NISTIR 7960

Programmers Guide to the Air-Handling Unit Performance Assessment Rule DLL

Michael A. Galler

National Institute of Standards and Technology
Technology Administration, U.S. Department of Commerce

NISTIR 7960

Programmers Guide to the Air-Handling Unit Performance Assessment Rule DLL

Michael A. Galler
Mechanical Systems and Control Group
Building Environment Division
Engineering Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899-8631

September 2013

Abstract

The air handling unit (AHU) Performance Assessment Rules (APAR) dynamically linked library (DLL) is an implementation of the APAR ruleset, which was developed for automated fault detection and diagnostics for air handling units [1-7]. While a list of the rules has been widely available, an implementation of them that is easy to access and use has not been. The APAR DLL was developed in Microsoft Visual Studio 2008, and was designed to increase ease of access to the analysis this rule set provides. By implementing the rules in a DLL, the rules are available for use by existing applications or new applications written specifically to take advantage of this new functionality. This includes commonly used tools such as MATLAB and LabView. By combining this with other programs such as the BACnet Communications DLL [8] or the BACnet Data Source [9], researchers can create powerful tools for Fault Detection and Diagnostics for use in buildings.

The purpose of this paper is to introduce the APAR DLL to heating, ventilation, and air conditioning (HVAC) designers, engineers, software developers, researchers, or anyone else interested in using the capabilities it provides. This paper provides instructions on how to interface with the DLL from a program written in C++. This information should be sufficient to allow users to interface from other computer languages or environments.

Disclaimers

Table of Contents

List of Tables

1. Introduction to APAR and the APAR DLL

The AHU Performance Assessment Rule (APAR) rule set is used to perform fault detection and diagnostics (FDD) on air handling units (AHUs). The APAR rule set was developed at NIST, and uses control signals and occupancy information to identify the particular mode of operation of the AHU, thereby identifying a subset of the rules that specify temperature relationships that are applicable for that mode [1]. The modes are described in Table 1. Subsets of the 28 APAR rules are applicable to each mode, as well as a subset of the rules which are applied in every mode. The APAR rule set has been used to evaluate building data from simulations and from field studies [1-7].

Table 1: Modes of Operation in APAR

Mode	Description
1	Heating
2	Cooling with outdoor air
3	Mechanical cooling with 100 % outdoor air
4	Mechanical cooling with minimum outdoor air
5	Unknown occupied modes
6	Stopped
7	Night cooling
8	Frost protection

The APAR rules have been encoded in a dynamically linked library (DLL) which allows them to be accessed by existing applications. They may also be integrated into new applications written for FDD or commissioning. NIST created the APAR DLL to become part of a semi-automated AHU commissioning tool [10]. It may also be used by other common analysis tools such as LabView and MATLAB[1].

The APAR DLL is written in C++, but may also be accessed by programs written in other computer languages, including C# and Visual Basic. The development environment used was Microsoft Visual Studio 2008. Detailed instructions on accessing the subroutines in the DLL from a program written in C++ are provided in this paper. The subroutines can be accessed using other computer languages or analysis tools by following the general instructions for accessing DLL's provided with the other language or analysis tool, and using the information provided in this guide. This guide documents Version 1.0 of the DLL.

[1] Any mention of commercial products in the APAR DLL or this guide is for information purposes only; it does not imply recommendation or endorsement by the National Institute of Standards and Technology (NIST).

2. Usage of the APAR DLL

Communications with the APAR DLL is performed by calling one of a number of subroutines. Each subroutine is designed to perform a specific task with a simple, easy to use interface.

The subroutines are divided into four categories: configuration, evaluation, data entry, and information. The available subroutines in each category are listed in Table 2. The APAR DLL uses several types of variables, with the usage and ranges as defined in Microsoft Visual Studio 2008 for C++. The usage and ranges are shown in Table 3.

