SAE Aerospace
An SAE International Group

AEROSPACE RECOMMENDED PRACTICE

SAE ARP4868 REV. A

Issued 2001-10
Revised 2011-05

Superseding ARP4868

(R) Function-Based API for Gas Turbine Engine Performance Programs

RATIONALE

Gas turbine engine manufacturers (suppliers) have long provided their customers with computer programs that simulate engine performance. Application manufacturers and others (customers) use these programs, often called models or simulations, in design studies, mission analysis, life cycle analysis, and performance prediction of their products. These models are used throughout the life of a product, from conceptual design through production, deployment, field use, maintenance, and overhaul. Communication between suppliers and customers is more productive and less error prone if all engine models adhere to common guidelines with respect to presentation of data and interface with other computer programs.

Rev A represents a substantial revision and simplification of the API presented in the original document. The intent of the document is the same but all function names and their arguments have been changed.

TO PLACE A DOCUMENT ORDER: **Tel: 877-606-7323 (inside USA and Canada)**
Tel: +1 724-776-4970 (outside USA)
Fax: 724-776-0790
Email: CustomerService@sae.org
SAE WEB ADDRESS: **http://www.sae.org**

SAE values your input. To provide feedback on this Technical Report, please visit
http://www.sae.org/technical/standards/ARP4868A

TABLE OF CONTENTS

1. SCOPE

The SAE Aerospace Standard document AS681 is the parent document of this SAE Aerospace Recommended Practice (ARP). AS681 applies to Engine programs written to conform to this document. This ARP specifies a set of functions and their expected behaviors that constitute a function based Application Program Interface (API) for gas turbine engine customer programs. The functions specified in this API are delivered by the Supplier as part of the Engine model.

This document defines generic language independent functions and specific appendices for implementations in C and Fortran.

The function based API specified in this ARP represents an alternative to the Fortran COMMON block structure, as specified in AS4191, historically used to communicate with an engine program. The customer may request emulation of the AS4191 interface if desired.

This document does not specify how the parameter names in the Engine program are constructed, how program capabilities might be expanded or altered, or how error messages are constructed. See AS755 for overall guidelines for nomenclature. See ARP5571 for information on nomenclature, expanding program operational capabilities and generating error values and messages for object-oriented models.

2. REFERENCES

2.1 Applicable Documents

The following publications form a part of this document to the extent specified herein. The latest issue of SAE publications shall apply. In the event of conflict between the text of this document and references cited herein, the text of this document takes precedence. Nothing in this document, however, supersedes applicable laws and regulations unless a specific exemption has been obtained.

2.1.1 SAE Publications

Available from SAE International, 400 Commonwealth Drive, Warrendale, PA 15096-0001, Tel: 877-606-7323 (inside USA and Canada) or 724-776-4970 (outside USA), www.sae.org.

AS681	Gas Turbine Engine Steady-State and Transient Performance Presentation for Computer Programs
AS755	Aircraft Propulsion System Performance Station Designation and Nomenclature
AS4191	Gas Turbine Engine Performance Presentation for Computer Programs Using Fortran
ARP5571	Gas Turbine Performance Presentation and Nomenclature for Object-Oriented Computer Programs

2.2 Definitions

API: Application Program Interface, a protocol agreed to between the supplier and the customer for interaction between the customer's computer program and the supplier's engine performance program, when the supplier's program is operating as a subprogram.

CALLING PROGRAM: The program responsible for interfacing with the Engine Program through the API functions defined in this document.

CUSTOMER: The person or organization which receives and executes the Engine Program.

EMPTY STRING: An empty string contains either an initial ASCII NULL character or all blank characters.

ENGINE PROGRAM: Gas turbine engine performance program. The API functions defined in this document are supplied as a part of the Engine Program.

ENGINEERING STATUS INDICATOR (ESI): An 8- or 9-digit integer conveying error and warning information defined in ARP5571. This is an extension of the 4-digit Numerical Status Indicator (NSI) concept defined in AS4191.

NUMERICAL STATUS INDICATOR (NSI): A 4-digit integer issued by AS4191 compliant Engine Programs.

PARAMETER NAME: Engine Program variable names as defined in the Engine Program documentation.

SEVERITY: Severity is an integer that indicates the seriousness of an Engine Program error condition. The valid values, 0-9 and 99 are described in ARP5571. The variable severityMax is the most serious/meaningful severity error/warning encountered.

STRING: Character strings in this document are structured as appropriate for the language used to implement the API functions.

SUPPLIER: The organization or person that delivered or created the Engine Program.

3. OVERVIEW

Historically gas turbine Engine Programs have used a Fortran-based API that relied on COMMON blocks for data communication. AS4191 defines this interface.

This document defines a function-based API for Engine Programs that can support any programming language and decouples the variable storage and naming between the Calling and Engine Programs. The Calling Program can set or get Engine Program parameter values by specifying their parameter names and the Calling Program can associate the engine parameter to any Calling Program variable name it chooses. All interactions between the Calling program and the Engine Program should be through the API functions.

