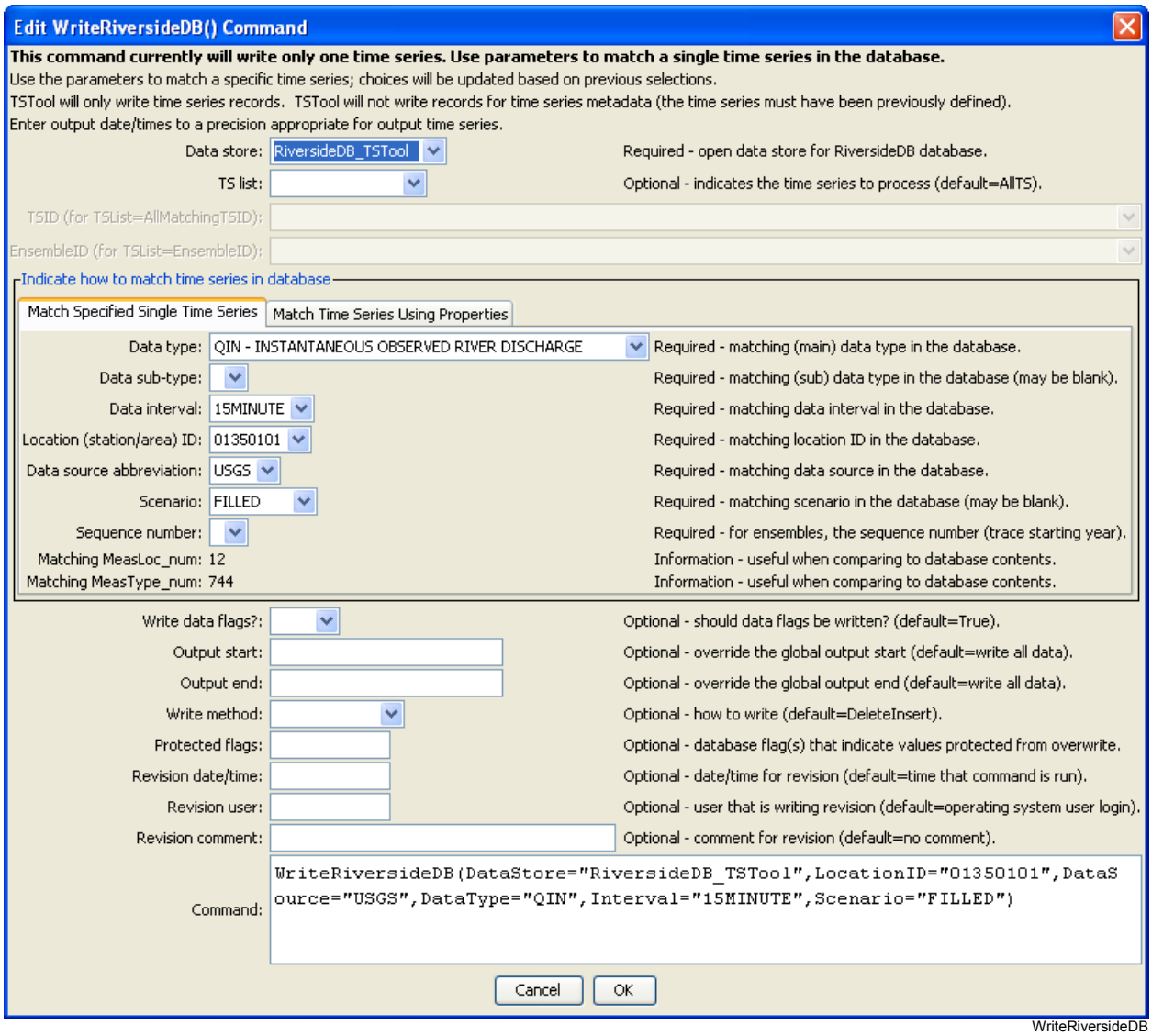
Command Reference: WriteRiversideDB()

Write time series to a RiversideDB database

Version 10.06.00, 2012-04-15

This command is under development – please provide feedback to developers. The `WriteRiversideDB()` command writes time series to a RiversideDB database. See the **RiversideDB Data Store Appendix** for more information about the database features and limitations. The command will not define a new time series but will replace or insert the data records for an existing time series. The current functionality allows a single time series to be written by matching a specific existing time series in the database; however, in the future time series properties may be used to write multiple time series with one command (e.g., use `%L` to match the location).

