CAPE-OPEN

Expanding Process Modelling Capability
through Software Interoperability Standards

Errata and Clarifications for

Parameter Common Specification 1.0

www.colan.org

ARCHIVAL INFORMATION

Filename	Parameter_Errata_1.0_1.022.docx
Authors	Methods & Tools SIG
Status	Authorized for public release
Date	February 2016
Version	Version 1.0.1.022
Number of pages	20
Versioning	18 February 2016 approved for public release by Mgt Bd
Additional material	
Web location	
Implementation specifications version	Version 1.0
Comments	

IMPORTANT NOTICES

Disclaimer of Warranty

CO-LaN documents and publications include software in the form of *sample code.* Any such software described or provided by CO-LaN --- in whatever form --- is provided "as-is" without warranty of any kind. CO-LaN and its partners and suppliers disclaim any warranties including without limitation an implied warrant or fitness for a particular purpose. The entire risk arising out of the use or performance of any sample code --- or any other software described by the CAPE-OPEN Laboratories Network --- remains with you.

CO-LaN is a non for profit organization established under French law of 1901.

Trademark Usage

Many of the designations used by manufacturers and seller to distinguish their products are claimed as trademarks. Where those designations appear in CO-LaN publications, and the authors are aware of a trademark claim, the designations have been printed in caps or initial caps.

Microsoft, and the Component Object Model (COM) are registered trademarks of Microsoft Corporation.

SUMMARY

This document describes clarifications on implementation of the CAPE-OPEN Parameter Common Interface Specification version 1.0. In particular, this document addresses the effect of various initialization and editing operations on the state of the Parameter Collection so that PMEs will know potential changes to the Parameter Collection that may result from these operations. This clarification is intended to improve the performance of a process simulation application by reducing the number of times the PME iterates through the Parameter Collection to update the state of the Parameter Collection in the PME's graphical user interface. This document clarifies what are the static elements of the Parameters and identifies the conditions under which the static aspects can be modified. The document clarifies the dimensionality object and defines new array elements that indicate use of absolute or relative values. The document clarifies minimum support for Parameters and array Parameters.

ERRATA

1. Description of *ICapeRealParameterSpec.UpperBound*

In *ICapeRealParameterSpec.UpperBound*, the description says "Gets/Sets the upper bound of the real valued parameter for which this is the specification". The upper bound is read-only so the description should say "Gets the upper bound of the real valued parameter for which this is the specification".

2. Description of *ICapeIntegerParameterSpec.DefaultValue*

In *ICapeIntegerParameterSpec.DefaultValue*, the description says "Gets/Sets the default value of the integer valued parameter for which this is the specification". The upper bound is read-only so the description should say "Gets the default value of the integer valued parameter for which this is the specification".

CLARIFICATIONS

3. Roles and Responsibilities of the Parameter Owner, Parameter Clients, and the Process Modeling Environment

***Overall Context*:** The Parameter Common Interface Specification document identifies two actors in Parameter Uses Cases: the Parameter Owner and the Parameter Clients. The Parameter Owner is the software component that implements the Parameter; typically, this is the PMC that either implements the *ICapeUtilities* interface, or provides the Parameter Collection service through methods in their defining interface. In addition, energy or information streams contain Parameters that are owned by the PME. A Parameter Client is defined as a human user or piece of software that will access the functionalities provided by the Parameter Package. Three distinct Parameter Clients are identified in the Parameter Common Interface Specification document: the Solver Manager, the Flowsheet Builder, and the Flowsheet User.

The Use Cases that can be performed by Parameter Clients include UC-002: Get List of Parameters, UC-003: Display List of Parameters, UC-004: Change Parameter, UC-005: Validate Parameter, and UC-006: Validate Parameters Owner List of Parameters. Use Cases UC-004, UC-005, and UC-006 change the state of the Parameter, and have been implicated in performance issues due to the fact that the Process Modeling Environment (PME) often maintains and/or displays a list of the current Parameter values to the Flowsheet User which must be updated when the list of Parameters is modified.

For the most part, modifications of Parameters originated by the Flowsheet User and the Flowsheet Builder occur as a result of an activity initiated through the PME, such as setting the value of an individual Parameter, calculation of the Flowsheet, or editing of a particular PMC. When these actions occur, the PME can respond by updating its information regarding the state of the Parameter or Parameter Collection, as appropriate. For Parameters that are owned by the PME, the PME is also aware of changes to the Parameter.

Other actors that can serve as Parameter Clients and modify the Parameter or Parameter Collection in general are identified CAPE-OPEN PMC Primary Objects, including Optimization PMCs and Flowsheet Monitoring PMCs. Actions of these actors need to be initiated by, or coordinated through the PME so that the PME is aware that actions have occurred, triggering an update of the PME's information regarding the state of the Parameters and Parameter Collection of the PMCs in the current Flowsheet.

The role of the PME in maintaining the state of the Parameters and Parameter Collections within a particular Flowsheet need to be formalized and incorporated into the Parameter Common Interface Specification. Parameter Clients that modify a Parameter or Parameter Collection need to act only under the direction of, or in coordination with the PME. The Parameter Client will be a defined CAPE-OPEN PMC Primary Object, and activities of that PMC Primary Object that modify the Parameter or Parameter Collection of other PMCs need to be clearly stated in the PMC Primary Object's defining interface specification document.

