Open Applications Group – Position Paper
Data Management Language Specification

Project Team Leader:
Steffen M. Fohn, Ph.D. – ADP

Authors:
Steffen M. Fohn, Ph.D. – ADP
Dave Carver – STAR
Isabel Espina – ADP
Kurt Kanaski – Merck
Santosh Krishnakumar – CISCO
Michael Rowell – OAGi

Reviewers:
David Connelly – OAGi
Chuck Allen – HRInterop
Paul Kiel – XML Helpline
Pat O'Connor – Infor

Version: 1.2
Document Number: 20090406-1

NOTICE

The information contained in this document is subject to change without notice.

The material in this document is published by the Open Applications Group, Inc. for evaluation. Publication of this document does not represent a commitment to implement any portion of this specification in the products of the submitters.

WHILE THE INFORMATION IN THIS PUBLICATION IS BELIEVED TO BE ACCURATE, OPEN APPLICATIONS GROUP, INC. MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANT ABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Open Applications Group, Inc. shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

For more information, contact:
Open Applications Group, Inc.
P.O. Box 4897
Marietta, Georgia 30061 USA
Telephone: 1.770.943.8364
Internet: http://www.openapplications.org

79 **Table of Contents**
80

103
102

Abstract

This document describes how to use the OAGIS Data Management Language to communicate data management instructions in OAGIS BOD (Business Object Documents) message instances.

Objective

The objective of this specification is to describe describes the language and guidelines for communicating and processing data management instructions (Create, Read, Update, and Delete operations) specified within message instances for messages defined in OAGi's Integration Specification (OAGIS).

The specification endeavors to attain the following design goals:

- A message encapsulates both behavior and structure – data management instructions should be contained in the contents of the message.
- The data management specification should be defined at the business layer. It is therefore agnostic to systems' physical database implementations.
- The data management specification should offer flexibility in accommodating different data management approaches (i.e., snapshot and incremental)
- The data management specification should enunciate concise language and guidelines for conveying data management instructions for each data management approach supported.
- The data management specification should promote simplicity so as to not add unnecessary complexity and overhead during message production and consumption.
- The data management specification should be technologically feasible across a majority of data binding frameworks.

Terminology

- Message – definition or schema of the information from which message instances are instantiated.
- BOD – (Business Object Document) is a message that assembled from OAGi's Integration Specification (OAGIS)

- Message instance – an instance of message that complies with the definition or schema of the message.
- BOD instance – is a message instance of a BOD.
- Get request – is a BOD instance of a BOD defined with the Get verb; it is used to request information from a system.
- Show response – is a BOD instance of a BOD defined with Show verb; it is used to respond to a Get request.

1.0 OVERVIEW

Although there are several criteria to successful application of a Message Library in an operational environment, there is one criterion that overshadows the others in relative importance. This is the consistent application of the message definitions and related guidelines across systems communicating with each other. Consistent application mandates standard representation and interpretation of the interchange language; this notion is otherwise referred to as the *contract* to which systems must adhere in the production and consumption of message instances.

Elements of the OAGIS interchange language support:

- Message transaction data (i.e., in the BOD Application Area)
- Message payload data (i.e., in the BOD Noun)
- Message data management instructions (i.e., in the BOD Verb)

This paper focuses on the last bullet. The objective is to describe both the language elements and the associated guidelines of how these language elements should be applied in order to meet their intended design purpose. This is necessary to ensure standard representation and interpretation of message data management instructions.

Note: OAGIS defines a message architecture called the Business Object Document (BOD) architecture. A given message definition, implementing this message architecture, is referred to as a Business Object Document (BOD). A BOD is composed of an ApplicationArea and a DataArea. The ApplicationArea acts as header to the message; the DataArea represents the message body that is comprised of a Verb and Noun.

2.0 RATIONALE

This document is intended to describe the specification (rules) for managing data communicated in OAGIS-based messages between systems. This is an essential part of the data management language specification.

