CAPE-OPEN

Expanding Process Modelling Capability
through Software Interoperability Standards

Errata and clarifications for Unit Operation specification 1.0

CO LaN

www.colan.org

ARCHIVAL INFORMATION

Filename	Errata_UO1.0_1.0.0015.doc
Authors	CO-LaN consortium: UNIT SIG
Status	Internal
Date	March 2015
Version	Version 1.0.0015
Number of pages	11
Versioning	Version 1.0.0000 created on Aug 31, 2012 by J. van Baten
	Version 1.0.0001 edited by Michel Pons on Sep 3, 2012
	Version 1.0.0002 edited by Michel Pons on Sep 3, 2012
	Version 1.0.0004: Null flows / Validate
	Version 1.0.008 Includes clarification on UC-020.
	Version 1.0.011 as revised by core UNIT SIG.
	Version 1.0.012 as revised by Michel Pons on July 21, 2014
	Version 1.0.013 as revised by Michel Pons on July 28, 2014
	Version 1.0.014 as revised by SIG on August 5, 2014
	Version 1.0.015 as revised by SIG on March 17, 2015 (sections on Reset and on modal dialogs)
Additional material	
Web location	
Implementation specifications version	Unit Operations 1.0 (version 6.25)
Comments	

IMPORTANT NOTICES

FOREWORD

The errata and clarifications have been discussed and developed by the CO-LaN Special Interest Group dedicated to CAPE-OPEN Unit Operations. Only the approved Unit Operations Interface Specification 1.0 document in its version 6.25 is to be followed when implementing CAPE-OPEN interfaces.

SUMMARY

This document provides three corrections to the specification document and clarifies the flow of events of three Use Cases (Check Unit, Save Flowsheet and Retrieve Flowsheet), gives rules for zero flows in Material Objects, expands the range of report formats, and makes recommendation about the Ports. The document proposes a new named value to deal with resetting Unit Operations. A recommendation is also issued about modal dialogs.

ERRATA AND CLARIFICATIONS

1. Errata

1.1 UC-015 Check Unit

The sentence
"The *Validate* method *must* be called before *Calculate* when any (one or more) of the following have occurred since the last **calculation**:"

should be replaced by

"The *Validate* method *must* be called before *Calculate* when any (one or more) of the following have occurred since the last **validation that leads to a CAPE_VALID status**:"

1.2 *Validate* method description

In Section 3.6.1 within the Notes of the description of *ICapeUnit::Validate*, the sentence

> "However, *Validate* must be called before *Calculate* in case any (one or more) changes to the following configuration aspects of the Unit Operation have changed since the last call to ***Calculate***:"

should be replaced by

> "However, *Validate* must be called before *Calculate* in case any (one or more) changes to the following configuration aspects of the Unit Operation have changed since the last call to ***Validate* that leads to a CAPE_VALID status**:"

1.3 *Validate* "message" argument description

Issue: the specification document states that the message argument of Validate is optional. However a message must be provided regarding the reason for a Unit Operation not being in a valid state so that the end-user may take the appropriate actions to overcome that situation. Additionally, the description speaks of validation failure which suggests that Validate raises an exception. What is meant is a successful validation resulting in a validation state of the Unit Operation being CAPE_INVALID and Validate returning FALSE.

Correction: description of the "message" argument of method Validate: "An optional message describing the cause of the validation failure" should be replaced by "If the Unit Operation is not in a valid state, a message conveying the reason for that situation."

2. Clarifications

2.1 Use case UC-001 “Add Unit to Flowsheet”

Clarification: if the creation of the Flowsheet Unit fails, for example if *ICapeUtilities:Initialize* fails on the Flowsheet Unit, no further calls on the Flowsheet Unit must be made.

2.2 Use case UC-018 “Save flowsheet”

The Simulator Executive is responsible for ensuring that the previously persisted state of each Flowsheet Unit is consistent with the current state of each Flowsheet Unit at the time of saving a Flowsheet.