The supplier includes the API functions as an integral part of the Engine Program. This document specifies only those aspects and behaviors of the API functions that are visible to the Calling Program and Suppliers are free to determine how they implement the API functions. The Supplier is not required to provide the API in all languages specified in this document.

The body of this document discusses the expected functionality and the inputs and outputs for each function. The API functions as implemented for specific programming languages are shown in the appendices. The Customer and Supplier must agree to any modifications or extension to the APIs for languages currently specified in this document or to extend this document to encompass an additional programming language.

4. PROGRAM DOCUMENTATION

The Engine Program supplier will supply all the documentation specified in AS681. The following additional information that is unique to Engine Programs conforming to ARP4868 should also be included with the user documentation:

a. The specific programming language used to implement the API functions.

b. List of supplied functions along with their implementation status.

c. Documentation of the Engine Program initialization process. This should include a listing of any initialization arguments, a list of files or strings to be given to the engine to be parsed, and any input parameter values that must be set before the Engine Program is ready to be run. If the program initialization is dependent on the order any of this information is specified, that fact should be noted and the correct order specified.

d. An example Calling Program illustrating the usage of the API in the requested language.

5. ATTRIBUTES AND CONVENTIONS

5.1 Function Names

All API functions exist in the global name space. The function names in this API are constructed to guarantee that name collisions with other global functions are avoided. Using the **`x4868getI1D`** function as an example, the following pattern is used to construct function names:

`x` Leading lower case letter or letters that uniquely indicate the programming language the function is written in. Recommended prefixes are listed below. In the body of this document, function names begin with an "**`x`**" as a placeholder.

Symbol	Language
c	C, C++
f	Fortran
j	Java
pl	Perl
py	Python
vb	Visual Basic, VBA (Excel)

`4868` Designates an ARP4868 API function, to avoid name collisions with other functions in the global name space.

`get` Indicates primary action of the function. Actions include: get or set data, perform initialization, parse additional input, run the Engine Program or terminate the Engine Program.

`I` Character indicating the type of the Calling Program variable being referenced. Data types supported are "**`I`**" – integer, "**`F`**" - single precision floating point, "**`D`**" - double precision floating point, and "**`S`**" - character string. Functions that do not operate on numerical data will not have these single letter type indicators. For example, the **`x4868getUnits`** function indicates that it is a get function returning the units string.

`1D` Indicates a 1D array is being referenced. Functions for 2D and 3D are also defined in this API. This portion of the name is only present for array handling functions.

To avoid undue repetition, related functions are grouped together in individual sections. Square bracket, regular expression, notation is used to indicate that multiple variations exist. For instance, "**`[IFDS]`**" indicates the various variable type options: integer, single precision real, double precision real, or character string data. For array data, "**`[123]`**" indicates the dimension of the array. This last designator is not present for scalar parameters. In the appendices, the specification for a given programming language will give the function prototypes for each function separately.

5.2 Engine Program Error Reporting

The API functions direct the Engine Program to perform actions. These actions may cause Engine Program errors and warnings, however these errors and warnings are not directly reported by the API functions. The **x4868run** function is the most obvious function that can produce Engine Program errors, however many other functions also can, for instance the functions that “parse” files or strings and “set” functions may set invalid values. Engine Program errors and warnings are communicated through Engine Program parameters. It is recommended that **x4868getSeverityMax** be used by the Calling Program after every **x4868run** call and as appropriate after other API function calls.

The status of the Engine Program is traditionally communicated through Numerical Status Indicators (NSI) or more recently Engineering Status Indicators (ESI). More than one NSI or ESI can occur during a given run, so these status indicators are held in a 1-D integer array parameter. This array can be obtained using the **`x4868getI1D`** function. The Engine Program may also have a 1-D string array parameter that contains descriptive strings corresponding to the status indicator values in the 1-D integer array.

5.3 API Function Error Reporting

Each API function reports a function status “functStatus” resulting from the execution of the API function itself, this is not to be confused with underlying Engine Program errors or warnings. Typically, “functStatus” is reported via the function return value, other mechanisms will be used if required by the language. A non-zero integer value indicates that the API function encountered one or more errors or warnings. The **`x4868getErrorMsg`** function can be called after each function to get a text description of the error message. The **`x4868activateLog`** and **`x4868closeLog`** functions are also available for reviewing the behavior of the API functions.

The “functStatus” approach should be used even in those languages that support exception handling, as exception handling should be reserved for situations that are difficult or impossible to handle.

The Engine Program is responsible for catching all exceptions, anything it can’t handle cleanly will be reported through “functStatus”. Likewise, if the Engine Program encounters a Severity of 99 the Engine Program will be terminated and any API “functStatus” will report this condition.

5.4 Memory Management

Memory management for all data returned to the Calling Program by the Engine Program is the responsibility of the Calling Program.

The Calling and Engine Programs are responsible for their own internal memory management. However, the Calling Program is responsible for calling the **`x4868terminate`** function prior to program termination in order to allow the Engine Program an opportunity to shut down in an orderly fashion, including releasing any memory it had allocated for internal use.