Clarification 1: It should be noted that most Parameter Client activities are initiated by the Flowsheet User and performed through a method invoked on the Parameter Owner by the PME. These activities

that modify either the Parameter Collection or individual Parameters should be stated in the PMC Primary Object defining interface specification document. The PME can assume that the value of the Parameters remains unchanged except as a result of invoking these methods.

4. Performance Issues

***Overall Context*:** Some PMEs are continually scanning the PMC's Parameter Collection, causing performance degradation. Scanning of the Parameter Collection includes the following actions: obtaining the Parameter Collection from the PMC by calling *ICapeUtilities.Parameters*, obtaining an individual Parameter by calling *ICapeCollection.Item*, accessing any of the Parameter properties by calling methods on the *ICapeParameter*, *ICapeParameterSpec*, or the individual Parameter specification interfaces (*ICapeRealParameterSpec*, *ICapeIntegerParameterSpec*, *ICapeArrayParameterSpec*, *ICapeOptionParameterSpec*, or *ICapeBooleanParameterSpec*).

The likely reason that PMEs excessively scan the Parameter Collection is that there are no established means to inform the PME that changes have occurred within the Parameter Collection. The purpose of this clarification is to identify those times at which the changes can occur with the Parameter Collection and the PME can limit its scan of the Parameter Collection to these times. In order to limit the number of times that the Parameter Collection is scanned by the PME, the methods that can modify either an individual parameter or the entire list of parameters, necessitating action by the PME to update its information about the Parameter or Parameter Collection have been identified. The following list of methods can modify the entire Parameter Collection, and after a call to these methods, the PME shall rescan the entire Parameter Collection:

- *ICapeUtilities.Initialize*
- *ICapeUtilities.Edit*
- *ICapeUnit.Calculate*

The following methods can only modify the Parameter that implements the method. Therefore, only the modified Parameter will require scanning.

- *ICapeParameter.SetValue*
- *ICapeParameter.Reset*
- *ICapeParameter.Validate* (Validation Status only)

Recommendation 1: PMEs should limit their scans of a PMC's Parameter Collection and the individual Parameters in these collections to only those times where either the Parameter Collection or individual Parameters in the collection may have been modified by a method call.

Clarification 1: PMCs must ensure that their Parameter Collection is only modified during the above calls from the PME. *ICapeParameter.Validate* may only change the Validation Status of the Parameter. It must not change value or other attributes of the Parameter. No other operations are allowed to modify any aspect of a PMC's Parameter Collection.

5. Parameter Specification Object

Issue: There is no guidance in the document for the purpose of the typed Parameter Specification interfaces, e.g. *ICapeRealParameterSpec*.

Discussion: Parameter specification object provides constraints on the Parameter's value, such as lower and upper bounds, default values, and the options list, indicating conditions under which the Parameter value is valid. In general, Parameters are objects that parameterize the PMC, this could include choice of calculation method/equation of state, initial temperatures/pressures, maximum number of iterations before a calculation is considered to not converge, whether the calculation converged, and a wide variety of other uses.

The default value of a Parameter is an attempt by the Parameter Owner's developer to provide a typical starting value. Examples of uses for the default value include: setting 100 as the default maximum number of iterations for a calculation, 1 atmosphere of pressure, 273.15 degrees Kelvin temperature, or using the "van der Waals" equation of state. In any case, these are not the only acceptable value for the Parameter, but just one that can typically be used and will likely result in convergence of the Flowsheet. Use of an *UNDEFINED* default value for *CAPE_INPUT* or *CAPE_INPUT_OUTPUT* Parameters implies that there is no obvious choice of default value, and that the Flowsheet User will need to specify these values prior to the Parameter validation method resulting in the Parameter taking a state of *CAPE_VALID*. An *UNDEFINED CAPE_OUTPUT* Parameter indicates that there is no predicted value for the Parameter.

Upper and Lower bounds, as well as the options list for Option Parameters provide limits on the values that the Parameter can take. In the case of numerical bounds, these may represent physical limitations of the system being simulated, limits to the size of a finite element grid, etc. For *CAPE_INPUT* and *CAPE_INPUT_OUTPUT* Parameters, these values may be used to check that input values are in an appropriate range, such as within a range where the calculation is likely to converge, or limits on the validity of models. For *CAPE_INPUT_OUTPUT* and *CAPE_OUTPUT* Parameters, the values indicate that the calculated values are within an acceptable range of values. Choosing an *UNDEFINED* numeric value for either the upper or lower bound implies that there is no limit on the value of the Parameter in that bound.

The Options List represents the list of choices the value of an Option Parameter make assume. For instance, it could be the choice of adiabatic, isothermal, or isochoric calculation.

The Parameter Specification provides the minimum constraints that can be placed on a Parameter value, and is the base line for determining whether the Parameter's value is valid.