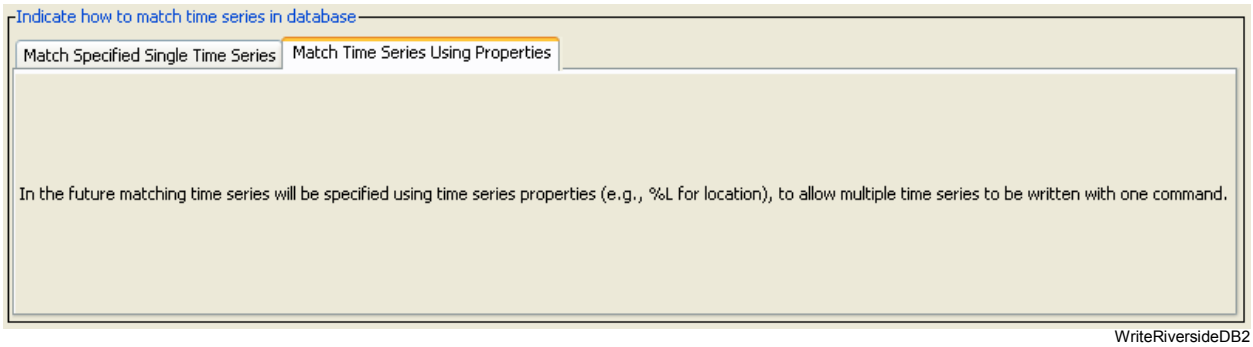
The following dialog is used to edit the command and illustrates the syntax of the command when writing a single time series to the database. In this case the choices are used to select a matching time series and only one time series can be in the list to process (otherwise a warning will result and nothing is written).



WriteRiversideDB

WriteRiversideDB() Command Editor when Matching/Writing a Single Time Series

The following figure illustrates use of the ***Match Time Series Using Properties*** tab, which is envisioned as a future enhancement. In this case, the parameter values do not match a specific time series in the database but instead use the properties from the list of time series being written to indicate how to match a time series in the database. For example, `DataType=%T` would use the data type from the time series to match the data type in the database and `Location=%L` would use the location identifier from the time series to match the location identifier in the database. Writing multiple time series requires fewer commands but the command editor will not be able to confirm that the time series are matched in the database (this can only occur when running the commands and data are actually processed).



WriteRiversideDB() Command Editor when Matching/Writing Multiple Time Series

The following technical issues apply when writing time series:

- Currently there is no authentication to prevent users from using this command. Options will be explored based on specific system requirements.
- Time series being written must have compatible units with the units of the time series in the database. The same units or conversion factor of 1.0 must be detected to allow writing.
- Missing values in time series are written as missing values (null) in the database. This allows missing values in the database to have flags.
    - Regular interval time series could be written in such a way that missing values are simply not written to the database. If this is required, then a new parameter `WriteMissingValues` could be added. However, tracking revisions might be difficult using this approach because older values would need more complex handling since no newer missing value record would be present.
    - Irregular interval time series in TSTool do not have missing value records unless they were specifically read from input.
- Irregular time series present challenges that may not be fully addressed with the current software features and may require additional enhancements. For example, `WriteMethod=TrackChanges` may not work properly if updates to irregular data have different date/times than the original values. In this case `WriteMethod=DeleteInsert` may be more appropriate; however, this does not check the `ProtectedFlags` parameter.

