CAPE-OPEN

Expanding Process Modelling Capability
through Software Interoperability Standards

Errata and Clarifications for

Persistence Common Interface specification

www.colan.org

ARCHIVAL INFORMATION

Filename	Persistence_Errata_1.0_0.008.docx
Authors	CO-LaN consortium: M&T SIG
Status	Public document
Date	July 2016
Version	Version 0.008
Number of pages	9
Versioning	0.000 created on May 7, 2015 by Jasper van Baten
	0.001 edited on November 4, 2015 by SIG
	0.002 edited on December 2, 2015 by SIG
	0.003 edited on January 6, 2016 by SIG
	0.006 edited on March 17, 2016 by SIG
	0.007 proofread by CTO and sent to Mgt Bd
	0.008 Readied for public release by CTO
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

The Errata and Clarifications document for the Persistence Common Interface specification 1.0 provides clarification on the use of the COM persistence interfaces in the context of CAPE-OPEN applications. The document provides background on COM persistence, clarifies the responsibilities of the PME and PMCs during persistence, enumerates what a PMC should save and restore, discusses the fall-back persistence procedure and defines the data types supported by *IPropertyBag* implementations.

For information concerning the life cycle management of PMCs, see the Utilities Common interface specification document and any current Errata and Clarifications associated with the document.

ERRATA

Page 11: for Use Case 003, the Actor should be the PME instead of the Flowsheet User and the first sentence of the flow of events should be “Upon request, the PME retrieves a previously stored Flowsheet.”

Page 11: the sentence “It requests each CAPE-OPEN PMC to connect to the simulator’s CAPE-OPEN objects, such as connecting their ports to the simulators streams in the case of unit.” should be revised in “It requests each CAPE-OPEN PMC to connect to the simulator’s CAPE-OPEN objects, such as reconnecting Ports of Unit Operations.”

Page 12: Use Cases [Add PMC to Flowsheet] and [Specify PMC connections to Simulator Objects] are not existing and should not be referred to in Use Cases 003 and 004.

Page 13: simplification of SQ-001 as follows: Flowsheet Unit should be PMC, Flowsheet Manager and Unit Manager should be pulled together as PME. This may lead to a simplified sequence diagram as well in the next version of the interface specification document.

Page 14: modification of SQ-002 as follows: Flowsheet Unit should be PMC, Flowsheet Manager should be PME, “Save Unit” should be “Save PMC”, Flowsheet User should not be in that Sequence Diagram.

Page 15: section 3 should be reformatted for the next version of the interface specification document.

Page 16: in section 3.4, the legend of Figure 4 should be “Persistence state diagram”.

Page 17: in section 3.5, within CO-001 Component Diagram, there is no ICapePersistence interface. This should say instead “Persistence interface”. The legend requires correction: “Persistable component diagram”.

Pages 22-24: within section 6, all VB6 examples should be removed from future interface specification document.

CLARIFICATIONS

1. Requirement to implement persistence

Persistence allows saving and restoring the configuration of PMCs by a PME in a simulation. A configurable PMC is one whose state can be configured by the Flowsheet User through the PME using *ICapeUtilities.Edit* method or by exposing Public Parameters. CAPE-OPEN standardizes persistence mechanisms to facilitate the saving and restoring a flowsheet that consists of configurable PMCs selected and arranged by the flowsheet user.

Configurable PMCs may also be configured by non-CAPE-OPEN mechanisms, and persistence mechanisms for these PMCs may be independent of the CAPE-OPEN specification.

***Clarification*:** PMCs are not required to support persistence. Persistence is not required for configurable PMCs, however configurable PMCs that do not support persistence cannot be expected to be restored to their configured state.

2. Manager Objects

Manager Objects are configurable PMCs. However, their configuration falls outside of the scope of the Flowsheet and therefore Manager Objects do not need to be persisted as part of the Flowsheet. Maintaining the private configuration of any Manager Object is the responsibility of the Manager Object itself.

Currently known Manager Objects are Property Package Managers, Thermo Systems and Reaction Package Managers.

***Clarification*:** Manager Objects must not be persisted or restored from persistence as part of the Flowsheet.

3. Background on persistence interfaces in COM

Persistence allows saving and restoring the configuration of PMCs by a PME in a simulation. PMCs are not required to support persistence unless the PMC has a state that can be configured by the Flowsheet User through the PME using *ICapeUtilities.Edit* method or by exposing public parameters.

