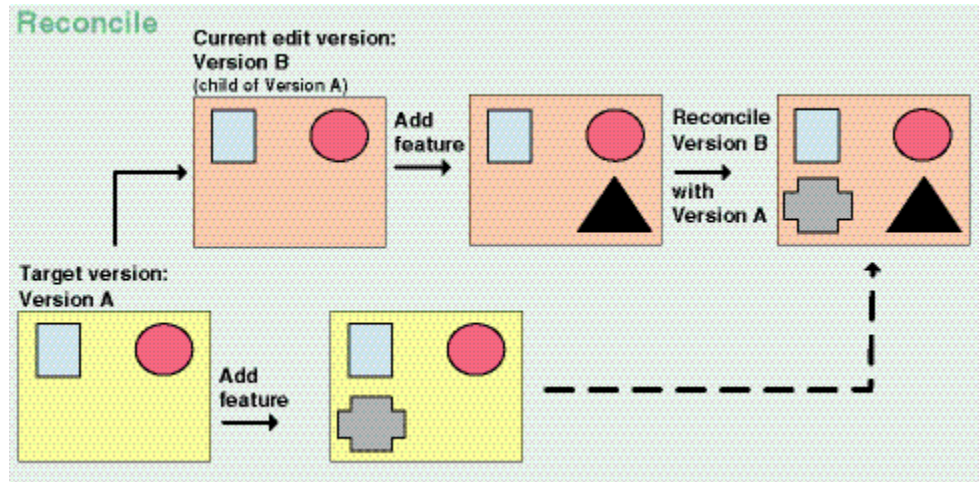
The changes introduced during each edit operation are not visible to other users until the edit session has been saved. When these edits are saved, the temporary internal version is automatically reconciled with the current edit version; any currently connected users must refresh their workspace connections to see the edits that have been made. For example, in ArcMap this workspace refresh option is available as a tool on the Versioning toolbar.



By design, ArcMap does not support editing multiple versions at the same time in the same map document. This is a deliberate restriction to safeguard against possible data corruption. However, if there is a requirement—for example, to update a parent and a child version at the same time—editing multiple versions during one edit session is supported programmatically with ArcObjects. Care should be taken to avoid introducing inconsistencies in the data.

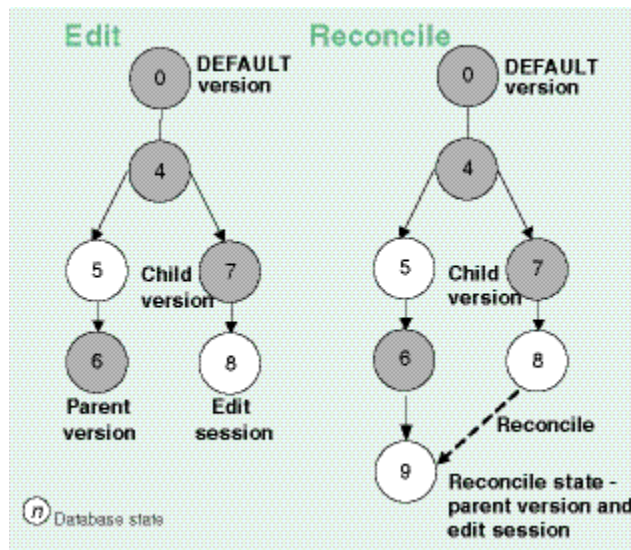
Reconciling versions

To merge the changes that have been made to two versions, the version currently being edited and a target version (a target version is any version in the direct ancestry of the current version being edited, such as the parent version or the DEFAULT version), the two versions must be reconciled. This reconcile process initially involves merging features and objects from the target version into the current edit session.



Once this merge has completed, an editor will review the changes that have been introduced into the current edit session.

Internally this reconcile operation has the following effect on the version state tree:



The state lineage for the target (parent) version is state zero, four, five, and six; the state lineage for the current edit session is state zero, four, seven, and eight—all features that

have been added, modified, or deleted at these combined states will be reconciled into the current edit session and a new state, state 9, will be created.

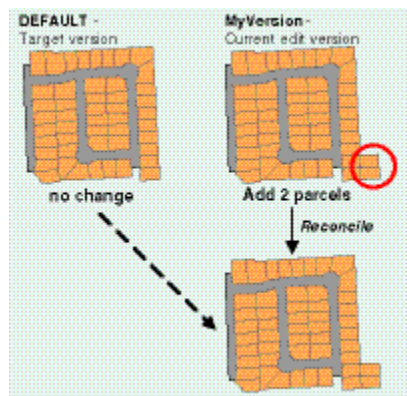
As a version spans all the versioned data in a geodatabase, any objects and features that were modified in the target version will be merged into the edit session. However, as the majority of these objects is not in conflict, they will merge seamlessly into the current edit session, replacing the current objects or features.

By default, features and objects modified in the parent version will take precedence over the same features and objects in the current edit version. In the event of a reconcile conflict, where the same feature or object has been modified in both versions, the editor will be notified as to which features are in conflict. [Conflict detection will be discussed in greater detail later in this paper.]

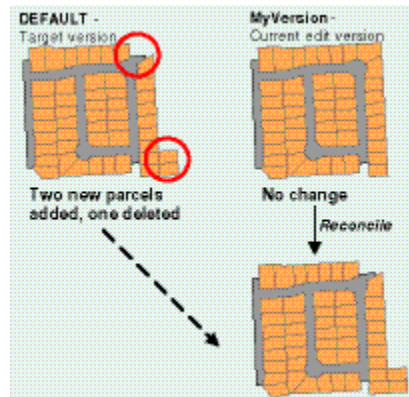
It's then up to the editor to decide if they want to keep the changes from the parent version or accept the changes they've made in the current edit session. For example, a new polygon feature has been added to the parent version—after the reconcile, this polygon feature appears in the current edit version. The editor has to decide if they want to keep this feature or not by either accepting the results of the reconcile or keeping the changes made during the current edit session—to keep the results of the reconcile process, the editor would simply save the changes; to undo the results of the reconcile, the editor would end the edit session without saving the changes.

The general rules for reconciling versions are:

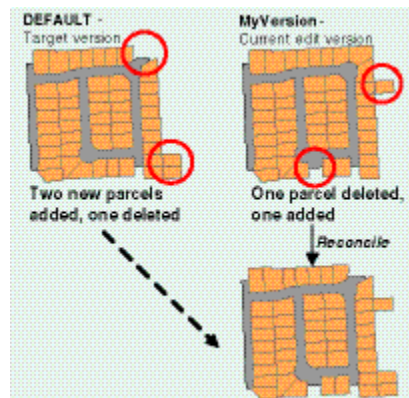
- When the target version has not been modified before the current edit session is reconciled, the result will be the representation of the current edit session.



- When the target version had been modified before the current edit session was reconciled, the target representation would take precedence and those features not in conflict with features in the current edit version would become the results of the reconcile.

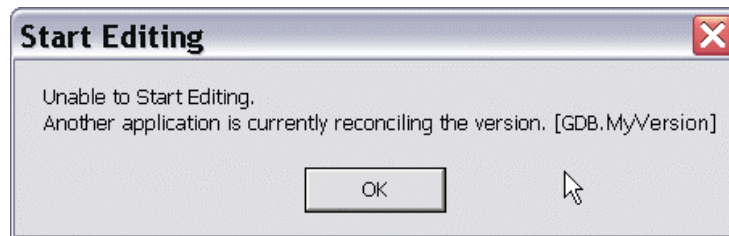


- When both the target and current edit versions have been edited, the results of the reconcile will be a merge of both sets of edits. (Any features modified in both versions will be highlighted as conflicts—see the next section on conflict detection.)

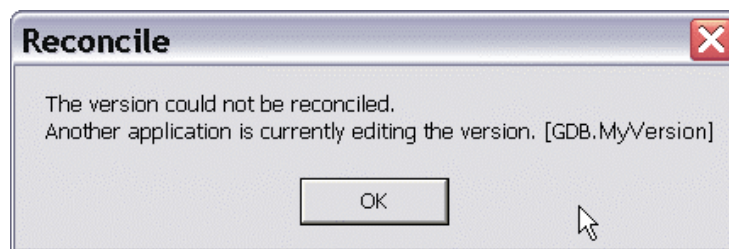


During the reconcile process, no spatial filters are applied, all the tables that have been modified are identified by querying the MVTABLES_MODIFIED table, and the version delta tables are queried to identify the differences between the states and any conflicts that may exist.

When an edit operation is started, a shared lock on the version is acquired. Once a shared lock has been assigned to the version, other users can still access the version but not make any schema changes until the shared lock is released when the edit operation has ended. When two versions are reconciled, the reconcile process promotes this shared lock to an exclusive lock. An exclusive lock prevents other users from getting a shared lock on the version and prevents the version from being edited by other users while the reconcile is taking place. It remains in effect throughout the reconcile process until the final save edits or version post operation has been completed. (Version posting will be covered in a later section.) When another editor tries to edit that version, the following message is displayed:



If this exclusive lock cannot be acquired by the editor who wants to reconcile their version, the reconcile process will terminate with the following warning:

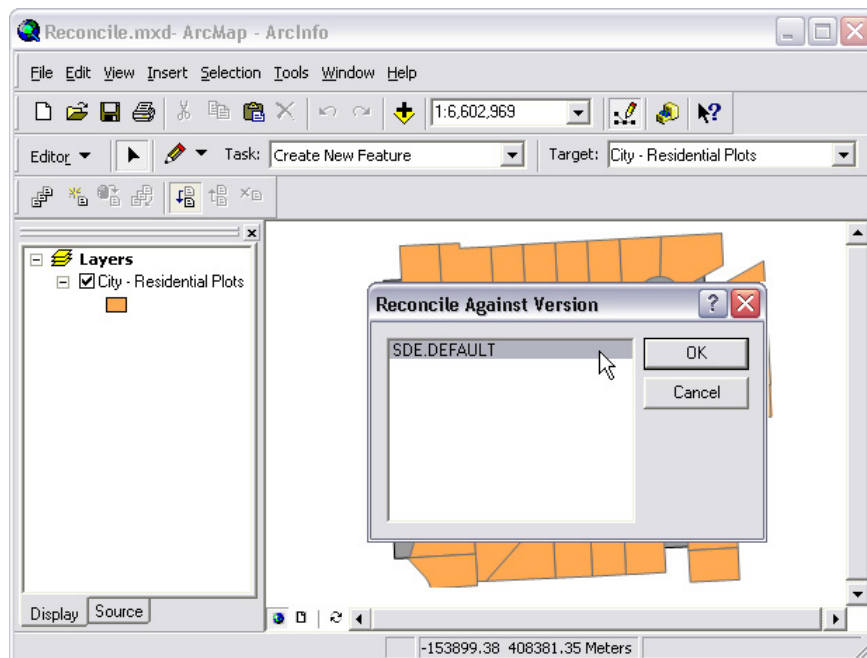
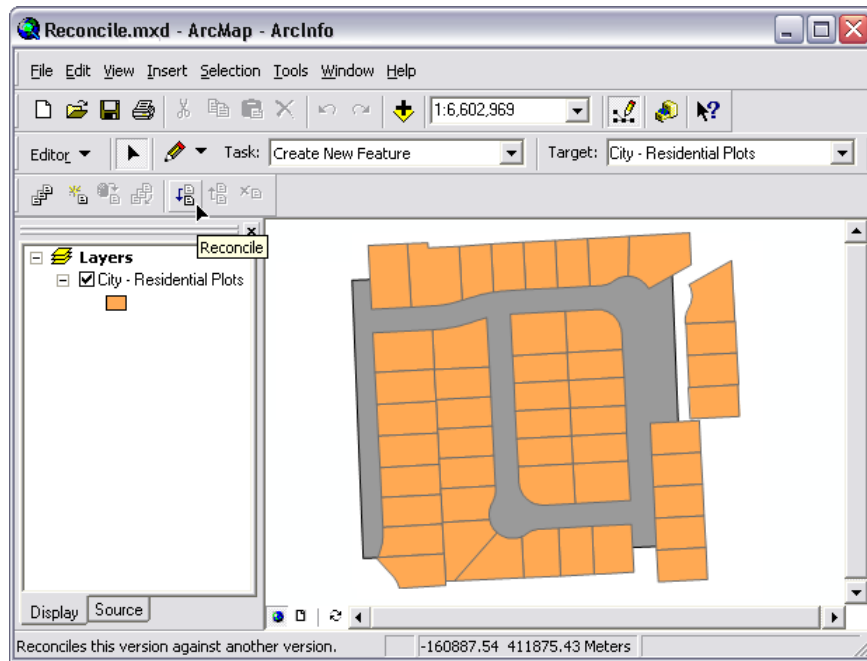


