WINDOGRAPHER DATABASE INTERFACE SPECIFICATION

Authors:	Tom Lambert, Windographer Team Lead, UL Renewables
	Tom Ferguson, Senior Software Engineer, UL Renewables

Applies to:	Windographer Database Interface revision 7
Document revision:	3
Date:	November 2022

1. DEFINITIONS

This document uses the term ***data point*** to mean a measurement of some variable (e.g. wind speed, wind direction, or temperature) made over a certain time interval and recorded for later analysis. The ***time step*** is the interval of time to which a data point corresponds (typically 10 minutes with measured data from meteorological towers). A ***data column*** is a chronological series of data points. A data column could contain, for example, the recorded mean wind speeds in each ten-minute time step from April 4 to July 28.

In the wind power industry, a data logger typically measures a particular variable many times within each time step, and then records four data points for that time step: the mean, minimum, maximum, and standard deviation of those measurements. The resulting four data columns are therefore closely associated with each other. This document uses the term ***data column group*** to refer to such a group of related data columns. This document will refer to the mean as 'the parent' and the other three (minimum, maximum, and standard deviation) as the 'children'.

The term ***dataset*** refers to a set of data columns that have a logical grouping. For meteorological data, this will typically consist of recorded measurements made at a particular location. This type of dataset will correspond to one measurement campaign made at a certain place by a meteorological tower or a SoDAR or LiDAR device. However, a database administrator might also choose to create datasets that consist of all of the measurements for a particular instrument, even though it has been installed in different locations.

The term ***database*** refers to a relational data storage system that can house multiple datasets. Some typical examples are SQL Server, Oracle, MySQL, and PostgreSQL.

A ***flag*** is a category that the Windographer user can use to mark and filter data points. A ***flagged data segment*** or simply ***flagged segment*** is a set of contiguous data points to which someone has applied a flag.

A ***stored procedure*** is a subroutine contained within a relational database system that outside applications can call to interact with the database.

* A note about the example statements found in this document: We mean for these examples to convey the gist of what Windographer will do, not to portray the precise syntax of commands sent to the database. In other words, these examples show the information (field names, parameter values, etc.) that Windographer will send to the database or will expect to receive from the database. The precise syntax will vary according to the database platform (SQL Server vs. MySQL vs. Oracle, etc.). The commands that Windographer actually sends to the database appear in the status window at the bottom of the Import From Database window.

OVERVIEW

This document describes how the Windographer user will import from and export to a central database of wind resource data and explains the requirements that the database must meet to enable this integration.

In Windographer a dataset comprises:

1. dataset properties (ID, name, longitude, latitude, elevation, change permissions, etc.)
2. data columns
3. data column properties (data type, name, units, measurement height, etc.)
4. a list of the properties of the flags used in the dataset (name and color of each)
5. a list of flagged segments
6. calibration history for each column of data
7. reviewer information (names of analysts who have reviewed the dataset and the periods they reviewed)
8. the dataset history (a list of all modifications made to the dataset and any notes about the dataset)

The database import and export processes encompass all 8 of these data types, though the last five are optional. If the database stores all eight types of data, one user can import a dataset from the database, flag some segments to indicate icing or tower shading, then write the data back to the database, and then another user can import the same dataset and see those flagged segments and the full list of modifications made to the dataset.

Because Windographer communicates with the database via stored procedures, it does not need to know anything about the structure of the database. In other words, Windographer does not need to know the names or structures of the tables that the database uses to store data. When Windographer calls the stored procedure to retrieve a particular type of data, that stored procedure queries the database as necessary and returns the data to Windographer in the format that Windographer requires. The burden of writing the stored procedures rests with the database administrator.

We have successfully tested import from and export to Microsoft SQL Server, MySQL, and PostgreSQL, and we have also successfully tested import from an Oracle v10.2 database. We have done most of our testing against SQL Server, and this document reflects that. Since Oracle recommends creating stored procedures and functions inside ‘packages’, Windographer's interface to Oracle databases looks slightly different. Instead of making calls to stored procedures like this:

```
Exec windog_ReadDataColumnPropertiesR2 'K5DO3AL9'
```

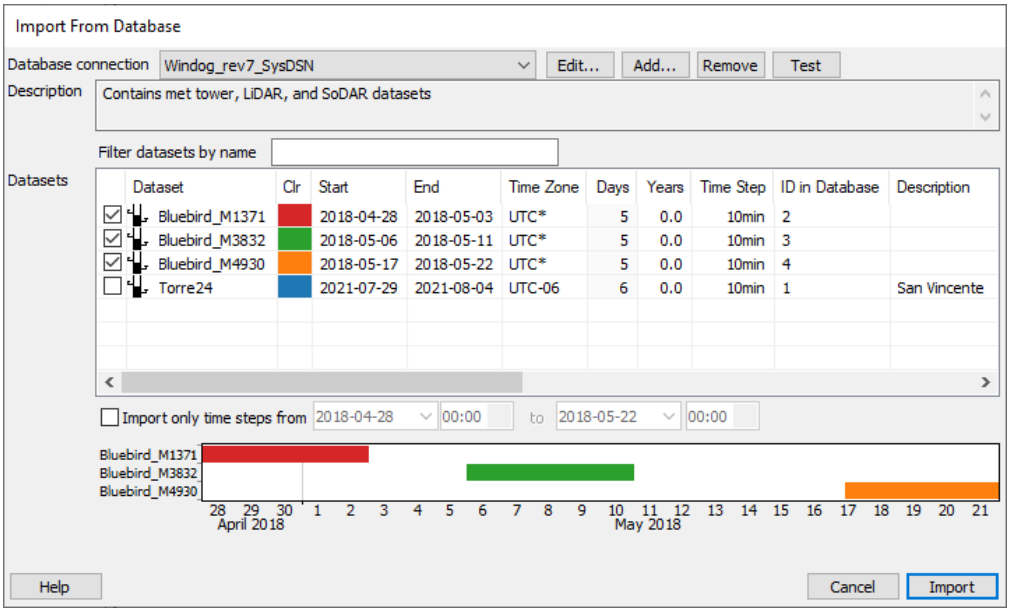
Windographer will expect the stored procedures to exist in a ‘WINDOG’ package, and then remove the ‘windog_’ prefix from each procedure name. So, the call to the ‘windog_ReadDataColumnPropertiesR2’ stored procedure will look like this for Oracle databases:

```
Call windog.ReadDataColumnPropertiesR2('K5DO3AL9')
```

2. User Interface

2.1 Importing Data

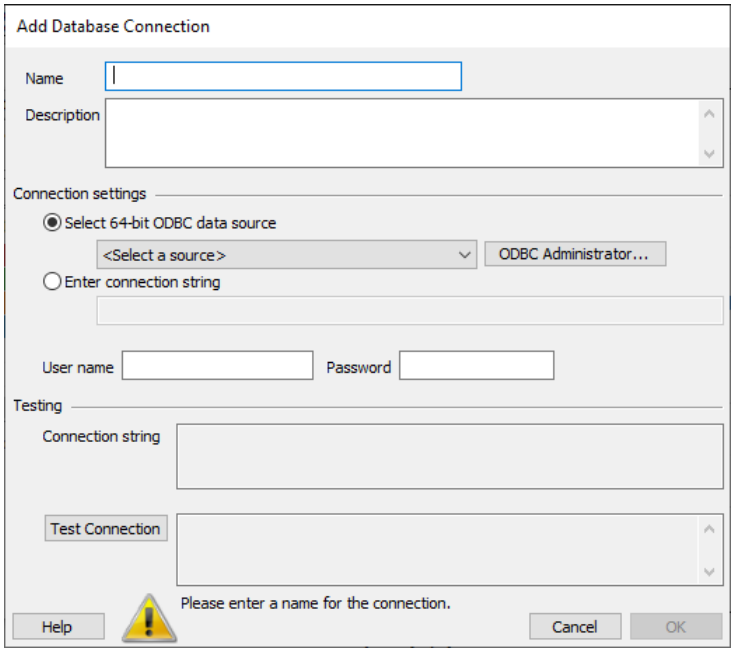
When the Windographer user chooses File > Import From Database, the following window appears:



This window serves as the main interface for importing data into Windographer from a database. The database connection drop-down box lists the database connections that the user has configured. We expect that in most cases that first drop-down box will contain only one item. The Edit, New, Remove, and Test buttons all refer to Windographer database connections. They allow users to change, add, delete, and test connections, respectively.

Check one or more datasets in the list, then click Import to import those datasets (or just the subset data range that you specify) into Windographer.

If you have not yet configured a database connection, clicking 'Add…' to cause the Add Database Connection window to appear, a screenshot of which appears below. Select from the list of ODBC data sources or else specify a connection string. Your database administrator should be able to help with this process. Click 'Test Connection' to confirm that the connection string or ODBC data source is working.



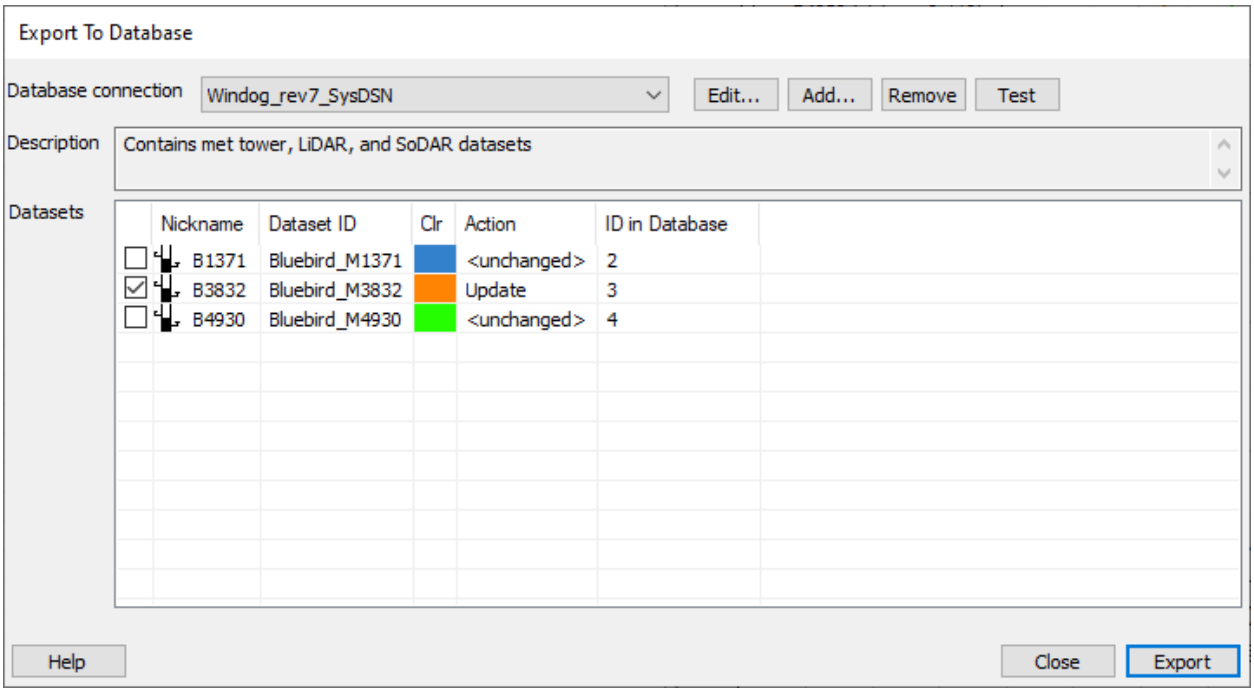
TESTING CONNECTIONS

Notice that both the Add Database Connection window and the Import From Database window have buttons that allow you to test a connection. The Test Connection button on the Add Database Connection window will attempt to connect to the database with the parameters you have supplied. If the connection succeeds, Windographer will show many connection parameters in the text box next to the Test Connection button. If not, Windographer displays errors there instead. Text that appears in this text box can be copied and pasted to help with diagnostics. **Please be careful when sending this information as this text may contain sensitive information like user names and passwords.**

The Test button on the top right of the Import From Database window conducts the same test as described above, plus it tests for the existence of the expected stored procedures. It runs the test on the connection displayed next to the Database label on the left. As above, the test results will appear in the text box at the bottom of the window, and can be selected and copied to help with diagnostics.

