PSK Propagation Reporter DLL Documentation 2013-Mar-10 Philip Gladstone <philip@gladstonefamily.net>

This describes the PSK Propagation Reporter API that is available on Windows and which is provided by the PSKReporter.DLL. This API does not only provide support for reporting PSK events, but also can report events from other modes.

This version of the document describes version 1.8 of the DLL.

This DLL is designed to have minimal dependencies on other DLLs in the system. It should run on Windows 2000 and above. For full protection against backwards compatibility problems, this DLL can be marked as delay loaded by the linker (with /delayload:pskreporter.dll). This means that it is not loaded until it is called for the first time. The advantage of this is that any dependencies required by this DLL cannot cause load failures in the main program, and the programming interface looks like regular DLL dynamic linking. The only point to note is that calls to the API (the first one at least) should be encapsulated inside a structured error handler. If the DLL fails to load (or is missing/broken), an exception is thrown and caught by the SEH handler. The DLL should then not be invoked any more until the calling program is restarted.

Different versions of the API may support more functions. If backwards compatibility is required (i.e. an application needs to support using an older version of the API), then loading the API DLL explicitly and then using GetProcAddress is probably the safest way to proceed.

The general use of the API is to call ReporterInitialize to provide the destination hostname (or ip address) and port number. After that, call ReporterSeenCallsign whenever a callsign is recognized. This may be automatically decoded off air, manually entered, or from a logfile. At regular intervals, the function ReporterTickle should be invoked to handle flushing of buffers etc. On program shutdown, the ReporterUninitialize function can be called to cleanup, however this will be automatically done as part of DLL unload.

This DLL is thread-safe. There is internal locking to ensure that bad things will not happen if multiple functions are invoked at the same time. There is no requirement that each function be called from the same thread.

ReporterGetInformation can be called after any API to retrieve information about the result of that API call. After an error, it will return an error message, and after a successful call, it will either return a null string or some informative text.

ReporterGetStatistics can be called at any time to return some basic information about the current session that may be of interest to the user.

For a detailed description of the APIs, please see the sections below. Note that the primary interfaces use UNICODE strings. There is a set of interfaces that use ANSI strings, but these are present for use from languages which find it difficult to manipulate UNICODE strings.

The kit includes the following files:

PSKReporter.DLL	The DLL that implements the API
PSKReporter.LIB	The link library that provides the stubs
PSKReporter.h	The header file that describes the functions
PSKReporterTest.EXE	A test program. If invoked with no arguments, it explains how to use it.
PSKReporterTest.CPP	The source of the test program.
ShortTest.txt	A simple test file that shows a working example of the test program.
PSKReporter.pdf	This file.

ReporterInitialize

Initialize the DLL with the hostname and port number.

```
DWORD __cdecl ReporterInitialize(
      const wchar_t *hostname,
      const wchar_t *port
);
```

Parameters

hostname	this is the hostname or ip address (in dotted quad notation) to which the reports will be sent. The default that will be used if a null pointer or null string is provided is “report.pskreporter.info”.
port	this is the portnumber to which the reports will be sent. The default that will be used if a null pointer or null string is provided is “4739”.

Return Value

0	good. Use ReporterGetInformation to retrieve the version number of the DLL.
-1	bad. On an error, use the ReporterGetInformation function to retrieve an English form of the error.

Notes

This must be the first function called. It can be called more than once – especially if the first call fails. If there is a running session, then the current buffers will be flushed and a new session started.

This is the only function that may block. This function does a DNS lookup to resolve the supplied DNS name into an IP address. All subsequent communication is performed using unacknowledged UDP, so there is no requirement to block.

ReporterSeenCallsign

Indicates that a Callsign has been detected and may be logged.

```
DWORD __cdecl ReporterSeenCallsign(
      const wchar_t *remoteInformation,
      const wchar_t *localInformation,
      DWORD flags
);
```

Parameters

remoteInformation

this is a list of ADIF field names followed by values in the following format. Each field name is null terminated and is followed by a null terminated value. There is an extra null at the end of the list. Null valued fields need not be included. The specific fields of interest are CALL, GRIDSQUARE, LATLNG, FREQ, MODE, QSO_DATE, TIME_ON, SNR. If the QSO_DATE and TIME_ON are omitted, then the current time will be used. If the time of the contact is unknown (e.g. automatic decode from a saved audio file), then do *not* use this API, or at least set the 'test' flags as described below. The SNR field is not a standard ADIF field and can be used to report the Signal to Noise Ratio of the decoded signal in dB. The range is -127 to +127.