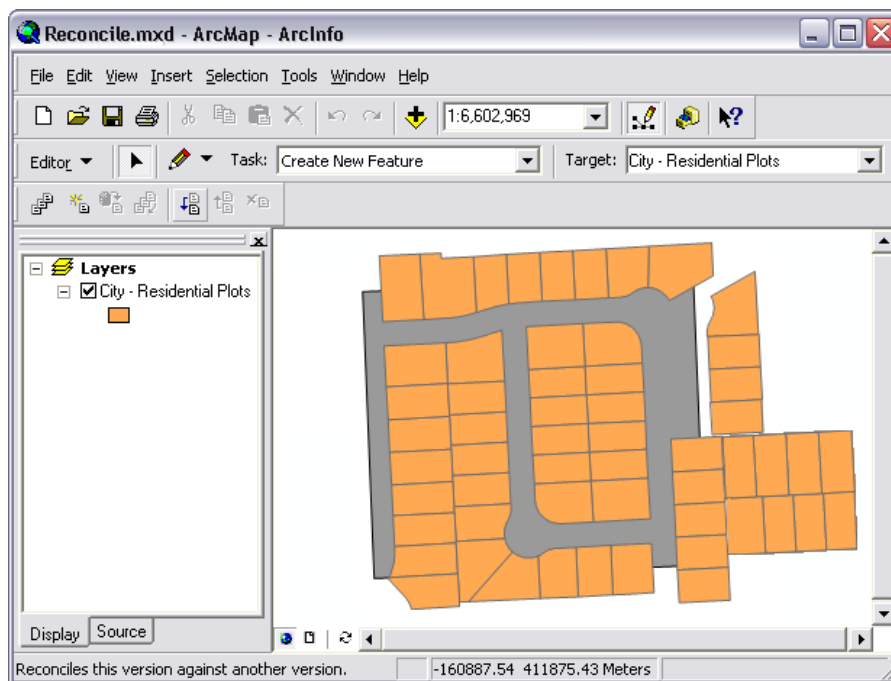
A shared lock is also acquired on the target version—this prevents multiple concurrent reconcile operations from taking place. A version that is being reconciled can't be edited by another edit session; a version that is being edited by other edit sessions can't be reconciled.

Reconcile is either a manual or automatic operation.

Manual reconcile

After completing some changes to a version, an editor selects another version, the target version, against which to reconcile.



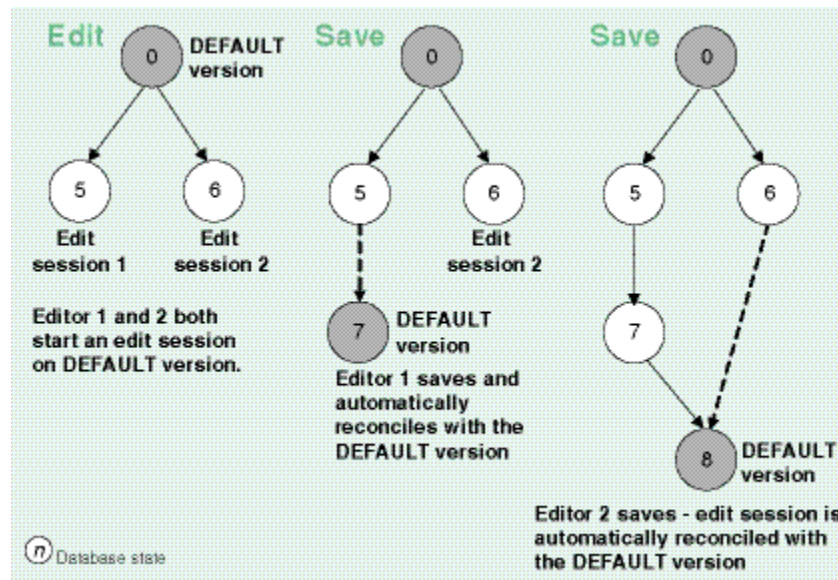


The relative cost of a reconcile operation is directly related to the complexity of the data model and the number of classes modified in the version being reconciled. For example, if geometric networks, topologies, or relationships are being reconciled, additional processing is required to ensure all associated geodatabase behavior is applied correctly to preserve the integrity of the data.

Although the volume of changes can also affect performance, as a general rule it's more efficient to reconcile a large number of changes in batches rather than reconciling a small number of changes at frequent intervals. The main performance cost of the reconcile process is opening the datasets in each version during the reconcile—in most cases, there is less cost associated with moving the changes between versions. However, if the volume of changes to reconcile is considerable, it may be more efficient to reconcile the changes in smaller batches.

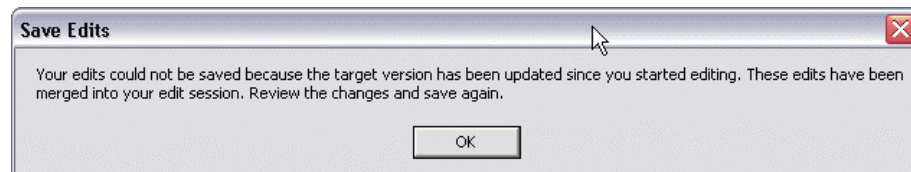
Automatic reconcile

Whenever an editor saves their work or ends the edit session, the temporary version, created by the edit session, is automatically reconciled with and posted to the target version. In this case, the target is the initial state of the current edit version. Automatic version reconcile also takes place when multiple editors are simultaneously editing the same version at the same time.



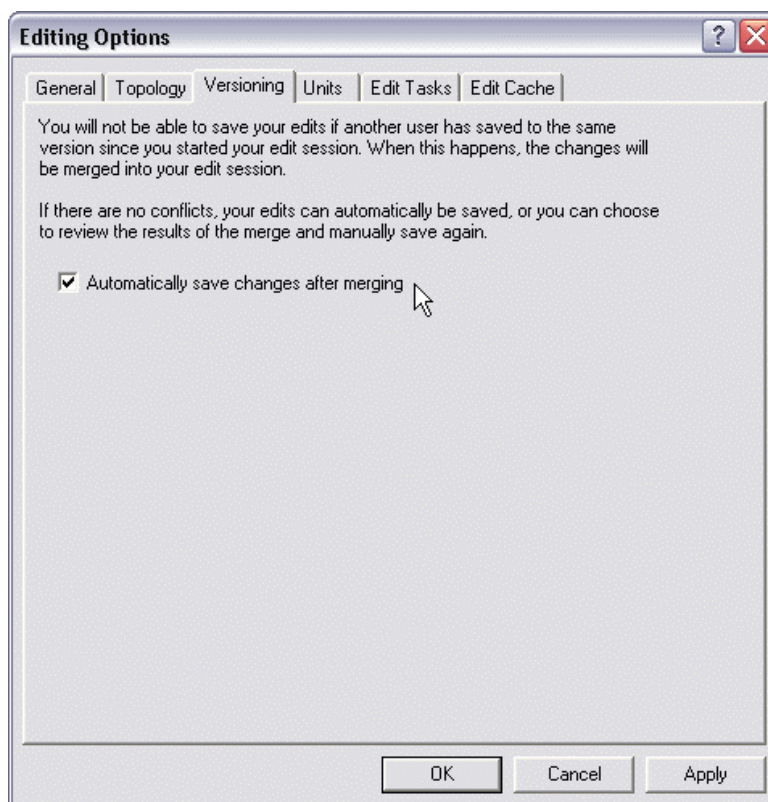
Note: Regardless of which reconcile option is used, reconcile can only take place within an active edit session.

After one editor has saved and automatically reconciled their changes, when a second editor then tries to save their changes, a warning will be returned to advise that the target version has been altered since the current edit session started. The second editor must review the changes from the target version that have now been merged with the current edit version and save it again.



Any conflicts introduced by the automatic reconcile would be highlighted and the editor would have to decide which version of the feature or rows in conflict was acceptable.

An auto reconcile option available in ArcMap can override this default behavior. If the target has been modified since the second editor's edit session began, the changes will be saved automatically, without notifying the editor, after performing the reconcile.



Reconcile notification can be useful, however, to allow the second editor to inspect the results of the reconciliation and review the differences that were introduced into the current edit version.

Conflict detection

To manage long transactions in the database, the geodatabase adopts an optimistic concurrency approach to data management. This means that when features and objects are modified, no locks are applied to the data; other editors may edit the same features, at the same time, in the same or in another version of the database. This will inevitably introduce the potential for feature conflicts when versions are reconciled.

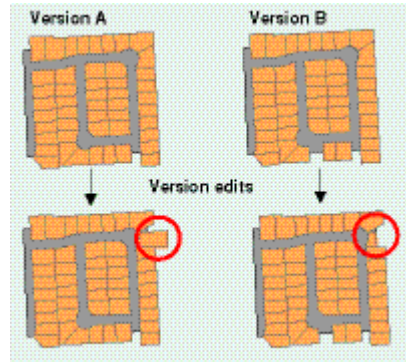
The absence of the feature locks delegates the responsibility for safeguarding against version conflicts to workflow managers. Structuring the workflow in such a manner to avoid overlapping edits will reduce the likelihood of feature conflicts.

A conflict will occur when a comparison of two version state lineages highlights features or rows that are found in both lineages but are different in some way. This situation can happen when:

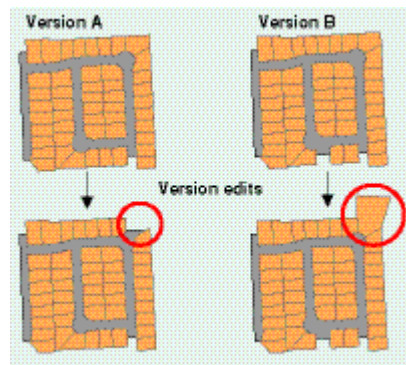
- Two editors are editing the same data in the same version at the same time.
- The same feature is modified in two different versions.

There are two categories of conflicts:

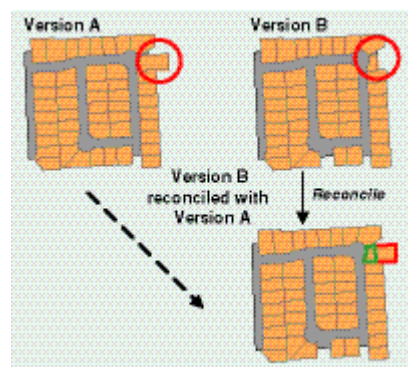
1. The same feature has been updated in each version.



2. The same feature has been updated in one version and deleted in the other.

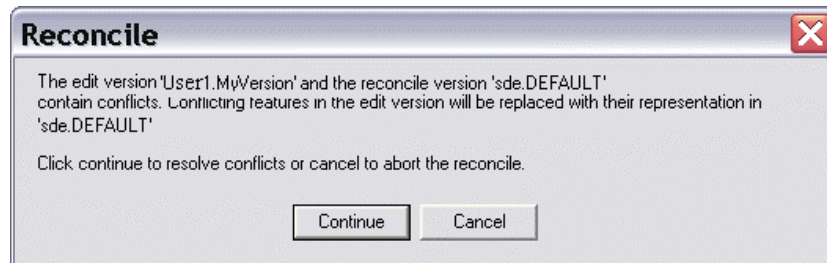


When conflicts are detected, the default behavior is for the target version's feature representation to take precedence over the edit session's representation. All conflicting features in the current edit session are initially replaced by their representation in the target version.