Clarification 1: The following values are allowed for properties of the Parameter Specification:

- Real
  - Default Value, upper and lower bounds – any real value, including *UNDEFINED* (Not-a-Number, NaN).
  - Upper bound must be greater than or equal to the lower bound, if both upper and lower bound are defined.
  - The Default Value can be *UNDEFINED* (NaN). If both Default Value and upper bound are not *UNDEFINED*, Default Value must not be larger than upper bound. If both Default Value and lower bound are not *UNDEFINED*, Default Value must not be smaller than lower bound.
  - A dimensionality, in the form of a double array.
- Integer
  - Default Value, upper and lower bounds – any integer value, including *UNDEFINED* (minimum integer value).
  - Upper bound must be greater than or equal to the lower bound, if both upper and lower bound are defined.

    - The Default Value can be *UNDEFINED*. If both Default Value and upper bound are not *UNDEFINED*, Default Value must not be larger than upper bound. If both Default Value and lower bound are not *UNDEFINED*, Default Value must not be smaller than lower bound.
- Boolean
    - Default Value – *TRUE* or *FALSE*.
    - If there is no clear preference for either *TRUE* or *FALSE*, an Option Parameter should be considered with an Options List including "*TRUE*", "*FALSE*", and "Make a selection", or other meaningful terms.
- Option
    - Default Value – any string value, including *UNDEFINED*.
    - RestrictedToList is *TRUE*
        - *OptionsList* – a string array containing at least one value.
    - RestrictedToList is *FALSE*
        - *OptionsList* – a string array. An empty string array is valid.
- Array Parameters shall have a valid specification for each element of the array based on the element type.

Clarification 2: An *UNDEFINED* default value can either mean that the default value is the *UNDEFINED* value, or that no default value is appropriate. The *ICapeParameter<Type>Spec.Validate* method can be used to distinguish between these two cases by checking if UNDEFINED is a valid value. If *UNDEFINED* is an invalid value, a default value is not available for the Parameter, and the value of the Parameter must be set by the Flowsheet User.

Clarification 3: *UNDEFINED* upper or lower bound values imply that the upper or lower bound are not available.

Clarification 4: The Parameter Specification can only be modified by calling *ICapeUtilities.Edit* on the Parameter Owner.

6. Use of Invalid Parameter Values

Issue: Some Parameter implementations do not allow the Parameter value to be invalid.

Discussion 1: Parameter validation provides that the Parameter can have a value that is either valid or invalid for the Parameter. A *CAPE_INVALID* validation status indicates that the value of the Parameter does not comply with its Parameter specification. The validation message argument returned by the *ICapeParameter.Validate* method must indicate the reason for which the Parameter value is not valid.

Discussion 2: The Parameter Specification (*ICape<Type>ParameterSpec* interface) provides a Validate method to test the compliance of a proposed Parameter value with its Parameter Specification. The message argument of the *ICape<Type>ParameterSpec.Validate* method must provide information regarding the non-conformance of the proposed value.

Clarification 1: Parameters are allowed to have a validation status of *CAPE_INVALID*, which means that the Parameter's value does not comply with its Parameter Specification.

Clarification 2: Attempting to change the value of a Parameter using the *ICapeParameter.value* to a value of type that does not match the *CapeParameterType* of the Parameter shall raise an *ECapeBadArgument*, and the validation status of the Parameter shall be set to *CAPE_INVALID*.

Clarification 3: When the value of a Parameter is changed using the *ICapeParameter.value* property, then its *ICapeParameter.ValStatus* property is changed to *CAPE_NOT_VALIDATED*. The *ICapeParameter.Validate* method can then be called to validate the Parameter and set the *ICapeParameter.ValStatus* property to either *CAPE_VALID* or *CAPE_INVALID*.

Clarification 4: If the new value of a Parameter being changed using the *ICapeParameter.value* property does not comply with its Parameter specification, the Parameter may perform one of the following actions:

1. Accept the invalid value. The Parameter will have a validation status of *CAPE_INVALID* after a subsequent call to *ICapeParameter.Validate*.
2. Reject the invalid value and raise an *ECapeInvalidArgument* error. In COM, this will be done through returning an *HRESULT* value of *ECapeInvalidArgumentHR* (0x80040506) (and the *ECapeInvalidArgument* interface must be exposed). The value of the Parameter remains unchanged. The *ECapeUser.description* string will provide the reason that the value was not accepted. The Validation Status of the Parameter remains unchanged.

Clarification 5: PME implementations are encouraged to inform the Flowsheet User of the cause of the validation failure provided by the Parameter *Validate()* method message argument.

7. Parameter Validation

Issue 1: The following behaviour by a PME was observed: upon instantiation of a CAPE-OPEN Unit Operation that carries a Parameter Collection, the PME navigated the contents of the Parameter Collection (type, mode, etc.) including the *ValStatus* of each Parameter, and each Parameter returned a *ValStatus* of *CAPE_NOT_VALIDATED*. The Parameter Client was therefore aware that the *Validate()* method had not been called on each Parameter. The *ICapeUtilities.Edit* method was never called within the sequence observed; however, the PME kept requesting the *ValStatus* of each Parameter.

Discussion 1: The behaviour described above indicates that the Parameter client was repeatedly checking the *ValStatus* of the Parameter having a *ValStatus* of *CAPE_NOT_VALIDATED* without an intervening change in the *ValStatus* of the Parameter. The *ValStatus* must be re-evaluated and set to the appropriate status (*CAPE_VALID* or *CAPE_INVALID*) during the call to *ICapeParameter.Validate*. It is not useful for the Parameter Client to obtain its validation state if the Parameter Client does not make any subsequent attempt to validate the Parameter in case the current validation state is CAPE_NOT_VALIDATED.