As of version 1.8, any non alphabetic character (also excluding underscore) can be used to delimit the field names and values. This may make constructing these strings much easier.

localInformation

this is a list of ADIF field names followed by values in the following format. Each field name is null terminated and is followed by a null terminated value. There is an extra null at the end of the list. Null valued fields need not be included. The specific fields of interest are STATION_CALLSIGN, MY_GRIDSQUARE, MY_LATLNG, PROGRAMID, PROGRAMVERSION and MY_ANTENNA. Note that the MY_ANTENNA field is not a standard ADIF field, but it should contain a textual description of the logging station's antenna (if possible).

As of version 1.8, any non alphabetic character (also excluding underscore) can be used to delimit the field names and values. This may make constructing these strings much easier.

flags — this defines the source of the callsign.

REPORTER_SOURCE_AUTOMATIC – the callsign was automatically extracted

REPORTER_SOURCE_LOG – the callsign was generated from a log record

REPORTER_SOURCE_MANUAL – the callsign was manually entered

REPORTER_SOURCE_TENTATIVE – the callsign should be considered tentative. This should be set if an automatic decode for the RF protocol only decodes a single copy of the callsign. This would typically be used in conjunction with the REPORTER_SOURCE_AUTOMATIC flag above.

REPORTER_SOURCE_TEST – this is a test record (and can be used in conjunction with one of the flags above).

Return Value

0	good
-1	bad.

Notes

Reports will only be sent if the callsign is 'new'. There is a holdback mechanism that prevents every record from being sent. A callsign will only be sent once every 30 minutes unless it moves onto a different band.

The frequency supplied should be the center frequency of the communication if available. If not, then the radio frequency. If nothing is available, then omit the frequency tag.

The values for the fields should all be in ADIF format. For example, the QSO_DATE field is eight digits: YYYYMMDD.

The ADIF field names are case insensitive. Extra fields may be passed, and will be ignored. This provides a level of backwards and forwards compatibility. However, if an unknown field is found, then the information string will note that fact. An example of a localInformation string is:

```
L"STATION_CALLSIGN,N1DQ,MY_GRIDSQUARE,FN42hn,"
```

In this example, a comma is being used to delimit field names and values. There is an implicit null character (C standard for string termination) on the end of the string which ends the whole string.

The two LATLNG fields are not standard ADIF fields. They can be used as alternatives to the two GRIDSQUARE fields. The data format associated with a LATLNG field is in accordance with ISO 6709, but only specifying the latitude and longitude (i.e. no elevation). If both LATLNG and GRIDSQUARE are present, then the LATLNG value will override (if it seems valid).

Test reports can be seen (albeit in a crummy format) at

http://www.pskreporter.info/query?test=1

Test reports may be deleted from the database after 24 hours. In any event, the query above only shows reports received more recently than that. The callsign tables for the test reports are shared with the live reports, so please do not fill them up with bad data.

The REPORTER_SOURCE_TENTATIVE flag indicates that the callsign may or may not be correct. This can happen if the underlying communication being decoded only sends a single copy callsign for the transmitter and there is no error correction. If the *same* callsign is received twice within a short period of time and within a small frequency difference (values could be 90 seconds and 500 Hz, though I am open to other suggestions), then the second report will have the TENTATIVE flag removed and forwarded.

On the backend, if a tentative report is received with the same callsign from multiple monitors at approximately the same time, on approximately the same frequency, then it is treated as a valid report. The rules for ‘approximately’ are likely to be about the same as in the reporting DLL – i.e. 90 seconds and 500 Hz. The frequency may be relaxed a little more as this would be comparing the frequencies reported by different radios.

ReporterTickle

Provides the DLL with processing time

```
DWORD __cdecl ReporterTickle(
);
```

Return Value

0	good
-1	something went wrong

Notes

This function is used to actually send the data to the collecting system. This function will not block. It should be called reasonably often – at least once every 30 seconds.

Now that the DLL is thread-safe, it is possible to call this from another thread at regular intervals.

ReporterGetInformation

Returns the last error as a text string.

```
DWORD __cdecl ReporterGetInformation(
      wchar_t *buffer,
      DWORD maxlen
);
```

Parameters

buffer	points to an area of memory that will be filled with a null terminated string that describes the last error (in English).
maxlen	the size of the buffer in characters.

Return Value

0	good
-1	bad, and calling this function again will not reveal anything about this error.

Notes

There is an internal buffer in the DLL that stores the result from the last API call (good or bad). The only function that does not update this buffer is ReporterGetInformation. This function just fetches the contents of that buffer.