6. PARAMETER ATTRIBUTES

The primary attributes of each parameter are its name and value, and as such are specified for each available parameter. Three additional attributes may also be available for each parameter. They are a description, a units specification and a data type. These additional attributes are character strings, and providing these additional attributes is optional. An empty string is the default value and can be automatically returned for any parameter and attribute for which the given attribute is not defined by the Engine Program.

The API functions **`x4868getDescription`**, **`x4868getUnits`**, **`x4868getDataType`** return these additional attributes for a given parameter. Additional attributes would have to be agreed to by the Customer and Supplier and would require an additional API function to return them.

6.1 Name

Parameter names as described in the Engine Program documentation.

6.2 Description

The description attribute is a character string, the content of which is at the discretion of the Engine Program supplier. The default is an empty string.

6.3 Units

The standard string for each type of unit is given in AS681. A unit string of "NONE" or "none" should be returned for integer and string parameters, which are naturally unit-less, as well as dimensionless real parameters. An empty string indicates that unit type is not defined or supplied.

6.4 Data Type

The data type attribute indicates whether the parameter holds an integer, single precision or double precision floating point value or character string, and whether it holds a single value or an array of values. The API implementation must clearly indicate any more abstract data types used by the Engine Program language and what corresponding data type they will be cast to. In general, the data type referred to in this document is that of the variable in the Calling Program and always so when used in a function name.

The Engine Program shall cast numeric variables, into the requested data type. However, this can cause problems:

- Setting Engine Program variables with types of lower range or precision than the "set" Calling Program function type will yield either truncated values or NANs (overflows).
- Getting Engine Program variables with types of higher range or precision than the "get" Calling Program function type will yield either truncated values or NANs (overflows).

If done intentionally, truncated values are expected and acceptable, but NAN (overflow) values can lead to unintended consequences. Thus the Calling Programmer uses mismatched types at their own risk.

The character strings representing the different data types available through this API are:

I	Integer
F	Single
D	Double
S	Character String
I1D	1D Integer
F1D	1D Single
D1D	1D Double
S1D	1D Character String
I2D	2D Integer
F2D	2D Single
D2D	2D Double
S2D	2D Character String
I3D	3D Integer
F3D	3D Single
D3D	3D Double
S3D	3D Character String

Repeat this pattern for higher dimensional arrays.

The default is an empty string, indicating that data type is undefined.

7. ARRAY DATA

Functions for setting and getting one to three dimensional data are specified in section 9.

Array data passed through these functions are assumed to be "square", in that all 1-D arrays in a 2-D array are assumed to have the same number of points, all the 2-D arrays in a 3-D array are assumed to have the same number of lines, and so forth in higher dimensional arrays.

Fortran arrays are referenced in column major order while C arrays are referenced in row major order. The meaning of the array indices follows those appropriate for the language of the API function. This means that the order of the indices are reversed between Fortran and C. The array storage order must be identified if this API is extended to languages other than Fortran and C.

8. CHARACTER STRING REQUIREMENTS

8.1 Character String Format

All strings passed through this API must be structured as appropriate for the language used to implement the API functions.

8.2 Character String Sizes

For situations where a character string is returned to the Calling Program through the call list, the Calling Program is responsible for declaring a character parameter of sufficient size to contain the returning string and passing a reference to the Engine Program which will then populate the string and return it to the Calling Program. If the allocated space is insufficient, then the Engine Program should write as much of the character string as will fit in the allocated space and then raise an error. It is recommended that the Engine Program put the required length in the error message string to allow the Calling Program to be modified to support the required length.

9. GENERIC API FUNCTION SPECIFICATIONS

The following sections define the required behavior of each function, its corresponding inputs and outputs, and a list of error conditions that might be generated. Each appendix contains the function prototypes for a given language. The actual structure of the functions may differ from what is outlined here due to aspects of language structure and language differences. The appendix will also specify any language specific exceptions for each function that are in addition to the general exceptions listed here.

All API functions exist in the global name space. To avoid name collisions with global functions from other sources, the function names are constructed to insure that they are unique. The leading "x" in each name is a placeholder, and is replaced by one or more letters to indicate language, such as "f" for Fortran, in the actual implementation of this API in a specific programming language. See Section 5.1 for further discussion of function name construction.

Many, but not all, functions defined in this document must be fully implemented to insure proper interaction between the Calling and Engine Programs. Some functions are optional and only have to be fully implemented if the functionality, information or data type covered by a particular function is included in the Engine Program. However, all functions defined in this API must be present in some form to insure proper linkage of the Calling and Engine Programs. If a given functionality is not available for a particular Engine Program, then calling the associated function should generate an error that informs the Calling Program that a requested action was not performed.

9.1 **x4868activateLog**: Activates API Function Logging

This function activates a log file to track all API function calls. The log file should indicate the name of each API function called, it's arguments and any associated API error or warning conditions with descriptive messages. Note that Engine Program error conditions are not reported.

fileName	Input	Name of log file. Path name is relative to program startup location. stdOut & stdErr names are permitted. An empty string as treated as stdOut.
functStatus	Output	Function status

Common error conditions:

- The indicated file name could not be created.
- Function was called before the **x4868initProg** function

A minimal implementation of this function is permitted.