Clarification 1: The outcome of a successful call to the *ICapeParameter.Validate* method is to set the Parameter *ValStatus* property to either *CAPE_VALID* or *CAPE_INVALID*. The Parameter cannot have a validation status of *CAPE_NOT_VALIDATED* following a successful call to *ICapeParameter.Validate*.

If the call to *ICapeParameter.Validate* is unsuccessful, the Parameter validation status must be set to *CAPE_NOT_VALIDATED*, and an appropriate error HRESULT returned for the *ICapeParameter.Validate* call.

Performance advisory: The Parameter client shall not make successive calls to *ICapeParameter.ValStatus* to obtain the validation status of the Parameter having a ValStatus of *CAPE_NOT_VALIDATED* without making an intervening call to *ICapeParameter.Validate.*

Issue 2: The events which may trigger a change in the validation status of a Parameter are not stated in the Parameter specification.

Discussion 2: The validation status of a Parameter can be changed by various actions, including PMC initialization, PMC configuration, PMC calculation, etc. This clarification will indicate the times at which a Parameter validation status may be set to *CAPE_NOT_VALIDATED*, requiring the PME to validate the Parameter to set its validation status to *CAPE_VALID* or *CAPE_INVALID*.

Clarification 2: A Parameter may have a validation status of *CAPE_NOT VALIDATED* at the following times:

1. After a successful call to *ICapeUtilities.Initialize*
2. After changing the *ICapeParameter.value* property
3. After a call to *ICapeParameter.Reset*
4. After a call to *ICapeUtilities.Edit*
5. After an unsuccessful call to *ICapeParameter.Validate*
6. Additional events stated in PMC business specifications, including for example:
    a. After a call to *ICapeUnit.Calculate*
    b. After a call to *ICapeThermoPropertyPackage.CalcEquilibrium*.
    c. After a call to *ICapeThermoPropertyPackage.CalcProp*.
    d. After a call to *ICapeThermoCalculationRoutine.CalcProp*.
    e. After a call to *ICapeThermoEquilibriumServer.CalcEquilibrium*.
    f. After a call to *ICapeThermoPropertyRoutine.CalcSinglePhaseProp*.
    g. After a call to *ICapeThermoPropertyRoutine.CalcTwoPhaseProp*.
    h. After a call to *ICapeThermoPropertyRoutine.CalcAndGetLnPhi*.
    i. After a call to *ICapeThermoequilibriumRoutine.CalcEquilibrium*.

Issue 3: The purpose of Parameter validation is not stated in the document.

Discussion 3: Validation (and the validation status) of the Parameter is a means by which a Parameter Owner can identify and communicate to the Parameter Client and Flowsheet User whether the Parameter's current value complies with its Parameter Specification and other applicable criteria. For input Parameters, this implies the Parameter value is not acceptable for subsequent calculations, while for output Parameters, likely means the result of the calculation is not within an acceptable design envelope. Examples of uses for Parameter validation and *ValStatus* include indicating that initial values are within a range where convergence is considered likely, or that calculated results represent value within an acceptable range, *e.g.* pressures and temperatures outside a safety threshold. Scanning the validation status of the Parameter Collection items provides a PME with a quick check of the state of the PMCs in the Flowsheet and enables the PME to quickly alert Flowsheet Users to potential problems.

Clarification 4: Parameter validation status is informational and shall not have any impact on PMC performance, ability to be edited using *ICapeUtilities.Edit*, or ability to perform operations except as specified by the PMC's controlling interface specification document or documented in the PMC's user's documentation. SIGs providing specifications for PMC Primary Objects, and developers of PMC objects must clearly state any effect Parameter validation status can have on a PMC's behavior.

Issue 4: The criteria for validation against the Parameter Specification is not stated in the Parameter Common Interface Specification document.

Discussion 4: Validation of a Parameter against its Parameter Specification is the minimum default validation that can be performed on a Parameter. The Parameter Specification provides bounds or other limitations on the value of the Parameter, such as restriction of the value to an element of the *OptionsList* for an Option Parameter. Validation against the Parameter Specification simply tests whether the value of the Parameter meets these criteria. The purpose or context of the Parameter may impose additional restrictions on the validity of values Parameters can take, in which case, the Parameter Owner's documentation must provide information regarding these additional constraints.

Clarification 4: When validated against its Parameter Specification, a Parameter is considered valid if it meets the following criteria:

- Real-valued Parameter
  - A value between upper and lower bounds.
  - A value of Infinity is invalid.
  - Not-a-Number (NaN) represents an UNDEFINED value, and is valid if either the upper or lower bound are UNDEFINED.
- Integer-valued Parameter
  - A value between upper and lower bounds
  - An UNDEFINED value may or may not be valid.
- Boolean-valued Parameter
  - A value of *TRUE* or *FALSE*.
- Option-valued Parameter
  - If the value of the *RestrictedToList* property is *TRUE*, the Parameter value must be one of the elements of the *OptionsList* string array.
  - If the value of the *RestrictedToList* property is *FALSE*, any string value is valid.
- Array Parameter
  - Must have the structure described below.
  - Each array element must have a valid value for the specification of the array element.
  - Each array element has a corresponding specification.