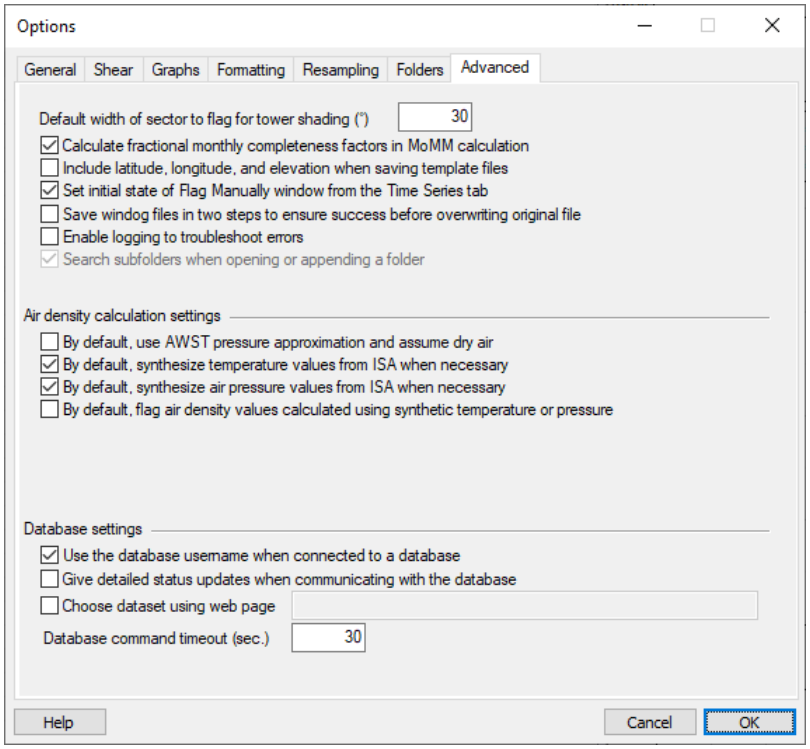
2.2 EXPORTING TO THE DATABASE

After having imported from the database one or more datasets, and/or importing from data files one or more datasets, the user can then choose to export some or all datasets to the database using File > Export to Database, which brings up the Export To Database window:



2.3 OTHER DATABASE INTERFACE OPTIONS

A few other interface features affect Windographer's communications with the database. The most important of these are the options listed in the 'Database settings' section at the bottom of Tools > Options > Advanced:



If the user selects the 'Use the database username when connected to a database' option, then Windographer will use the database username (the username used to make the connection to the database) for document history events. The checkbox labeled 'Give detailed status updates...' controls the detail level of the status updates as Windographer communicates with the database. The detailed setting might be helpful when initially setting up communications with the database, or if you experience any trouble with importing from or exporting to the database. Windographer will use the time entered in the 'Database command timeout' field to determine the length of time (in seconds) it waits for communication with the database. If your stored procedures start returning values but don't finish, you can increase this value to give the database more time to finish.

The 'Choose dataset using web page' option is an advanced feature in Windographer Enterprise that allows a company to specify how users select datasets using a web page designed by the company. The web page completely replaces the controls on the Import From Database window. For details about using this option, please contact Windographer support.

3. Reading Database Specifications

Windographer begins its communication with the database by calling the stored procedure `windog_ReadSpecifications`, which must return a recordset containing one record and at least 2 integer fields. As the table below summarizes, the first field indicates the revision number of the Windographer interface that the database implements, and the second specifies which of three possible methods Windographer should use to query data columns from the database. The interface revision number will increment with future changes, but the latest revision number is 7, so the first field of this recordset must return a value equal to or less than 7. This document applies to interface revision 7. Older documents describe older interface versions. The second field can contain a value of 0, 1, or 2 depending on how the database administrator wants Windographer to query data.

Table 1 – Structure of Recordset Returned by `windog_ReadSpecifications`

Field Name	Type	Meaning	Possible Values
interface_version	integer	Database interface revision number	An integer between 1 and 7
data_column_query_method	integer	Data column query method	If 0, query each data column separately. If 1, query all data columns at once. If 2, query columns separately using a time step number.

Returning 'Recordset' Variables to Windographer

Using SQL Server, a person can return a recordset to Windographer by including a 'SELECT…' statement directly in the stored procedure. For example, a stored procedure like the following would fulfill Windographer's requirement for `windog_ReadSpecifications`.

```
CREATE PROCEDURE [windog_ReadSpecifications]
AS
BEGIN
 SELECT 5 as interface_version, 1 as data_column_query_method;
END
```

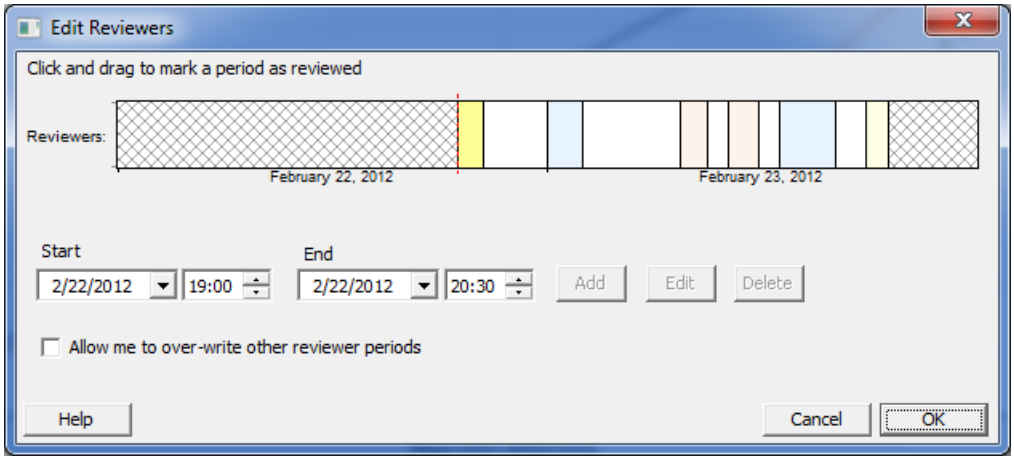
We have tested the above code using SQL Server Express version 10.5.

In addition to the two fields mentioned above, starting in revision 5 of the database interface, you can supply two optional fields:

Table 2 – Optional Fields Returned by `windog_ReadSpecifications`

Field Name	Type	Meaning	Possible Values
force_reviewer_popup	integer	Pop-up reviewer window before exporting to database	An integer: 0 means do not pop-up the window, any other value will cause Windographer to pop-up the window.
reviewer_name	text	Supply the name of the reviewer to be used in the reviewer window	Any text string

If the `windog_ReadSpecifications` stored procedure returns the 'force_reviewer_popup' field to Windographer as an integer value other than zero, Windographer will show the Edit Reviewers window when a user selects File > Export to Database:



This window allows the user to mark which sections of the dataset they have reviewed. Windographer will attempt to save this information back to the database using the `windog_WriteReviewerInfo` stored procedure as outlined below (see section 7.8). Users can also edit this information at any time by selecting View > Document History > Reviewers.

If the `windog_ReadSpecifications` stored procedure returns the 'reviewer_name' field, Windographer will use the value of this field when adding reviewer information. If the field is not supplied, or is empty, Windographer will use the value from the 'User name' field in Tools > Options > General for the reviewer name.

4. Reading List of Datasets

When the user clicks Retrieve Datasets on the Import From Database window, Windographer queries the database for a list of datasets by calling `windog_ReadListOfDataSetsR3`. This stored procedure should take the user name as its only input parameter. If the option to 'Use the database username when connected to a database' is checked in Tools > Options > Advanced, Windographer will attempt to get the user name from the login used to connect to the database. If that fails, or if the user has not selected that option, Windographer will take the user name from the 'User name' field in Tools > Options > General.

For example, to retrieve the list of datasets for a user named 'John Doe', Windographer will execute the following statement:

```
windog_ReadListOfDataSetsR3 ('John Doe')
```

That stored procedure must return a recordset with the structure shown in the table below. If the recordset lacks any of the mandatory fields, Windographer will report an error and stop the import process.

Table 3 – Structure of Recordset Returned by `windog_ReadListOfDataSetsR3`

Field	Type	Comments
dataset_id	text	**Mandatory**
dataset_name	text	**Mandatory**
dataset_timestep	integer	**Mandatory, in minutes**
dataset_timezone	integer	**Mandatory, minutes offset from UTC (NULL means UTC-unspecified)**
dataset_start	text	**Mandatory, format YYYY-MM-DD HH:MM**
dataset_end	text	**Mandatory, format YYYY-MM-DD HH:MM**
dataset_inst_ht_meas_units	text	Optional, m or ft, default is m
dataset_description	text	Optional, default = ''
dataset_displacement_height	float	Optional, default = 0
dataset_elevation	float	Optional, default = 0
dataset_elevation_units	text	Optional, default = m
dataset_latitude	double	Optional, default = 0
dataset_longitude	double	Optional, default = 0
doc_history_notes	text	Optional, default = ''
permission_change_dataset_info	boolean	Optional, default = false
permission_change_numeric_data	boolean	Optional, default = false
permission_change_column_properties	boolean	Optional, default = false
permission_change_data_structure	boolean	Optional, default = false
permission_change_flag_properties	boolean	Optional, default = false
permission_change_flagged_segments	boolean	Optional, default = false
permission_change_calibration_constants	boolean	Optional, default = false

If the recordset lacks one of the optional fields, Windographer will set that field to its default value, indicated in Table 3 above. The fields that start with '**permission_change'** indicate the willingness of the database to allow Windographer to write data back to the database after the user has changed the data in certain ways. The section on exporting data to the database will cover this topic in more detail.