Table 2- Subroutines in the APAR DLL

Category	Available Subroutines
Configuration	setAHUconfig
	getAHUconfig
	setAHUparameters
	getAHUparameters
	setAHUoccupancy
	getAHUoccupancy
Evaluation	evalData
Data Entry	addAHUrecord
	ReadFile
Information	getCauseDescription
	getRuleDescription

Table 3- Value ranges for variable types used in the APAR DLL

Variable Type	Description	Value Range
char	Used for storing a single character or a small number.	-128 to 127. 1 byte long.
double	Used for storing a floating point number.	(+/-) 1.7e(+/-) 308. 8 bytes long.
int	Used for storing an integer.	−2,147,483,648 to 2,147,483,647. 4 bytes long.
struct tm	Stores a time value, separated by date and time components.	This is a struct with several components. See Table 3 for the full description.
time_t	The number of seconds elapsed since 00:00 hours, Jan 1, 1970 UTC, stored as a single number. This is usually calculated by passing a tm value to a standard library function.	Same as int.
unsigned char	Used for storing a single character or a small number.	0 to 255. 1 byte long.
unsigned int	Used for storing an integer.	0 to 4,294,967,295. 4 bytes long.

The struct tm differs from the other variables in that it is a structure, which consists of several variables associated in one composite type. The members of the struct tm structure are shown in Table 4.

Table 4- The members of the tm structure

Variable Name	Description	Value Range
tm_sec	Seconds after the minute	0-60
tm_min	Minutes after the hour	0-59
tm_hour	Hours since midnight	0-23
tm_mday	Day of the month	1-31
tm_mon	Months since January	0-11
tm_year	Years since 1900	Same as int
tm_wday	Days since Sunday	0-6
tm_yday	Days since January 1	0-365
tm_isdst	Daylight Saving Time flag	0 or 1

Note that the range of tm_sec is 0 – 60, not 0 – 59. The extra range is to allow for leap seconds.

2.1. Use of the setAHUconfig subroutine

2.1.1. Description

The setAHUconfig subroutine allows the user to set the details of the configuration of the AHU under test. An array containing configuration options is passed to the subroutine. All parameters will be set to the values in this array when this subroutine is called. If not modifying all of the values, it is recommended to initialize the array by calling getAHUconfig first and then modifying only the values you wish to change.

2.1.2. Declaration

The subroutine is declared as:
```
__declspec(dllexport) int __cdecl setAHUconfig(unsigned char
*ip);
```

The declaration in the calling program should be:
```
extern __declspec(dllimport) int __cdecl setAHUconfig(unsigned
char *ip);
```

2.1.3. Arguments and Return Value

The input argument is an array of unsigned characters indicating configuration options of the AHU under test, and is described in Table 5. The array is eight bytes long, and functions like an array of Boolean values. The AHU options are listed in Table 6. To enable an option, set the value of its position to 1. To disable an option, set the value to 0.

Table 5- Explanation of arguments to the setAHUconfig subroutine

Argument	Description
ip	A pointer to an array of unsigned characters eight bytes long. A description of the values for each position is found in Table 6.

Table 6- Options for setAHUconfig

Position	Label	Description
1	heat	Does the AHU have heating capability
2	cooling	Does the AHU have cooling capability
3	recovery	Does the AHU have heat recovery capability
4	mixing	Does the AHU have a mixing box
5	humidity	Does the AHU have humidity control
6	nightCooling	Is the AHU configured for night cooling
7	frostProtection	Is the AHU configured for frost protection
8	econ_Flag	Does the AHU have an economizer

The return values for setAHUconfig are described in Table 7. The subroutine will return a value of 1 if there were no errors, and a value of 0 if there was an error.

Table 7- Description of return values for the setAHUconfig subroutine

Return Value	Description
1	No errors.
0	An error was encountered.

2.2. Use of the getAHUconfig subroutine

2.2.1. Description

The getAHUconfig subroutine allows the user to retrieve the details of the configuration of the AHU under test. An empty array is passed to the subroutine, and filled with the configuration information on return. The array is identical to that used for setAHUconfig, and is described in Table 6.

2.2.2. Declaration

The subroutine is declared as:

```
__declspec(dllexport) int __cdecl getAHUconfig(unsigned char *ip);
```

The declaration in the calling program should be:

```
extern __declspec(dllimport) int __cdecl getAHUconfig(unsigned char *ip);
```

2.2.3. Arguments and Return Value

The input argument is an empty array of unsigned characters indicating configuration options of the AHU under test. See Table 6 for details of the array. The array will be overwritten with the details of the current configuration of the AHU under test.

Table 8- Explanation of arguments to the getAHUconfig subroutine

Argument	Description
ip	A pointer to an array of unsigned characters eight bytes long. A description of the values for each position is found in Table 6. This array will be filled by the getAHUconfig subroutine.

The return values for getAHUconfig are described in Table 9. The subroutine will return a value of 1 if there were no errors, and a value of 0 if there was an error.