This document is to be used as a guide by Application Architects, Information Architects, Business Analyst, and Developers to assist in the creation of system interfaces that produce and consume BOD messages.

3.0 APPROACH

The term data management in this document is used as an umbrella term to represent all CRUD (Create, Read, Update, and Delete) operations.

The remainder of this document describes the Data Management Language Specification, specifically:

- The constructs (vocabulary and syntactical structure) of the OAGIS data management language defined and available in the OAGIS message library
- The data management operations and the rules of how the data management language constructs, above, are to be applied

To assist in the readability of this document, discussion of the data management operations is subdivided into two major sections. The first section will address the Create, Update, and Delete (CUD) data management operations. The second section will address the Read (R) data management operations.

> Note:
>
> All rules of the specification are prefixed with an "R" and sequentially numbered (i.e., R1). This is applicable to all BODs and is independent of the BOD's version.

4.0 Data Management Language Specification

4.1 Constructs of the OAGIS Data Management Language

OAGIS defines three verb types: Action, Response, and Request. Table 1 shows the verbs classified by Verb Type.

Verb	Verb Type
Cancel	Action
Change	Action
Load	Action
Notify	Action
Post	Action
Process	Action
Sync	Action
Update	Action
Acknowledge	Response
Confirm	Response
Respond	Response
Show	Response
Get	Request

Table 1: OAGIS Verbs and Verb Types

The Verb itself represents a coarse-grain action, response, or request related to the respective Noun of the BOD. For example, the Process verb communicates a *request* to

the receiver system to process the message instance (for example, a ProcessPersonRegistration BOD communicates a request to the Registry Service to Process a Person Registration). More detailed constructs are available within the verb that enables communication of finer-grain actions on a Noun's components (vs. the coarser Noun level).

4.1.1 Action Verbs

The action verbs are the OAGi language elements through which Create, Update, and Delete data management instructions are conveyed by message senders to receivers. Table 1 lists the action verbs: Cancel, Change, Load, Post, Process, Sync, and Update.

Figure 1, shows the elements of the ActionVerbType schema definition. An action verb supports zero-to-many ActionCriteria. Each ActionCriteria supports zero-to-many ActionExpressions and zero-to-one ChangeStatus. The ActionExpression is the mechanism used to represent the data management instructions in a BOD instance; specifically, this includes identification of the element(s) and the action to be taken on those elements. Identification of the element may consist of its location in the schema structure and possibly additional key value information if it is necessary to identify a specific element of interest in a BOD instance. ChangeStatus may be used to communicate state change information (e.g. the EffectiveDateTime and ReasonCode for the state change as well as the FromStateCode and ToStateCode).

The ActionExpression has two attributes: actionCode and expressionLanguage. The ActionExpression.actionCode specifies an action to be taken by the receiver of the BOD instance. The ActionExpression.actionCode is restricted to a value domain. The actionCode's value domain includes: Add, Change, Delete, and Replace.

Recall that the BOD architecture specifies that a BOD instance must have exactly one Verb instance and may have one-to-many Noun instances. Since an action verb (e.g., process) supports multiple ActionCriteria and multiple ActionExpressions, a many-to-many relationship exists between the ActionExpression and the Noun. More specifically, an ActionExpression must be associated with one-to-many Nouns and a Noun may be associated zero-to-many ActionExpressions.

The noun of the BOD instance is used to represent the added entities for the element identified in the add action code, the changed entities for the element identified in the change action code, the deleted entities for the element identified in the delete action code, and the replacement entities for the element identified in the replace action code.