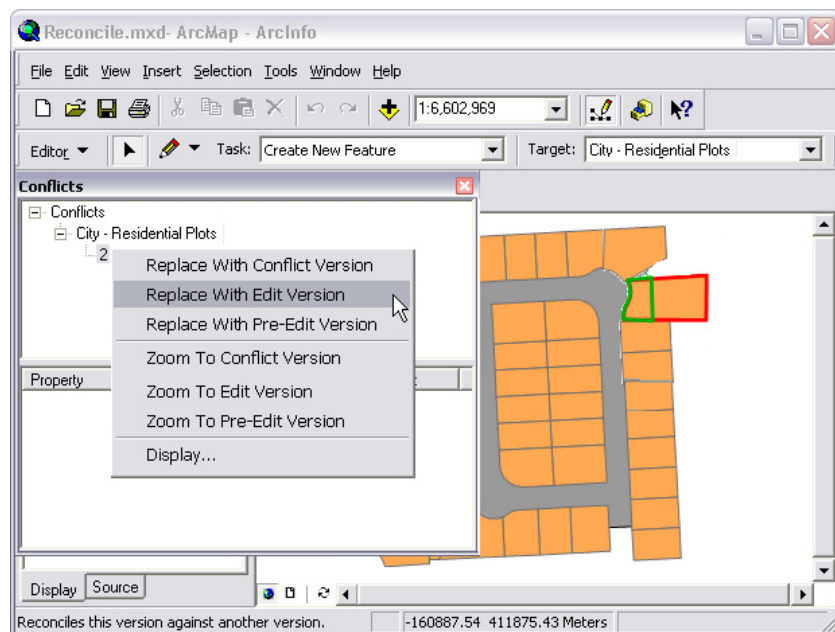
If multiple users are editing the same version and conflicts are detected, the feature or row that was first saved becomes the target version's representation and takes precedence over the second edit session's representation.

ArcMap ensures the integrity of the data by forcing editors to interactively inspect and resolve each conflict.



If the editor wishes to undo the results of the reconcile, selecting cancel from the Reconcile warning dialog box will undo the Reconcile, and the current edit session will revert to the prereconcile configuration of the data.

If the editor opts to continue the process and resolve the conflicts, the features in conflict will be highlighted in a conflict dialog box. This dialog box also provides a list of alternatives for selecting the required representation of the data.



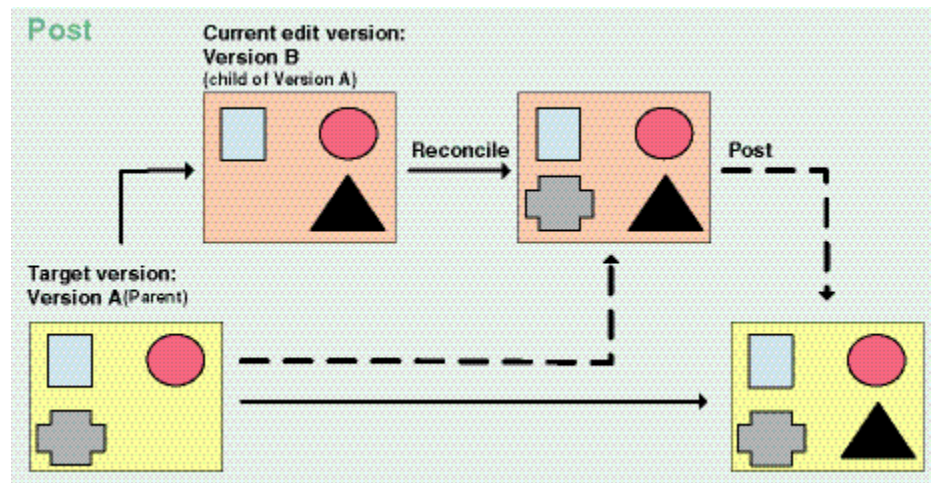
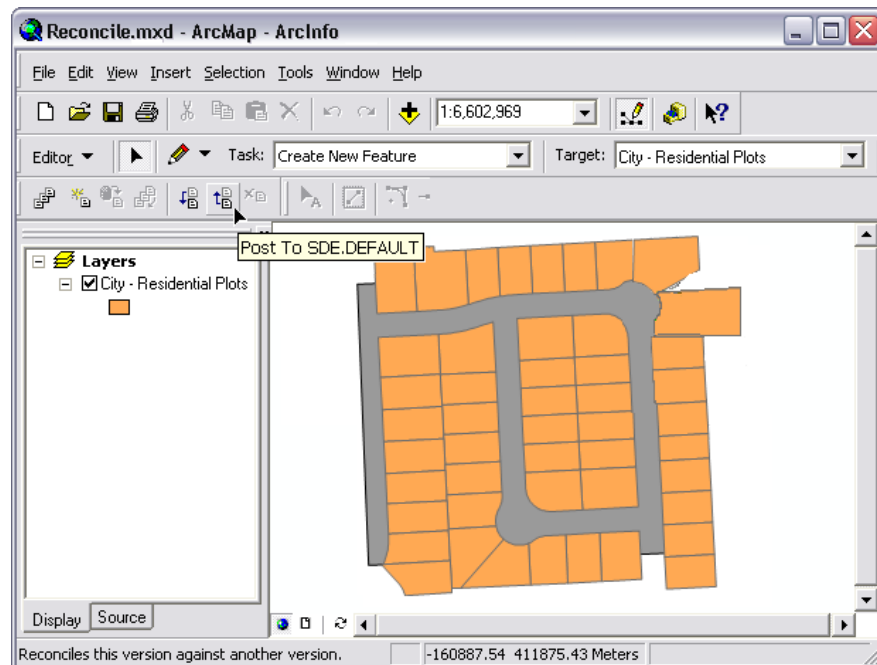
Options for dealing with conflicts include:

- Replacing the features in conflict with the target version's representation

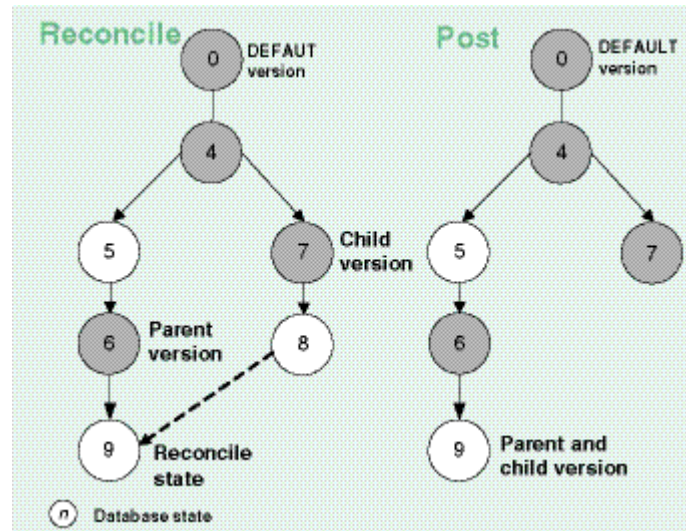
- Replacing the features in conflict with the current edit version's representation
- Replacing the features in conflict with the pre-edit version—that is, reverting to the representation of the data in the current edit session before the conflicting edit was made

Posting versions

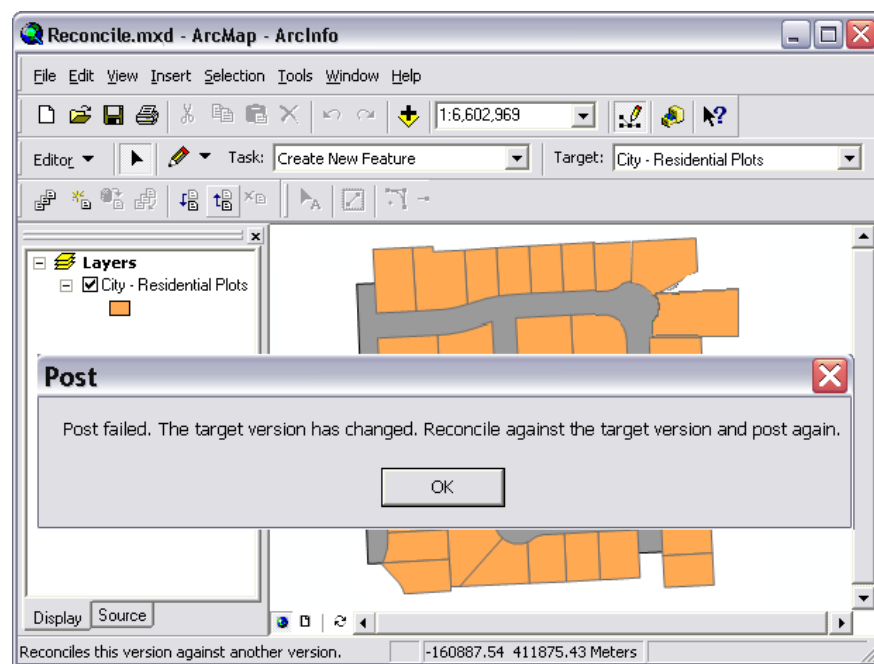
Once the reconcile operation has been completed, and the editor has decided which representation of the features and rows is acceptable, the results of the reconcile can be posted from the current edit version to the target version.

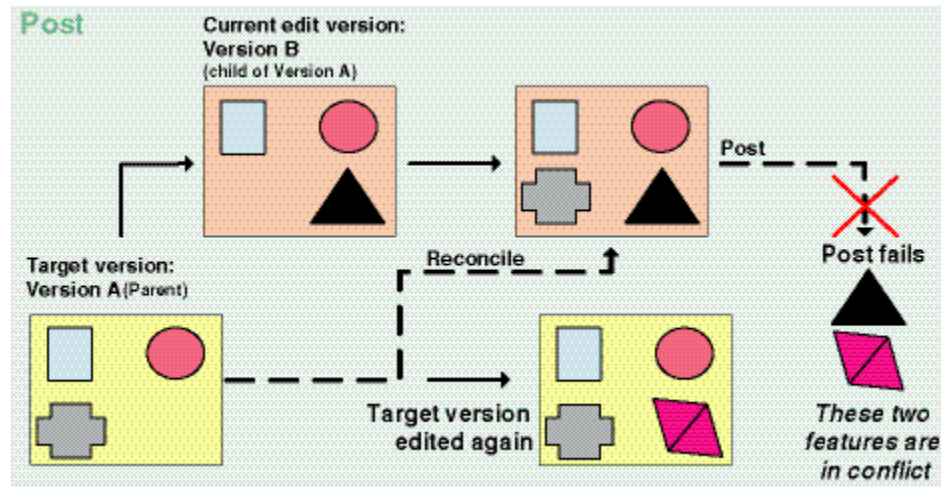


The post operation first saves the edit session to the reconciled state, synchronizing the edit session with the target version. At this point the edit session state and the target version are identical. Posting then relabels the target version to the new state.



This post operation can only be completed if the target version has not been modified since the reconcile operation completed. If any changes have been made to the target version in the interim, the post will fail as the now modified target version would contain changes that had not been reconciled with the current edit session.





This situation could arise if other editors have continued to make changes to the target version. Changes made after the reconcile operation could be in conflict with features in the current edit version.

Before the changes made to the current edit version can be posted to the target version, the two versions must be reconciled again to take into consideration the latest changes made to the target. Any conflicts introduced in the target version would be identified at this stage. Once this second reconcile was complete, and assuming no further changes were made to the target, the post operation can be successfully completed.

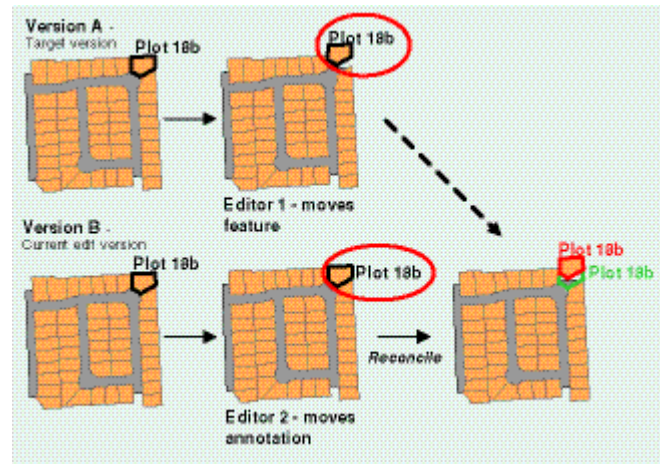
Special reconcile cases

The previous examples illustrated the effect of version reconcile on simple data—that is, data that is geometrically unrelated to other data. While the same basic reconcile mechanisms are in operation, resolving conflicts between features that are related to other features, as with feature-linked annotation, relationship classes, geometric networks, and topologies, is different to resolving conflicts between simple features. As these geodatabase objects have specific internal behavior that can affect other features, resolving a feature conflict in these cases may trigger this behavior and produce some unanticipated results.