9.2 **x4868closeLog**: Closes API Function Logging

This function deactivates API function logging and closes the associated file if it is a named file.

functStatus	Output	Function status

Common error conditions:

- Function was called before the **x4868initProg** function

A minimal implementation of this function is permitted.

9.3 **x4868defineDataList**: Define a List of Parameters to Set or Get Later

This function defines a list of Engine Program parameters so they can be efficiently used by the **x4868getDataList** or **x4868setDataList** functions. One motivation for this function is real time models where speed is of the essence.

paramList	Input	List of parameter names, must all be scalar and of the same type
size	Input	Number of names in the list
dataType	Input	Calling program data type for all Engine Program paramList variables
listID	Output	Integer identifier associated with this defined list of names
functStatus	Output	Function status

Common error conditions:

- List of parameter names contains names that do not exist, are not scalar values or are not of a type that can be cast to dataType
- Function was called before the **x4868initProg** function.

A minimal implementation of this function is permitted.

9.4 **x4868getArraySize[123]D**: Get Size of Dimensions in Array Parameters

This family of functions obtains the number of elements in each dimension for a requested Engine array parameter. The x4868getArraySize1D function can also be used to obtain the length of a non-array character parameter, since a character string can be viewed as a 1-D array of individual characters.

paramName	Input	Parameter name (array)
num[XYZ]	Output	Number of elements in the array. One argument for each dimension
functStatus	Output	Function status

Common error conditions:

- The variable type of the referenced parameter is not an array.
- The number of dimensions of the referenced array parameter does not match that of the function called.
- This function was called before the **x4868initProg** function.

Implementation of this function is required for Engine Programs which can accept and return array data.

9.5 **x4868getDataList[IFD]**: Get an Array of Values Associated with a List of Parameters

These functions get an array associated with a list of Engine Program parameters, the motivation for this function is real time models where speed is of the essence. A previous call to **x4868defineDataList** is required.

listID	Input	Integer identifier associated with a previously defined list of names
size	Input	Number of elements allocated in the array, same size as in the associated defined list
array	Output	Array values being returned
functStatus	Output	Function status

Common error conditions:

- Invalid listID value, indicates **x4868defineDataList** was not called or not called successfully.
- Function was called before the **x4868initProg** function.

A minimal implementation of these functions should check if listID is valid, if not it should generate an error.

9.6 **x4868getDataType**: Get the Data Type Attribute for the Named Parameter

This function returns a string with the data type attribute for the requested parameter. Section 6.4 defines the strings to be returned for scalar and array parameters. The Engine Program may have data types that are more abstract than this API supports, for example NPSS Option variables, in those situations the API implementation should choose the most appropriate matching type.

paramName	Input	Parameter name
buffSize	Input	Buffer size of returned string
dataType	Output	Character string containing the data type string.
functStatus	Output	Function status

Common error conditions:

- The requested parameter name was not found in the Engine Program.
- This function was called before the **`x4868initProg`** function.

An acceptable implementation is one that returns an empty string for all parameters.

9.7 **`x4868getDescription`**: Get the Description Attribute for the Named Parameter

This function returns a string with the description attribute for the requested parameter.

paramName	Input	Parameter name
buffSize	Input	Buffer size of returned string
descr	Output	Character string containing the description string
functStatus	Output	Function status

Common error conditions:

- The requested parameter name was not found in the Engine Program.
- This function was called before the **`x4868initProg`** function.

An acceptable implementation is one that returns an empty string for all parameters.

9.8 **`x4868getErrorMsg`**: Get Function Error Description String

This function returns a descriptive string of any error that was generated in the immediately prior API function call. It will be an empty string if the preceding function returns zero. The content of the error message string is cleared at the start of each API function. Thus, error messages are available only for the most recent called API function.

buffSize	Input	Buffer size of returned string, recommend setting large, e.g. 1000.
message	Output	Character string containing the description string
functStatus	Output	Function status

Common error conditions:

- This function was called before the **`x4868initProg`** function.

A full implementation of this function is required.

9.9 **`x4868get[IFDS]`**: Get a Scalar Parameter Value

This family of functions returns the value of scalar parameters. A separate function is defined for each data type to be returned.

paramName	Input	Parameter name
buffSize	Input	Buffer size of returned string
value	Output	Value of parameter
functStatus	Output	Function status

Common error conditions:

- The requested parameter name was not found in the Engine Program.
- The variable type of the parameter referenced is not compatible of the variable type handled by the API function.
- Function was called before the **`x4868initProg`** function.

A minimal implementation of these functions should check if the parameter name is valid, if not it should generate an error.