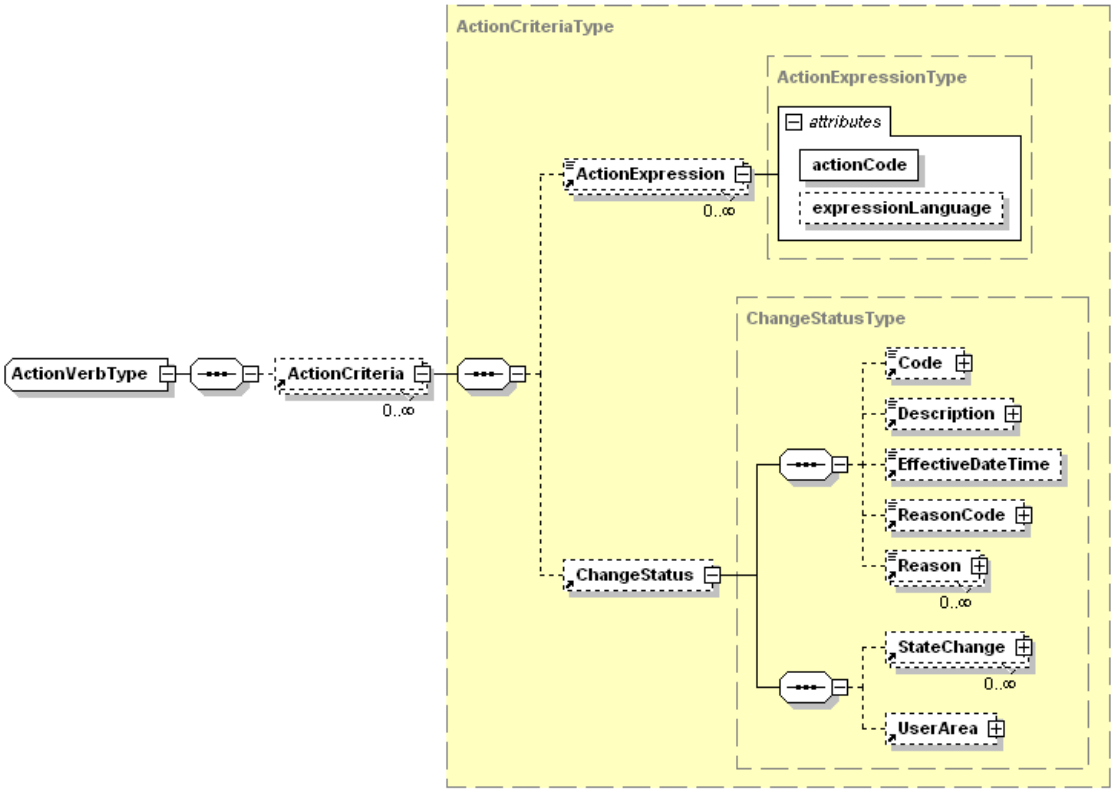

Figure 1: Action Verb Type

4.1.2 Request Verbs

The request verbs are the OAGi language elements through which Read data management instructions are conveyed by message senders to receivers. Table 1 lists the request verb: Get.

Figure 2, shows the elements of the RequestVerbType schema definition. The Expression has one attribute: expressionLanguage.

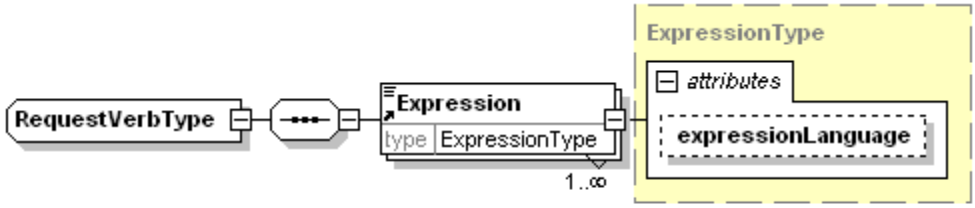

Figure 2: Request Verb Type

Figure 3, below, shows that the GetType is defined as an extension to the RequestVerbType. The Get verb is instantiated from the GetType.