The Persistence Common Interface specification states that "rather than defining CAPE-OPEN specific persistence interfaces and data structure, it was best to reuse the COM Persistent native mechanisms." COM provides a variety of interfaces, derived from *IPersist* that can be used to store the persistent state of objects. CAPE-OPEN requires implementation of either the *IPersistStream* or *IPersistStreamInit* interfaces for objects that support persistence. COM requires that objects support either *IPersistStream* or *IPersistStreamInit*. Based upon limited testing conducted as part of developing this document, Visual Basic 6 (VB6) calls the *IPersistStreamInit.InitNew* method when creating or loading an object from persistence. For this reason, PMCs that support *IPersistStreamInit* should not raise an error if both the *InitNew* and *Load* methods are called.

Both *IPersistStream* and *IPersistStreamInit* use an *IStream* object to persist the PMC. *IStream* provides structured storage, presented as a contiguous sequence of bytes that can be read and written. Streams can be marshalled to other processes, allowing applications to share data in storage without using global memory. As a result, *IPersistStream* or *IPersistStreamInit* fulfils most of CAPE-OPEN's common persistence needs, primarily, persisting an object to a file or other appropriate storage medium.

COM also provides additional interfaces that may be supported by PMC objects and PMEs: *IPersistStorage*, *IPersistMemory*, *IPersistPropertyBag*, *IPersistMoniker*. Support for any of these additional interfaces is optional, and may be considered for particular cases, as follows:

- Persist to an allocated memory location (*i.e.*, fixed-size byte array): *IPersistMemory*
- Persist to a property bag container, such as an XML text file: *IPersistPropertyBag*
- Persist to structured storage: *IPersistStorage*
- Persist to a moniker: *IPersistMoniker*

***Clarification*:** PMCs that do not support persistence cannot be expected to be restored to their configured state.

The PME may utilize any of the additional persistence mechanisms listed above if exposed by the PMC. If a PMC exposes any of these additional interfaces, the PMC must fully implement that interface.

4. What should be saved and restored?

***Context*:** Any PMC may implement persistence to store its modifiable information and any PME is expected to utilize persistence to save and restore PMCs.

Issue: The Persistence Common Interface specification document did not indicate what exactly is to be stored by the PMC as part of persistence. Also, PMEs are not using persistence systematically when saving and restoring PMCs. As a result, Flowsheets are not consistently restored by PMEs.

Clarification: Ideally, persistence allows the user to save the flowsheet at one time on a given computer and restore it at a later time on either the same computer or another computer having the same versions of the PME and PMCs installed. Via persistence, a PMC stores its internal state so that it can be restored to an equal state.

PMC Responsibilities: The internal state to be persisted includes all modifiable data of the PMC, for example:

- Name and description (if modifiable) of the PMC,
- State of all Parameters (such as their current values)
- Parameter specifications (such as default values, upper and lower bounds, and options list) that are unmodifiable through CAPE-OPEN interfaces but modifiable through internal configuration mechanism (typically for Unit Operations via *ICapeUtilities::Edit*)
- Anything that depends on the system configuration, such as the local setup of a Property Package Manager.

There is no need to persist data that can be reconstructed by the PMC to the same state it was in at the time the PMC was saved. However, the PMC needs to persist anything it cannot re-create on demand. For example reports that can be immediately re-created should not be persisted, however, reports that cannot be quickly and easily regenerated should be persisted. Context dependent objects are not saved, for example the active Material Object that is set using *ICapeThermoMaterialContext::SetActiveMaterial*.

PME Responsibilities:

The PME needs to save the object type of the PMC to instantiate the appropriate PMC object. In COM, this is the class id (CLSID) that the class is registered. For PMCs that are created through one of these manager, the PME needs to store the name of the managed PMC in order to recreate the PMC. Currently, there are three "manager" PMCs: Thermo System, Property Package Manager, and the Reaction Package Manager. After the PMC has been created, the PME can call load on the managed PMC.

Persistence of any PMC that is managed by a manager is managed by the manager through persistence of the manager.

The PME is responsible for resetting the Simulation Context of the PMCs.

Port connections are not saved by the PMC but are restored by the PME (after initialization of the PMC).

The PME should not merely save and restore a PMC's parameter values (via the *ICapeParameter* interface) as a substitute for persistence of the PMC itself.

5. Persistence fall-back

Context: Persistence mechanisms other than *IPersistStream* or *IPersistStreamInit* are optional. The level of support for the format in which the data is stored is not necessarily clear. For example, the data types of the values that can be stored via *IPropertyBag* may vary from implementation to implementation. Persistence via *IPersistPropertyBag* may unexpectedly fail.