When using the COM persistence interfaces, a Flowsheet Unit is responsible for ensuring that its state is properly reportable to the Simulator Executive. This state is reported through the *IsDirty* method on the *IPersistStream* or *IPersistStreamInit* interface implemented by the Flowsheet Unit.

As part of the “Save Flowsheet” Use Case (UC-018), the Simulation Executive is expected to perform the “Save Unit” Use Case (UC-009) on the Flowsheet Unit.

This flow of events is relaxed as follows:

“The Simulator Executive can choose, without checking the *IsDirty* method on each Flowsheet Unit, to save the Flowsheet Unit by executing the [Save Unit] Use Case.

Alternatively, the Simulator Executive has the option to check the persisted state of each Flowsheet Unit by calling the *IsDirty* method. Only if the *IsDirty* method of a Flowsheet Unit returns S_FALSE, the Simulator Executive will not execute the [Save Unit] Use Case on that Flowsheet Unit.”

Consequently, it is the responsibility of the Flowsheet Unit to return S_OK from *IsDirty* in case a configuration change (through UC-008) has been made (or may have been made) that affects the data that is persisted by the Flowsheet Unit since the last time the Flowsheet Unit was either initialized, loaded or saved with a ClearDirty flag.

2.3 Use Case UC-020 “Retrieve Flowsheet”

Clarification 1: it is not required to validate a Flowheet Unit immediately after retrieving the Flowsheet. Validation may be postponed as long as it is performed before *Calculate* (in accordance with UC-015).

So the Flow of events is relaxed as follows:

« The Simulator Executive retrieves the native Flowsheet Units in its usual way. For each CAPE-OPEN Flowsheet Unit in the Flowsheet, it recovers the Flowsheet Unit type. The Flowsheet Unit is created. The Flowsheet Unit’s data is restored as in [Restore Unit].
The Simulator Executive recovers details of the Flowsheet Unit’s stream connections (which are saved in the simulator's native format) and asks each CAPE-OPEN Flowsheet Unit to connect their Ports to those specific streams.
If the Flowsheet Unit fails to restore, the Flowsheet Builder is notified. »

Consequently the post-conditions of this Use Case now include: <Flowsheet Unit validation status is CAPE_NOT_VALIDATED>.

Clarification 2: failure to restore a Flowsheet Unit should be dealt with in a manner consistent with failure to create a Flowsheet Unit. Consequently if a Flowsheet Unit fails to restore, for example because *Load* fails or because *Initialize* fails, no further calls on the Flowsheet Unit must be made.

2.4 Material Objects with zero flow

Problem:

A stream can be defined by total flow rate and composition, or by compound flow rates. In the latter case, it is possible for all compound flow rates to be zero, which leaves composition undefined.

Typically, a stream is defined by the user (feed stream) or as the product of a unit operation. A unit operation must set, according to the specification, overall total flow rate and composition, or compound flow rates, and request a phase equilibrium calculation on the stream.

Composition may subsequently be requested by the PME, or by any PMC operating on the stream. Particularly, a scenario may appear in which a Unit Operation sets all zero compound flow rates and two flash conditions on a stream, and requests a phase equilibrium calculation. The phase equilibrium calculator (e.g. a Property Package) takes overall composition as input and would fail if the composition is undefined.

Hence, for simplicity of PME operation, composition must always be present in Material Objects that represent material streams. As some PMEs require a valid composition even for material streams with zero flowrate, the strong requirement of material feed and product streams to be in thermodynamic phase equilibrium remains. Considering the fact that most Unit Operations have a material feed stream, it is always possible for a Unit Operation to use the composition of a feed stream for a product stream. Considering that a PMC knows best about the possible interaction of inlet information with its model, it has been decided that it is the responsibility of the PMC to provide a composition on its outlet streams. This may not be an intuitive or physically sensible solution but it is a pragmatic approach. In order to prevent a situation in which composition is not known on a stream, setting compound flows is disallowed in case all compound flows are zero. The following paragraphs have been changed:

Section 2.1.6 Conceptual examples

Material Objects connected to outlet ports: the Unit must have specified the conditions of each of the material outlet ports at the end of a successful calculation. It must specify either overall composition and total flow or overall compound flows (provided that at least one compound flow is non-zero and positive) and it must instruct the thermodynamic services to calculate the thermodynamic phase equilibrium on the outlets by specification of two additional variables (e.g. temperature and pressure, or pressure and enthalpy) and call of *CalcEquilibrium* method on the Material Object attached to each outlet stream.