Table 9- Description of return values for the getAHUconfig subroutine

Return Value	Description
1	No errors.
0	An error was encountered.

The subroutine will return a value of 1 if there were no errors, and a value of 0 if there was an error.

2.3. Use of the setAHUparameters subroutine

2.3.1. Description

The setAHUparameters subroutine allows the user to set parameters used by the ruleset when evaluating data. An array containing parameter values is passed to the subroutine. All parameters will be set to the values in this array when this subroutine is called. If not modifying all of the values, it is recommended to initialize the array by calling setAHUparameters first and then modifying only the values you wish to change.

2.3.2. Declaration

The subroutine is declared as:
```
__declspec(dllexport) int __cdecl setAHUparameters (double *ip);
```

The declaration in the calling program should be:
```
extern __declspec(dllimport) int __cdecl setAHUparameters (double *ip);
```

2.3.3. Arguments and Return Value

The input argument is an array of double values indicating parameters to be used by the APAR ruleset. The input argument is described in Table 10. The array is 37 units long. The parameters are listed in Table 11. Each position should have its value set before calling the subroutine. The set of current values can be retrieved by calling the getAHUparameters subroutine.

Table 10- Explanation of arguments to the setAHUparameters subroutine

Argument	Description
ip	A pointer to an array of 37 double variables. A description of the values for each position is found in Table 11.

Table 11- Values for setAHUparameters

Position	Label	Description	Default
0	Epsilon_hc	Heating coil signal threshold.	0.005
1	Epsilon_cc	Cooling coil signal threshold	0.05
2	Epsilon_d	Mixing box damper control signal threshold	0.155
3	Epsilon_f	Air fraction threshold	0.3
4	Epsilon_h	Humidify signal threshold	2.33
5	Qoa_frac_min	Minimum outdoor air fraction	0.15
6	Epsilon_t	Temperature threshold	1.0
7	DeltMin	Minimum change in temperature	5.56
8	DelTsf	Change in temperature across supply fan	1.11
9	DelTrf	Change in temperature across return fan	1.0
10	Tco	Outdoor air changeover temperature	36
11	Uccmin	Minimum cooling coil signal	0.0
12	Uccmax	Maximum cooling coil signal	1.0
13	Uhcmin	Minimum heating coil signal	0.0
14	Uhcmax	Maximum heating coil signal	1.0
15	Udmin	Minimum mixing box damper	0.0
16	Udmax	Maximum mixing box damper	1.0
17	Uhmin	Minimum humidity command	0.0
18	Uhmax	Maximum humidity command	1.0
19	Urmin	Minimum recovery command	0.0
20	Urmax	Maximum recovery command	1.0
21	Epsilon_r	Recovery signal threshold.	1.0
22	Epsilon_Tmax	Maximum allowable temperature difference	100
23	Hrasmin	Minimum return air humidity set point	20.0
24	Hrasmax	Maximum return air humidity set point	60.0
25	DelccMax	Number of sign changes of the cooling coil control signal per hour.	1.0
26	DelHcmax	Number of sign changes of the heating coil control signal per hour.	1.0
27	DelHmax	Number of sign changes of the humidifier control signal per hour.	1.0
28	TsetMax	Maximum air temperature set point	24
29	TsetMin	Minimum air temperature set point (for band)	16
30	MTmax	Number of mode transitions per hour.	4
31	OccMin	Minimum value for Occupancy.	0
32	OccMax	Maximum value for Occupancy.	1
33	UodMin	Minimum value for outdoor air damper.	0.0
34	UodMax	Maximum value for outdoor air damper.	1.0
35	Epsilon_Uod	Outdoor air damper threshold.	0.005
36	sensitivity	Threshold value modifier.	1.0

The return values for setAHUparameters are described in Table 12. The subroutine will return a value of 1 if there were no errors, and a value of 0 if there was an error.

Table 12- Description of return values for the setAHUparameters subroutine

Return Value	Description
1	No errors.
0	An error was encountered.

2.4. Use of the getAHUparameters subroutine

2.4.1. Description

The getAHUparameters subroutine allows the user to retrieve the values of the parameters used by the APAR ruleset. An empty array is passed to the subroutine, and filled with the parameter values on return. The array is identical to that used for setAHUparameters, and is described in Table 11.