Issue: If persistence fails via any persistence mechanism other than *IPersistStream* or *IPersistStreamInit*, it is not clear how the PME should act.

Clarification: PMCs that support persistence must implement support for *IPersistStream* or *IPersistStreamInit*, these can be used as backup method in case saving via any other persistence mechanism fails. If the PME determines that a persistence mechanism other than *IPersistStream* or *IPersistStreamInit* can be used to save a PMC, and persistence via the selected mechanism fails, the PME should attempt to save the PMC via its *IPersistStream* or *IPersistStreamInit* implementation. If the fall-back mechanism is used when storing the PMC, the same persistence mechanism should be used upon restoring the PMC from persistence.

6. Minimum level of support for data types of IPropertyBag

***Context*:** Implementation of *IPersistPropertyBag* is optional. Whereas *IPersistStream* and *IPersistStreamInit* provide the ability to persist to a binary stream (*IStream*), implementation of *IPersistPropertyBag* provides storage in the form of key/value pairs (*IPropertyBag*) that are more suitable for storage in human readable format, such as textual input files or XML.

***Issue*:** The documentation of *IPropertyBag* does not impose a minimum level of support for data types. It is up to an *IPropertyBag* implementation to decide which value types it supports. In case a value type is not supported, the *Write* member of *IPropertyBag* indicates the data type is not supported by returning E_FAIL. It is not clear for PMCs that implement *IPersistPropertyBag* what level of support for data types can be expected from PMEs that implement *IPropertyBag*. As a result PMCs may fail to persist.

***Discussion*:** It stands to reason that PMCs should be able to store all data types that are commonly used in CAPE-OPEN. The data types that must be supported in this context are therefore CapeInteger (a.k.a. CapeLong), CapeDouble, CapeBoolean, CapeString, CapeShort, CapeFloat, CapeChar, CapeDate, CapeURL, CapeArrayInteger, CapeArrayDouble, CapeArrayBoolean, CapeArrayString and CapeArrayVariant. In addition, it is foreseen that PMCs should be able to store parameter values directly to the PropertyBag. This requires CapeArrayVariant to be supported, where each element of the array may contain a CapeInteger, CapeDouble, CapeBoolean, or CapeString. CapeArrayVariant is also used for functionality like compound constants, phase attributes and universal constants. Support for CapeInterface can be expected only in case the object that implements the interface also implement *IPersistStream*, in accordance with the *IPropertyBag* documentation. Finally, it is convenient to store binary data as an array of bytes. The VARIANT types required for the CAPE-OPEN types follow.

In the light of the minimum requirements for Parameter support, PMEs are *not* required to support Parameter values that consist of nested Arrays. Consequently, support for persisting and loading nested CapeArrayVariant is not required *if* Parameters with this data type are not supported by the PME. Nevertheless it is recommended to support nested CapeArrayVariant so that PMCs that have Parameters of that data structure can be persisted via *IPersistPropertyBag*. For the same reason, support for multi-dimensional CapeArrayVariant values is recommended but not required.

Clarification: If the PME provides persistence through *IPersistPropertyBag*, the *IPropertyBag* object of the PME must support the following VARIANT types:

- VT_BOOL (CapeBoolean)
- VT_BSTR (CapeString, CapeURL)
- VT_DATE (CapeDate)
- VT_I1 (CapeChar)
- VT_I2 (CapeShort)
- VT_I4 (CapeLong, CapeInteger and enumerations)
- VT_R4 (CapeFloat)
- VT_R8 (CapeDouble)
- VT_UNKNOWN, where the object implements *IPersistStream* (CapeInterface)
- VT_ARRAY|VT_BOOL (CapeArrayBoolean)
- VT_ARRAY|VT_BSTR (CapeArrayString)
- VT_ARRAY|VT_I4 (CapeArrayInteger)
- VT_ARRAY|VT_UI1 (Binary data)
- VT_ARRAY|VT_R8 (CapeArrayDouble)
- VT_ARRAY|VT_VARIANT (CapeArrayVariant), where each element can contain
    - VT_I4 (CapeInteger or enumeration)
    - VT_R8 (CapeDouble)
    - VT_BOOL (CapeBoolean)
    - VT_BSTR (CapeString)
    - A nested VT_ARRAY|VT_VARIANT (recommended)