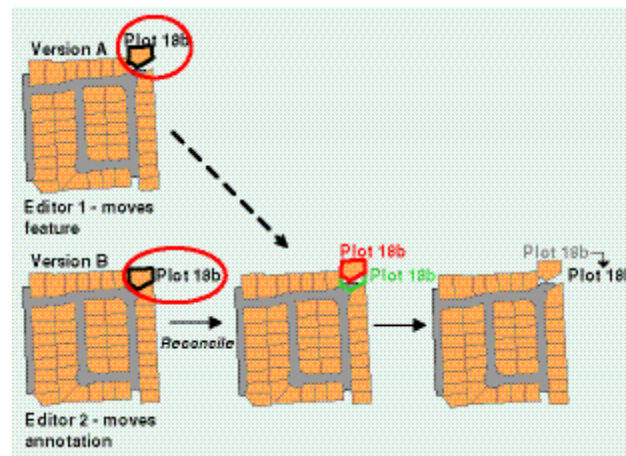
The next sections illustrate how version reconciliation and conflict detection operate with respect to these objects.

Feature-linked annotation

When reconciling feature-linked annotation, one general rule applies—reconciling a feature with feature-linked annotation replaces both the feature and annotation. In this next example, some feature-linked annotation is repositioned in the current edit version. In the target version, Version A, the feature associated with this annotation is also moved. This triggers internal geodatabase behavior to reposition the related annotation by a commensurate adjustment. Version B is then reconciled against Version A. All four objects, the two features and the two related annotation elements, are subsequently highlighted as conflicts following the reconcile operation.



To resolve these conflicts, the same three choices are available: accept the target version's representation, choose the current edit version's representation, or revert to the pre-edit feature configuration. If the decision is to accept the target version's representation, then this action will delete the existing feature-linked annotation, insert the conflict feature, and create new annotation. Subsequent edit operations to reposition this new annotation may be required as necessary.

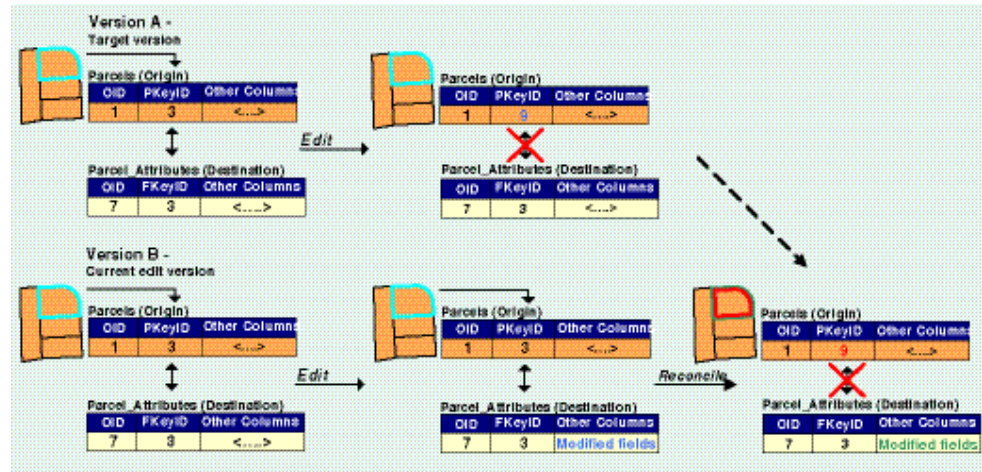


Relationship classes

Relationship classes have similar dependencies to feature-linked annotation. If the relationship is simple—that is, the related features or rows can exist independently—the foreign key value in the destination class will be nullified if a feature or row is deleted in the origin class. If the relationship is composite, deleting a feature from an origin class will trigger internal geodatabase behavior to delete the related feature or row from the destination class. As with feature-linked annotation, modifications to features that are related to other features or rows may introduce additional conflicts when two versions are reconciled.

In the following example, one of the primary key field values in an origin feature class is modified in version A. This edit breaks the relationship with the destination class, a

related attribute table. In version B, the related row in the destination class is updated. When the two versions are reconciled, given the dependency of rows in the destination class on features in the origin class, both updates are highlighted as conflicts. As geodatabase internal behavior notified both the origin and the destination class of the change to the affected, related row, both the feature class and the attribute table edits show up as conflicts.



To retain the relationship between the feature class and the table but keep the attribute change, the editor would accept the current edit version's representation of the data.

Geometric networks

The next section will concentrate on how geometric networks are affected by the reconcile process. A brief summary of the principal elements that make up a geometric network is included for reference. To learn more see the *ArcGIS Network Model* technical document in ArcObjects Developer Help or online at <http://arcobjectsonline.esri.com/>.

Networks maintained in a geodatabase have two main components: a geometric network and a logical network. The geometric network is the actual feature classes that make up the network; the logical network is the physical representation of the network connectivity in the database (the geodatabase tables that record all the information). Each feature in the geometric network is associated with one or more elements in the logical network.

Networks features include:

- Edges—for example, a water main. Edge features are related to edge elements in the logical network.
- Junctions—for example, a water valve. Junction features are related to junction elements in the logical network, and edges must be connected to other edges through junctions. Orphan junctions are created to hold edge endpoints that are not identified by another junction feature class, either during network building or editing.

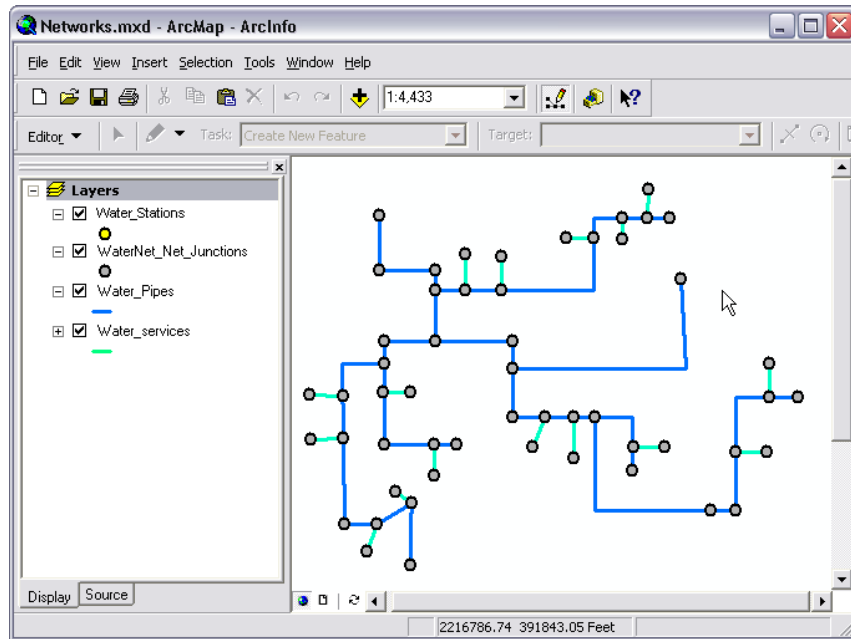
Network features are further categorized as either simple or complex:

- **Simple**—features correspond to a single element in the logical network. Simple edges are always connected to exactly two junction features, one at each end. If a new junction feature is snapped mid-span on a simple edge, establishing connectivity, then that simple edge feature is physically split into two new features.
- **Complex**—features correspond to more than one element in the logical network. Complex edges are always connected to at least two junction features at their endpoints but can be connected to additional junction features along their length. If a new junction feature is snapped mid-span on a complex edge, that complex edge remains a single geometric feature, although logically it has now been split.

The geodatabase automatically maintains the explicit topological relationships, or connectivity, between features in a geometric network. Connectivity is based on the geometric coincidence of participating network features; it is this connectivity that affects the version reconcile process.

When editing network features, changes to either the geometric features or logical elements may create conflicts. For example, by adding a service to a main, which is a complex feature, the main itself is not physically split into two smaller sections but has been logically split to accommodate the new service feature that is now topologically related to the main. So while the main's geometry has not been edited directly, its connectivity has been modified. If the target version for the reconcile operation has also modified the main, then the new service that has been inserted will create a conflict with the main.

The following examples illustrate how reconcile and conflict detection rules are applied to geometric networks. The data is a simple utility network, comprising complex edges (water pipes), simple edges (water services), and orphan junctions.



In all cases, conflicts have arisen as a result of two versions being edited independently and reconciled, during which connectivity is rebuilt. The current edit version, Version B, is a child of Version A, the target version (itself a child of the DEFAULT version). Version B is reconciled against Version A. The default behavior will be for the features in Version A to take precedence over the features in Version B.

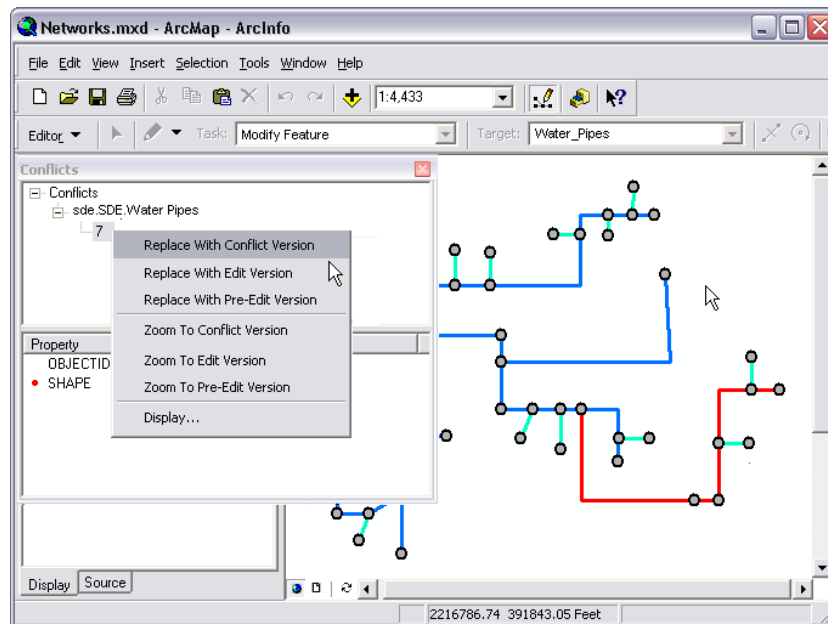
Note: It is always recommended that the changes made to a geometric network be saved before versions are reconciled—this avoids the possibility of any inconsistencies between the logical and geometric networks.

The conflicting features are identified as follows:
 Red—Version A, the target version’s representation
 Green—Version B, the current edit version’s representation

The features are identified with the following legend:

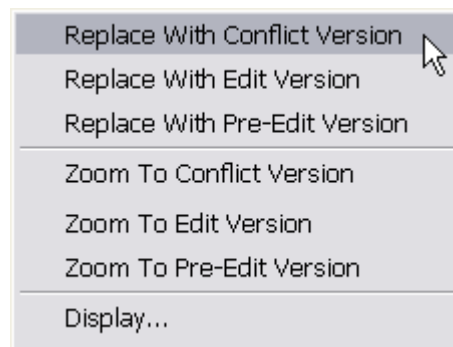


Where conflicts arise, they are inspected and resolved from the conflict dialog box in ArcMap.



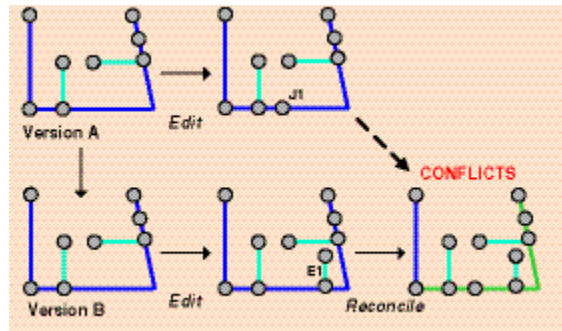
Editors have the choice of accepting the feature representation in the target version, accepting the representation in the current edit session, or reverting to the state of the data before the current edit session began.

In all cases, satisfactorily resolving the conflicts involving geometric network classes requires an understanding of how the Replace With command operates to update the network topology in the current edit version.



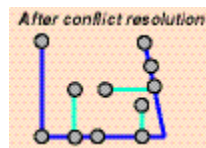
Resolving any conflicts will involve investigating the differences, determining if the conflict is valid, and deciding if further editing and reconciliation is required.

- Conflict 1*
- Version A—a standard junction, J1, is added.
 - Version B—a simple edge, E1, is added to the same edge.