The returned string will always be null terminated (provided that maxlen >= 1). The string will not normally include newlines.

ReporterUninitialize

Cleans up resources consumed and flushes any buffer

```
DWORD __cdecl ReporterUninitialize(
);
```

Return Value

0	good. Use ReporterGetInformation to retrieve some simple statistical information about the number of reports sent during this session, and the same information across all sessions (based on information held in the registry).
-1	something went wrong

Notes

This will close the connection to the collection server after flushing any buffers.

ReporterGetStatistics

Returns statistics on the current session.

```
DWORD __cdecl ReporterGetStatistics(
      REPORTER_STATISTICS *statistics,
      DWORD maxlen
);
```

Parameters

statistics	points to an area of memory that will be filled with the statistics.
maxlen	the size of the statistics structure in bytes.

Return Value

0	good
-1	bad.

Notes

The size of the structure must be passed in so that backwards and forwards compatibility is preserved. Unused fields will be set to zero.

The structure definition is:

```
typedef struct {
      wchar_t       hostname[256];
      wchar_t       port[32];
      bool          connected;
      unsigned int  callsigns_sent;
      unsigned int  callsigns_buffered;
      unsigned int  callsigns_discarded;
      unsigned int  last_send_time;
      unsigned int  next_send_time;
      wchar_t       last_callsign_queued[24];
      unsigned int  bytes_sent;
      unsigned int  bytes_sent_total;
      unsigned int  packets_sent;
      unsigned int  packets_sent_total;
} REPORTER_STATISTICS;
```

Future versions of this API will only add fields to the end of this structure.

Fields

hostname	The current hostname that is being used as the destination of the reporting messages.
port	The current port number that is being used as the destination of the reporting messages.

connected	True if the DLL is ‘connected’ to the server. Since the channel is one-way UDP, all this really means is that the hostname/port look valid and that the names resolve.
callsigns_sent	The total number of records actually transmitted to the server.
callsigns_buffered	The number of records currently buffered and waiting to be sent.
callsigns_discarded	The number of records that were discarded due to the holdback timer.
last_send_time	The unix time (seconds since 1/1/1970) that the last packet was sent to the server. This may be zero if no packet has been sent.
next_send_time	The unix time that the next packet will be sent. This may be zero if there is no buffered information. This is only approximate, as packets are only sent when ReporterTickle or ReporterSeenCallsign is called.
last_callsign_queued	The last callsign that was not discarded and will be (or was) sent to the server.
bytes_sent	The number of bytes sent to the server in the current session.
bytes_sent_total	The number of bytes sent to the server over all sessions.
packets_sent	The number of packets sent to the server in the current session.
packets_sent_total	The number of packets sent to the server over all sessions.

There are a set of function designed to be callable from Visual Basic. These all have the __stdcall calling convention, and take ANSI strings as arguments. No description of the function itself is provided here. Please see the section above for the description of the wchar_t version of the function. It is possible to call APIs with wchar_t arguments from Visual Basic, however invoking __cdecl functions is very difficult and prone to error.

All these functions are implemented as stubs that perform argument conversion and then invoke the wchar_t variants. This means that you could mix and match them (though it is probably not wise to do so).

ReporterInitializeSTD

See the ReporterInitialize function above.

```
DWORD __stdcall ReporterInitializeSTD(
      const char *hostname,
      const char *port
);
```

ReporterSeenCallsignSTD

See the ReporterSeenCallsign function above.

```
DWORD __stdcall ReporterSeenCallsignSTD(
      const char *remoteInformation,
      const char *localInformation,
      unsigned long flags
);
```

ReporterTickleSTD

See the ReporterTickle function above.

```
DWORD __stdcall ReporterTickleSTD(
);
```

ReporterGetInformationSTD

See the ReporterGetInformation function above.

```
DWORD __stdcall ReporterGetInformationSTD(
      char *buffer,
      unsigned long maxlen
);
```

ReporterGetStatisticsSTD

See the ReporterGetStatistics function above. Note that this function returns the same structure, **including** the embedded wchar_t strings. This means that if the application wants to process those strings, then it needs to be cognizant that these strings are not converted to ANSI.

```
DWORD __stdcall ReporterGetStatisticsSTD(
      REPORTER_STATISTICS *buffer,
      unsigned long maxlen
);
```

ReporterUninitializeSTD

See the ReporterUninitialize function above.

```
DWORD __stdcall ReporterUninitializeSTD(
);
```