5. Importing a Dataset

To import a dataset, Windographer queries the database for five types of information:

1. a list of the properties of each data column
2. the data columns themselves
3. a list of the properties of the flags used in the dataset
4. a list of flagged segments
5. a list of the calibration history for each data column
6. a list of the sections of the dataset that have been reviewed
7. a document history list, meaning a list of modifications made to the dataset

As Windographer calls the appropriate stored procedures, it reports progress in the text box at the bottom of the Import From Database window. If no errors occur, the window closes and Windographer displays the dataset in its main window. If an error occurs, the window stays open and reports the error.

5.1 Importing Data Column Properties

Before it retrieves the data columns, Windographer has to know how much and what kind of data to expect. So it begins by calling the `windog_ReadDataColumnPropertiesR2` stored procedure to retrieve the properties of the data columns in the dataset. For example, to retrieve data column information for the dataset with the `dataset_id` value of 'K5DO3AL9', Windographer will execute the following statement:

```
windog_ReadDataColumnPropertiesR2 ('K5DO3AL9')
```

That stored procedure must return a recordset with the structure shown in the table below. If the recordset lacks any of the mandatory fields, Windographer will report an error and stop the import process.

Table 4 – Structure of Recordset Returned by `windog_ReadDataColumnPropertiesR2`

Field	Type	Comments
column_id	text	Mandatory
group_id	text	Mandatory
type	integer	Mandatory
sub_type	integer	Mandatory
label	text	Mandatory
height	float	Mandatory
units	text	Mandatory
verbose_label	text	Optional, default = ''
color	integer	Optional, default = -1
is_visible	boolean	Optional, default = true

For example, `windog_ReadDataColumnPropertiesR2` might return a recordset like the one shown in the table below.

Table 5 – Sample Recordset Returned by `windog_ReadDataColumnPropertiesR2`

column_id	group_id	type		sub_type	height	label	units	verbose_label	color	is_visible
D8iG-08	1	1		1	40	Spd 40m	m/s	Average wind speed at 40m	3428285	true
D8iG-03	1	1		2	40	Spd 40m SD	m/s	Std. dev. of wind speed at 40m	2594834	true
D8iG-04	1	1		3	40	Spd 40m Max	m/s	Minimum wind speed at 40m	2594582	false
D8iG-05	1	1		4	40	Spd 40m Min	m/s	Maximum wind speed at 40m	2594330	true
D8iG-20	2	1		1	20	Spd 20m	m/s	Average wind speed at 20m	2594078	true
D8iG-15	2	1		2	20	Spd 20m SD	m/s	Std. dev. of wind speed at 20m	2593826	true
D8iG-16	2	1		3	20	Spd 20m Max	m/s	Minimum wind speed at 20m	2593574	false
D8iG-17	2	1		4	20	Spd 20m Min	m/s	Maximum wind speed at 20m	2593322	true
D8iG-34	5	3		1	38	Dir 38m	°	Average wind direction at 38m	-1	true
D8iG-35	5	3		2	38	Dir 38m SD	°	Std. dev. of wind dir at 38m	-1	true
D8iG-57	7	4		1	0	Temperature	°C	Average temperature in °C	-1	true
Qi-14	D8iG-08, D8iG-20, D8iG-34	100		1	0	Quality Indicator	%	Quality Indicator	-1	true

Windographer uses the `label` field to identify each data column in its graphs and tables. The `type` and `sub_type` fields must contain one of the values shown in Table 6 and Table 7 below. The `type` field identifies the type of data the data column contains (wind speed, wind direction, temperature, etc.) while the `sub_type` field indicates whether its data points record the mean, minimum, maximum, or standard deviation values measured in each time step. The `height` field indicates the measurement height above ground. (The dataset properties specify the units of these measurement height values.) The `color` field will contain an RGB value, or, if Windographer should automatically assign a color, this field can be absent or contain -1. Windographer will use the `group_id` field to associate the data columns that belong to the same data column group (see definition at the top of the document). The `is_visible` field will set the initial value for the visible setting for the column.

In the example table above, columns D8iG-34 & D8iG-35 are part of the same group, where D8iG-35 represents the standard deviation of the D8iG-34 wind speed column. Column Qi-14 contains quality information that Windographer will associate with columns D8iG-08, D8iG-20, and D8iG-34.

Table 6 - Allowable Values of `Type`

Value	Meaning
1	Horizontal wind speed
2	Vertical wind speed
3	Wind direction
4	Temperature
5	Air pressure
6	Relative humidity
7	Measured turbulence intensity
8	Battery voltage

Value	Meaning
9	Air density
10	Wind turbine output
11	GHI (global horizontal irradiance)
12	DNI (direct normal irradiance)
13	DHI (diffuse horizontal irradiance)
100	Quality
1000	Other (unknown type)

Table 7 - Allowable Values of `Sub_Type`

Value	Meaning
1	Average
2	Standard Deviation
3	Minimum
4	Maximum

5.2 Importing Data Columns

As section 4 explains, the recordset returned by `windog_ReadSpecifications` determines how Windographer queries data columns. The following subsections describe the available approaches.

5.2.1 Importing One Data Column at a Time

Windographer queries the data columns one at a time by calling the stored procedure `windog_ReadDataColumn` once for every data column specified in the data column properties recordset. This stored procedure takes four parameters: `dataset_id`, `column_id`, `start_date`, and `end_date`. The start and end dates come from the date range inputs the user specified in the Import From Database window. The `column_id` field comes from the data column properties recordset. The `dataset_id` field comes from the list of datasets retrieved by `windog_ReadListOfDataSetsR3`.

For example, the call to this stored procedure might look like this:

```
windog_ReadDataColumn('D8iG-16', 'F7_15', '2009-03-27 18:50', '2009-03-27 23:00')
```

This stored procedure must return a recordset containing one column of data, such as the example shown in the table below.

Table 8 – Sample Recordset Returned by `windog_ReadDataColumn`

field
11.5
12.7
9.9
9.7
10.3
10.9
11.9
12.8
...

Windographer will ignore the name of the field. Each call to `windog_ReadDataColumn` must return the same number of records, and that number of records must equal the number of time steps between `start_date` and `end_date`. In the example above, Windographer will expect to import 25 records if the time step is 10 minutes, since there are 250 minutes from 18:50 to 23:00, and 250 minutes divided by 10 minutes per time step equals 25 time steps. If Windographer encounters a recordset with the wrong number of records, it will report an error and stop the data import process.

5.2.2 Importing One Data Column at a Time Using Time Step Numbers

Using this method, Windographer calls the stored procedure `windog_ReadDataColumnWithStepNumber` once for every data column specified in the data column properties recordset. This stored procedure takes four parameters: `dataset_id`, `column_id`, `start_date`, and `end_date` (the same as `windog_ReadDataColumn`). The start and end dates come from the date range inputs the user specified in the Import From Database window. The `column_id` field comes from the data column properties recordset. The `dataset_id` field comes from the list of datasets retrieved by `windog_ReadListOfDataSetsR3`.

For example, the call to this stored procedure might look like this:

```
windog_ReadDataColumnWithStepNumber('D81', 'F7', '2009-03-27 18:50', '2009-03-27
23:00')
```

This stored procedure must return a recordset containing two columns. Windographer will expect the first column to contain the time step number, and the second column to contain numerical values, as in the example in the table below.

Table 9 – Sample Recordset Returned by `windog_ReadDataColumnWithStepNumber`

time step number	data
0	11.5
1	12.7
2	9.9
7	9.7
10	10.9
8	10.3
11	11.9
12	12.8

Windographer ignores the name of the fields and will fill the values of its data column with data from the second column of this recordset. The time step numbers should be zero-indexed, meaning the first time step in the requested date range should have the number zero. Time steps for which the recordset specifies no data will be treated as gaps in Windographer. For instance, if a dataset with 10min time steps spans a time period from 2013-06-15 9:10 to 2013-06-15 11:20, Windographer will expect to fill its data column with 13 values (10 min x 13 = 130min). If the table above were returned to Windographer for one of the columns of that dataset, the column in Windographer would have empty values for the time steps with numbers 3, 4, 5, 6, and 9, as shown in Table 10:

Table 10 – Values Entered into Windographer for the Example Above

time step number	data
0	11.5
1	12.7
2	9.9
3	
4	
5	
6	
7	9.7
8	10.3
9	
10	10.9
11	11.9
12	12.8

Windographer does not require the records to be returned in order of ascending time step. It will put the data values into the positions specified by the time step numbers regardless of order. It will ignore any extra values returned by the recordset.

5.2.3 Importing All Data Columns at Once

To query all data columns at once, Windographer calls the `windog_ReadAllDataColumns` stored procedure once, with one parameter specifying the dataset ID, one for the start date, and one for the end date. For example, the call to this stored procedure might look like this:

```
windog_ReadAllDataColumns ('D8iG-16', '2009-03-27 18:50', '2009-03-27 23:00')
```

That stored procedure should return a recordset whose number of fields corresponds to the number of records in the recordset returned by `windog_ReadDataColumnPropertiesR2`. Each field corresponds to one data column, and the name of each field must correspond to the `column_id` of that data column.

For example, if `windog_ReadDataColumnPropertiesR2` returns a recordset like the one shown in Table 11, then `windog_ReadAllDataColumns` should return a recordset like the one shown in Table 12:

Table 11 – Sample Recordset Returned by `windog_ReadDataColumnPropertiesR2`

column_id	group_id	type	sub_type	height	label	units	color	is_visible
Qe3s-12	1	1	1	40	Spd 40m	m/s	3428285	true
Qe3s-41	1	1	2	40	Spd 40m SD	m/s	2594834	true
Qe3s-08	1	1	3	40	Spd 40m Max	m/s	2594582	true
Qe3s-05	1	1	4	40	Spd 40m Min	m/s	2594330	false
Qe3s-32	2	3	1	38	Dir 38m	°	-1	true
Qe3s-02	2	3	2	38	Dir 38m SD	°	-1	true
Qe3s-16	3	4	1	0	Temperature	°C	-1	true

Table 12 – Sample Recordset Returned by `windog_ReadAllDataColumns`

Qe3s-12	Qe3s-41	Qe3s-08	Qe3s-05	Qe3s-32	Qe3s-02	Qe3s-16
11.5	2.0	12.0	11.0	250	5	21
12.7	2.5	13.5	11.0	255	25	21
9.9	1.5	11.0	8.0	268	55	21
9.0	1.8	10.5	6.0	251	45	21
10.3	2.1	11.0	8.9	240	56	21
10.9	2.2	14.7	7.6	233	74	21
11.9	1.5	12.5	8.5	247	35	21
12.8	1.7	15.0	5.0	258	94	21
…	…	…	…	…	…	…

The order of these fields does not matter; Windographer will identify each data column by its `column_id`, which appears as the field name. In this example, Windographer will interpret the third field in Table 12 (highlighted) as the maximum wind speed measured at 40 meters, since its field name matches the `column_id` of the third record of Table 11 (also highlighted). Likewise, it will interpret the 5th column as the wind direction measured at 38 meters, and so on.