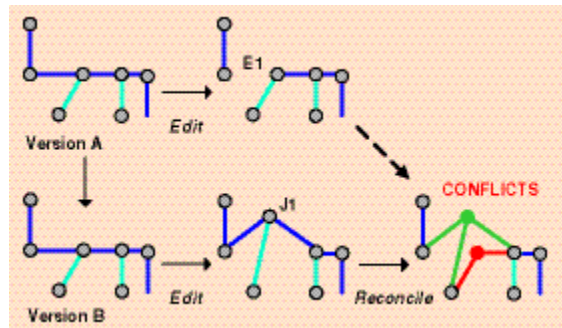
Conflict type = update/update

The addition of two new junctions results in a change to the logical network of the complex edge and introduces a feature conflict. To resolve this conflict, accepting the current edit version's representation will integrate the changes without requiring any post-reconcile editing.



Conflict 2

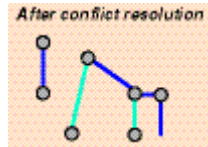
Version A—a complex edge, E1, is deleted.
Version B—an adjacent junction, J1, is moved.



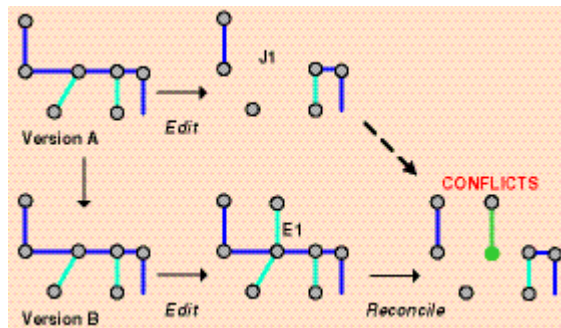
Conflict type = update/delete

All the modified features are reported as conflicts in the current edit version, and the modified complex edge, simple edge, and junction feature are reported as conflicts in the target version. Resolving the conflicts in this case will involve:

- Accepting the current edit version's representation for the repositioned water services (simple edge), water pipe (complex edge), and junction.
- Accepting the target version's representation for the deleted water pipe (complex edge).



Conflict 3 Version A—an orphan junction, J1, is deleted.
Version B—an adjacent simple edge, E1, is added.

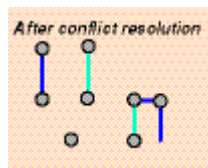


Conflict type = update/delete

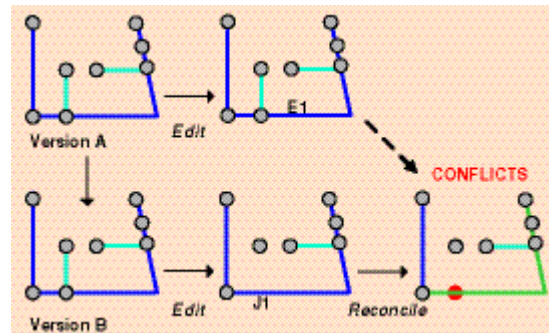
The new simple edge and deleted junction are reported as conflicts in the current edit version. Resolving the conflicts in this case will involve:

- Accepting the current edit version's representation of the new water services (simple edge) feature

This will leave an unconnected or standalone orphan junction, which resulted from the deletion of an edge feature. As junction features do not require a minimum number of connected edge features, junctions are not deleted when a connected edge is deleted.



Conflict 4 Version A—an attribute updated (ENABLED value) on a complex edge, E1.
Version B—an orphan junction, J1, on the same complex edge is deleted.

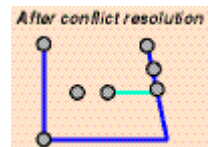


Conflict type = delete/update

The complex edge is returned as a conflict in the current edit session (change to the logical network), and the junction feature is reported as a conflict in the target version. Resolving the conflicts in this case will involve:

- Accepting the target version's representation of the modified water pipe (complex edge) attribute.
- Accepting the current edit version's representation of the deleted junction.

Again, this reconcile will leave an unconnected orphan junction, which may be deleted as required.



Topologies

The next section will concentrate on how topologies are affected by the reconcile process. A brief description of topologies and how they are managed in the geodatabase is provided for reference.

A geodatabase topology is a collection of rules and relationships, which, in conjunction with a set of editing tools and techniques, enables the geodatabase to accurately and realistically model the relationships between different geometry objects. Topology ensures data quality by providing the framework to control these relationships and maintain the geometric integrity between features.

The principal elements that make up a topology are:

- Rules—define the permissible relationships between features
- Rank—determines which features may be moved to other features when snapping the topology together during validation operations
- Cluster tolerance—defines how close vertices must be to each other to be considered coincident and limits the distance features can move during validation

- Clustering—part of the validation process; vertices that fall within the cluster tolerance are snapped together
- Cracking—part of the validation process; vertices are created at the intersection of feature edges
- Dirty areas—areas that have been modified. Dirty areas are created when:
 - A feature is created or deleted
 - A feature's geometry is modified
 - A feature's subtype is changed
 - Versions are reconciled
 - The topology properties are modified
- Errors and exceptions—violation of topology rules are stored as features in the database
- Validation—evaluates the features against the rules to identify any errors and removes errors related to rules or feature classes that have been removed or corrected

Versioning and topologies

In a versioned geodatabase the feature classes that participate in a topology do not have any special version reconciliation or conflict detection and resolution behavior. However, the dirty areas, error features, and exceptions that are created and maintained by the topology itself do have special version-related behavior, which is required to ensure the integrity of the topology. The nature of topology edits and the effects of the validate process (cracking and clustering) on feature geometries may introduce conflicts during the version reconciliation process.

The following sections on dirty areas, errors and exceptions, and conflict detection describe and illustrate the effects of a reconcile operation on dirty areas, errors, and exceptions. In each example, Versions A and B derive from a common ancestor; both have been modified since they were created, and Version B is reconciled against Version A.

Dirty areas

Dirty areas are the result of a spatial modification to the geometry of a feature or features. When an edit occurs, the area around the feature that was modified is marked as dirty, and a dirty area, whose extent corresponds to the bounding rectangle of the modified geometry, is automatically created. These dirty areas represent areas of change that have not been checked to ensure there are no violations of the geodatabase geometry rules. Dirty areas also allow the geodatabase to limit the area that has to be checked during the validation process; there is no mandatory requirement to revalidate an entire feature class if only a small section has been modified. In the context of versioning and version reconciliation, although it is not required to validate dirty areas before reconcile and posting, no topology operations can be run on the data until the data has been revalidated.

The general rules for dirty areas and version reconciliation are:

- If dirty areas existed in the parent and child versions before reconcile, the areas are unioned to create one large dirty area; the sum of the two extents must be revalidated.

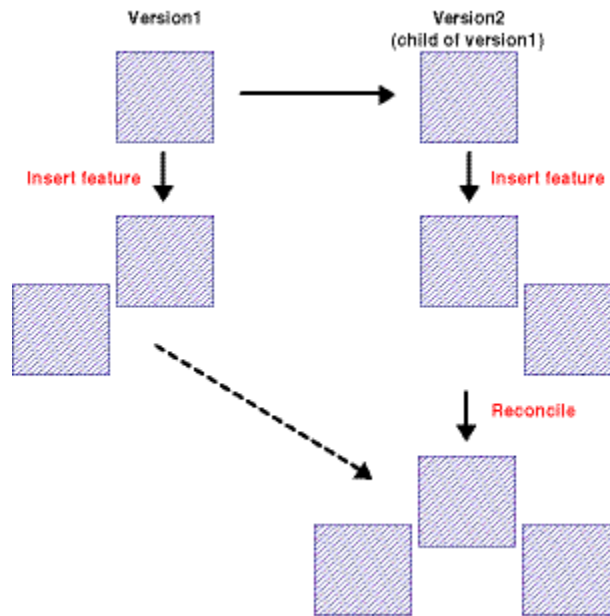
- Any dirty area the child version inherited from the parent version, whether it is validated in the child version or not, will be a dirty area after the reconcile process.
- Any dirty areas in the parent and child versions that had been validated are reactivated and intersected to create a smaller dirty area in the reconcile version; only the areas common to both dirty areas have to be revalidated.
- Any dirty area that was created for any feature that was created, updated, or deleted in the child version, whether it is validated or not, will be a dirty area after the reconcile. Reconciling two versions that do not contain active dirty areas may still result in dirty areas in the reconcile version.

During the course of a reconcile operation between two versions, new topology errors may be introduced despite the fact that the dirty areas within each individual version had been validated and were free of errors. As each version has no knowledge of the changes made in other versions, edits made to one feature class may violate one of the topology rules when reconciled with the target version. As a result, when two versions are reconciled, regardless of their error status, any areas of change are flagged as a new dirty area. The following samples will illustrate the specific logic that has been implemented in the geodatabase to reconcile dirty areas.

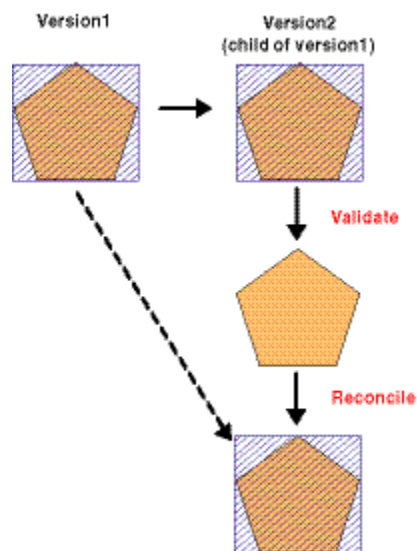
The legend below has been adopted for all examples.



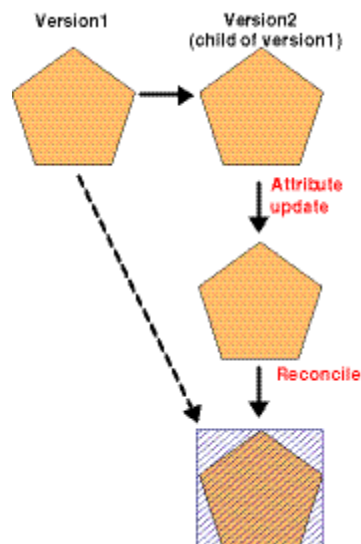
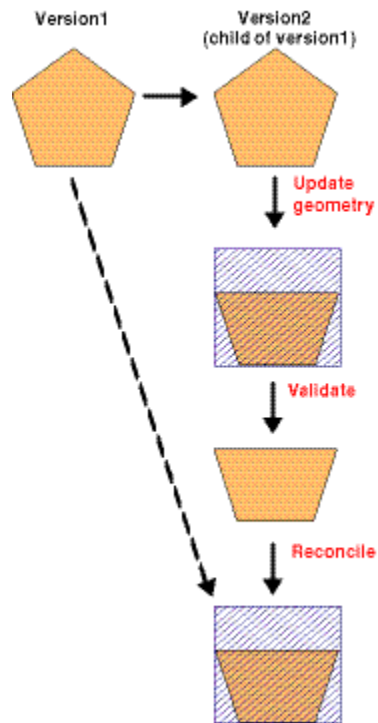
Any dirty areas present in the parent or child version that did not exist in the common ancestor—that is, they did not exist before the parent and child version were created and subsequently edited—will remain dirty as a result of the reconcile.



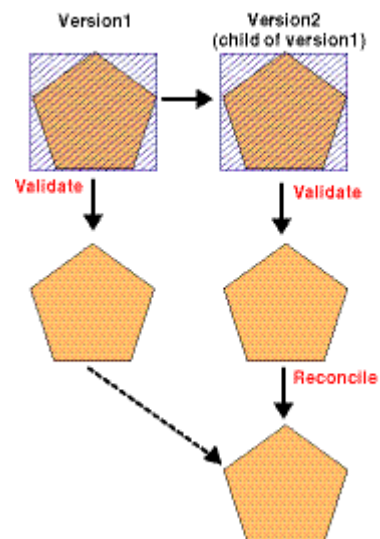
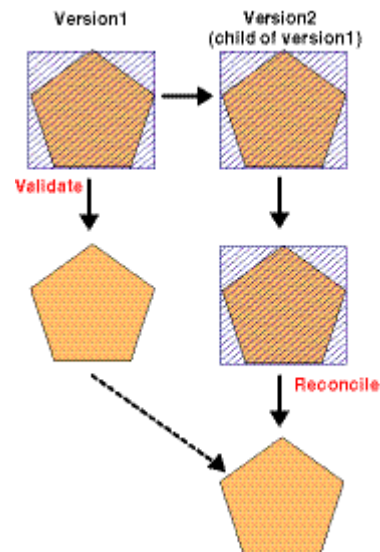
Any dirty area that was present in the common ancestor and validated in the child version will become dirty as a result of the reconcile process.