9.10 **`x4868get[IFD]1D`**: Get1D Array Parameter Values

These functions return array data from the Engine Program.

paramName	Input	Parameter name
numGet	Input	Number of paramName elements to get, array and paramName should be dimensioned at least this large
array	Output	Array values being returned
functStatus	Output	Function status

Common error conditions:

- The requested parameter name was not found in the Engine Program
- The variable type and/or dimensional size of the parameter referenced is not compatible with the variable type and/or dimensional size handled by the API function
- Function was called before the **`x4868initProg`** function.

A minimal implementation of these functions should check if the parameter name is valid, If not it should generate an error.

9.11 **`x4868get[IFDS][123]Dentry`**: Get Array Parameter Entry Value

These functions return a single array value (entry) from the Engine Program. The call list of each function will contain the array name being retrieved as well as the index of each dimension in the array

paramName	Input	Parameter name
index[123]	Input	Index value for each dimension in the array
buffSize	Input	Buffer size of returned string
value	Output	Value of parameter
functStatus	Output	Function status

Common error conditions:

- The requested parameter name was not found in the Engine Program
- The variable type and/or dimensional size of the parameter referenced is not compatible with the variable type and/or dimensional size handled by the API function
- Insufficient memory was allocated in the array passed in by the Calling Program for the Engine Program to return the requested array parameter.
- Function was called before the **`x4868initProg`** function.

A minimal implementation of these functions should check if the parameter name is valid, If not it should generate an error.

9.12 **`x4868getSeverityMax`**: Get the Max Severity from Engine Program

Gets the maximum severity Engine Program error condition encountered.

severityMax	Output	Integer value containing the maximum severity encountered in the Engine Program
functStatus	Output	Function status

Common error conditions:

- Function was called before the **`x4868initProg`** function.

9.13 **`x4868getUnits`**: Get the Units Attribute for the Named Parameter

This function returns a string with the units attribute for the requested parameter. See section 6.3 for more information about the units attribute of a parameter.

paramName	Input	Parameter name
buffSize	Input	Buffer size of returned string
units	Output	Character string containing the units.
functStatus	Output	Function status

Common error conditions:

- The requested parameter name was not found in the Engine Program.
- Function was called before the **`x4868initProg`** function.

An acceptable implementation is one that returns an empty string for all parameters.

9.14 **x4868initProg**: Initialize the Engine Program

This function initializes the Engine Program. The Calling Program must call this function before any other API function. The **x4868initProg** function may be called only once per Engine Program execution session, which can be ended by a call to the **x4868terminate** function. Allowing **x4868initProg** to be called after a call to **x4868terminate** to begin another Engine Program session is an optional capability. All other API functions will generate an error if they are called before this function.

This function, while required to be the first API function to be called, may not complete the Engine Program initialization process. A given Engine Program may require additional input, either structured input passed through one of the parse functions, or specific data values to be set before the Engine Program is ready to be executed. Program documentation should specify all steps required to fully initialize the Engine Program.

The usage of this function will likely vary greatly between Engine Program environments, therefore this unique documentation will need to be supplied.

numArgs	Input	Number of strings in the character array
args	Input	Character array containing initialization arguments
functStatus	Output	Function status

Common error conditions:

- There are a number of different reasons why initialization might fail, and those reasons may vary between Engine Programs and their associated initialization methods.

A full implementation of this function is required, however, multiple initializations is optional.

9.15 **x4868isValidParamName**: Confirms If Parameter is Present in Engine Program

This function determines if the requested parameter name is valid. This function returns a Boolean through the function return value to indicate if the requested parameter is present in the Engine Program. Determining if the parameter is present or not is the normal operation of this function, therefore, an error should be generated if the requested parameter is not present.

paramName	Input	Parameter name
valid	Output	Boolean TRUE/FALSE (1/0), indicating if the name is valid or not
functStatus	Output	Function status

Common error conditions:

- Function was called before the **x4868initProg** function.

A full implementation of this function is required.

9.16 **x4868parseEfile**: Parse an Encrypted File

The file name given must be sufficient to allow the Engine Program to locate the file. Further, the file must be accessible by the Engine Program, and have contents that can be decrypted and parsed by the Engine Program. The entire specified file is decrypted, read and parsed and any actions specified by the input in the file are performed before this function returns.

filename	Input	Name of file
functStatus	Output	Function status

Common error conditions:

- The specified file could not be located
- Function was called before the **x4868initProg** function.
- The Engine Program does not support the parsing of encrypted files.

A minimal implementation of this function that generates an error when called is acceptable.

9.17 **x4868parseFile**: Parse a Plain Text File

The file name given must be sufficient to allow the Engine Program to locate the file. The file must be accessible by the Engine Program, and have content that can be parsed by the Engine Program. The entire specified file is read and parsed and any actions specified by the input in the file are performed before this function returns.

filename	Input	Name of file
functStatus	Output	Function status

Common error conditions:

- The specified file could not be located
- Function was called before the **x4868initProg** function.
- The Engine Program does not support the parsing of files.

A minimal implementation of this function that generates an error when called is acceptable.