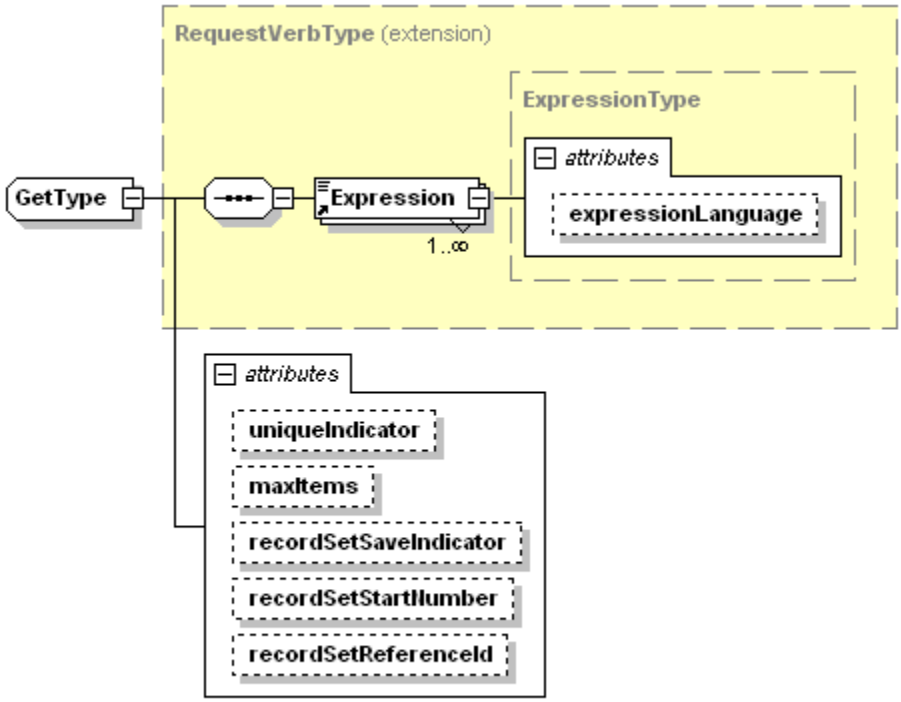

Figure 3: Get Type

The extension includes several attributes whose values may be set as part of a Get request. The attributes are defined as follows:

- **uniqueIndicator –** Indicates whether duplicates should be filtered out.
- **maxItems** – Communicates the maximum number of records of a recordSet that should be returned in a Show response.
- **recordSetSaveIndicator** – A true value indicates that receiver should save the record set.
- **recordSetStartNumber** – The record number identifying the first record that should be returned in the Show response. This attribute is specified on subsequent Get requests, not the initial Get request[1]. The requesting system may determine this number from the prior Show response (see the Show verb attributes for more information)

[1] This document differentiates, as needed, initial Get requests from subsequent ones. The two types of requests are related by a single read operation (selection and filter criteria). Subsequent

- **recordSetReferenceID** – **Unique identifier of** the RecordSet. It is generated by the producer of the Show response as a result of the original Get request.

4.1.3 Response Verbs

The response verbs are the OAGi language elements through which message receivers can convey meta-data on the response to the sender of the original message instance. The response verbs are used in message instances that respond to action verb-based message instances (i.e., an Acknowledge response to a Process action), request-verb based messages instances (i.e., a Show response to a Get request), or even other response-verb messages instances (i.e., a **Confirm response to** a **Show response).** Table 1 lists the response verbs: Acknowledge, Confirm, Respond, Show.

Figure 4, below, shows the elements of the ResponseVerbType schema definition. The ResponseExpression has two attributes: actionCode and expressionLanguage. The actionCode specifies an action that was taken by the receiver of the BOD instance.

The ResponseExpression.actionCode is restricted to a value domain. The actionCode's value domain includes: Accepted, Modified, and Rejected.

A response verb, specifically the Show verb, is the OAGi language element through which the results from processing the Read data management instruction are returned to the sender of the Get request. Table 1 shows Get as the only request verb.

Figure 5 shows that the ShowType is defined as an extension of the RequestVerbType.