2.4.2. Declaration

The subroutine is declared as:

```
__declspec(dllexport) int __cdecl getAHUparameters (double *ip);
```

The declaration in the calling program should be:

```
extern   __declspec(dllimport)   int   __cdecl   getAHUparameters
(double *ip);
```

2.4.3. Arguments and Return Value

The input argument is an empty array of double values. See Table 13 for details of the array. The array will be overwritten with the details of the current configuration of the AHU under test.

Table 13- Explanation of arguments to the getAHUparameters subroutine

Argument	Description
ip	A pointer to an array of 37 double variables. A description of the values for each position is found in Table 11.

The return values for getAHUparameters are described in Table 14. The subroutine will return a value of 1 if there were no errors, and a value of 0 if there was an error.

Table 14- Description of return values for the getAHUparameters subroutine

Return Value	Description
1	No errors.
0	An error was encountered.

2.5. Use of the setAHUoccupancy subroutine

2.5.1. Description

The setAHUoccupancy subroutine allows the user to set occupancy schedule, indicating the time the building is occupied or unoccupied. A variable indicating the day or days to be set, and values indicating the start and stop time for occupancy are passed to the subroutine. There is only one week schedule and changes to the schedule will persist in the current session until they are modified again. The occupancy is stored internally as an array of length 1440 (24 x 60) for each day, giving one-minute resolution to the occupancy schedule.

2.5.2. Declaration

The subroutine is declared as:

```
__declspec(dllexport) int __cdecl setAHUoccupancy (unsigned int
day, int hourStart, int minStart, int hourEnd, int minEnd, int
occ);
```

The declaration in the calling program should be:

```
extern   __declspec(dllimport)   int   __cdecl   setAHUoccupancy
(unsigned int day, int hourStart, int minStart, int hourEnd, int
minEnd, int occ);
```

2.5.3. Arguments and Return Value

Calling setAHUoccupancy sets a period of time to occupied or unoccupied status. There are six input arguments to setAHUoccupancy. The inputs are described in Table 15. The first is an unsigned integer indicating the day which is being set. The values for this are shown in Table 16. The second and third input arguments are integer values indicating the hour and minute being set. The fourth and fifth input arguments are integer values indicating the hour and minute being set. The sixth input argument indicates whether the period is set to occupied or unoccupied status. Calling setAHUoccupancy will overwrite the portion of the array indicated by the period. The current occupancy schedule can be retrieved by calling the getAHUoccupancy subroutine.

Table 15- Explanation of arguments to the setAHUoccupancy subroutine

Argument	Description
day	An unsigned integer value representing the day of week. This should be set to one of the values in Table 16.
hourStart	An integer variable 0 - 23, indicating the first hour being set.
minStart	An integer variable 0 - 59, indicating the minute of the first hour being set.
hourEnd	An integer variable 0 - 23, indicating the last hour being set.
minEnd	An integer variable 0 - 59, indicating the minute of the last hour being set.
occ	An integer variable representing the status being set. 1 = occupied, 0 = unoccupied

Table 16- Values for the day of week parameter

Value	Day of Week
1	Sunday
2	Monday
3	Tuesday
4	Wednesday
5	Thursday
6	Friday
7	Saturday
8	All weekdays (Monday – Friday)
9	All weekend days (Saturday and Sunday)
10	All days

The return values for setAHUoccupancy are described in Table 17. The subroutine will return a value of 1 if there were no errors and a value of 0 if there was an error.

Table 17- Description of return values for the setAHUparameters subroutine

Return Value	Description
1	No errors.
0	An error was encountered.

2.6. Use of the getAHUoccupancy subroutine

2.6.1. Description

The getAHUoccupancy subroutine allows the user to retrieve the occupancy schedule, which indicates the time the building is occupied. A variable indicating the day to be read, and an array which will contain the occupancy values are passed to the subroutine. The schedule for one day can be retrieved at a time.

2.6.2. Declaration

The subroutine is declared as:
```
__declspec(dllexport) int __cdecl getAHUoccupancy (unsigned int
day, unsigned char *ip);
```

The declaration in the calling program should be:
```
extern __declspec(dllimport) int __cdecl getAHUoccupancy
(unsigned int day, unsigned char *ip);
```

2.6.3. Arguments and Return Value

There are two input arguments to getAHUoccupancy. The arguments are described in Table 18. The first is an unsigned integer indicating the day which is being read. The values for this are 1-7 as shown in Table 16. The second input argument is an array of unsigned char values. The array is 1440 (24 x 60) members long, with each member representing one minute of the day. This array will be filled with the occupancy data. The occupancy is represented in each member as 0 for unoccupied, and 1 for occupied. The first position in the array corresponds to 12:00 AM, the second to 12:01 AM, and so on. The final position in the array corresponds to 11:59 PM.