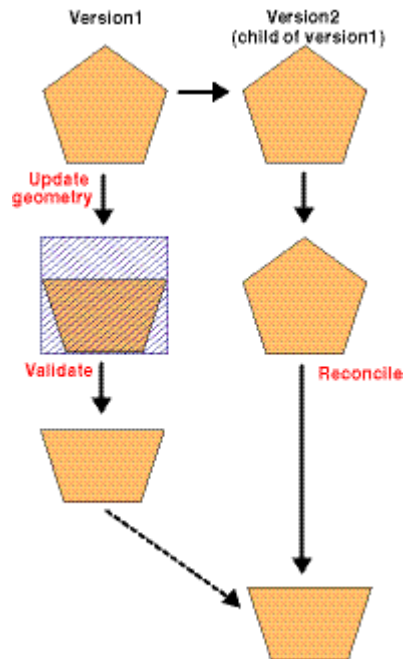


Any edits made to topology features in the child version will result in a dirty area after the reconcile operation, regardless of whether the dirty area resulting from the edit is validated in the child version. This is also the case when the original edit did not result in a dirty area, such as an attribute update.



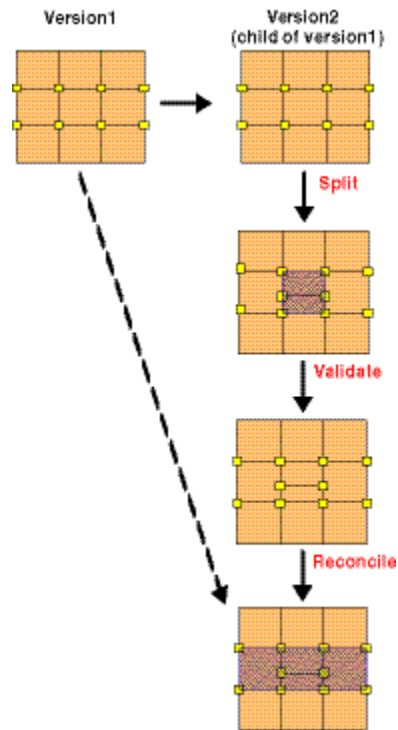
Any dirty area that is validated in the parent version, whether it was present in the common ancestor or created in the parent version, will remain validated as a result of the reconcile.





There are a number of scenarios in which reconcile can result in new dirty areas that did not exist in either the parent or child, due to cracking and clustering during the validate process. Both versions contain polygons that share edges in a topology that existed in the common ancestor. A polygon is split in the child version, and the dirty area is validated.

Splitting the polygon deletes the original feature and replaces it with two new ones. When the dirty area is validated, cracking and clustering introduce new vertices into the shared edges of the adjacent polygons.



When the versions are reconciled, dirty areas cover all of the features that have been modified in the child version—the split polygons and the polygons modified by cracking and clustering.

In the next example, new features were added to one of the participating feature classes in each version, and the resulting dirty areas were validated. The changes do not introduce any new topology errors in either version.

After the reconcile, dirty areas must be reactivated so any errors introduced by merging the changes from the two versions together can be discovered. In this case, a new topology error has been introduced—the no-overlap rule has now been violated as a result of this reconcile.

Errors and exceptions

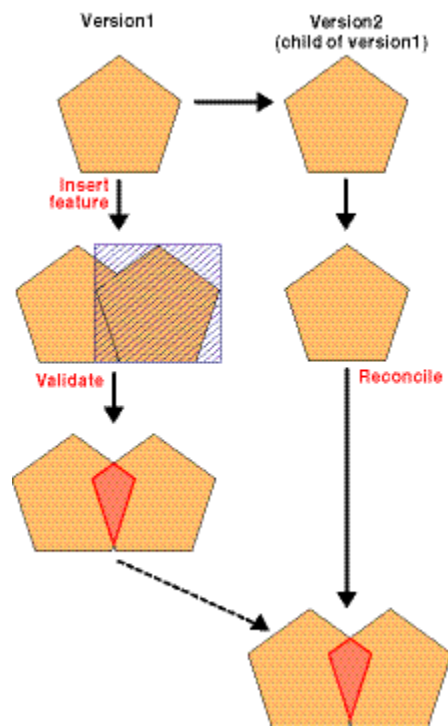
A topology error, or flag, is generated for each instance of a topology rule that has been violated. This is similar in concept to a network error, although the topology errors are not stored in any user-managed geodatabase table. They are managed instead by a special error feature class that is hidden from the user. Topology errors are created and managed by the topology and have an associated geometry that is used for display purposes.

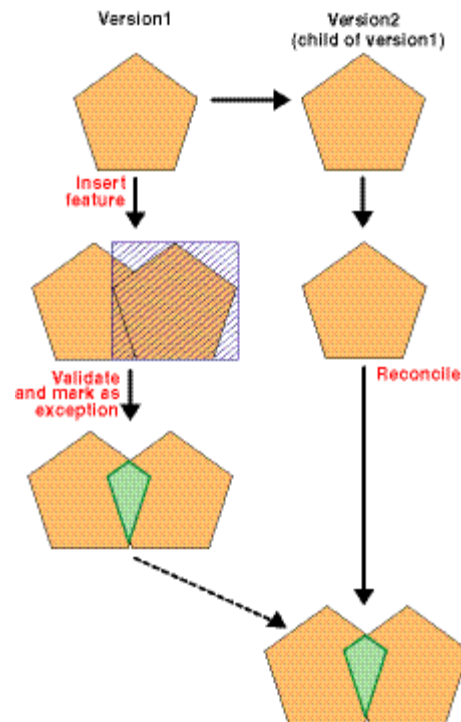
Error features and error features marked as exceptions have special behavior with respect to how they are treated during the version reconciliation process. As errors and exceptions cannot be edited directly, the reconcile process will not report conflicts for them between two versions. Errors are only created by the topology validation process and can be deleted by correcting the error using the topology error correction tools or by

editing features and using the validation process. Error features can only be updated by marking an error as an exception or by marking an exception as an error.

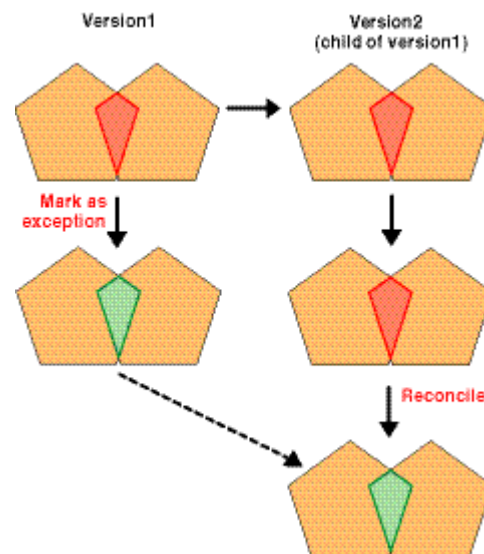
The following sections summarize the results of a reconcile operation on errors and error exceptions in the parent version. In each case, the errors have arisen when a no-overlap rule was violated.

Any error created in the parent version, whether or not it is marked as an exception, will be brought into the child version as a result of the reconcile; the parent version feature representation takes precedence.



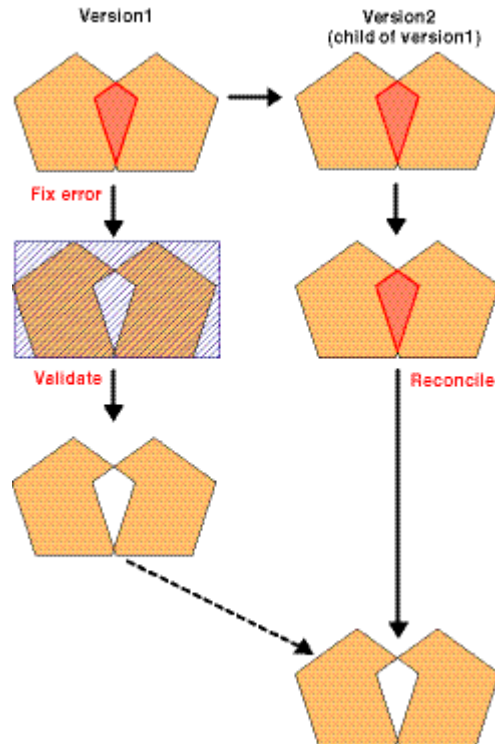


Any error that was marked as an exception in the parent version will still be marked as an exception after the reconcile; parent takes precedence.



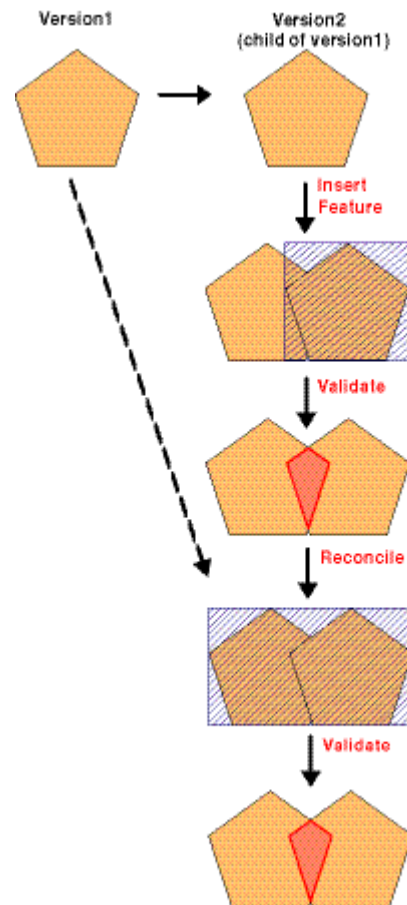
Any error or exception that is deleted in the parent version, either by fixing the error or by the validation process, will be deleted from the child version as a result of the

reconcile. This includes errors that were in the common ancestor and errors that were created in the parent version; parent takes precedence.

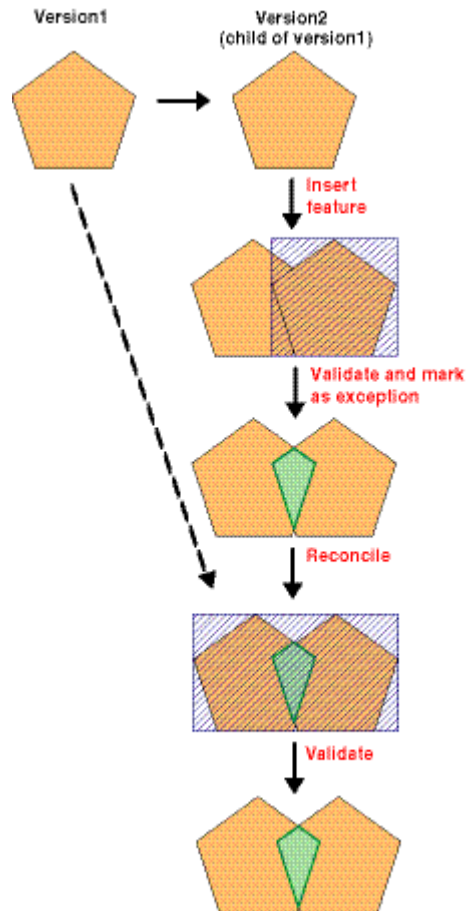


The next section summarizes the results of a reconcile operation on errors and exceptions in the child version.