9.18 **x4868parseString**: Parse a String

This function passes a plain character string to the Engine Program. The Engine Program is responsible for parsing its contents. All of the string is parsed and any actions specified by the contents of the string are performed before this function returns. The string can be of any length and may contain statement termination characters. As such any number of Engine Program statements may be contained with the specified string.

str	Input	Name of string to parse
functStatus	Output	Function status

Common error conditions:

- A problem occurred during the parsing of the string or in the subsequent execution of the parsed contents.
- Function was called before the **x4868initProg** function.
- The Engine Program does not support the parsing of strings.

A minimal implementation of this function that generates an error when called is acceptable.

9.19 **x4868run** – Execute the Engine Program

This function executes the Engine Program using the values of the input parameters at the point this function is called.

Engine Program errors and warnings do not result in an API error. See Section 5.2 for discussion on how Engine Program errors and warnings are communicated through the API.

functStatus	Output	Function status

Common error conditions:

- Function was called before the **x4868initProg** function.

A full implementation of this function is required.

9.20 **x4868setDataList[IFD]**: Set a List of Engine Program Parameters Values

These functions set a list of Engine Program parameters based on a predefined list of names and an associated array of values. The motivation for this function is real time models where speed is of essence. A previous call to **x4868defineDataList** is required.

listID	Input	Integer identifier associated with a previously defined list of names
size	Input	Number of elements allocated in the array, same as in associated defined list
array	Input	Array of values, sets Engine Program variables as indicated in predefined list
functStatus	Output	Function status

Common error conditions:

- Invalid listID value, indicates **x4868defineDataList** was not called or not called successfully.
- Function was called before the **x4868initProg** function.

A minimal implementation of these functions should check if listID is valid, if not it should generate an error.

9.21 **x4868set[IFDS]**: Set a Scalar Parameter Value

This family of functions set the specified parameter to the specified scalar value. A separate function is defined for each data type to be returned.

paramName	Input	Parameter name
value	Input	Value of parameter
functStatus	Output	Function status

Common exceptions:

- The requested parameter name was not found in the Engine Program.
- The variable type of the parameter referenced is not compatible with the variable type handled by the API function.
- Function was called before the **`x4868initProg`** function

A minimal implementation of these functions should check if the parameter name is valid, and generate an error if is not.

9.22 **`x4868set[IFD]1D`**: Set 1D Array Parameter Values

These functions pass array data to the Engine Program. The call list of each function will contain the array being passed as well as the number of elements to set.

paramName	Input	Parameter name
numSet	Input	Number of paramName elements to set, both array and paramName need to be dimensioned at least this large
array	Input	Array values being set
functStatus	Output	Function status

Common error conditions:

- The requested parameter name was not found
- The variable type and/or dimensional size of the parameter referenced is not compatible with the variable type and/or dimensional size handled by the API function
- Function was called before the **`x4868initProg`** function.

A minimal implementation of these functions should still check if the parameter name is valid, and generate an error if it is not.

9.23 **`x4868set[IFDS][123]Dentry`**: Set an Array Parameter Entry Value

These functions pass a single data array value (entry) to the Engine Program. The call list of each function will contain the array name being set as well as the index of each dimension in the array.

paramName	Input	Parameter name
index[123]	Input	Index value for each dimension in the array
value	Input	Value of array entry being passed
functStatus	Output	Function status

Common error conditions:

- The requested parameter name was not found
- The variable type and/or dimensional size of the parameter referenced is not compatible with the variable type and/or dimensional size handled by the API function
- Function was called before the **`x4868initProg`** function.

A minimal implementation of these functions should still check if the parameter name is valid, and generate an error if it is not.

9.24 **`x4868terminate`**: Instructs Engine Program to Terminate

This function instructs the Engine Program to perform an orderly shut down of the current execution session. This includes, but is not limited to, releasing any Engine Program allocated memory, writing buffered output to files, closing open files and unloading any dynamic libraries that it opened. This function must be called by the Calling Program before the Calling Program terminates. Further, it must be the last API function called.

functStatus	Output	Function status

Common error conditions:

- Function is called before the **x4868initProg** function. This function might generate an error for other reasons as well.

A full implementation of this function is required.

10. NOTES

A change bar (|) located in the left margin is for the convenience of the user in locating areas where technical revisions, not editorial changes, have been made to the previous issue of this document. An (R) symbol to the left of the document title indicates a complete revision of the document, including technical revisions. Change bars and (R) are not used in original publications, nor in documents that contain editorial changes only.

APPENDIX A - C IMPLEMENTATION

This appendix provides function prototypes for each API function, as they would appear for a C implementation of this API.

It is the responsibility of the Calling Program to allocate memory for all arguments being passed through the API functions and to free that memory prior to program termination.