Table 18- Explanation of arguments to the getAHUoccupancy subroutine

Argument	Description
day	An unsigned integer value representing the day of week. This should be set to a value of 1 – 7, as described in Table 16.
ip	An unsigned character array, length 1440.

The subroutine will return a value of 1 if there were no errors, and a value of 0 if there was an error. Note that the subroutine can only get the schedule for one day at a time. If a value of 8, 9 or 10 is sent as the day of week parameter an error will be returned.

2.7. Use of the evalData subroutine

2.7.1. Description

Calling the evalData subroutine causes the APAR DLL to evaluate the HVAC data either at a specific date/time or over a time span. The HVAC data must be passed to the APAR DLL before the evalData subroutine is called. The results of the evaluation are returned in the rules and causes arrays.

2.7.2. Declaration

The evalData subroutine is declared with four configurations:

```
__declspec(dllexport) int __cdecl evalData(time_t time, int *rules, int *causes);

__declspec(dllexport) int __cdecl evalData(struct tm *intime, int *rules, int *causes);

__declspec(dllexport) int __cdecl evalData(time_t tstart, time_t tend, int *rules, int *causes);

__declspec(dllexport) int __cdecl evalData(struct tm *tstart, struct tm *tend, int *rules, int *causes);
```

The declaration in the calling program should be:

```
extern __declspec(dllimport) int __cdecl evalData(time_t time, int *rules, int *causes);

extern __declspec(dllimport) int __cdecl evalData(struct tm *intime, int *rules, int *causes);

extern __declspec(dllimport) int __cdecl evalData(time_t tstart, time_t tend, int *rules, int *causes);

extern __declspec(dllimport) int __cdecl evalData(struct tm *tstart, struct tm *tend, int *rules, int *causes);
```

2.7.3. Arguments and Return Value

Each of the configurations has two types of arguments. The first type passes date/time values into the APAR DLL, while the second is used for return values. The input arguments are described in Table 19.

The first two configurations have one `time` argument. If called, the APAR DLL will evaluate the data at the date/time passed in the `time` argument. The `time` argument may be passed as a `time_t` variable or a pointer to a `tm` struct. The second two configurations are used to evaluate a time span, and have two arguments related to time. The `tstart` argument indicates the beginning of the time span, and the `tend` argument indicates the end. Both may be passed as a `time_t` variable or a pointer to a `tm` structure. Note that when setting the year, the value is set as the number of years since 1900. The year 2012 would be entered as 112.

Table 19- Explanation of arguments to the evalData subroutine

Argument	Description
time	The time and date information for the evaluation. This variable is of type time_t, which is explained in Table 3.
intime	The time and date information for the evaluation. This variable is of type struct tm, which is explained in Table 3.
tstart	When entering a time span for the evaluation, this is the time and date information for the start of the span. Depending on the configuration used it may be of type time_t or struct tm.
tend	When entering a time span for the evaluation, this is the time and date information for the end of the span. Depending on the configuration used it may be of type time_t or struct tm.
rules	A pointer to an array of integers, allocated to a length of at least 30 members. It will be returned with a list of rules triggered at the time or during the time span of evaluation.
causes	A pointer to an array of integers, allocated to a length of at least 25 members. It will be returned with information of which causes were triggered at the time or during the time span of evaluation.

The return values are identical in all four configurations. Two arrays of integer values are passed to the APAR DLL, which fills them with the results of the analysis. The `rules` array must be allocated to a length of 30 members. The contents of the rules array will be filled with a list of rules that the APAR DLL found were triggered by the evaluated data. The `causes` array must be allocated to a length of 25 members. The causes array is filled with a count of the times each one of the 25 possible causes is signaled by a rule. For example if the data indicated a violation of Rule 2, then the `rules` array would hold a value of [2], and the `causes` array would have members 2, 3, 4, 18, and 19 (the causes indicated by Rule 2) set to a value of 1. Descriptions of the rules and causes referenced by the arrays can be retrieved by use of the getCauseDescription and getRuleDescription subroutines. Note that the number of rules and causes may change in future versions of this program, and that the current value for each can also be obtained by use of the getCauseDescription and getRuleDescription subroutines.

