Draft Version 3

API Requirements for DNP3-to-DFS Translator

Overview

For the purposes of this document, we will use "API" when referring to DFS accessing the code that we are contracting to be written. We will use "Code" when referring to features or aspects of the code that will be part of the code that we are contracting to be written.

Specific API calls referenced in this document will follow this format

```
return_type functionName([parameter1, parameter2....])
```

As a general rule, each function call will return at least a success/fail boolean unless the call is trivial. Please default to returning a boolean.

Connections To DNP3 Devices

DNP3 devices connect usually through port 20000. The Code will need to manage each connected device, whether they are Masters or Outstations. We will need to be able to specify a different port via

```
bool setCommunicationPort(int port)
```

Determine connection count via

```
int connectedDevices()
```

Determine which Master or Outstations are connected. How this will be accomplished is not known at this time

```
list getConnectedDevices()
```

Set the API mode we going to run as

```
void useAsMaster()
```

```
void useAsOutstation()
```

Messaging

The general messaging needs are as follows:
- Note: the parameter "index" in the function calls below is the DNP3 point index.
- The Code will need to manage the necessary queues that the DNP3 spec requires.
- The Code will use the DNP “index” based address structure.
- API users will translate the DNP index for their specific needs

- Function Calls When Running As Outstation

determine if Master accepts unsolicited event traffic

```
bool acceptsUnsolicitedEvents()
```

send unsolicited event

```
bool sendDigitalEvent(int index, int value)
bool sendDigitalEvents(list events)
bool sendAnalogEvent(int index, float value)
bool sendAnalogEvents(list events)
bool sendAnalog32Event(int index, double value)
bool sendAnalog32Events(list events)
```

add an event to the queue for when the Master requests status updates

```
bool queueDigitalEvent(int index, int value)
bool queueDigitalEvents(list events)
bool queueAnalogEvent(int index, float value)
bool queueAnalogEvents(list events)
bool queueAnalog32Event(int index, double value)
bool queueAnalog32Events(list events)
```

add an event and time of event to the queue.

```
bool queueDigitalEvent(int index, int value, time_t time)
bool queueDigitalEvents(list events, time_t time)
bool queueAnalogEvent(int index, float value, time_t time)
bool queueAnalogEvents(list events, time_t time)
bool queueAnalog32Event(int index, double value, time_t time)
bool queueAnalog32Events(list events, time_t time)
```

direct status requests from Master

```
list statusRequests() #returns list of indices
```

class based status request

```
bool classOneRequested()
bool classTwoRequested()
bool classThreeRequested()
list classRequests() #returns list of requested classes
```

register function for Control Actions

```
bool registerControlActions(void *controlActionsFunction)
```

receive control action from Master

```
list controlActions()
```

- Function Calls When Running As Master

determine if outstation points are online.

```
list anyOffline()
```

send controls

```
bool setDigital(int index, int value)
bool setDigitals(list digitals)
bool setDigital(int index, bool value)
bool setAnalog(int index, float value)
bool setAnalogs(list analogs)
bool setAnalog32(int index, double value)
bool setAnalog32s(list analogs)
```

register function for notification that an unsolicited event has been received

```
bool registerDigitalEventsReceived(void *digitalEventReceivedFunction)
bool registerAnalogEventsReceived(void *analogEventReceivedFunction)
bool registerAnalog32EventsReceived(void *analog32EventReceivedFunction)
```

receive unsolicited traffic

```
list getDigitalEvents()
list getAnalogEvents()
list getAnalog32Events()
```

Returned in the list are classes with point value and time tag.

request status

```
bool requestClass(int class_number)
bool requestDigitalStatus(int index)
bool requestDigitalStatus(list digitals)
bool requestAnalogStatus(int index)
bool requestAnalogStatus(list analogs)
bool requestAnalog32Status(int index)
bool requestAnalog32Status(list analogs)
```

Outstation configuration

```
list getOutstationConfiguration()
bool setAcceptUnsolicitedEvents(bool value)
```

DFS Addressing to/from DNP3 Indices

DNP3 assigns an index to each telemetry point. The DNP3 Spec suggests that the indices be contiguous. Furthermore, the indices can be grouped into Classes. DNP3 allows for a Master to request status based on one or more indices, or the Master can query a Class for status.The API will use one DNP3 index to refer to a discrete telemetry point.

Example 1

The outstation receives a DNP3 message asking for a status update on the analog point at index 14. The Code adds index 14 to its queue named "status_requested". The API will provide a call that returns a copy of that "status_requested" queue.

```
list statusRequests()
```

The API user will then translate that index 14 on his/her own. The user would then make an API call to return the status of index 14.

```
bool queueDigitalEvent(int index, int value)
bool queueDigitalEvent(int index, bool value)
bool queueAnalogEvent(int index, float value)
bool queueAnalog32Event(int index, double value)
```

Example 2

The outstation receives a DNP3 message asking for Class 2 statuses.

```
bool classTwoRequested()
```

The above call would return true so the API user would gather the relevant data. The API would then add all points in Class 2 to the queue. The list contains classes with index, value & type.

```
bool classTwoResponse(list points)
```