A.1 **c4868activateLog**: ACTIVATES API FUNCTION LOGGING

Function prototype:

```
int c4868activateLog (const char* fileName);
```

A.2 **c4868closeLog**: CLOSES API FUNCTION LOGGING

Function prototype:

```
int c4868closeLog ();
```

A.3 **c4868defineDataList**: DEFINE A LIST OF PARAMETERS TO SET OR GET LATER

Function prototype:

```
int c4868defineDataList (const char** paramList, int size, const char* dataType,
                int* listID);
```

A.4 **c4868getArraySize[123]D**: GET SIZE OF DIMENSIONS IN ARRAY PARAMETERS

Function prototypes:

```
int c4868getArraySize1D (const char* paramName, int* numX);

int c4868getArraySize2D (const char* paramName, int* numX, int* numY);

int c4868getArraySize3D (const char* paramName, int* numX, int* numY, int* numZ);
```

A.5 **c4868getDataList[IFD]**: GET AN ARRAY OF VALUES ASSOCIATED WITH A LIST OF PARAMETERS

Function prototypes:

```
int c4868getDataListI(int listID, int *array,  int size)

int c4868getDataListF(int listID, float *array, int size)

int c4868getDataListD(int listID, double *array, int size)
```

A.6 **c4868getDataType**: GET THE DATA TYPE ATTRIBUTE FOR THE NAMED PARAMETER

Function prototype:

```
int c4868getDataType (const char* paramName, char* dataType, int buffSize);
```

A.7 **c4868getDescription**: GET THE DESCRIPTION ATTRIBUTE FOR THE NAMED PARAMETER

Function prototype:

```
int c4868getDescription (const char* paramName, char* descr, int buffSize);
```

A.8 **c4868getErrorMsg**: GET FUNCTION ERROR DESCRIPTION STRING

Function prototype:

```
int c4868getErrorMsg (char* message, int buffSize);
```

A.9 **c4868get[IFDS]**: GET A SCALAR PARAMETER VALUE

Function prototypes:

```
int c4868getI(const char* paramName, int* value)

int c4868getF(const char* paramName, float* value)

int c4868getD(const char* paramName, double* value)

int c4868getS(const char* paramName, char* value, int buffSize)
```

A.10 **c4868get[IFD]1D**: GET 1D ARRAY PARAMETER VALUES

Function prototypes:

```
int c4868getI1D(const char* paramName, int *array,  int numGet)

int c4868getF1D(const char* paramName, float *array, int numGet )

int c4868getD1D(const char* paramName, double *array, int numGet )
```

A.11 **c4868get[IFDS][123]DENTRY**: GET ARRAY PARAMETER ENTRY VALUE

Function prototypes:

```
int c4868getI1Dentry(const char *paramName, int index1, int* value)

int c4868getI2Dentry(const char *paramName, int index1, int index2, int* value)

int c4868getI3Dentry(const char *paramName, int index1, int index2, int index3,
                     int* value)

int c4868getF1Dentry(const char *paramName, int index1, float* value)

int c4868getF2Dentry(const char *paramName, int index1, int index2, float *value)

int c4868getF3Dentry(const char *paramName, int index1, int index2, int index3,
                    float *value)

int c4868getD1Dentry(const char *paramName, int index1, double* value)

int c4868getD2Dentry(const char *paramName, int index1, int index2, double *value)

int c4868getD3Dentry(const char *paramName, int index1, int index2, int index3,
                     double *value)

int c4868getS1Dentry(const char *paramName, int index1, char *value, int buffSize)

int c4868getS2Dentry(const char *paramName, int index1, int index2, char *value,
                     int buffSize)
```

A.12 **c4868getSeverityMax**: GET THE MAX SEVERITY

Function prototype:

```
int c4868getSeverityMax (int severityMax);
```

A.13 **c4868getUnits**: GET THE UNITS ATTRIBUTE FOR THE NAMED PARAMETER

Function prototype:

```
int c4868getUnits (const char* paramName, char* units, int buffSize);
```

A.14 **c4868initProg**: INITIALIZE THE ENGINE PROGRAM

Function prototype:

```
int c4868initProg(int numArgs, char *args[]);
```

A.15 **c4868isValidParamName**: CONFIRMS IF PARAMETER IS PRESENT IN ENGINE PROGRAM

Function prototype:

```
int c4868isValidParamName(const char* paramName, int *valid);
```

A.16 **c4868parseEfile**: PARSE AN ENCRYPTED FILE

Function prototype:

```
int c4868parseEfile(const char* filename)
```

A.17 **c4868parseFile**: PARSE A PLAIN TEXT FILE

Function prototype:

```
int c4868parseFile(const char* filename)
```

A.18 **c4868parseString**: PARSE A STRING

Function prototype:

```
int c4868parseString(const char* str)
```

A.19 **c4868run** – EXECUTE THE ENGINE PROGRAM

Function prototype:

```
int c4868run()
```

A.20 **c4868setDataList[IFD]**: SET A LIST OF ENGINE PROGRAM PARAMETERS VALUES

Function prototypes:

```
int c4868setDataListI(int listID, int *array, int size)

int c4868setDataListF(int listID, float *array, int size)

int c4868setDataListD(int listID, double *array, int size)
```

A.21 `c4868set[IFDS]`: SET A SCALAR PARAMETER VALUE

Function prototypes:

```
int c4868setI(const char* paramName, int value)

int c4868setF(const char* paramName, float value)

int c4868setD(const char* paramName, double value)

int c4868setS(const char* paramName, char* value)
```