Clarification 5: Parameter Owners can establish additional or alternative validation protocols or criteria. The PMC developer must document these additional validation protocols so that Flowsheet Users know the reason for, and implications of *CAPE_VALID* and *CAPE_INVALID* Parameter validation states on the results of the process simulation. Examples of alternative Parameter validation include:

- String Parameters that contain file paths must contain a valid file path.
- The validity of Parameters that are inter-dependent may depend on the values of other Parameters.
- An optional input, such as a temperature override, may be valid if UNDEFINED,
- A PMC that sets Boolean Parameter to *FALSE* to indicate that the last calculation attempt failed to converge, can make a *FALSE* value invalid.

Clarification 6: Validation messages for invalid Parameters shall state the criterion that invalidated the Parameter.

8. Implementation of the Parameter Specification Interfaces

Issue: In one example, a Parameter Client is attempting to obtain an *ICapeIntegerParameterSpec* interface from Parameter Specification having a *CapeParameterType* of CAPE_BOOLEAN.

Discussion: The specification document requires that the Parameter Specification must implement the *ICapeParameterSpec* interface, but the document does not indicate which of the typed Parameter specification interfaces that a Parameter Specification should implement. In particular, it does not indicate whether a Parameter Specification should implement all of the *ICape<Type>ParameterSpec* interfaces, or just the interface corresponding to the *CapeParamType* of the Parameter. Section 3.3 and Figure 8 of the document provide the UML class diagrams for the Parameter objects. The *ICape<Type>ParameterSpec* interfaces are shown as specializations of the *ICapeParameterSpec* interface, which implies that the specialization interfaces are only required on the particular specialization type.

Clarification: The Parameter Specification must expose both the *ICapeParameterSpec* interface and the *ICape<Type>ParameterSpec* interface implied by the *CapeParamType* of the Parameter returned by the *ICapeParameterSpec.Type*. The following list indicates which Parameter specification interface must implemented for the *CapeParamType* of a Parameter:

- CAPE_REAL implements *ICapeRealParameterSpec*,
- CAPE_INT implements *ICapeIntegerParameterSpec*,
- CAPE_OPTION implements *ICapeOptionParameterSpec*,
- CAPE_BOOLEAN implements *ICapeBooleanParameterSpec*,
- CAPE_ARRAY implements *ICapeArrayParameterSpec*

It should be noted that some Parameter implementations use the same Parameter Specification for all Parameter types, therefore the Parameter Specification may implement multiple *ICape<Type>ParameterSpec* interfaces. No Parameter Client shall attempt to obtain or use an *ICape<Type>ParameterSpec* interface from a Parameter that does not match the current *CapeParamType* value returned by the *ICapeParameterSpec.Type* method of the Parameter.

9. Inconsistencies in the Parameter Specifications

Issue: The document is inconsistent in whether the specification accessed via the *ICapeRealParameterSpec*, *ICapeIntegerParameterSpec*, *ICapeArrayParameterSpec* *ICapeOptionParameterSpec*, or *ICapeBooleanParameterSpec* are read only (just gets), or can be modified (get and set methods). For instance, the *ICapeRealParameterSpec.LowerBound* property description indicates that its value may only be obtained (the description indicated that the method can "Gets the lower bound of the real valued Parameter for which this is the specification"), while the *ICapeRealParameterSpec.UpperBound* property can be changed (the description indicated that the method can "Gets/Sets the upper bound of the real valued Parameter for which this is the specification").

Modifications of the Parameter Specification have been found to add complexity to the Parameter Common Interface as it may cause PMEs to excessively scan the Parameter Collection, having significant implications for the performance of the application. For this reason, modification of a Parameter Specification is limited to actions of a custom editor of the Parameter Owner accessed through the *ICapeUtilities.Edit* method.

Clarification: The properties accessed through the Parameter Secification interfaces are read-only. The Parameter Collection can only be modified through the custom editor of the Parameter Owner accessed through the *ICapeUtilities.Edit* method.

10. Parameter Mode

Issue: The mode of the Parameter determines when the Parameter value can be modified. Parameters have three possible modes of operation: *CAPE_INPUT*, *CAPE_INPUT_OUTPUT*, and *CAPE_OUTPUT*. *CAPE_INPUT* Parameters are used to configure the state of the PMC such that the Flowsheet User is able to modify its value, but the value will not be changed by calculating the PMC. While different values of these Parameters are possible, changing their value during calculation would change the nature of the PMC. For a PMC, *CAPE_OUTPUT* Parameters have the opposite role – providing the state of the PMC after calculation, and are evaluated during the calculation process. *CAPE_INPUT_OUTPUT* Parameters have values that can be set prior to a calculation and can be changed by the calculation process. An initial estimated value for a Parameter calculated iteratively could be one possible use of a *CAPE_INPUT_OUTPUT* Parameter.

The values of energy and information streams are also represented by Parameters. The Parameter owner in this case is the PME. The Parameters are exposed to the Unit Operation PMCs by means of connection of an energy or information object to a Unit Operation port. For Parameters on INLET ports, the Parameter mode should be *CAPE_OUTPUT* (the Parameter is read-only from the point of view of the Unit Operation); the Parameter mode on OUTLET ports should be *CAPE_INPUT* (the Parameter value must be specified by the Unit Operation as part of its *Calculate* method).