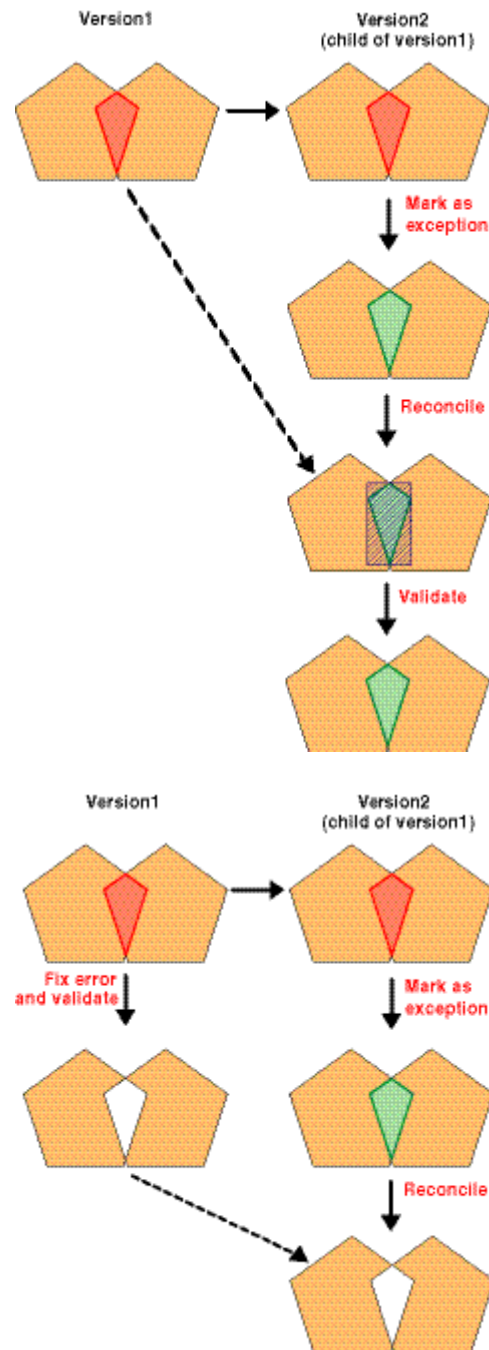
Any error created in the child version will be deleted as a result of the reconcile process. This area will be covered by a dirty area, which must be validated again.



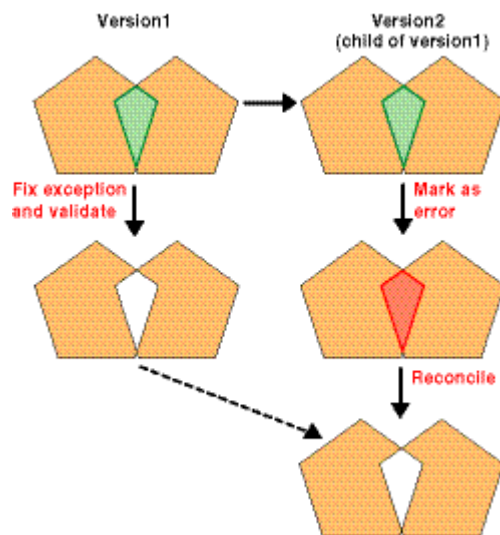
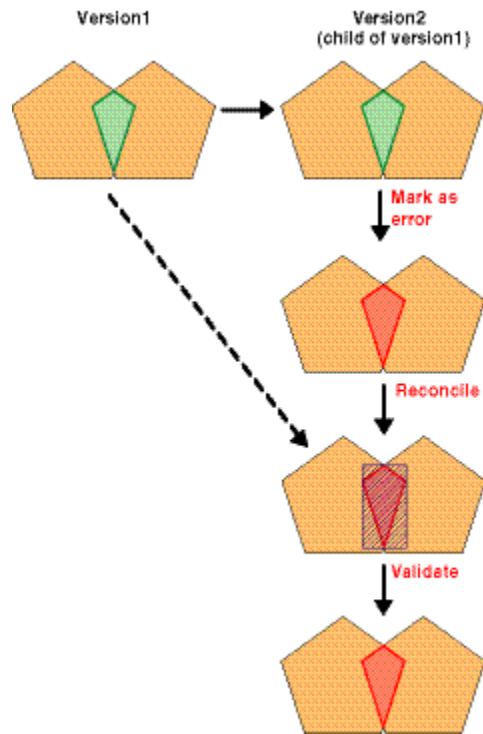
Any error created in the child version and marked as an exception will remain an exception as a result of the reconcile process. As in the previous example, a dirty area will cover the exception.



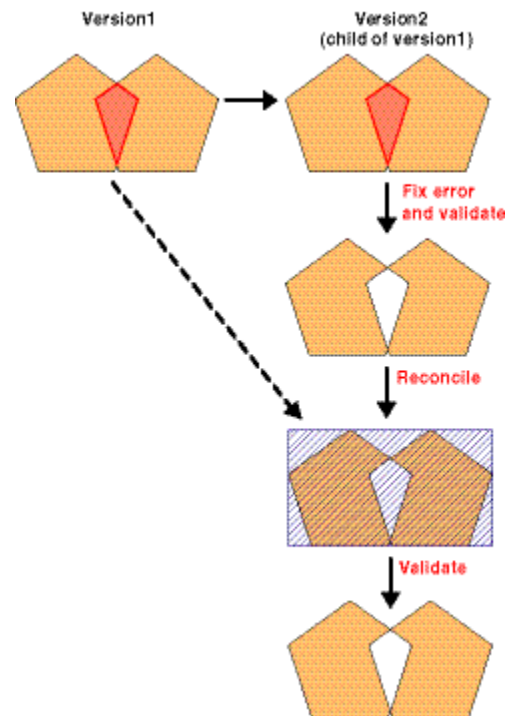
An error that existed in the common ancestor and is marked as an exception in the child version will remain an exception as a result of the reconcile process. It will be covered by a dirty area. However, if the error was deleted in the parent version, then it will remain deleted in the child version.

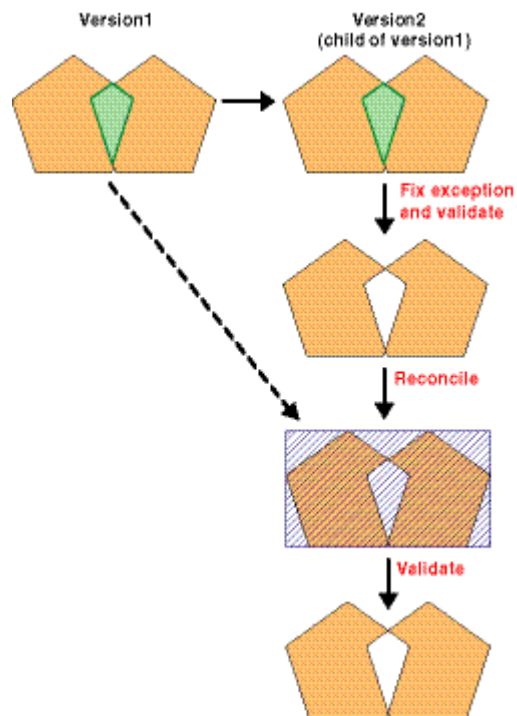


An exception that existed in the common ancestor, marked as an error in the child version, will remain an error as a result of the reconcile process. Again, a dirty area will cover it. However, if the exception was deleted in the parent version, then it will remain deleted in the reconcile version.

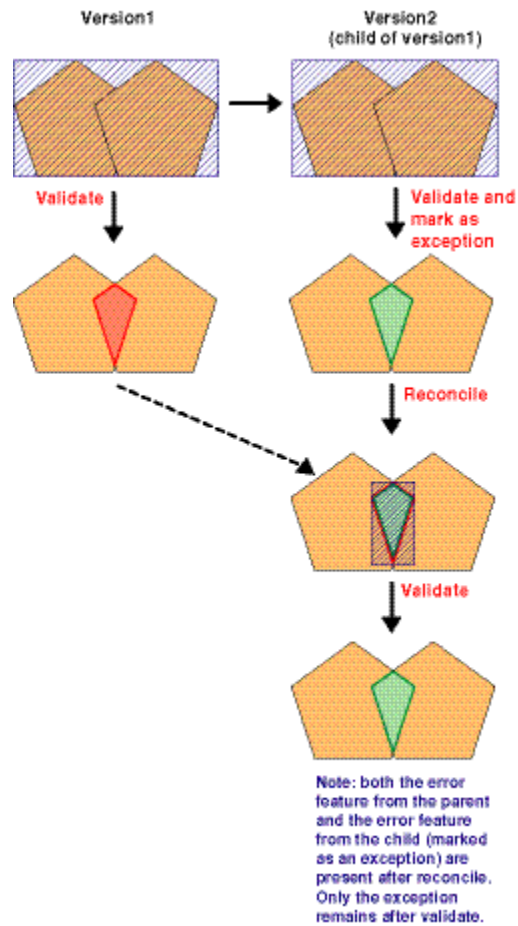


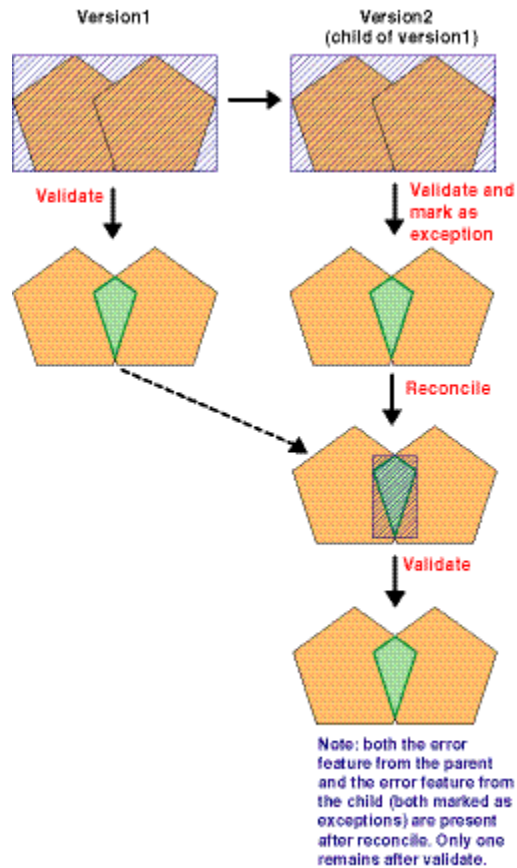
An error or exception that existed in the common ancestor and is deleted in the child version will remain deleted as a result of the reconcile.





There are cases in which the same error can be created in both the parent and child versions by validating a dirty area that existed in the common ancestor. If this error is marked as an exception in either the parent or child version, the reconcile will result in duplicate error features. In these cases, the error features will be covered by a dirty area and will be reduced to a single error or exception when the dirty area is validated.





Conflicting topology features

The preceding sections dealt with topology version reconcile scenarios where no conflict existed. The next section deals with cases where conflicts may be introduced during reconcile and how they are handled.

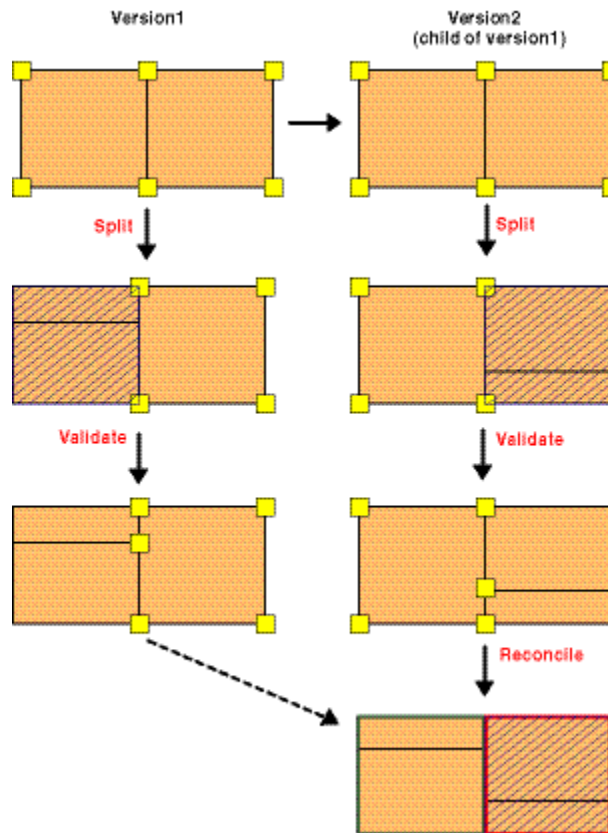
There is no special behavior for topology feature classes with respect to conflict detection and resolution. If the same feature is edited in two separate versions and those versions are reconciled, they will be in conflict, just the same as with nontopological data. However, when a topology feature class is edited, other topologically related data may be modified automatically at the same time.

The changed features may belong to the same feature class or to one or more other feature classes. New topology errors may occur when edited parent and child versions are reconciled; the cracking and clustering processes can introduce changes to the geometry that have to be revalidated, even when the dirty areas within each version have been validated and are free of errors. The type of edits that are made to topologies and the results of the validation process can potentially result in a large number of conflicts, so it is important to consider this when designing the data model and workflows.