The command syntax is as follows:

```
WriteRiversideDB(Parameter=Value,…)
```

Command Parameters

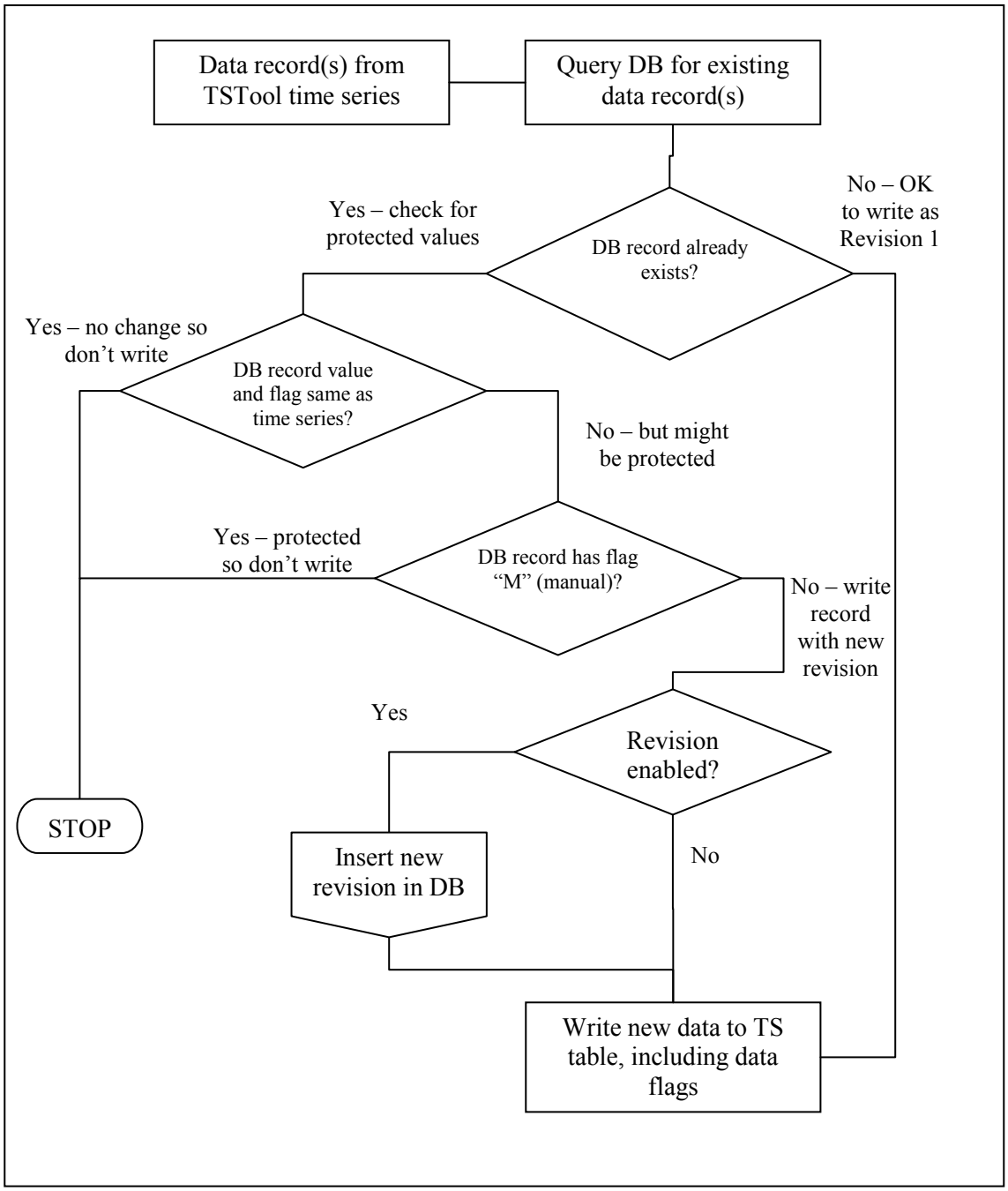
Parameter	Description	Default
DataStore	The identifier for the RiversideDB data store to use for the database.	None – must be specified.
TSList	Indicates the list of time series to be processed, one of: • AllMatchingTSID – all time series that match the TSID (single TSID or TSID with wildcards) will be processed. • AllTS – all time series before the command. • EnsembleID – all time series in the ensemble will be processed. • FirstMatchingTSID – the first time series that matches the TSID (single TSID or TSID with wildcards) will be processed. • LastMatchingTSID – the last time series that matches the TSID (single TSID or TSID with wildcards) will be processed. • SelectedTS – the time series are those selected with the SelectTimeSeries() command.	AllTS
TSID	The time series identifier or alias for the time series to be processed, using the * wildcard character to match multiple time series.	Required if TSList=*TSID.
EnsembleID	The ensemble to be processed, if processing an ensemble.	Required if TSList=EnsembleID.
WriteMode	This parameter is envisioned if writing multiple time series is enabled in the future (currently only WriteSingle is enabled). Indicates how time series are being matched, one of: • WriteSingle (the only mode that is enabled), in which case the ***Match Specified Single Time Series*** tab is used to enter command parameters. • WriteMultiple (future enhancement), in which case the ***Match Time Series Using Properties*** tab is used to enter command parameters. Ensemble time series may fit into either of the above depending on how the sequence numbers are handled.	WriteSingle
DataType	The data type abbreviation in the database to match.	None – must be specified.
DataSubType	The data sub-type in the database to match (may be blank).	Not used if blank.
Interval	The data interval in the database to match.	None – must be specified.
LocationID	The location identifier in the database to match.	None – must be specified.
DataSource	The data source abbreviation in the database to match.	Not used if blank.
Scenario	The scenario in the database to match (may be blank).	Not used if blank.
SequenceNumber	Used for ensembles – the trace number, often the starting year of the trace (may be blank).	Not used if blank.
WriteDataFlags	Indicate whether data flags should be written.	True