UC-025 Flow of events

As in [Set Stream Data through Outlet Ports] except: the streams requested are material streams represented by Material Objects, and the Ports are material Ports. The Flowsheet Unit must specify either both overall composition and total flow, or overall compound flows provided that at least one compound flow is non-zero and positive. In addition the Flowsheet Unit must instruct the thermodynamic services to calculate the thermodynamic phase equilibrium on the outlets by specification of two additional variables (e.g. temperature and pressure, or pressure and enthalpy). Note that either compound flows or composition as well as the two additional variables must be specified prior to the thermodynamic phase equilibrium calculation, whereas the total flow may be set either before or after the phase equilibrium calculation.

Outlet material streams with zero flow rate must be defined by setting the overall total flow to a value of zero and by setting a valid overall composition. Setting all zero values for overall compound flow is not permitted. As necessary, a dummy valid composition must be provided so that the state of the material stream may be set. Such a composition may be obtained from any inlet stream.

Note: equimolar or other arbitrary composition may not be supported by the underlying thermodynamic system and may lead to a flash failure. In the absence of a composition prediction in the limit of zero flow, using an inlet composition should avoid this problem.

2.5 UC-015 Check Unit

Currently the second paragraph of the Flow of events for that Use Case states:

> "During validation, the Flowsheet Unit may typically obtain information regarding the configuration (not the state) of the objects connected to the Ports. For example, it may ask for a list of compounds on each Port to see whether a) the Flowsheet Unit can deal with this list and b) the compounds on in- and outlet Ports are consistent. Therefore, a call to *Validate* may be relatively time consuming, and it is not recommended to call *Validate* before each call to *Calculate*, unless the configuration of the Unit has changed since the last call to *Calculate*."

The last part of the last sentence of this paragraph does not reflect the entire set of conditions calling for a *Validate* to be used. The entire set of conditions may be found in the Notes of the *Validate* method description, as well as further down in UC-015. It is considered that further insight on the use of *Validate* should be brought in, giving the following text in replacement.

> "During validation, the Flowsheet Unit may typically obtain information regarding the configuration (not the state) of the objects connected to the Ports. For example, it may ask for a list of compounds on each Port to see whether a) the Flowsheet Unit can deal with this list and b) the compounds on in- and outlet Ports are consistent. Therefore, a call to *Validate* may be relatively time consuming, and it is not recommended to call *Validate* before each call to *Calculate*. In particular, calling *Validate* as part of each iteration in solving a Flowsheet should be avoided if not required."

2.6 ICapeUnitReport

Issue 1: formatting reports during *Calculate* may impede on performance, particularly if the Unit Operation is included in a recycle loop. At intermediate recycle calculations, the report content will likely not be required.

Recommendation 1: if formatting of a report requires significant computational effort (note that formatting numbers is potentially expensive), it should be postponed until *ProduceReport*.

Issue 2: reports produced by *ICapeUnitReport* are text reports and there is no common consensus on how to render these reports. This makes difficult for Unit Operations to produce reports that contain data formatted for example in columns.

Recommendation 2a: pertains to plain text reports.

- Recommendation 2a.1 on PME: fixed-pitch fonts are to be used by PME to visually render reports supplied by PMCs.
- Recommendation 2a.2 on PME: combinations of carriage return/line feed should be treated as line feed.
- Recommendation 2a.3 on PME: all printable UNICODE characters are allowed.
- Recommendation 2a.4 on PME: there should be no limitation on the length of a report.
- Recommendation 2a.5 on PME: there should be no limitation on the line length of a report. Consequently PME should avoid breaking or wrapping lines.
- Recommendation 2a.6 on PMC: carriage return characters should not appear without a line feed character immediately after. Line feed characters may appear by themselves.
- Recommendation 2a.7 on PMC: the PMC should not put control characters (page break, form feed, back space) in their reports. TAB characters are not to be included either in the Report. Instead space characters should be used to align data.