If Windographer successfully queries the data columns (one at a time, or altogether) without encountering any errors, it will move on to querying flag properties.

5.3 Importing Flag Properties

To retrieve a list of the flags used in a particular dataset, Windographer calls the stored procedure `windog_ReadFlagProperties` with one parameter specifying the dataset ID. For the dataset with an ID of 'K5DO3AL9', for example, Windographer will execute the following statement:

```
windog_ReadFlagProperties('K5DO3AL9')
```

That stored procedure must return a recordset with the structure shown in the table below. If the recordset lacks any of the mandatory fields, Windographer will report an error and stop the import process.

Table 13 – Structure of Recordset Returned by `windog_ReadFlagProperties`

Field	Type	Comments
flag_id	`text`	Mandatory
name	`text`	Mandatory
Description	`text`	Mandatory
Color	`bigint`	Mandatory
include_in_calcs	`bit`	Mandatory
show_in_graphs	`bit`	Mandatory
tower_shading	`bit`	Mandatory
invalid_data	`bit`	Mandatory
synthesized_data	`bit`	Mandatory

For example, `windog_ReadFlagProperties` might return a recordset like the one shown in Table 14. The Windographer database interface requires all the fields shown in Table 13. If the database administrator decides to use the same set of flags for every dataset, this stored procedure can ignore the dataset ID parameter – and the

`permission_change_flag_list` field should have a value false (otherwise each user would be writing to the same table, potentially overwriting each other's changes).

Table 14 – Sample Recordset Returned by `windog_ReadFlagProperties`

flag_id	name	description	color	include_in_calcs	show_in_graphs	tower_shading	invalid_data	synthesized_data
38	Icing	Data affected by sensor icing	245754	false	true	false	false	false
39	Invalid	Error code value from data logger	216526	false	false	false	true	false
4a	Low quality	Signal-to-noise ratio lower than threshold	487298	false	false	false	false	false
4b	Tower shading	Shading of an anemometer by the tower	358070	false	true	true	false	false
4c	Synthesized	Synthesized by Windographer	328842	true	true	false	false	true
4d	Cold weather	Chilly	304210	true	true	false	false	false
4e	Summertime	Nice and warm	274982	true	true	false	false	false

The **name, description, color, include_in_calcs**, and **show_in_graphs** fields that appear in this table correspond to the properties of flags in Windographer, and the Windographer online help system contains further information about them. The last three fields (**tower_shading, invalid_data, and synthesized_data**) tell Windographer which flags will be used as special purpose flags. Only one of the flags can be marked as the tower shading flag, invalid data flag, and synthesized data flag – that is why there is a single 'true' value in each of those columns. In this example, the 'Tower shading' flag will become Windographer's default flag for marking points as tower shading. Similarly, 'Invalid' will get used to flag invalid data, and 'Synthesized' will get used to mark data that Windographer synthesizes. If more than one record has a 'true' value in any of these last three fields (**tower_shading, invalid_data,** or **synthesized_data**), Windographer will return an error and stop the import process. Windographer will also return an error if any of those fields do not have any records with a 'true' value.

HOW WINDOGRAPHER HANDLES FLAGS

Each row returned by the `windog_ReadFlagProperties` stored procedure corresponds to a flag in Windographer. When Windographer imports data from a database, the list of flags returned by `windog_ReadFlagProperties` will appear in the list of flags in the Define Flags dialog box. Windographer users can also define a set of 'favorite flags', which are not associated with any single dataset. Starting in revision 2 of the database interface (the initial version used in Windographer v3), Windographer added these 'favorite flags' to a new dataset only if the `windog_ReadFlagProperties` stored procedure did not return any records from the database. Otherwise, Windographer would only include the flags returned by the stored procedure in the list of flags in the Define Flags window

By default, Windographer defines 5 favorite flags which correspond to the first 5 flags in Table 14: Icing, Low Quality, Tower shading, Invalid, and, Synthesized. Also by default, Windographer will assign the last three (Tower shading, Invalid, and Synthesized) as the 'special purpose' flags, which is uses for particular situations.

Starting in revision 2 of the database interface, Windographer completely separates its flag IDs from those used by the database. The database can assign any format for a flag ID (integers, characters, or a mixture of the two) and Windographer will use those values when exporting data back to the database. This means that when a user creates a new flag in Windographer, the export process must get the new flag ID from the database before it can save the flagged segments associated with this flag. Thus, the `windog_AddOrUpdateFlagR2` stored procedure must return the new flag ID from the database when it is called to add a flag.

Starting in revision 3 of the database interface, Windographer imports/exports special purpose flags from/to the database. We have updated the specifications for the stored procedure that writes flag properties in the database and given it a new name: `windog_AddOrUpdateFlagR2.`

5.4 Importing Flagged Segments

To retrieve flagged segment data for a particular dataset, Windographer will call the stored procedure `windog_ReadFlaggedSegments`, passing in the dataset ID as a parameter. For a dataset with an ID of 'K5DO3AL9', for example, Windographer will execute the following statement:

```
windog_ReadFlaggedSegments('K5DO3AL9')
```

This stored procedure must return a recordset of the format shown in the example in the table below.

Table 15 – Sample Recordset Returned by `windog_ReadFlaggedSegments`

flag_id	column_id	start_time	end_time
4b	84	2009-07-08 16:30	2009-07-10 09:40
4b	84	2009-07-12 01:00	2009-07-12 04:20
4b	86	2009-07-12 01:00	2009-07-12 04:20
39	92	2009-07-12 01:00	2009-07-12 04:20
4b	85	2009-07-08 16:30	2009-07-10 09:40
39	92	2009-07-08 16:30	2009-07-10 09:40

The `windog_ReadFlaggedSegments` stored procedure does not take a date range as parameters. It should return all flagged segments, regardless of whether the user is reading the entire date range of a dataset, or just a subset thereof. Windographer will sort out which flagged segments apply to the retrieved date range and ignore those that do not.

Please note that if the user has imported only a subset of the dataset, meaning only a particular date range, then if Windographer encounters any flagged segments that span the start or end of that date range, it will prevent the user from editing flagged segments and then exporting back to the database. In this situation, if the user does flag or unflag segments, Windographer will break the link to the database, so that the user can save the dataset to a .windog file but not back to the database. This situation will never arise if the user imports the complete dataset.

5.5 Reading Calibration Data

Starting in revision 4 of the database interface, Windographer imports calibration settings from the database and exports them back to the database. To read the calibration history for a dataset, Windographer calls the `windog_ReadCalibrationEvents` stored procedure, passing one parameter specifying the dataset ID, one for the start date, and one for the end date. For example, for the dataset with ID '23-31567', Windographer will execute the following statement:

```
windog_ReadCalibrationEvents ('23-31567', '2009-03-01 00:00', '2009-04-01 00:00')
```

The call to this stored procedure must return a recordset of the format shown in the table below.

Table 16 – Sample Recordset Returned by `windog_ReadCalibrationEvents`

column_id	start_time	slope	offset	boom_orientation	serial_number
456	2009-03-01 00:00	0.35	0.0	270	1565848
456	2009-03-12 01:00	0.4	0.0	271	4565-DD15
477	2009-03-01 00:00	0.3	0.0	90	8755889
478	2009-03-01 00:00	0.3	8.5	180.5	65411
479	2009-03-01 00:00	0.1	0.50	0	17
482	2009-03-01 00:00	1.25	0.51	125	87H7

The **column_id** must match the column ID of the column to which the calibration information applies. The **start_time** is the date and time where that calibration period started to apply. The first calibration **start_time** for a column should match the start time of the dataset. Windographer will assume that a calibration period applies beginning with its **start_time** and continuing until it encounters another calibration event for that column. If no other calibration event applies to that column, the initial one applies to the whole column. In the example above, only the '456' column has more than one calibration period. The first begins at '2009-03-01 00:00' and applies up to '2009-03-12 01:00'. From that point on, the second calibration period applies. The **slope, offset, and boom_orientation** fields should be floating point values, and the **serial_number** field will be handled as text. If the stored procedure does not return any records, Windographer will assign default calibration constants (0 for the offset and boom orientation, and 1 for the slope) to each parent (mean) column.

5.6 Reading Reviewer Information

Starting in revision 5 of the database interface, Windographer reads and writes reviewer information. Reviewer information is simply a list of names with associated dates that is meant to keep track of who has reviewed different parts of the dataset. To read the reviewer information for a dataset, Windographer will call the `windog_ReadReviewerInfo` stored procedure, passing one parameter specifying the dataset ID, one for the start date, and one for the end date. For example, for the dataset with ID '3651', Windographer will execute the following statement:

```
windog_ReadReviewerInfo ('3651', '2009-03-01 00:00', '2009-04-01 00:00')
```

The call to this stored procedure must return a recordset of the format shown in the table below.

Table 17 – Sample Recordset Returned by `windog_ReadReviewerInfo`

reviewer_name	start_time	end_time
Tom Ferguson	2009-03-01 00:00	2009-03-03 00:00
Linda Sloka	2009-03-08 14:00	2009-03-11 00:00
Tom Lambert	2009-03-11 00:00	2009-03-21 00:00

If the stored procedure does not return any records, Windographer will simply leave the reviewer information empty for this dataset. If the stored procedure returns any records, then all the fields shown in the table above are mandatory. The **reviewer_name** field should be a text field, while **start_time** and **end_time** are date/time fields.

5.7 Reading Document History Data

As of revision 4 of the database interface, Windographer will call the `windog_ReadDocumentHistoryR2` stored procedure to retrieve the document history for a particular dataset. Windographer will pass in one parameter to indicate the dataset. For the dataset with ID 'K5DO3AL9', for example, Windographer will execute the following statement:

```
windog_ReadDocumentHistoryR2('K5DO3AL9')
```

This stored procedure must return a recordset with the structure shown in the table below. As with flagged segments, the database will always provide the entire document history list even if the user has retrieved only a portion of the entire dataset.

Table 18 – Structure of Recordset Returned by `windog_ReadDocumentHistoryR2`

Field	Type
type	integer
user	text
modification_time	date/time
start_time	date/time
end_time	date/time
time_steps	integer
data_columns	text
text1	text
text2	text
float_array	text
binary_blob	varbinary(max)

For Windographer to accept the recordset as valid, the stored procedure must return all fields listed in Table 18. These specifications do not provide details about each field since these are very specific to Windographer and can simply be retrieved from a table with the same structure as the recordset shown above. Windographer will update the table using the `windog_WriteDocumentHistoryEventR3` stored procedure whenever a user saves information back to the database.

6. Writing Data to the Database

Windographer can write the same types of information to a database that it reads. If you imported the dataset from a database, then Windographer will make the exporting process more efficient by writing only the data that have changed since the import. For example, if the user has applied flags but not changed the numeric data in any way, then Windographer will write the flag data to the database, but it will not write the numeric data. But the write process will always encompass all the modified data. For example, if the user has changed both flag data and numeric data, he or she will not have the option of writing just the modified flag data or just the modified numeric data; the only option will be to write all modified data, both flag and numeric, back to the database.