A.22 `c4868set[IFD]1D`: SET 1D ARRAY PARAMETER VALUES

Function prototypes:

```
int c4868setI1D(const char* paramName, int *array, int numSet)

int c4868setF1D(const char* paramName, float *array, int numSet)

int c4868setD1D(const char* paramName, double *array, int numSet)
```

A.23 `c4868set[IFDS][123]DENTRY`: SET ARRAY PARAMETER ENTRY VALUE

Function prototypes:

```
int c4868setI1Dentry(const char *paramName, int index1, int value)

int c4868setI2Dentry(const char *paramName, int index1, int index2, int value)

int c4868setI3Dentry(const char *paramName, int index1, int index2, int index3,
                     int value)

int c4868setF1Dentry(const char *paramName, int index1, float value)

int c4868setF2Dentry(const char *paramName, int index1, int index2, float value)

int c4868setF3Dentry(const char *paramName, int index1, int index2, int index3,
                     float value)

int c4868setD1Dentry(const char *paramName, int index1, double value)

int c4868setD2Dentry(const char *paramName, int index1, int index2, double value)

int c4868setD3Dentry(const char *paramName, int index1, int index2, int index3,
                     double value)

int c4868setS1Dentry(const char *paramName, int index1, const char *value)

int c4868setS2Dentry(const char *paramName, int index1, int index2, const char
                     *value)
```

A.24 `c4868terminate`: INSTRUCTS ENGINE PROGRAM TO TERMINATE

Function prototype:

```
int c4868terminate()
```

APPENDIX B - FORTRAN IMPLEMENTATION

This appendix provides function prototypes for each API function.

The following specification assumes a Fortran 90 (F90) or later compiler, but only to the extent of allowing extended length parameter and function names. All other aspects of the functions reflect features and capabilities present in Fortran 77 (F77). F90 constructs, such as “Type” and “Module”, that allow an object-oriented programming style, are not used. The Calling Program is free to employ the dynamic memory capabilities of F90 to dynamically resize numerical and character arrays. The function to obtain the array size for a specified array parameter is retained in the Fortran implementation to support Fortran Calling Programs which opt to use this information to dynamically allocate array storage. However, a F77 Calling Program should function correctly with static array definitions of sufficient size.

Fortran functions can return character strings through the function return. However, the length of the string must be specified directly in the function definition. It is the responsibility of the Calling Program to declare a character string of sufficient size to contain the string to be returned. If the allocated space is insufficient, then the Engine Program should write as much of the character string as will fit, and then generate an error.

Dynamic memory allocation is not assumed for a Fortran implementation. If sufficient space is not allocated by the Calling Program, code changes and recompilation of the Calling Program may be required

B.1 **f4868activatelog**: ACTIVATES API FUNCTION LOGGING

Function prototype:

```
integer*4 function f4868activatelog (filename)
      character*(*) filename
```

B.2 **f4868closelog**: CLOSES API FUNCTION LOGGING

Function prototype:

```
integer*4 function f4868closelog ()
```

B.3 **f4868definedatalist**: DEFINE A LIST OF PARAMETER NAMES TO SET OR GET LATER

Function prototype:

```
integer*4 function f4868definedatalist (paramlist, size, datatype, listid)
      parameter character*(*) paramlist(size), datatype
      integer*4 size, listid
```

B.4 **f4868getarraysize[123]d**: GET SIZE OF DIMENSIONS IN ARRAY PARAMETERS

Function prototypes:

```
integer*4 function f4868getarraysize1d (paramname, numx)
      character*(*) paramname
      integer*4 numx
```

```
integer*4 function f4868getarraysize2d (paramname, numx, numy)
     character*(*) paramname
     integer*4 numx, numy
integer*4 function f4868getarraysize3d (paramname, numx, numy, numz)
     character*(*) paramname
     integer*4 numx, numy, numz
```

B.5 **f4868getdatalist[ifd]**: GET AN ARRAY OF VALUES ASSOCIATED WITH A LIST OF PARAMETERS

Function prototypes:

```
integer*4 function f4868getdatalisti (listid, array, size)
     integer*4 listid, size
     integer*4 array(size)
integer*4 function f4868getdatalistf (listid, array, size)
     integer*4 listid, size
     real*4 array(size)
integer*4 function f4868getdatalistd (listid, array, size)
     integer*4 listid, size
     real*8 array(size)
```

B.6 **f4868getdatatype**: GET THE DATA TYPE ATTRIBUTE FOR THE NAMED PARAMETER

Function prototype:

```
integer*4 function f4868getdatatype (paramname, datatype)
     character*(*) paramname, datatype
```

B.7 **f4868getdescription**: GET THE DESCRIPTION ATTRIBUTE FOR THE NAMED PARAMETER

Function prototype:

```
integer*4 function f4868getdescription(paramname, descr)
     character*(*) paramname, descr
```

B.8 **f4868geterrormsg**: GET FUNCTION ERROR MESSAGE DESCRIPTION STRING

Function prototype

```
integer*4 function f4868geterrormsg (errormsg)
     character*(*) errormsg
```