Parameter	Description	Default
`OutputStart`	The date/time for the start of the output.	Use the global output period.
`OutputEnd`	The date/time for the end of the output.	Use the global output period.
`WriteMethod`	The approach for writing time series, one of: • `Delete` – delete the time series records but do not write the time series (useful for testing and database maintenance) • `DeleteInsert` – delete time series records and then insert. Revision data are not inserted into the database. • `TrackRevisions` – track revisions as per the logic described after this table.	None – must be specified.
`ProtectedFlags`	Indicate the flag values for database data records that should NOT be modified, when `WriteMethod=TrackRevisions`. Typically these records indicate that data have been validated and/or manually entered and automated processes should not overwrite the values. Currently only one flag value can be specified; however, multiple values or patterns may be supported in the future.	None – no data will be protected.
`ComparePrecision`	The number of digits after the decimal used when comparing previous database values with values in the time series. This may be required where input/output operations result in truncations or round-off.	`4`
`RevisionDateTime`	The date/time for a new revision, if needed, one of: • `CurrentToSecond` syntax – see `SetOuputPeriod()` command for more information. • `${Property}` – property value string • `YYYY-MM-DD hh:mm` or similar date/time string	Time from computer system clock at the time the command is run.
`RevisionUser`	User identifier for revisions. Currently this is NOT taken from the RiversideDB user table. `${Property}` notation can be used.	User's login from the operating system.
`RevisionComment`	Comment for revisions. `${Property}` notation can be used.	No comment.

Revision information will be utilized only if configured in the database. The time series data table must contain a Revision_num column and the Revision table must exist. The time series table layout information will indicate whether revision numbers are used. If `WriteMethod=DeleteInsert` and `RevisionComment` is provided, then all old data will be deleted (all revisions) and a new revision will be added corresponding to all inserted records. This ensures that the database size does not grow quickly. If `RevisionComment` is not specified, then a revision will not be added in the database (revision will be set to zero, which corresponds to the "original data" revision).

The following figure explains the logic when `WriteMethod=TrackRevisions`, which can be used when RiversideDB is configured with time series tables that use the Revision_num column. The following flow chart focuses on the case where a data record to write to the database (consisting of a date/time, a value, and a flag) already exists for the date/time. This indicates that this data record

previously has been written to the database. In this case, the existing data record in the database should be overwritten with the new record unless the existing record is flagged as protected (in this example, manually adjusted with `ProtectedFlags=M`). In this case, the manually adjusted value persists and can be overwritten only if the new data record also is flagged with `M`, indicating that an additional manual adjustment has occurred. The following figure illustrates the logic performed for each value. However, several steps are performed in bulk to improve performance.



Logic for Writing Time Series Values with Revisions (ProtectedFlags="M")

Revisions to existing data records are stored in the Revision table in RiversideDB. Original data records in the time series tables have a revision number of '1', which refers to the revision number '1' in the Revision table. Any time a new revision is needed due to changes in a data value, a new entry is created in the Revision table, populated with the pertinent information (date/time of revision, user, and a comment) and the corresponding revision number is assigned to the revised data record in the time series table. Revision numbers are not incremented for each data value but are incremented for the bulk database operation (similar to a commit in content management systems). Care should be taken in using `WriteRiversideDB()` commands in order to minimize the number of revisions that are tracked. For example, rather than relying on the default value for the `RevisionDateTime` command parameter, a better approach may be to define a property at the top of the command file using a `SetProperty()` command and then refer to that property when specifying the `RevisionDateTime` property. This will ensure that multiple `WriteRiversideDB()` commands in a single command file utilize the same revision information.

If a data revision is detected based on the logic in the above figure, the corresponding revision will be searched for in the Revision table. If not found, then a new revision record will be inserted into the Revision table and that information will be used in the time series data table.

Time Series Table

MeasType_num	Date_Time	Val	Revision_num	Quality_flag
1	2011-04-11 14:35	585.98	1	
1	2011-04-11 14:50	586.02	1	
1	2011-04-11 15:00	585.98	1	
1	2011-04-11 15:10	585.98	1	
1	2011-04-11 15:10	585.97	2	M
1	2011-04-11 15:15	585.99	1	
1	2011-04-11 15:15	585.98	3	M
1	2011-04-11 15:15	585.97	4	M
1	2011-04-11 15:25	586	1	

Revision Table

Revision_num	Date_Time	User	Comment
1			NO REVISION
2	2011-04-11 17:00	JP	Manual Change
3	2011-04-11 17:01	JP	Manual Change
4	2011-04-11 18:01	JP	Manual Change after review

TSTool commands that read RiversideDB time series, such as the `ReadRiversideDB()` command, sort time series data records by the Date_Time and revision number (Revision_num) prior to transferring the data into data objects. Consequently, the entry with the largest revision number for records at the same sensor (MeasType_num) and time (Date_Time) is used as the valid data record because the record will be processed last. In the above example, this is the record with Revision_num `4` and a value of `585.97`. This approach may result in performance degradation over time if many revisions are made to data values for the same date/time and consequently it may be appropriate to remove or archive very old revisions as part of database maintenance. The trade-off between performance and the ability to track revisions may vary between systems and in general there should be few revisions for the same data point because data loading will move forward through time without reloading the entire period.