To ensure data integrity, Windographer gives the database administrator a great deal of control over the process of writing data to the database. That control begins with the ability to specify the kinds of modifications that the Windographer user can make to the dataset, and still write that dataset back to the database.

Windographer allows users to insert datasets into the database. All stored procedures that add new items (datasets, columns, and flags) to the database therefore require at least one output parameter from the database: the ID of the newly-added item. These stored procedures (`windog_AddOrUpdateDatasetR4`, `windog_AddOrUpdateColumnR2`, and `windog_AddOrUpdateFlagR2`) each work in a similar way. When Windographer wants to modify a dataset that it imported from the database, it will send the dataset ID as the input parameter into the stored procedure. When Windographer wants to insert a new item that is part of a dataset that did not originate in the database, it will specify an empty dataset ID as the input parameter, and it will expect the stored procedure to specify the new dataset ID as an output parameter.

For example, the code below contains a few lines from the `windog_AddOrUpdateDatasetR4` stored procedure in our test database.

Example T_SQL code for returning new dataset ID from the data base (this is NOT a complete stored procedure):

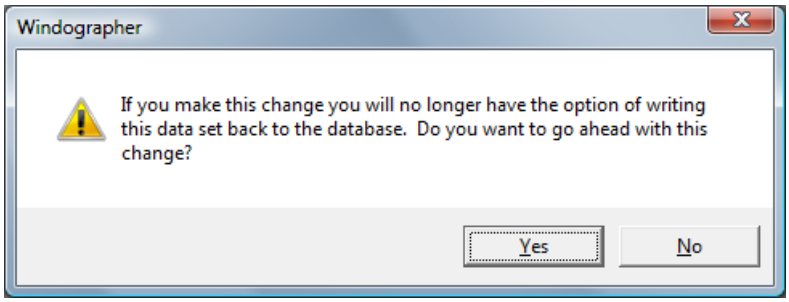
```
...

@new_dataset_id varchar(50) output,

...

@retmsg varchar(200)=null output

...

Insert into [tblSites] ([name], [description], [timestep], [elevation]...
select @name, @description, @timestep, @elevation, @longitude, @latitude...

if exists(select id from [tblSites] where id=(select SCOPE_IDENTITY()) )
    begin
        set @new_dataset_id = (select SCOPE_IDENTITY())
        return 1
    end
else
    begin
        set @retmsg='Error adding a record into the tblSites table.'
        return 0
    end
```

Note: Unlike reading data from a database, the technique that Windographer uses to write data to a database changes substantially for different database platforms. This results from the fact that different databases handle stored procedures differently. MySQL in particular handles return values from stored procedures in a distinctive way.

6.1 Permission to Write Data to the Database

The Windographer user will be able to do anything to a dataset imported from a database that he or she could do to any other dataset. Possible changes the user can make include applying flags, applying slopes and offsets to numeric data, applying time shifts, deleting data, filling gaps, changing data column properties, creating a new wind speed data column by vertical extrapolation, updating calibration constants, appending rows and/or columns using File > Append, and adding calculated data columns. The question for the database administrator is this: Given that a certain dataset has been changed in a certain way, does Windographer have permission to write that dataset back to the database or not? (The user will always have the option of saving the dataset to a .windog file using File > Save, or exporting to a text file using File > Export Data.)

The six `permission_change` fields in Table 3 specify the types of modifications the user can make to a dataset while still preserving the option to write that dataset back to the database. The `permission_change_numeric_data` field, for example, specifies whether a dataset can be written back to the database after the user has made one or more changes to its numeric data, such as the application of a slope or offset to some or all data points, or the deletion of some data points. If that permission field had a value of false, then just at the point that the user was about to execute some change to the numeric data, Windographer would issue the following warning message:



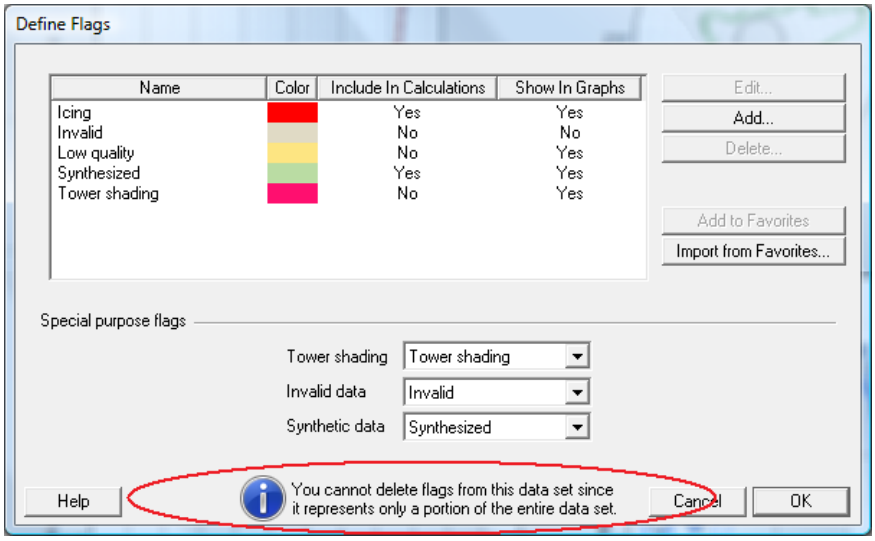
If the user clicks Yes, Windographer will disable the Export to Database menu item, switch out of database mode, and cease to ask such questions for the current dataset.

The table below lists the modifications governed by each of the six permission fields. As mentioned in the section on reading data, these fields form part of the recordset returned by the `windog_ReadListOfDataSetsR3` stored procedure. Their values could vary by dataset if the database administrator decides that the database can accept certain types of modification in some datasets, but not in others. However, in most cases we would expect the database administrator to apply the same restrictions to all datasets.

Table 19 – Changes Governed by Permission Fields

Permission Field	Changes Governed by Permission Field
permission_change_dataset_info	Changes to the meta data like the dataset name, description, longitude, latitude, elevation, measurement units, etc.
permission_change_numeric_data	Filling of gaps, application of slope or offset, calibration changes where user selects option to update the data, deletion of data, or time shifting data within the existing period of record (with Revise > Time Shift)
permission_change_column_properties	Changes to data column properties (label, type, units, color, associations with other data columns)
permission_change_data_structure	Addition of time steps (with File > Append) or removal of time steps (with Revise > Delete Data). A future version may also allow addition of data columns (with File > Append or vertical extrapolation), or removal of data columns (with Revise > Delete Data or Revise > Configure Dataset), or time shifting data such any of the new data falls outside of the starting period of record (with Revise > Time Shift).
permission_change_flag_list	Changes to the properties of a flag, addition of flags, deletion of flags*
permission_change_flagged_segments	Application of a flag to a segment, removal of a flag from a segment
permission_change_calibration_constants	Calibration changes

*Note: the scenario in which the Windographer user imports only a portion of a dataset (e.g. the last month of a two-year dataset) requires special handling. When the user deletes a flag, Windographer also deletes all the associated flagged segments (those that refer to that flag) in the dataset. But if Windographer cannot access the entire dataset it cannot perform that step in its entirety, nor even correctly inform the user as to whether it must be performed, since it only sees a subset of the dataset. To avoid this problem, in the scenario in which the database import covers only a portion of the entire dataset, the Define Flags window will disable the Delete button and inform the user that he or she cannot delete flags, as shown in the screenshot below.



6.2 WRITING DATASET INFORMATION

Windographer will start its export to the database by opening a transaction in the database. If Windographer finds that its connection already has a transaction open, it will add a warning to the status text box. Windographer will conclude the transaction after it calls `windog_WriteConcluded`.

Windographer begins exporting data to the database by calling the `windog_AddOrUpdateDatasetR4` stored procedure. If the user imported the current dataset from a database, or if the dataset came from a file but has already been saved to the database, then the first parameter (the dataset ID from the database) will have a value and Windographer will ignore the output parameters of this stored procedure. If the dataset did not originate from a database, then the first parameter will be empty, and Windographer will use the output parameters. Specifically, Windographer will need parameter 18, the new ID for this dataset, which was assigned by the database.

> **SQL Server**: Windographer will query the database to determine the list of parameters for the `windog_AddOrUpdateDatasetR4` stored procedure. Windographer will expect to find the parameters listed in the table below and will report an error message otherwise. The stored procedure should return an integer: 1 for success, or 0 for failure. If the stored procedure returns 0, and has an extra output parameter of type `varchar`, Windographer will treat this as an error message and will display it in the status text box. If the stored procedure returns 1 and is being called to add data from a file into the database, then the stored procedure should also return all the parameters listed as 'out' parameters in the table below. The parameters, their types, and the values that Windographer will pass in or retrieve appear in the table below:

Table 20 – Parameters for `windog_AddOrUpdateDatasetR4`

Parameter	Direction	Type	Comments
1	in	varchar	dataset ID
2	in	varchar	dataset name (or file name if it's empty)
3	in	varchar	dataset description
4	in	varchar	document history notes
5	in	integer	time zone (minutes from UTC; NULL means UTC-unspecified)
6	in	integer	timestep (minutes)
7	in	float	displacement height (m)
8	in	float	elevation (m)
9	in	float	longitude
10	in	float	latitude
11	in	varchar	instrument height measurement units
12	in	varchar	elevation units
13	in	boolean	Permission to change the dataset information
14	in	boolean	Permission to change numeric data
15	in	boolean	Permission to change column properties
16	in	boolean	Permission to change data structure

Table 20 Continued – Parameters for `windog_AddOrUpdateDatasetR4`

Parameter	Direction	Type	Comments
17	in	boolean	Permission to change flag properties
18	in	boolean	Permission to change flagged segments
19	in	boolean	Permission to change calibration constants
20	out	varchar	New dataset ID
21	out	bit	Permission to change dataset info
22	out	bit	Permission to change numeric data
23	out	bit	Permission to change the column properties
24	out	bit	Permission to change the data structure
25	out	bit	Permission to change flag properties
26	out	bit	Permission to change flagged segments
27	out	bit	Permission to change calibration constants

Example T-SQL code setting up the parameters for the stored procedure:

```
CREATE proc [dbo].[windog_AddOrUpdateDatasetR4]
@dataset_id varchar(50),
@name varchar(250),
@description varchar(250)=null,
@doc_history_notes varchar(4000)=null,
@timezone int,
@timestep int,
@displacement_height float,
@elevation float,
@longitude float,
@latitude float,
@height_meas_units varchar(50),
@elevation_units varchar(50),
@permission_change_dataset_info bit,
@permission_change_numeric_data bit,
@permission_change_column_properties bit,
@permission_change_data_structure bit,
@permission_change_flag_list bit,
@permission_change_flagged_segments bit,
@permission_change_calibration_constants bit,
@new_dataset_id varchar(50) output,
@permission_change_dataset_info_out bit output,
@permission_change_numeric_data_out bit output,
@permission_change_column_properties_out bit output,
@permission_change_data_structure_out bit output,
@permission_change_flag_list_out bit output,
@permission_change_flagged_segments_out bit output,
@permission_change_calibration_constants_out bit output,
@retmsg varchar(200)=null output
as
...
```

Please note that none of the return parameters is defined as '`varchar(max)`'. The current versions of Windows drivers do not handle that data type correctly – values being returned to Windographer must have the variable size specified; for example: `varchar(50)` or `nvarchar(2000)`.