Runtime modifications of the Parameter Mode outside of configuration using the *ICapeUtilities.Edit* method has been found to add complexity to implementation of the Parameter Common Interface Specification as it may require PMEs to excessively scan the Parameter Collection, having significant implications for the performance of the application.

Reccommended Change to the Parameter Interface: The mode of the Parameter will be deemed read-only. Parameter clients should not modify the mode of a Parameter, and calls to the *ICapeParameter.SetMode* method shall result in an error condition being raised. The Parameter mode can only be changed through the custom editor of the owning PMC accessed through the *ICapeUtilities.Edit* method by the PME.

11. Dimensionality

Parameter dimensionality allows communication of the units of measurement for the Parameter's value between the Parameter and its clients. The CAPE-OPEN standard includes the *Dimensionality* on the *ICapeParameterSpec* interface. The *ICapeParameterSpec* interface is exposed by all Parameters. According to the document:

> The dimensionality represents the physical dimensional axes of this Parameter. It is expected that the dimensionality must cover at least 6 fundamental axes (length, mass, time, angle, temperature and charge).

In implementation, a number of issues have arisen regarding the use of the dimensionality of the Parameter. These issues include format of the dimensionality of a non-dimensioned Parameter, and absolute vs. relative Parameter values.

11.1 Expression of Parameter Dimensions as an Array

The value of all CAPE-OPEN Parameters is always returned using International System of Units (SI) units. The SI dimensionality is expressed as a *CapeArrayDouble*, an array of real-valued elements

wrapped as a VARIANT in COM-based implementations. The use of real-valued elements allows for situations where the dimension is expressed fractionally or as an exponential root (*e.g.* square root). The array will follow the C++ convention where the index of the first element is zero (0). The array will include up to 9 elements, which are defined as listed in Table 1.

It should be noted that the dimensionality of a Parameter does not reflect on the underlying physical quantities. For example, surface area of packing per unit volume has a dimensionality of reciprocal distance. Further, the dimensionalities of mass fraction, volume fractions, and mole fraction cannot be distinguished. In addition, the dimensionality object cannot distinguish between different physical quantities that share the same dimensionalities, such as mass transfer coefficient and velocities sharing length per time units. Log-valued Parameters (e.g. log of vapour pressure having units that are the logarithm of Pascal unit) may also be encountered. This clarification does not attempt to address these shortcomings of the dimensionality array.

The absolute value flag is used to indicate whether the dimensionality of the Parameter value is absolute or relative to a defined state. Examples include temperature and pressure where the Parameter value can be measured using an offset scale (e.g. degrees Celsius or gauge pressure), or an absolute scale (e.g. degrees Kelvin or absolute pressure). Using an absolute value flag of zero (0) indicates that the Parameter value is using an absolute (e.g. degrees Kelvin), and the flag set to one (1) indicates the use of a relative value (e.g. degrees Celsius). It should be noted that thermodynamic properties available from objects covered under the thermodynamics specifications, such as pressure and temperature are not Parameters using this specification; therefore, the distinction between absolute and relative values described here does not apply to values passed using the thermodynamic interfaces. Interested parties are directed to the Thermodynamics and Physical Properties Interface Specification documents and documentation for the particular property model implementations for discussion of the units of measure used.

Table 1. Dimensionality Array Specification

Index	Dimension	SI Base Unit	
		Name	Symbol
0	Length	meter	m
1	Mass	kilogram	kg
2	Time	second	s
3	Electrical current	Ampere	A
4	Temperature	Kelvin	K
5	Amount of Substance	Mole	mol
6	Luminous intensity	candela	cd
7	Angle	radian	rad
8	Absolute or Relative Flag	-	-

Dimensionality arrays may be encountered that have fewer than 9 elements. Trailing zeros in the dimensionality array are optional. Should trailing zeros be omitted, the index of the element determines the base dimensions included in the dimensionality array as indicated in the table above. For instance, an absolute temperature Parameter will always have all array elements except index four (4) as zero, and index four will contain the value 1 <0,0,0,0,1>. Likewise, absolute pressure (force per area) will always have an array with the first three values of <-1, 1, -2>, and the remaining values, if present, will be zero (0). No error condition should be raised by the client in this circumstance.

The relative/absolute value flag can also be used to distinguish between non-dimensioned values such as dimensionless numbers (e.g. Reynolds numbers) and fractional values (e.g. mole fractions,

efficiencies, yields). In the case where the value of the absolute/relative flag is zero (0), *i.e.* <0,0,0,0,0,0,0,0,0,0>, <0,0> or an empty array, the *ICapeParameter.value* will be interpreted as dimensionless. In the case where the value of the absolute/relative flag is the only non-zero value in the array, such as <0,0,0,0,0,0,0,0,0,1>, the *ICapeParameter.value* will be interpreted as a fractional value, for example, 90 per cent or 900,000 parts per million is represented as 0.9.

The specification for the dimensionality array is *CapeArrayDouble*. It should be noted that legacy implementations of this specification may return a *CapeArrayInteger* for the dimensionality. No error condition should be raised by the client in this circumstance.