The subroutine will return a value of 1 if there were rules returned, a value of 0 if there were no rules returned, and a value of -1 if there was an error.

2.8. Use of the addAHUrecord subroutine

2.8.1. Description

The addAHUrecord subroutine is called to add HVAC data to the APAR DLL for later evaluation. HVAC data must be passed to the APAR DLL before the evalData subroutine is called.

2.8.2. Declaration

The addAHUrecord subroutine is declared with 3 configurations:

```
__declspec(dllexport) int __cdecl addAHUrecord(time_t time, double *ip);

__declspec(dllexport) int __cdecl addAHUrecord(struct tm *time, double *ip);

__declspec(dllexport) int __cdecl addAHUrecord(int year, int month, int day, int hour, int minute, int second, double *ip);
```

The declaration in the calling program should be:

```
extern __declspec(dllimport) int __cdecl addAHUrecord(time_t time, double *ip);

extern __declspec(dllimport) int __cdecl addAHUrecord(struct tm *time, double *ip);

extern __declspec(dllimport) int __cdecl addAHUrecord(int year, int month, int day, int hour, int minute, int second, double *ip);
```

2.8.3. Arguments and Return Value

Each of the configurations has two types of arguments. The first type passes date/time values into the APAR DLL, while the second is used to pass the building performance data to be analyzed.

The first two configurations have one `time` argument. The `time` argument may be passed as a `time_t` variable or a pointer to a `tm` struct. The third configuration uses separate arguments for each component of the date and time- year, month, day, hour, minute, and second. Note that when setting the year, the value is set as the number of years since 1900. The year 2012 would be entered as 112.

The `ip` argument contains a record of HVAC data. The data in a record must be in the order shown in Table 20.

Table 20- Order of input variables for the addAHUrecord subroutine

ID	Label	Units
1	Occupancy	n/a
2	Supply air setpoint	C
3	Supply temp	C
4	Return temp	C
5	Mixed air temp	C
6	Outdoor air temp	C
7	Cooling coil	%
8	Heating coil	%
9	Mixing box damper	%
10	Recirculation damper- return, recovery command	%
11	Humidity control valve for supply air	%
12	RH of outside air	% RH
13	Mixing temp	C
14	Return air setpoint	C
15	Outdoor air damper	%
16	Return air humidity	% RH

The values with units listed as % or % RH have a range of 0.0 – 1.0. Occupancy must be set to 0 or 1. The subroutine will return a value of 1 if there were no errors, and a value of 0 if there was an error. The return values for setAHUoccupancy are described in Table 21.

Table 21- Description of return values for the addAHUrecord subroutine

Return Value	Description
1	No errors.
0	An error was encountered.

2.9. Use of the ReadFile subroutine

2.9.1. Description

The ReadFile subroutine is used to enter an entire file of data into the APAR DLL. The file must be formatted to have one data record on each line. Lines that start with the hash symbol '#' will be treated as comments, as will lines with any letters in them. Each line starts with a date and time, followed by data. The date must be in the format DD/MM/YY. The time must be in the format hh:mm:ss, where hh is in 24 hour format. The time is followed by the data, in the order given in Table 20. The date, time, and all other entries must be separated by commas.

2.9.2. Declaration

The ReadFile subroutine is declared with two configurations:

```
__declspec(dllexport) int __cdecl ReadFile(char *ip);

__declspec(dllexport) int __cdecl ReadFile(FILE *ip);
```

The declaration in the calling program should be:

```
extern __declspec(dllimport) int __cdecl ReadFile(char *ip);
extern __declspec(dllimport) int __cdecl ReadFile(FILE *ip);
```

2.9.3. Arguments and Return Value

In the first configuration listed, there is one argument to ReadFile, the name of the input file in a character string. In the second configuration, the argument is a pointer to a FILE variable. This handle must already be opened before passing it to ReadFile. The arguments are described in Table 22.

Table 22- Explanation of arguments to the ReadFile subroutine

Argument	Description
char ip	The name of the input file in a character string. Example: “AHU data.csv”
FILE *ip	A FILE variable for the input file. This should be set using the fopen() function. See Figure 1 for a full example.

The subroutine will return a value of 1 if there were no errors, and a value of 0 if there was an error. The return values for ReadFile are described in Table 23.