6.3 DATA COLUMN PROPERTIES

If the call to `windog_AddOrUpdateDatasetR4` succeeds, then Windographer will determine whether to call the `windog_AddOrUpdateColumnR2` stored procedure. It needs to do so in the following two scenarios:

1) If the user is exporting from a file into the database and creating a new dataset in the database, Windographer will have to call `windog_AddOrUpdateColumnR2` for each data column to inform the database of the properties of each data column, and to retrieve from the database the new column IDs for use in later stored procedure calls. These calls to `windog_AddOrUpdateColumnR2` will happen just after the call to `windog_AddOrUpdateDatasetR4`, as part of the process of changing the dataset structure.

2) If the current dataset originated from a database, but the user has changed the properties of any data column such as a measurement height or a color, Windographer will call `windog_AddOrUpdateColumnR2` for each data column. These calls to `windog_AddOrUpdateColumnR2` will happen just after the call to `windog_WriteFlaggedSegmentR2` (see section 7.6), as part of the routine updates to column properties.

If Windographer intends for the column properties to update existing values in the database, then it will pass the current column ID as the second parameter to this stored procedure. If the data are being added to the database for the first time, then Windographer will pass an empty string as the second parameter, and the database must assign a new column ID for the data column and return it as parameter 12. Windographer stores this returned column ID and uses it in later stored procedure calls. If the database does not return a new column ID when Windographer has passed in an empty column ID, Windographer will stop the export process and report an error.

We introduced the 'Parent Column ID' field in revision 2 of Windographer's database interface to help with the file export process. When exporting data from a file, Windographer must provide the database with a way to maintain referential integrity for columns in the same data column group (see definition at the top of the document). In the process of exporting from a file to a database, Windographer will first call the `windog_AddOrUpdateColumnR2` stored procedure for all 'parent' columns. For each one, Windographer will record the new column ID returned by the database (parameter 12 in the table below). Then, as Windographer calls `windog_AddOrUpdateColumnR2` for each child column, Windographer will pass in the column ID (assigned by the database) for the parent column as parameter number 10.

> **SQL Server**: Windographer will query the database to determine the list of parameters for the `windog_AddOrUpdateColumnR2` stored procedure. Windographer will expect to find the parameters listed in the table below and will report an error message otherwise. The stored procedure should also return an integer: 1 for success, or 0 for failure. If the stored procedure returns 0, and has an extra output parameter of type `varchar`, Windographer will treat this as an error message and will display it in the status text box. The parameters, their types, and the values that Windographer will pass in appear in the table below:

Table 21 – Parameters for `windog_AddOrUpdateColumnR2`

Parameter	Type	Value That Windographer Will Pass In
1	varchar	Dataset ID
2	varchar	Column ID
3	int	Type
4	int	Sub-type
5	varchar	Label
6	float	Height
7	varchar	Units
8	varchar	Verbose Label
9	long	Color
10	varchar	Parent column ID
11	bit	Is visible
12	varchar (OUTPUT)	New column ID

As of Windographer v3.1.10, released in November of 2013, Windographer will send the value 1000 for columns of type 'Other'. Prior versions of Windographer sent the value of 0 (zero).

6.4 Changing Data Structure

Two types of data structure changes are possible in Windographer: addition or removal of time steps, and addition or removal of data columns. In the current version of Windographer, only the first type of change can be exported to the database, and then only if the database specifies the `permission_change_data_structure` field as true. We have not yet implemented the ability to update a dataset in the database after the addition or removal of data columns. That means that if a user imports a dataset and adds or removes a data column, she will lose the ability to write the dataset back to the database. As mentioned above, users always have the option to save the dataset to a .windog file or export to a text file.

When exporting to the database, Windographer handles added and deleted time steps by calling the `windog_WriteDataPoint` and `windog_WriteDataFollowUp` stored procedures, as the following section explains.

6.5 Writing Numeric Data

If the call to `windog_AddOrUpdateDatasetR4` succeeds, Windographer will call the stored procedure `windog_WriteDataPoint` once for each data point that the user has modified, for example by applying a scale or offset factor, or by filling a gap or deleting a data segment. If a data point has been deleted, Windographer will pass a NULL value in as the 4th parameter to `windog_WriteDataPoint`. Applying a flag to a data point does not count as a modification in this context, since the flagging operation does not affect the actual numeric value of the data point. If the user has not changed any data points, Windographer will not call `windog_WriteDataPoint` or `windog_WriteDataFollowUp`.

> **SQL Server**: Windographer will query the database to determine the list of parameters for the `windog_WriteDataPoint` stored procedure. Windographer will expect to find the parameters listed in the table below and will report an error message otherwise. The stored procedure should return an integer: 1 for success, or 0 for failure. If the stored procedure returns 0, and has an extra output parameter of type `varchar`, Windographer will treat this as an error message and will display it in the status text box. The input parameters, their types, and the values that Windographer will pass in appear in the table below:

Table 22 – Parameters for `windog_WriteDataPoint`

Parameter	Type	Value That Windographer Will Pass In
1	`varchar`	Dataset ID
2	`varchar`	Column ID
3	`datetime`	Start of time step
4	`float`	Value within time step, or NULL for deleted data

After all the calls to `windog_WriteDataPoint` (one for each data value that changed), Windographer will call the `windog_WriteDataFollowup` stored procedure once. This stored procedure should take a single input parameter specifying the dataset ID.

> **SQL Server**: Windographer will query the database to determine the list of parameters for the `windog_WriteDataFollowUp` stored procedure. Windographer will expect to find the parameters listed in the table below and will report an error message otherwise. The stored procedure should return an integer: 1 for success, or 0 for failure. If the stored procedure returns 0, and has an extra output parameter of type `varchar`, Windographer will treat this as an error message and will display it in the status text box. The input parameters, their types, and the values that Windographer will pass in appear in the table below:

Table 23 – Parameters for `windog_WriteDataFollowUp`

Parameter	Type	Value That Windographer Will Pass In
1	`varchar`	Dataset ID

The database can use the `windog_WriteDataFollowUp` stored procedure as a trigger to update tables that summarize the data, such as a table that contains the start date and end date of each dataset.

6.6 Writing Flags and Flagged Segments

If previous stored procedures calls succeed, Windographer will next attempt to update the dataset's flags and flagged segments, if necessary. Because flagged segments refer to flag IDs, to insure referential integrity Windographer first deletes flagged segments, then makes the necessary changes to the list of flags, and then writes the flagged segments.

6.6.1 Deleting Flagged Segments

If the user has made any change to the set of flagged segments, either by applying a flag to one or more data segments or by removing a flag from one or more data segments, Windographer will instruct the database to delete and rewrite all flagged segments. To delete the flagged segments, it calls the stored procedure `windog_DeleteFlaggedSegments`, passing in as parameters the dataset ID, start date, and end date. The `windog_DeleteFlaggedSegments` stored procedure must delete all flagged segments for the current dataset that fall within, or partially overlap this date range.

> **SQL Server**: In SQL Server, the T-SQL necessary to carry out this stored procedure will look something like this:
>
> ```
> delete from tblWindogFlaggedSegments
> where (dataset_id = @dataset_id)
> AND (
>  ((start_time >= @start_time) AND (start_time < @end_time))
>         OR
>  ((end_time > @start_time) AND (end_time <= @end_time))
>         OR
>  ((start_time < @start_time) AND (end_time > @end_time))
>  );
> ```

In the example above, the parameters passed in were `@dataset_id`, `@start_time`, and `@end_time`. As with other stored procedures that are part of the writing process, Windographer will query the database to determine the list of parameters for the `windog_DeleteFlaggedSegments` stored procedure. Windographer will expect to find the parameters listed in the table below and will report an error message otherwise. The stored procedure should also return an integer: 1 for success, or 0 for failure. If the stored procedure returns 0, and has an extra output parameter of type `varchar`, Windographer will treat this as an error message and will display it in the status text box. The input parameters, their types, and the values that Windographer will pass in appear in the table below:

Table 24 – Parameters for `windog_DeleteFlaggedSegments`

Parameter	Type	Value That Windographer Will Pass In
1	`varchar`	Dataset ID
2	`datetime`	Start time of dataset or subset
3	`datetime`	End time of dataset or subset

6.6.2 WRITING FLAGS

The user can modify the set of flags in three ways: by adding a flag, deleting a flag, or changing the properties of a flag. Windographer updates the list of flags in the database by calling three stored procedures, one for each of those three ways. First it calls the `windog_DeleteFlagR2` stored procedure once for each flag that the user has deleted since importing the dataset.

> **SQL Server**: Windographer will query the database to determine the list of parameters for the `windog_DeleteFlagR2` stored procedure. Windographer will expect to find the parameters listed in the table below and will report an error message otherwise. The stored procedure should also return an integer: 1 for success, or 0 for failure. If the stored procedure returns 0, and has an extra output parameter of type `varchar`, Windographer will treat this as an error message and will display it in the status text box.

Table 25 – Parameters for `windog_DeleteFlagR2`

Parameter	Type	Value That Windographer Will Pass In
1	varchar	Dataset ID
2	varchar	Flag ID

Next, Windographer will call the `windog_AddOrUpdateFlagR2` stored procedure for each flag whose properties the user has changed in the current session and for each flag that has been added. If Windographer intends to update the flag properties, then the flag ID passed into the stored procedure will have a non-empty value. In this case, Windographer will ignore the value returned for the 'new flag ID'. Otherwise, the flag ID will be an empty string and Windographer intends for the flag to be added into the database. When adding a flag, Windographer will expect the database to return the new flag's ID in the eleventh parameter. Windographer will then use the new flag ID when updating or deleting this flag in future exports to the database. If the database does not return a new flag ID when Windographer has passed in an empty flag ID, Windographer will stop the export process and report an error.

Starting in revision 3 of Windographer's database interface, Windographer imports/exports special purpose flags from/to the database. Up to three different flags can be marked as special purpose flags, one each for tower shading, invalid data, and synthesized data. However, a single flag can also be used as more than one special purpose flag. For instance, if a dataset has only a single flag, it will be marked as all three special purpose flags (and Windographer will use that flag to mark synthesized data, invalid data, and tower shading). As this can lead to confusing situations, we don't recommend using a single flag.