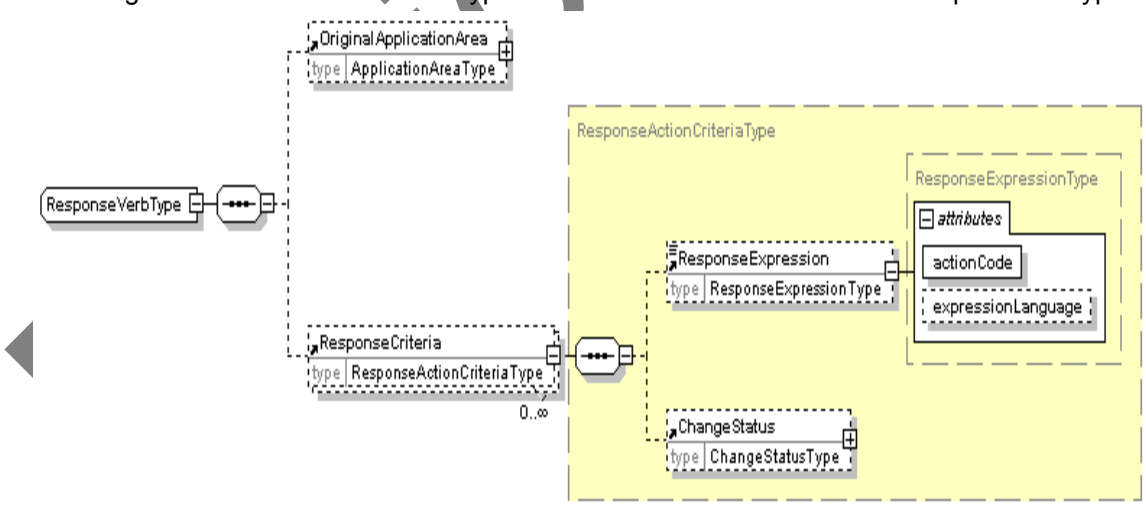

Figure 4: Response Verb Type

Get request(s) may be communicated when the initial Get request results in more records that can be returned in a single Show response.

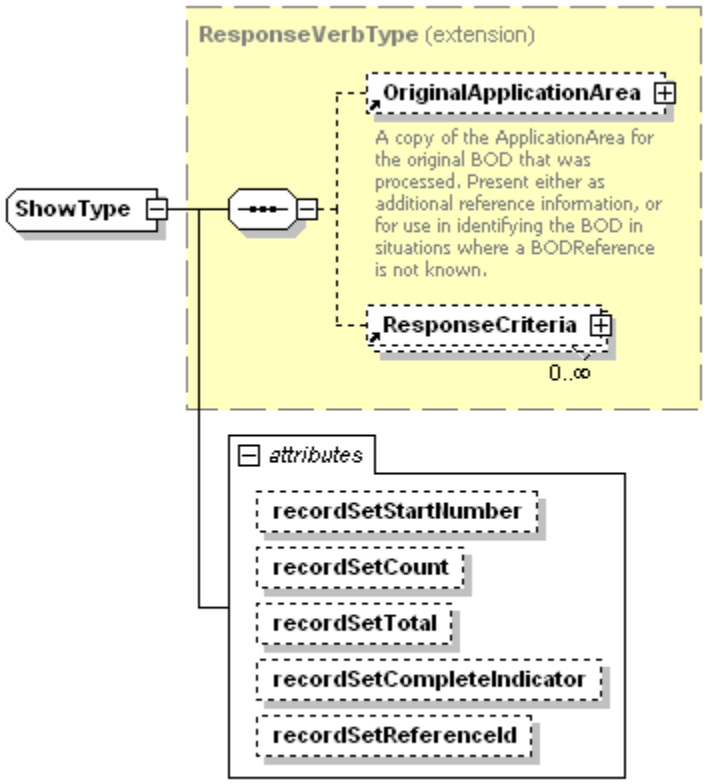

Figure 5: Show Type

The extension includes several attributes whose values may be set of part of a Show response. The attributes are defined as follows:

- **recordSetStartNumber –** The record number identifying the first record returned in the Show response. The producer of the Show response generates this number. It used by the requesting system to determine the start number of the subsequent Get request
- **recordSetCount** – Number of records in the recordSet**.**
- **recordSetTotal –** Number of total records in a recordSet**.**
- **recordSetCompleteIndicator –** Indicates whether the Show response represents the end of the recordSet**.**
- **recordSetReferenceID –** Unique identifier of the RecordSet. It is generated by the producer of the Show response as a result of the original Get request.

4.2 Data Management Approaches and Operations

This section describes how the OAGIS data management language constructs, above, are to be applied in describing and conveying data management operations in BOD message instances. As mentioned above, this is described in two parts: first, the Create, Update, and Delete (CUD) data management operations, and second, the Read (R) data management operation.

4.2.1 Create, Update, and Delete Data Management

Create, update, and delete data management, as referred to in this document, considers two related aspects:

- the data management approach
- the data management instructions, needed to convey the approach and the create, update, and delete operations

There are two data management approaches:

- the Snapshot or Full Refresh
- the Incremental or Delta

While the Snapshot approach is generally considered simpler to implement than the Incremental approach, message instances based on the Snapshot approach are larger in size and may require longer processing time than those based on the Incremental approach.

Recall, that the action verbs are the OAGi language elements through which create, update, and delete data management instructions are conveyed by message senders to receivers. This section will describe how the language elements are to be used to clearly communicate the data both the data management approach and operations.

There are standard conventions and guidelines that form the basis of the data management specification that are applicable to both the incremental and snapshot approaches; they may be considered rules and are stated below.

For any action verb-based BOD instance the following rules (R) apply:

R1: An entity[2] represented by an element in a noun instance, *should* be identified by an ID or set of IDs (in the case of a composite key)[3].

[2] Entity is an instance of an entity class and is characterized by having properties.

R2: The management of entity IDs (e.g., surrogate keys) must not occur in message instances communicating business transactions; management of IDs (e.g., migrate one ID value to another ID value in a merge process) must occur in a message specialized designed for this purpose.

R3: A business task identifier[4] *may* be specified in the ApplicationArea.Sender.TaskID as an annotation on the message instance

R4: The flexibility of the schema supports a many-to-many relationship between the verb's ActionCriteria and the Noun.
1. A Noun instance *may* be associated with multiple ActionCriteria instances.
2. An ActionCriteria instance *may* be associated with multiple Noun instances.

R5: The flexibility of the schema supports a many-to-many relationship between the verb's ActionExpression and the Noun.
1. A Noun instance may be associated with multiple ActionExpression instances.
2. An ActionExpression instance may be associated with multiple Noun instances.

Figure 1 in the previous section, shows that the ActionExpression and ChangeStatus are related through the ActionCriteria. The cardinalities of these elements mandates that the set of ActionExpressions within the ActionCriteria may be associated with at most one ChangeStatus.

R6: If one-to-many ActionExpressions are associated with a ChangeStatus, then that association *must* be represented with exactly one ActionCriteria.

R7: An actionCode *may* be specified in the ActionExpression.actionCode

R8: actionCode="Add" in the Action Expression *must* be used to indicate the creation/addition of an entity represented by the element identified in the expression.

R9: actionCode="Change" in the Action Expression *must* be used to indicate the modification of an entity represented by the element identified in the expression.

R10: actionCode="Delete" in the Action Expression *must* be used to indicate the removal/deletion of an entity represented by the element identified in the expression.

R11: actionCode="Replace" in the Action Expression *must* be used to indicate the replacement of an entity represented by the element identified in the expression.

R12: The expression of the ActionExpression *must* specify the element[5] of the noun instance that represents the managed entity.