11.2 Dimensionality for a Non-Dimensionable Parameter

Dimensionality applies to Parameters that represent measurable quantities, the values of which are typically expressed using continuous, real-valued numbers. As such, integer-, Boolean-, and option Parameters are not dimensioned. The same applies to array Parameters with elements that are of type integer, Boolean, or option. However, according to the CAPE-OPEN specifications, the *Dimensionality* property is exposed on the *ICapeParameterSpec* interface by all Parameters. As the dimensionality is not supported for integer, Boolean, and option Parameters, PMEs should not call the *ICapeParameterSpec.Dimensionality* method for integer, Boolean, or Option Parameters. In the event that the PME inadvertently calls the *ICapeParameterSpec.Dimensionality* method for any of the Parameter types that are not dimensioned, the recommended response is to indicate an error condition by returning the *ECapeNoImplHR* (0x80040509) HRESULT value.

Legacy Parameter implementations may return an S_*OK* HRESULT and attempt to return the dimensionality as an empty VARIANT (VT_EMPTY), an array indicating a non-dimensional value as described above, or return an *ECapeUnknown* error, by returning an *ECapeUnknownHR* HRESULT. No error condition should be raised by the Parameter client in any of these circumstances.

11.3 Missing Dimensionalities

Parameters that do not provide a properly formatted dimensionality array or return an error HRESULT should be treated as dimensionless values. No error condition should be raised by the client in this circumstance.

12. Expected Array Parameter Support

Issue: The existing interface for array Parameters (*ICapeArrayParameterSpec*) was designed to be very general and flexible and allows amongst other things:

- definition of multi-dimensional arrays with an arbitrary number of dimensions
- definition of an array of CAPE_REALs where each element has different default value, lower bound and upper bound and dimensionality
- definition of an array where each element is of a completely different type; rather like a programming language structure
- two different mechanisms for defining a multi-dimensional array:
  - either, explicitly by having *ICapeArrayParameterSpec::NumDimensions* return a value greater than 1, or
  - defining a 1D array whose elements are of ARRAY type

There is a consensus amongst CAPE-OPEN developers that the array Parameter is unnecessarily 'general' and that this generality provides too much flexibility in the implementation of the array Parameters. In order to provide a consistent implementation of the array Parameter, what follows is an expected structure for the array Parameter in COM-based CAPE-OPEN implementations. The following structure allows both homogeneous arrays (arrays where all elements have the same, or equivalent, specification), and generic, non-homogeneous arrays.

Generic Array Parameter Structure: Array Parameters will contain an array whose elements may include real numbers, integers, strings, Boolean, and/or array values. Arrays having elements of mixed types, and nested arrays (arrays having elements that are themselves array) are allowed. In COM-based CAPE-OPEN implementations, the array will have the following structure:

1. *ICapeParameter.value* is a COM *VARIANT*, with variant type (*VARTYPE*) set to *VT_ARRAY|VT_VARIANT*.
2. The *VARIANT* will contain a *SAFEARRAY(VARIANT)* stored at the *VARIANT*'s *pArray* member.
3. The lower bound for each dimension shall be zero (0), and the number of elements of each dimension shall be the size of the dimension obtained from the element of the array obtained from the *ICapeArrayParameterSpec.Size* method corresponding to the dimension.
4. Each *VARIANT* element in the *SAFEARRAY* will have one of the following types (*VARTYPE*):
    - *VT_R8* (CapeDouble),
    - *VT_I4* (CapeInteger),
    - *VT_BSTR* (CapeString),
    - *VT_BOOL* (CapeBoolean), or
    - *VT_VARIANT* (CapeVariant containing a nested *SAFEARRAY*).
5. A nested array indicated by a *VT_VARIANT* array element will contain a *VARIANT* of type *VT_ARRAY|VT_VARIANT*. Contents of the array will be as described in 2, 3, and 4.
6. The *ICapeArrayParameterSpec.ItemsSpecification* will consist of a *VARIANT* having type *VT_ARRAY|VT_DISPATCH*.
7. The *VARIANT* will contain a *SAFEARRAY(LPDISPATCH)* located at the *VARIANT*'s *pArray* member.
8. Each element in the specification *SAFEARRAY* will be a *VARIANT* containing an *IDispatch* pointer to a specification object exposing *ICapeParameterSpec* and the appropriate *ICape<TYPE>ParameterSpec* interface.
9. There will be one element in the *ICapeArrayParameterSpec.ItemsSpecification* array for each element in the *ICapeParameter.value* array. The number of dimensions and array bounds of the specification array must match the dimensions and bounds of the value array, described in item 2 and 3, above.
10. A nested array element in the Parameter value will have a matching *ICapeArrayParameterSpecification* element in the specification array. Specifications for nested arrays will have structures as described in 5, 6, 7, 8, and 9. All specifications shall meet the requirements stated above in Section 3.

Parameter Client Interaction with an Array Parameter: The Parameter value will have the structure as described above, which shall be the return value of the *ICapeParameter.GetValue* method. The Parameter Client will also use this structure for calls to *ICapeParameter.SetValue()*. There are two mechanism that can be used to obtain the structure of the Array Parameter:

- The Parameter Client can use the return value of a call to the ICapeParameter.GetValue method to obtain the structure of the Array Parameter.