> **SQL Server**: Windographer will query the database to determine the list of parameters for the `windog_AddOrUpdateFlagR2` stored procedure. Windographer will expect to find the parameters listed in the table below and will report an error message otherwise. The stored procedure should also return an integer: 1 for success, or 0 for failure. If the stored procedure returns 0, and has an extra output parameter of type `varchar`, Windographer will treat this as an error message and will display it in the status text box. The parameters, their types, and the values that Windographer will pass in appear in the table below:

Table 26 – Parameters for `windog_AddOrUpdateFlagR2`

Parameter	Type	Value That Windographer Will Pass In
1	varchar	Dataset ID
2	varchar	Flag ID
3	varchar	Flag Name
4	varchar	Flag description
5	bigint	Flag color
6	bit	Include in calculations
7	bit	Show in graphs
8	bit	Tower shading
9	bit	Invalid data
10	bit	Synthesized data
11	varchar (OUTPUT)	New flag ID if exporting data from a file

6.6.3 Writing Flagged Segments

If previous stored procedures calls succeed, and if the user has changed the set of flagged segments, Windographer will then write the flagged segments to the database one by one by calling the `windog_WriteFlaggedSegmentR2` stored procedure once for each flagged segment.

SQL Server: Windographer will query the database to determine the list of parameters for the `windog_WriteFlaggedSegmentR2` stored procedure. Windographer will expect to find the parameters listed in the table below and will report an error message otherwise. The stored procedure should also return an integer: 1 for success, or 0 for failure. If the stored procedure returns 0, and has an extra output parameter of type `varchar`, Windographer will treat this as an error message and will display it in the status box. The input parameters, their types, and the values that Windographer will pass in appear in the table below:

Table 27 – Parameters for `windog_WriteFlaggedSegmentR2`

Parameter	Type	Value That Windographer Will Pass In
1	varchar	Dataset ID
2	varchar	Flag ID
3	varchar	Column ID
4	datetime	Start date/time of segment
5	datetime	End date/time of segment

6.7 Writing Calibration Data

If previous stored procedures calls succeed, and if the user has changed any calibration information, Windographer will then attempt to write the new calibration information to the database using a two-step process for each column: first it will call `windog_DeleteCalibrationEvents` to remove existing calibrations events, and then it will call `windog_WriteCalibrationEventR2` for each calibration change to each column.

6.7.1 Deleting Existing Calibration Information for a column

Windographer will call the `windog_DeleteCalibrationEvents` once for each 'parent' column in the dataset since parent columns are the only columns for which Windographer recognizes calibration information.

> **SQL Server**: Windographer will query the database to determine the list of parameters for the `windog_DeleteCalibrationEvents` stored procedure. Windographer will expect to find the parameters listed in the table below and will report an error message otherwise. The stored procedure should also return an integer: 1 for success, or 0 for failure. If the stored procedure returns 0, and has an extra output parameter of type `varchar`, Windographer will treat this as an error message and will display it in the status box. The input parameters, their types, and the values that Windographer will pass in appear in the table below:

Table 28 – Parameters for `windog_DeleteCalibrationEvents`

Parameter	Type	Value That Windographer Will Pass In
1	`varchar`	Dataset ID
2	`varchar`	Column ID
3	`datetime`	Start time of dataset or subset
4	`datetime`	End time of dataset or subset

Windographer expects the database to remove all calibration information for the column between the start date and the end date because the following calls to `windog_WriteCalibrationEventR2` for that column should replace the calibration information for the same time period.

6.7.2 Updating Calibration Information

After calling `windog_DeleteCalibrationEvents` once for a particular column, Windographer will then call the `windog_WriteCalibrationEventR2` once for each calibration change to that column. Windographer will call `windog_WriteCalibrationEventR2` at least once for each column to create the initial calibration event for each column (the calibration period that begins at the start date). If the calibration does not change over the time period from start date to end date, that will be the only call; otherwise, Windographer will call `windog_WriteCalibrationEventR2`, with appropriate start and end dates, for each subsequent calibration event.

> **SQL Server**: Windographer will query the database to determine the list of parameters for the `windog_WriteCalibrationEventR2` stored procedure. Windographer will expect to find the parameters listed in the table below and will report an error message otherwise. The stored procedure should also return an integer: 1 for success, or 0 for failure. If the stored procedure returns 0, and has an extra output parameter of type `varchar`, Windographer will treat this as an error message and will display it in the status box. The input parameters, their types, and the values that Windographer will pass in appear in the table below:

Table 29 - Parameters for `windog_WriteCalibrationEventR2`

Parameter	Type	Value That Windographer Will Pass In
1	varchar	Dataset ID
2	varchar	Column ID
3	datetime	Start date/time of calibration period
4	datetime	End date/time of calibration period
5	float	Offset
6	float	Slope
7	float	Boom orientation
8	varchar	Serial number

Windographer will call the `windog_WriteCalibrationEventR2` stored procedure using chronological order for the calibration change events. Windographer does not allow overlapping calibration periods or gaps in between calibration periods. The end of one period can never occur between the start and end of another period. Similarly, the start of a period can never fall between the start and end of another period. Also, the end date of one period is guaranteed to coincide with the start of the next calibration period. If there are no following calibration changes (for instance, if there is only one, or on the last calibration change for a column), Windographer will pass the dataset end date as the ending date for the calibration period.

As the list of parameters shows, Windographer will pass an end date for each calibration change. Since the end date is redundant information, the database administrator may choose to ignore the end date and set calibrations to last until the next calibration starts, or until the end of the dataset.

6.8 Writing Reviewer Information

If previous stored procedures calls succeed, and if the user has changed any reviewer information, Windographer will then attempt to write the new reviewer information to the database using a two-step process: first it will call `windog_DeleteReviewerInfo` to remove existing reviewer information, and then it will call `windog_WriteReviewerInfo` for each period that has been marked as reviewed. If the `windog_ReadSpecifications` stored procedure returned the '**reviewer_name**' field, Windographer will use the value of this field when a user adds reviewer information through Windographer's graphic user interface. If the field is not supplied, or is empty, Windographer will use the value from the 'User name' field in Tools > Options > General for the reviewer name.

6.8.1 Deleting Reviewer Information

Windographer will first call the `windog_DeleteReviewerInfo` stored procedure to remove reviewer information in the database over the time period that the user.

> **SQL Server**: Windographer will query the database to determine the list of parameters for the `windog_DeleteReviewerInfo` stored procedure. Windographer will expect to find the parameters listed in the table below and will report an error message otherwise. The stored procedure should also return an integer: 1 for success, or 0 for failure. If the stored procedure returns 0, and has an extra output parameter of type `varchar`, Windographer will treat this as an error message and will display it in the status text box. The input parameters, their types, and the values that Windographer will pass in are listed below:

Table 30 – Parameters for `windog_DeleteReviewerInfo`

Parameter	Type	Value That Windographer Will Pass In
1	varchar	Dataset ID
2	varchar	Reviewer name
3	datetime	Start time of dataset or subset
4	datetime	End time of dataset or subset

Windographer expects the database to remove all reviewer information for this dataset between the start date and the end date because the following calls to `windog_WriteReviewerInfo` for that column should replace the reviewer information for the same time period.

6.8.2 Updating Reviewer Information

After calling `windog_DeleteReviewerInfo` once for the dataset, Windographer will then call the `windog_WriteReviewerInfo` once for each period that has been marked as reviewed.

> **SQL Server**: Windographer will query the database to determine the list of parameters for the `windog_WriteReviewerInfo` stored procedure. Windographer will expect to find the parameters listed in the table below and will report an error message otherwise. The stored procedure should also return an integer: 1 for success, or 0 for failure. If the stored procedure returns 0, and has an extra output parameter of type `varchar`, Windographer will treat this as an error message and will display it in the status box. The input parameters, their types, and the values that Windographer will pass in appear in the table below:

Table 31 – Parameters for `windog_WriteReviewerInfo`

Parameter	Type	Value That Windographer Will Pass In
1	varchar	Dataset ID
2	varchar	Reviewer name
3	datetime	Start time of review period
4	datetime	End time of review period

Windographer will call the `windog_WriteReviewerInfo` stored procedure using chronological order for the reviewed periods. Windographer does not allow overlapping reviewed periods; however, it does allow for gaps between reviewed periods. So, the end date of one reviewed period is not guaranteed to coincide with the start of the next time period, since there may be a gap. But, the end of one period will never come between the start and end of another period. Similarly, the start of a period will never fall between the start and end of another period.

6.9 Writing Document History Data and Notes

If previous stored procedures calls succeed, Windographer will call the stored procedure `windog_WriteDocumentHistoryEventR3` once for each document history event that has been added since the data was imported from the database.

SQL Server: Windographer will query the database to determine the list of parameters for the `windog_WriteDocumentHistoryEventR3` stored procedure. Windographer will expect to find the parameters listed in the table below and will report an error message otherwise. The stored procedure should also return an integer: 1 for success, or 0 for failure. If the stored procedure returns 0, and has an extra output parameter of type `varchar`, Windographer will treat this as an error message and will display it in the status text box. The input parameters, their types, and the values that Windographer will pass in are listed below:

Table 32 – Parameters for `windog_WriteDocumentHistoryEventR3`

Parameter	Type	Value That Windographer Will Pass In
1	`int`	Event ID (generated by Windographer)
2	`varchar`	Dataset ID
3	`int`	type (code used by Windographer)
4	`varchar`	User name (database login)
5	`varchar(3500)`	Verbose description (created by Windographer)
6	`datetime`	modification_time
7	`datetime`	start_time
8	`datetime`	end_time
9	`int`	time_steps
10	`varchar(3500)`	data_columns
11	`varchar(3500)`	text1
12	`varchar(3500)`	text2
13	`varchar(3500)`	array of values used in doc history event
14	`varbinary(max)`	byte array used to store details of the event

We expect that the database will contain a dedicated table for Windographer document history events since these are very specific to Windographer. Table 32 above can serve to guide the design of this database table.

If call(s) to `windog_WriteDocumentHistoryEventR2` succeed, Windographer will call the `windog_WriteDocumentHistoryNotes` stored procedure once, passing in the text found on the "Notes" tab of the Document History window. This stored procedure does not have to carry out any actions on the database, but it does have to exist with parameters as specified below and return a 1 (for success), for Windographer to successfully complete an export to the database using interface revision 5 or later.

SQL Server: Windographer will query the database to determine the list of parameters for the `windog_WriteDocumentHistoryNotes` stored procedure. Windographer will expect to find the parameters listed in the table below, and will report an error message otherwise. The stored procedure should also return an integer: 1 for success, or 0 for failure. If the stored procedure returns 0, and has an extra output parameter of type `varchar`, Windographer will treat this as an error message and will display it in the status text box. The input parameters, their types, and the values that Windographer will pass in are listed below:

Table 33 - Parameters for `windog_WriteDocumentHistoryNotes`

Parameter	Type	Value That Windographer Will Pass In
1	`varchar`	Dataset ID
2	`varchar`	Document History Notes

6.10 Concluding the Write Process

If previous stored procedure calls succeed, Windographer will call the `windog_WriteConcluded` stored procedure once. Windographer will call this stored procedure regardless of whether any data has changed; it will always be the last stored procedure called. This stored procedure does not have to carry out any actions on the database, but it does have to exist with parameters as specified below and return a 1 (for success), for Windographer successfully to complete an export to the database using interface revision 5 or later.