The most common source of conflicts resulting from the validation process with topologies is the introduction of vertices due to the integration of features during the

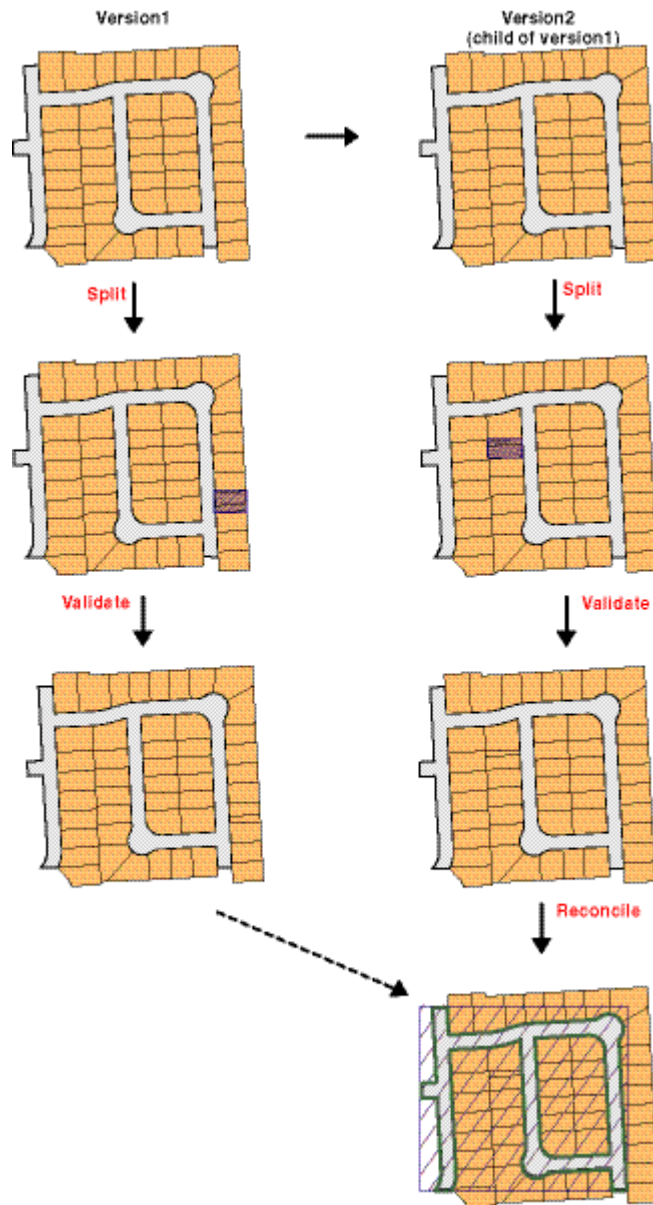
cracking and clustering phase of validation. The following examples illustrate how conflicts can result from the validate process.

Polygons that share edges in a topology in the parent version are inherited by the child version. One polygon is split in the parent version, an adjacent polygon is split in the child version, and the dirty areas are validated. Splitting the polygons deletes the original features and replaces each of them with two new ones. When the versions are reconciled, both original polygons are reported as update–delete conflicts. In other words, the feature deleted in the parent version was updated by the cracking and clustering process in the child version; the feature deleted in the child version was updated by the cracking and clustering process in the parent version.



In this example, the split parcel polygons share edges with a large right-of-way polygon. Cracking and clustering during validation alter the geometry of the parcel or right-of-way boundary; the reconcile process highlights these changes as a conflict in the child version.

A good data model design can help reduce the types of conflicts illustrated in the example below. Subdividing the right-of-way polygon into smaller sections could reduce the number of conflicts.



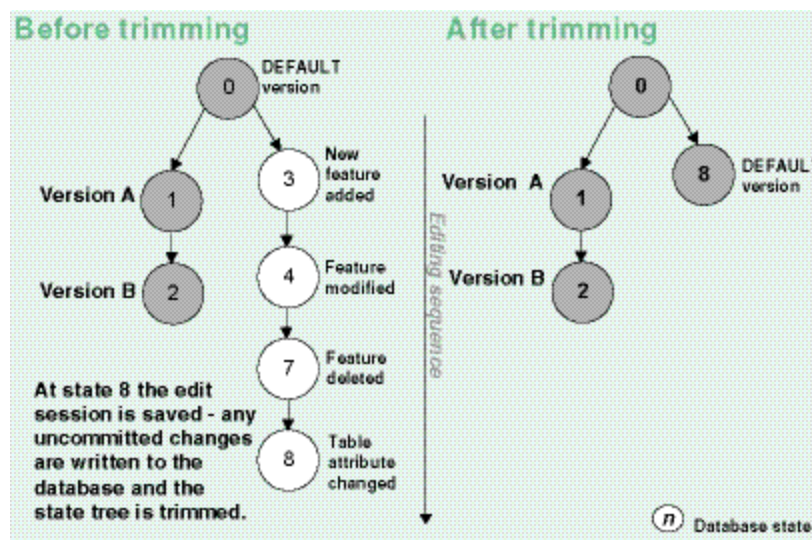
Generally, conflicts like those described here can be avoided by structuring the workflow so that editors working with topology data work in different areas. One option for workflow managers may be to use disconnected editing to control where users can and cannot edit.

Version administration

Previous sections have described how geodatabase versions evolve through a succession of database states: every edit operation creates a new state, which increases the complexity of the state tree that represents each version. If this state tree was allowed to grow unchecked, this would adversely affect database performance. To help keep the state tree manageable and to maintain acceptable performance, regular version and state maintenance is required. This is undertaken automatically by the ArcGIS application software and manually by SDE® administrators.

Database trimming (automatic)

The state tree is automatically trimmed every time a save operation is executed—for example during or at the end of an edit session. Trimming shrinks a linear branch of the state tree by removing states that are no longer required to represent a version. Child states will replace parent states within the branch, and all states that deviate from the direct path will be discarded. Trimming reduces the depth of the state tree and improves query performance.



Trimming also occurs automatically when the number of states in a lineage exceeds an application threshold determined by a registry setting—`HKEY_CURRENT_USER\Software\ESRI\Geodatabase\Settings\UndoRedoSize`. The default maximum number of states in a lineage is 30—above 30, the state tree is automatically trimmed. Although this application setting may be altered, the potential negative impact on performance should be considered.

State tree trimming does not occur when many editors are simultaneously editing one version—for example, the DEFAULT version. As the state trees of any active edit session are not trimmed, this can result in a number of orphan states.

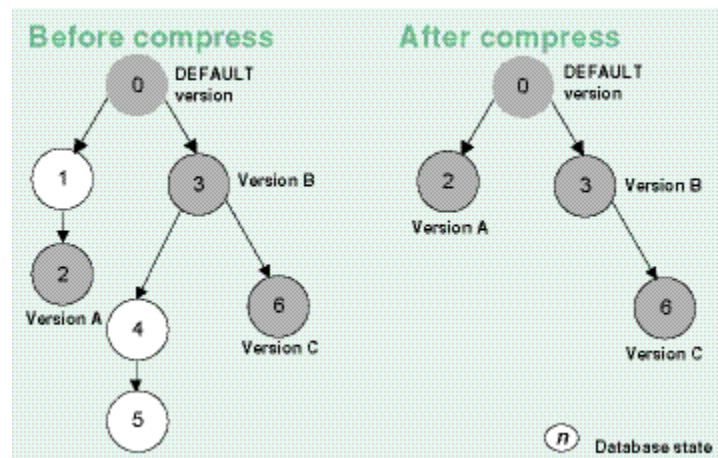
Trimming does also not occur when a child version is reconciled with the parent. To reduce the state tree after a round of version reconcile and post, the database should be compressed.

Database compression (manual)

To further improve performance, the state tree can be compressed. All versions reference a state, but not all states are referenced by a version. Compressing simplifies the state tree again by removing each state that is not referenced by a version and is not the parent of multiple child states. This operation will be executed against all states, regardless of owner, in the geodatabase and can only be performed by the SDE administrator.

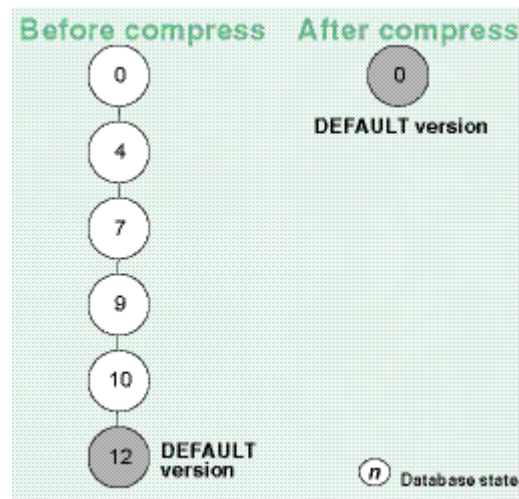
As with trimming, compressing reduces the depth of the state tree and shortens the state lineage, which will improve query performance. All states that are referenced by a version will remain following the compress operation.

As the data is edited, rows are added to the delta tables to record each change made to the version. Many of these rows are subsequently superseded by more recent changes to the data, and they are no longer required to represent the current state of a version. Compressing a versioned database also eliminates these redundant rows from the delta tables and moves all rows that are common to all versions into the base tables. This has the two-fold benefit of helping to conserve disk space and again improving query performance. All versioned geodatabases should be compressed periodically—the more dynamic the data, the more frequently the database should be compressed.



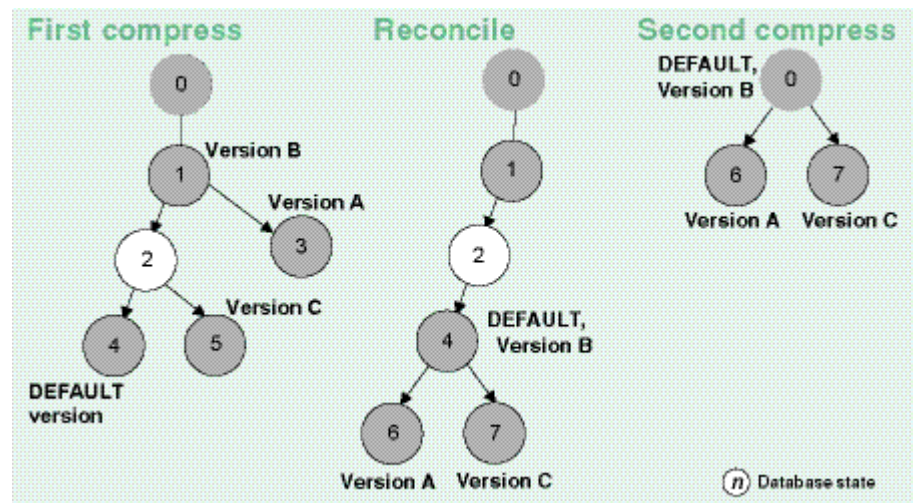
Note: At the 9.x release of ArcGIS, a compress operation will no longer require an exclusive lock to prevent inconsistent reads on the database. Compress will trim state trees and purge the delta tables for all versions that are not currently being edited. The state trees of any active edit sessions are not affected.

To achieve the optimum versioned geodatabase state configuration when running the compress—that is, no rows in the delta tables and the state tree trimmed back to 0—any outstanding changes to child versions should be reconciled and posted to the DEFAULT version and the versions themselves deleted.



However, given operating constraints imposed by organizational workflow requirements, this is not always possible. For example, organizations that must maintain historical versions will not be able to compress back to state 0. Although this state is desirable, it is not essential to good database performance. Trimming the state tree is the key to good performance; a regular database compress is recommended if the data changes frequently.

Before compressing the database, reconcile and post each version with its parent version, propagating the changes through the version tree to the DEFAULT version. Then reconcile and save each version again to ensure all versions now have the latest snapshot of the now modified DEFAULT version. Finally, perform the compress; this will flush all edits for the DEFAULT version, currently in the delta tables, to the base table.



The compress can still be executed without first reconciling, posting, and (optionally) deleting each version, but without a comprehensive reduction of the state tree and purge of the delta tables, there will be no significant performance improvements.

The results of each compress operation are written to a log, `SDE.compress_log`, maintained in the database by ArcSDE. This log records statistics on the compression start time, end time, the number of states that have been compressed, and the status of the operation. After running a compress on the database, it is recommended that the DBMS statistics be updated by running the Analyze command.