In addition to plain-text reports, various formatting protocols could be envisioned for reports in order to enhance reporting. Rather than defining a formatting protocol proprietary to CAPE-OPEN, it is decided to adopt a widely used and open format protocol (HTML) for which standard rendering software components are available.

Recommendation 2b: HTML provides formatting of text in a very flexible way. Reports from Unit Operations may require such flexibility. Reports formatted as HTML can only be used if this functionality is supported by both the PME and the PMC. Support for HTML formatted reports is optional, and subject to the following requirements. No particular HTML version or character encoding is enforced: it is up to the PMC to ensure that its report may be properly parsed by mainstream HTML rendering components. The intent is to provide support for text formatting, not object embedding functionality. The HTML report should preferably be self-contained to ensure proper rendering. If HTML contains links to external objects (such as images, movies, etc...), it is the responsibility of the PMC to manage the life cycle of these objects.

- Requirement 2b.1 on PME: PME should advertise to PMCs that HTML format is accepted. This is done by exposing the Boolean NamedValue "HTMLReportSupport" with a value of true. This NamedValue is exposed by *ICapeCOSEUtilities* interface.
- Requirement 2b.2 on PMC: PMC must not provide reports in HTML format unless the PME accepts this format.
- Requirement 2b.3 on PME: PME should detect that a report is formatted as HTML. In this case the report starts with a DOCTYPE declaration as follows: <!DOCTYPE html>
- Requirement 2b.4 on PMC: HTML reports must begin with a DOCTYPE declaration as per requirement 2b.3.

2.7 Port Collection

It is not reasonable to assume the PME must re-evaluate the Port Collection each time the value of a Parameter changes or a Port gets disconnected. Therefore the Unit Operation must keep the Port Collection constant (order, name, type, direction of Ports need to remain the same) after changing a Parameter value or after connecting or disconnecting a Port. The only places suitable for changing the Port Collection is during *Edit* and *Initialize*. So the PME should re-evaluate the Port Collection after *Edit* is completed. The PME should exercise caution regarding change of names for Ports: equivalence of Ports before and after *Edit* can be asserted through the objects still connected to the Ports. During *Edit*, the PMC should retain connections of Ports that are not deleted.

2.8 Reset Unit Operations

Issue: it is desired to notify UOs that the Flowsheet has been reset to its initial state so that the Unit Operations can perform internal reset actions such as removing initial guesses. The Unit interface has no method to allow for this.

Recommendation: a Boolean named value "ResetOnValidate" may be exposed by ICapeCOSEUtilities to convey to any Unit Operation that the PME is resetting its Flowsheet (for example because the end-user has asked for that Reset to happen). Such a Reset could mean to a Unit Operation that solution guesses specific to the Unit Operation have to be set back to their initial values. Once the PME has changed the value of ResetOnValidate to TRUE, the PME will call Validate on each Unit Operation. Unit Operations that support resetting check for the named value within their Validate method. If the named value is present and is TRUE, the UO internally performs a reset. Once Validate has been called on each Unit Operation, the named value is set back to FALSE.

During normal validation, the PME should make sure that ResetOnValidate (if exposed) has a value of FALSE.

This scenario can also be used to reset a subset of Unit Operations upon end-user request.

2.9 Modal dialogs

Issue: in single-threaded operations, unexpected interruption of the flow of events can occur from a Unit Operation displaying dialogs while appearing in a recycle. In multi-threaded operations, modal dialogs can lead to a dead-lock of an application. In headless operations (operation without a monitor, for example scheduled runs on cloud computers, SaaS – Software as a Service, etc…), dialogs should be avoided altogether.

Clarification: PMCs including Unit Operations should refrain from showing modal dialogs except for during the call to ICapeUtilities::Edit.