[3] IDs used to identify entity(s) being managed should be universally agreed-to across applications that are participating in an integration initiative. An ID Registry should be established that defines for each message the entity IDs required.

[4] A business task is a generalization of business action and business event. Business actions correspond to commands and requests; business events correspond to event notifications. Readers are referred to the document, "Business Task Message Framework" for further information on business tasks

[5] An element of the noun may include the noun, itself, or a constituent element (i.e., Component, Field).

Note: Given a BOD instance with multiple Noun instances, if an expression applies to *all* of the noun instances, the expression *must* not specify a noun instance. (In other words, specification of the noun instance must be avoided since the expression is intended to apply to all of the noun instances.)

Note: Given a BOD instance with multiple Noun instances, if an expression applies uniquely to a noun *instance*, the expression *must* specify the noun instance being managed.

R13: The expression of the Action Expressions *must* be written in an xml expression language (i.e., XPath, XQuery).

R14: The set of elements being managed must be well-defined and understood by message senders and receivers.

4.2.1.1 The Snapshot Approach

The Snapshot or Full Refresh approach is defined by the following:

- A subset of a Noun is communicated in a message instance; note that the subset could be the Noun, itself, or any element[6] therein. This subset corresponds to scope of data being managed, in other words the scope of data in the snapshot.
    - Any subset of elements managed together should be aggregated as an element in the message and be identifiable through a standard ID(s) (i.e., the properties of a person would be aggregated in a person element that has an ID).
    - Elements in a message instance include all elements in the scope of the snapshot.
    - The scope of data being managed must be *well-defined* and understood by the senders and receivers
- A snapshot by definition is a *refresh* or *replacement* of some set of data for a defined scope. Processing a snapshot may result in data having been created, updated, deleted, and/or not changed. This is because a snapshot contains all the data in a defined scope, regardless of whether or not a given element within the scope has changed.
- Detailed data management instructions for Create, Update, and Delete operations *are not* communicated in the message instance.

[6] Element, as used herein, is equivalent to the concept of a schema element that may represent entity classes and their properties.

The sender of a Snapshot message may be either a System of Record (SOR) publishing a snapshot of data or a non-SOR system that is requesting its targeted receivers to process a snapshot of data. In both cases, the Sender may communicate the business task that caused the message instance to be created and communicated.

The receiver of a Snapshot message instance MUST update its system with all the elements of the message instance that it manages. This may result in "creating" entities that were in the message but not in the system, updating" entities that were in the message and also in the system, and "deleting" entities that were not in the message but present in the system.

There are standard conventions and guidelines that form the basis of the data management specification that are applicable to the Snapshot approach; they may be considered preconditions.

For any action verb-based BOD instance used in the Snapshot approach the following rules (R) apply:

R17: Any of the action verbs may be used.

R18: The ActionExpression.actionCode *must* be restricted to the set of values: {Replace}

The Replace action code must be used in the snapshot data management approach where a "snapshot" of the entity(s) (and all its constituent entity(s)) is taken by the sending system and published. The snapshot is considered to be a "refresh" of the data; adds, changes, deletes are not explicitly indicated in the message.

For any action verb-based BOD instance used in the Snapshot approach the following rules (R) apply:

R14.1: The set of elements being managed must be well-defined and understood by message senders and receivers.

4.2.1.2 The Incremental Approach

The Incremental or Delta approach is defined by the following:

- A subset of a Noun is communicated in a message instance; note that the subset could be the Noun, itself, or any element therein.
    - Any subset of elements managed together should be aggregated as an element in the message and be identifiable through a standard ID(s) (i.e., the properties of a person would be aggregated in a person element that has an ID).

- Elements (with the exception of ID(s)) in a message instance are limited to those that contain entities that have been created, updated or deleted[7].
- The scope of data be managed must be *well-defined* and understood by the senders and receivers

- Detailed data management instructions for create, update, and delete operations *are* communicated in the message instance
- Only entities that are created, updated, or deleted are communicated

The sender of a Incremental message may be either a System of Record (SOR) publishing a Create, Update or Delete operation or a non-SOR system that is requesting its targeted receivers to process a Create, Update, or Delete operation.