Table 23- Description of return values for the ReadFile subroutine

Return Value	Description
1	No errors.
0	An error was encountered.

2.10. Use of the getCauseDescription subroutine

2.10.1. Description

The getCauseDescription subroutine is used to get a text description of one of the 25 causes.

2.10.2. Declaration

The getCauseDescription subroutine is declared as:
```
__declspec(dllexport) int __cdecl getCauseDescription(int cause,
char *text);
```

The declaration in the calling program should be:
```
extern __declspec(dllimport) int __cdecl getCauseDescription(int
cause, char *text);
```

2.10.3. Arguments and Return Value

There are two arguments to getCauseDescription. The first argument is an integer which must be set to the number of the cause being referenced. The second argument is a character string which will contain the text of the description. If the first argument is set to a value of 0, the `text` variable will be returned with the number of cause descriptions available, as a text string, i.e. “25” not the value 25. The arguments to getCauseDescription are described in Table 24.

Table 24- Explanation of arguments to the getCauseDescription subroutine

Argument	Description
cause	The index of the cause being requested. Set to 0 to retrieve the number of cause descriptions available.
text	A character string which will be filled with the text describing the cause requested. See Figure 1 for a full example.

The subroutine will return a value of 1 if there were no errors, and a value of 0 if there was an error. The return values for setAHUoccupancy are described in Table 25.

Table 25- Description of return values for the getCauseDescription subroutine

Return Value	Description
1	No errors.
0	An error was encountered.

2.11. Use of the getRuleDescription subroutine

2.11.1. Description

The getRuleDescription subroutine is used to get a text description of one of the 28 possible rules.

2.11.2. Declaration

The getRuleDescription subroutine is declared as:
```
__declspec(dllexport) int __cdecl getRuleDescription (int rule,
char *text);
```

The declaration in the calling program should be:
```
extern __declspec(dllimport) int __cdecl getRuleDescription (int
rule, char *text);
```

2.11.3. Arguments and Return Value

There are two arguments to getRuleDescription. The first argument is an integer which must be set to the number of the rule being referenced. The second argument is a character string which will contain the text of the description. If the first argument is set to a value of 0, the `text` variable will be returned with the number of rule descriptions available, as a text string, i.e. “28” not the value 28. The arguments to getRuleDescription are described in Table 26.

Table 26- Explanation of arguments to the getRuleDescription subroutine

Argument	Description
rule	The index of the rule being requested. Set to 0 to retrieve the number of rule descriptions available.
text	A character string which will be filled with the text describing the rule requested. See Figure 1 for a full example.

The subroutine will return a value of 1 if there were no errors, and a value of 0 if there were an error. The return values for setAHUoccupancy are described in Table 27.

Table 27- Description of return values for the getRuleDescription subroutine

Return Value	Description
1	No errors.
0	An error was encountered.

3. Using the APAR DLL- Code Samples

The following code snippets demonstrate how to use the APAR DLL by telling it to analyze an entire data file. Not every function is shown, but the techniques needed for each function are demonstrated.

3.1. Basic Use

This sample demonstrates the methods needed for a basic implementation of the code.

```
struct tm tms;
int   rules[61];  // NOTE- length + 1
int   causes[26];  // NOTE- length + 1
int   i; // NOTE- check i for errors after each function call (not shown)
char  t1[360];
int   nRule;
double params[37];
unsigned char ahuconfig[] = {1,1,1,1,1,1,1};
FILE *ip;

memset(&tms,0,sizeof(struct tm));
// set tms to first line time in file- 7:00 AM on August 16, 2010
tms.tm_hour = 7;
tms.tm_min = 0;
tms.tm_sec = 0;
tms.tm_year = 110;
tms.tm_mon = 8;
tms.tm_mday = 16;
ttstart = mktime(&tms);

i = setAHUconfig(ahuconfig);  // Set the AHU config
i = getAHUparameters(params);  // Retrieve parameter array
params[5] = 0.15; // set Qoa_frac_min to 15%
                  // Note that index is 5, not 6, due to 0-based arrays
i = setAHUparameters(params); // Set parameter array after modification

i = ReadFile("APARWeek34.csv");  // Read in an entire data file
ip = fopen(“APARWeek35.csv”,”r”);
i = ReadFile(ip); // Read in an entire data file using file pointer
i = getCauseDescription(0,t1);  // Retrieve count of available causes
printf("There are %d causes\n",i);

nRule = getRuleDescription(0,t1);  // Retrieve count of available rules
printf("There are %d rules\n",nRule);

i = evalData(tms,rules,causes);  // evaluate stored data
// Print index for each fault found
for(i=0;(rules[i] > 0)&&(i < nRule);i++){
    printf(“APAR found fault %d\n”,rules[i]);
    getRuleDescription(i,t1);
    printf(“Rule description: %s\n”,t1);
}
```