SQL Server: Windographer will query the database to determine the list of parameters for the `windog_WriteConcluded` stored procedure. Windographer will expect to find the parameters listed in the table below and will report an error message otherwise. The stored procedure should also return an integer: 1 for success, or 0 for failure. If the stored procedure returns 0, and has an extra output parameter of type `varchar`, Windographer will treat this as an error message and will display it in the status text box. The table below shows the input parameters, their types, and the values that Windographer will pass in to this stored procedure:

Table 34 – Parameters for `windog_WriteConcluded`

Parameter	Type	Value That Windographer Will Pass In
1	`varchar`	Dataset ID
2	`varchar`	User name

The following T-SQL would create a stored procedure in MS SQL Server sufficient to fulfill Windographer's demand for `windog_WriteConcluded`:

```
CREATE proc [dbo].[windog_WriteConcluded]
@dataset_id nvarchar(50),
@user_name nvarchar(400),
@retmsg nvarchar(500)=null output
AS
BEGIN
	SET NOCOUNT ON
	return 1
END
```

The call to the `windog_WriteConcluded` stored procedure allows for any final data manipulations that need to occur within the database. After a successful call to `windog_WriteConcluded`, Windographer will commit the transaction that includes the whole export process. Any other outcome (for instance, user cancellation or an error) will cause Windographer to roll back the transaction.

7. List of Required Stored Procedures

Table 35 - Required Stored Procedures

#	Stored Procedure	Arguments	Return Value
1	windog_ReadSpecifications		recordset[1]
2	windog_ReadListOfDataSetsR3	user name	recordset
3	windog_ReadDataColumnPropertiesR2	dataset ID	recordset
4	windog_ReadDataColumn, or windog_ReadAllDataColumns, or windog_ReadDataColumnWithStepNumber	dataset ID, column ID, start date, end date	recordset
5	windog_ReadFlagProperties	dataset ID	recordset
6	windog_ReadFlaggedSegments	dataset ID	recordset
7	windog_ReadCalibrationEvents	dataset ID, start date, end date	recordset
8	windog_ReadReviewerInfo	dataset ID, start date, end date	recordset
9	windog_ReadDocumentHistoryR2	dataset ID	recordset
10	windog_AddOrUpdateDatasetR4	dataset ID, dataset name, 17 other dataset properties, plus 8 output parameters	integer code and 8 output parameters
11	windog_AddOrUpdateColumnR2	dataset ID, column ID, parent column ID, 8 parameters describing the data column, an output parameter of the new column ID	integer code and 1 output parameter
12	windog_WriteDataPoint	dataset ID, column ID, start of time step, value within time step	integer code
13	windog_WriteDataFollowUp	dataset ID	integer code
14	windog_DeleteFlaggedSegments	dataset ID, start date, end date	integer code
15	windog_DeleteFlagR2	dataset ID, flag ID	integer code
16	windog_AddOrUpdateFlagR2	dataset ID, flag ID, 8 parameters describing flag, an output parameter with the new flag ID for flags being added to the database	integer code and 1 output parameter
17	windog_WriteFlaggedSegmentR2	dataset ID, flag ID, column ID, start and end time of segment	integer code
18	windog_DeleteCalibrationEvents	dataset ID, column ID, start and end of dataset or subset	integer code
19	windog_WriteCalibrationEventR2	dataset ID, column ID, 6 others	integer code
20	windog_DeleteReviewerInfo	dataset ID, reviewer name, start and end of dataset or subset	integer code
21	windog_WriteReviewerInfo	dataset ID, reviewer name, start and end of review period	integer code
22	windog_WriteDocumentHistoryEventR3	dataset ID, event ID, and 12 parameters describing the event	integer code
23	windog_WriteDocumentHistoryNotes	dataset ID, doc history notes	integer code
24	windog_WriteConcluded	dataset ID, user name	integer code

[1] – Returning a 'recordset' from a stored procedure in SQL Server can be accomplished with a 'Select...' statement directly in the stored procedure code.

8. Updating from Previous Database Interface Revisions

If your database implements an older revision of the Windographer database interface, you can follow these steps to update your database to revision 7. Not every step listed below is mandatory. If you do not intend to export data from Windographer to the database, for example, then you need not perform any of the steps that relate to writing data to the database. The relevant sections of this document explain each of the new and revised stored procedures in detail.

8.1 To Update from Rev1 to Rev7

1. You can now specify three additional data column types: relative humidity, wind turbine output, and other.
2. Add windog_ReadDataColumnWithStepNumber if you wish to specify time series data in that way.
3. If you wish to store reviewer info in the database, add to the recordset returned by windog_ReadSpecifications two new fields: 'force_reviewer_popup' and 'reviewer_name'.
4. Add windog_ReadListOfDataSetsR3 to replace windog_ReadListOfDataSets.
5. Add windog_ReadDataColumnPropertiesR2 to replace windog_ReadDataColumnProperties.
6. Add three new fields to the recordset returned by windog_ReadFlagProperties, to identify special-purpose flags.
7. Add windog_ReadCalibrationEvents.
8. Add windog_ReadDocumentHistoryR2.
9. Add windog_AddOrUpdateDatasetR4.
10. Add windog_AddOrUpdateColumnR2.
11. Add windog_AddOrUpdateFlagR2.
12. Add windog_DeleteFlagR2.
13. Add windog_WriteFlaggedSegmentR2.
14. Add windog_DeleteCalibrationEvents.
15. Add windog_WriteCalibrationEventR2.
16. Add windog_ReadReviewerInfo, windog_DeleteReviewerInfo, and windog_WriteReviewerInfo.
17. Add windog_WriteDataFollowUp and windog_WriteConcluded.
18. Add windog_WriteDocumentHistoryEventR3.

8.2 To Update from Rev2 to Rev7

1. You can now specify the data column type 'other'.
2. Add windog_ReadDataColumnWithStepNumber if you wish to specify time series data in that way.
3. Two new optional fields in the recordset returned by windog_ReadSpecifications: 'force_reviewer_popup' and 'reviewer_name'.
4. windog_ReadListOfDataSetsR3 replaces windog_ReadListOfDataSets.
5. windog_ReadDataColumnProperties replaced by windog_ReadDataColumnPropertiesR2, with an additional field 'is_visible' indicating whether the data column is visible or hidden.
6. Add three new fields to the recordset returned by windog_ReadFlagProperties, to identify special-purpose flags.
7. Add windog_ReadCalibrationEvents.
8. Add windog_ReadDocumentHistoryR2.
9. Add windog_AddOrUpdateDatasetR4 to replace windog_AddOrUpdateDataset.
10. Add windog_AddOrUpdateColumnR2.
11. Add windog_AddOrUpdateFlagR2.

12. Add windog_DeleteFlagR2.
13. Add windog_WriteFlaggedSegmentR2 to replace windog_WriteFlaggedSegment_v2.
14. Add windog_DeleteCalibrationEvents and windog_WriteCalibrationEvent.
15. Add windog_ReadReviewerInfo, windog_DeleteReviewerInfo, and windog_WriteReviewerInfo.
16. Add windog_WriteDataFollowUp and windog_WriteConcluded.
17. Add windog_WriteCalibrationEventR2 to replace windog_WriteCalibrationEvent.
18. Add windog_WriteDocumentHistoryEventR3 to replace windog_WriteDocumentHistoryEventR2.

8.3 To Update from Rev3 to Rev7

1. Two new optional fields in the recordset returned by windog_ReadSpecifications: 'force_reviewer_popup' and 'reviewer_name'.
2. windog_ReadListOfDataSetsR3 replaces windog_ReadListOfDataSets.
3. windog_ReadDataColumnProperties replaced by windog_ReadDataColumnPropertiesR2, with an additional field 'is_visible' indicating whether the data column is visible or hidden.
4. Added windog_ReadCalibrationEvents.
5. Added windog_ReadDocumentHistoryR2.
6. Add windog_AddOrUpdateDatasetR4 to replace windog_AddOrUpdateDataset.
7. Add windog_AddOrUpdateColumnR2.
8. Add windog_DeleteCalibrationEvents and windog_WriteCalibrationEvent.
9. Added windog_ReadReviewerInfo, windog_DeleteReviewerInfo, windog_WriteReviewerInfo, and windog_WriteDataFollowUp, and windog_WriteConcluded.
10. windog_WriteCalibrationEventR2 replaces windog_WriteCalibrationEvent. The new stored procedure has one more input parameter specifying boom orientation.
11. windog_WriteDocumentHistoryEventR3 replaces windog_WriteDocumentHistoryEventR2. The new stored procedure has one more input parameter: a binary byte array.

8.4 To Update from Rev4 to Rev7

1. Two new optional fields in the recordset returned by windog_ReadSpecifications: 'force_reviewer_popup' and 'reviewer_name'.
2. windog_ReadListOfDataSetsR3 replaces windog_ReadListOfDataSets.
3. windog_ReadCalibrationEvents now can include an additional field named 'boom_orientation' in the recordset it returns.
4. windog_ReadDocumentHistoryR2 now must include an additional field named 'binary_blob' in the recordset it returns.
5. windog_AddOrUpdateDatasetR4 replaces windog_AddOrUpdateDataset.
6. Added windog_ReadReviewerInfo, windog_DeleteReviewerInfo, windog_WriteReviewerInfo, and windog_WriteDataFollowUp, and windog_WriteConcluded.
7. windog_WriteCalibrationEventR2 replaces windog_WriteCalibrationEvent. The new stored procedure has one more input parameter specifying boom orientation.
8. windog_WriteDocumentHistoryEventR3 replaces windog_WriteDocumentHistoryEventR2. The new stored procedure has one more input parameter: a binary byte array.

8.5 To Update from Rev5 to Rev7

1. windog_ReadListOfDataSetsR3 replaces windog_ReadListOfDataSetsR2.

2. windog_ReadCalibrationEvents now can include an additional field named ‘boom_orientation’ in the recordset it returns.
3. windog_ReadDocumentHistoryR2 now must include an additional field named ‘binary_blob’ in the recordset it returns.
4. windog_AddOrUpdateDatasetR4 replaces windog_AddOrUpdateDatasetR2. The new stored procedure has additional input parameters specifying displacement height, document history notes, and time zone.
5. windog_WriteCalibrationEventR2 replaces windog_WriteCalibrationEvent. The new stored procedure has one more input parameter specifying boom orientation.
6. windog_WriteDocumentHistoryEventR3 replaces windog_WriteDocumentHistoryEventR2. The new stored procedure has one more input parameter: a binary byte array.

8.6 To Update from Rev6 to Rev7

1. Replace windog_ReadListOfDataSetsR2 with windog_ReadListOfDataSetsR3. The new stored procedure must return an additional field named ‘time_zone’.
2. Replace windog_AddOrUpdateDatasetR3 with windog_AddOrUpdateDatasetR4. The new stored procedure has two additional input parameters specifying document history notes and time zone.