- The Parameter Client can infer the Parameter structure from metadata [e.g. number of dimensions, size of the dimensions, and the specifications of the array items] about the array obtained from the Array Parameter specification.

There is no mechanism to set the value of a single element of an Array Parameter. As such, the value of an Array Parameter must be set as a whole. A Parameter Client will construct the argument to an *ICapeParameter.SetValue* call using information about the Parameter structure determined using one of the methods above.

The CAPE-OPEN specification defines the details of the array for the purpose of the *ICapeParameter.Get/SetValue* method calls. The mechanism used by the Parameter Client (typically a PME) to present the Array Parameter to the Flowsheet User to view and/or modify the Parameter's value is not covered by the CAPE_OPEN specification, which is consistent with scalar Parameters. There are a number of ways that the Parameter Client can display or allow the Flowsheet User to view or modify the values of an Array Parameter. If the PME developer chooses to display or allow the Flowsheet User to modify the Array Parameter, the PME developer will need to consider:

- The array may be multidimensional
- Any element of the array may be an array.
- Each array element will have its own specification (whether homogeneous or not).

Likewise, the developer of the Parameter Owner (typically a PMC) will need to consider the complexity of displaying a complex Array Parameter value. An example of a complex array structure might be an Array Parameter for a tray mole fractions by phase of a distillation column. In this array, the elements are a two-element array, whose elements are (1) vapor phase and (2) liquid phase array of reals containing the composition of each phase for each tray in the column.

In contrast to the general structure, a more limited structure can be formulated as follows:

Homogeneous One Dimensional Array Parameter Structure: Homogeneous Array Parameters contain a one-dimensional array whose elements are of the same type and are described by the same, or equivalent Specification: upper and lower bounds, default values, dimensionality, etc. In COM-based CAPE-OPEN implementations, the array will have the following structure:

1. *ICapeParameter.value* is a COM *VARIANT*, with variant type (*VARTYPE*) set to *VT_ARRAY|VT_VARIANT*.
2. The *VARIANT* will contain a *SAFEARRAY(VARIANT)* with one dimension stored at the *VARIANT*'s *pArray* member.
3. The lower bound shall be zero (0), and the number of elements of the array shall be the first (and only) element of the array obtained from the *ICapeArrayParameterSpec.Size* method.
4. Each *VARIANT* element in the *SAFEARRAY* will have the same type, being one of the following types (*VARTYPE*):
    - *VT_R8* (CapeDouble),
    - *VT_I4* (CapeInteger),
    - *VT_BSTR* (CapeString),
    - *VT_BOOL* (CapeBoolean), or
5. The *ICapeArrayParameterSpec.ItemsSpecification* will consist of a *VARIANT* having type *VT_ARRAY|VT_DISPATCH*.
6. The *VARIANT* will contain a *SAFEARRAY(LPDISPATCH)* located at the *VARIANT*'s *pArray* member.
7. Each element in the specification *SAFEARRAY* will be a *VARIANT* containing an *IDispatch* pointer to a specification object exposing *ICapeParameterSpec* and the appropriate

ICape<TYPE>ParameterSpec interface. For a homogeneous array parameter, this pointer points to the same, or equivalent, specification object.

8. There will be one specification object in the *ICapeArrayParameterSpec.ItemsSpecification* array (lower bound of zero) corresponding to the element in the *ICapeParameter.value* array.

Although each of the elements of an Array Parameter carries a Parameter specification (*ICapeParameterSpecification*), and therefore a dimensionality, the Parameter itself also has a specification (*ICapeParameterSpecification*), and therefore can have a dimensionality. The specification of the Parameter itself should only expose a dimensionality (different from dimensionless) in case all elements of the array are of type CAPE_REAL and have the same (or equivalent) specification. In other words, if the dimensionality provided by the Parameters itself is present (and not dimensionless) this is a promise that the parameter is a Homogeneous One Dimensional Array Parameter.

Clarification 1: The Parameter Client will at a minimum provide the ability to utilize a homogeneous one-dimensional Array Parameter having the structure described above. It is expected that the elements of such an array may be accessed, used as controller inputs or outputs, etc. to the same extent as is supported for scalar Parameters.

Clarification 2: An Array Parameter must not expose a dimensionality (other than dimensionless) in case its structure deviates from the homogenous one-dimensional structure described above.

13. Expected Parameter Support

Issue: Some PMEs are supporting only a limited set of Parameter types, preventing configuration of some CAPE-OPEN Unit Operations. The requirement for support of all types was not clearly stated in the Parameter Common Interface Specification.

Clarification: It is expected that PMEs support Parameters to aid the configuration of Unit Operations and other PMCs. Support for Parameters means that the PME needs to provide any Parameter Client the ability to inspect and modify the Parameter's value. The PMEs are required to support not just CAPE_REAL typed Parameters but also Parameters of types CAPE_INT, CAPE_BOOL, CAPE_OPTION and type CAPE_ARRAY containing CAPE_REAL, CAPE_INT, CAPE_BOOL or CAPE_OPTION elements. Support for CAPE_ARRAY supporting CAPE_ARRAY is optional.

However, support for Array Parameters in general is complex and the minimum level of support for Array Parameters provided by PMEs is as described in issue 12: one dimensional homogeneous arrays.

References:

1. Parameter Common Interface Specification Document