Figure 1- Sample Code for Multiple Subroutines

3.2. Using the addAHUrecord subroutine

The following code snippet demonstrates how to send a single set of data to the APAR DLL. The subroutine SendDataToAPAR would be declared as part of a program or class. Note that the timestamp and data are passed into the example subroutine.

```
int SendDataToAPAR(struct tm tms, double *myData)
{
      int   i;

      i = addAHUrecord(tms,myData);  // Pass the data to the DLL

      // NOTE- check i for errors after each function call (not shown)

      return(i);
}
```

Figure 2- Sample Code for the addAHUrecord Subroutine

3.3. Using the getAHUoccupancy and setAHUoccupancy subroutines

The following code snippet demonstrates how to modify the occupancy schedule used by the APAR DLL. The subroutine setAHUoccupancy would be declared as part of a program or class. Note that the timestamp and data are passed into the example subroutine.

```
// variable to hold occupancy data
unsigned char sched[1440];

// not shown: check return value for errors after each function call
//-----
// set occupied on Thursday from 6 – 7 PM.
i = setAHUoccupancy(5,18,0,19,0,1); // see Section 2.5 for parameter info
//-----
// set unoccupied on Friday from 3:15 (15:15) to 6:45 PM (18:45).
i = setAHUoccupancy(6,15,15,18,45,0);
//-----
// set unoccupied on Saturday and Sunday from 10 AM to 1:30 PM.
i = setAHUoccupancy(9,10,0,13,30,0);
//-----
// set unoccupied on all of Monday
i = setAHUoccupancy(2,0,0,23,59,0);
//-----
// get the occupancy schedule for Thursday
i = getAHUoccupancy(5,sched);
```

Figure 3- Sample Code for Occupancy Subroutines

4. Summary

The use of the NIST APAR DLL has been documented in order to enable the adaptation of this software by a wide range of users. It is hoped that improved access to these rules will encourage the use of fault detection and diagnostics by a wider audience. When combined with the NIST BACnet DLL software, users can create powerful tools for detecting faults and analyzing building performance.

5. References

[1] House, J.M., Vaezi-Nejad, H., and Whitcomb, J.M., *An Expert Rule Set for Fault Detection in Air-Handling Units*, ASHRAE Transactions, Vol. 107, Pt. 1, 2001.

[2] Milesi-Ferretti, N.S., Schein, J, Park, C.D. , Galler, M.A., Bushby, S.T., House, J, *Results from Simulation and Laboratory Testing of Air Handling Unit and Variable Air Volume Box Diagnostic Tools*, NISTIR 6964, January 2003.

[3] Schein, J, Bushby, S.T., Milesi-Ferretti, N.S., House, J, *Results from Laboratory Testing of Embedded Air Handling Unit and Variable Air Volume Box Fault Detection Tools*, NISTIR 7036, August 2003.

[4] Schein, J, Bushby, S.T., House, J, *Results from Field Testing of Air Handling Unit and Variable Air Volume Box Fault Detection Tools*, NISTIR 6994, April 2003.

[5] Schein, J, Bushby, *Automated Fault Detection and Diagnostics for Air Handling Units and VAV Boxes*, ASHRAE Journal Vol. 47 No. 7, July 2005

[6] Schein, J, Bushby, Milesi-Ferretti, N.S., *A Rule-Based Fault Detection Method for Air Handling Units*, Energy & Buildings Vol. 38, Issue 12, December 2006, pp. 1485 – 1492, 2006

[7] Schein, J., *Results from Field Testing of Embedded Air Handling Unit and Variable Air Volume Box Fault Detection Tools*, NISTIR 7365, 2006

[8] Galler, M.A., *Using the BACnet Communications DLL v1.0*, NIST TN 1607, 2008

[9] Galler, M.A., *Using the BACnet Data Source v1.7*, NISTIR 7825, 2011

[10] CITE-AHU, Version 2.0, NIST, 2013.