The receiver of an Incremental message instance must update its system per the data management instructions (create, update, or delete operations on some set of elements) conveyed in the message instance through the ActionExpressions. This may result in the "creation", "update", or "deletion" of entities in the system as specified in the data management instructions. It should be noted that the receiving system may interpret a "deletion" as either a physical delete or logical delete and is dependent upon the receiving systems data retention policies.

There are standard conventions and guidelines that form the basis of the data management specification that are applicable to the Incremental approach; they may be considered preconditions.

For any action verb-based BOD instance used in the Incremental approach the following rules (R) apply:

R19: Any of the action verbs *may* be used.

R20: The ActionExpression.actionCode *must* be restricted to the set of values: {Add, Change, Delete}

R14.2: The set of elements being managed must be well-defined and understood by message senders and receivers.

For any "add" operation, the following rules (R) apply:

R21: The message instance *must* contain an ActionExpression with an actionCode of "Add".

R12.1: The expression of the ActionExpression *must* specify the element of the noun instance that represents the created entity.

For any "delete" operation, the following rules (R) apply:

R22: The message instance *must* contain an ActionExpression with an actionCode of "Delete".

[7] This characteristic is a key differentiator from the snapshot approach.

R12.2: The expression of the ActionExpression *must* specify the element of the noun instance that represents the deleted entity.

For any "delete" operation, where the entity being deleted is identifiable with ID(s) the following rules (R) apply:

R23: The message instance noun *must* only provide a reference to the entity via its ID(s)

4.2.1.3 Identifying Noun Instances Managed

The flexibility of the BOD architecture allows:

- Multiple noun instances to be communicated in a single BOD message instance.
- Multiple ActionExpressions to be communicated in the verb of the single BOD message instance.

It is therefore possible to have a single BOD message instance with both multiple nouns instances and multiple ActionExpressions.

Recall Rule 12, in particular the last statement: "If an ActionExpression applies uniquely to a noun instance then it should identify that noun instance."

R12: An expression in the ActionExpression *must* specify the element[8] of the noun (or occurrence thereof[9]) that is being managed.

In the case where an expression applies to **one or more** noun instances, the expression must specify the noun element (i.e., node set) being managed (Note: in this case, specification of the noun instance is not required since the expression is intended to apply to all of the noun instances.)

In the case where an expression applies uniquely to a noun *instance*, the expression *must* specify the noun instance being managed

In support of this need, a noun may be defined with DocumentID property element. The DocumentID serves as an identifier of an entity corresponding to the noun. In addition to the explicit DocumentID, the position of the noun instance in the BOD instance may also serve to identify a specific noun instance. The positions of the sequence are specific to the message instance (e.g., in the case of a message instance with two noun instances, the first noun instance is understood to be in the first position of the sequence and the second noun instance is understood to be in the second position of the sequence. The noun instance position is referred to as the DocumentSequence below.

[8] An element of the noun may include the noun, itself, or a constituent element (i.e., Component, Field).
[9] Multiple occurrences of a noun may exist within a single message instance.

For any "create", "update", or "delete" operation, where multiple noun instances are communicated in a single BOD message instance the following rules (R) apply:

R27: Either the DocumentID or the DocumentSequence *may* serve as the identifier of the noun instance and used in the ActionExpression to identify the noun instance.

DocumentID *should* be used if systems are maintaining noun instance IDs (i.e., PurchaseOrder Reference Number) for the entity corresponding to the noun.

DocumentSequence *should* be used if systems are not maintaining noun instance IDs.

4.2.1.4 Summary of the Approaches

Figure 6 provides an overview diagram of the message concepts, discussed above, for the representation and communication of create, update, and delete operations in data management for BODs. The note construct at the bottom of the figure highlights additional constraints not already captured in the model.

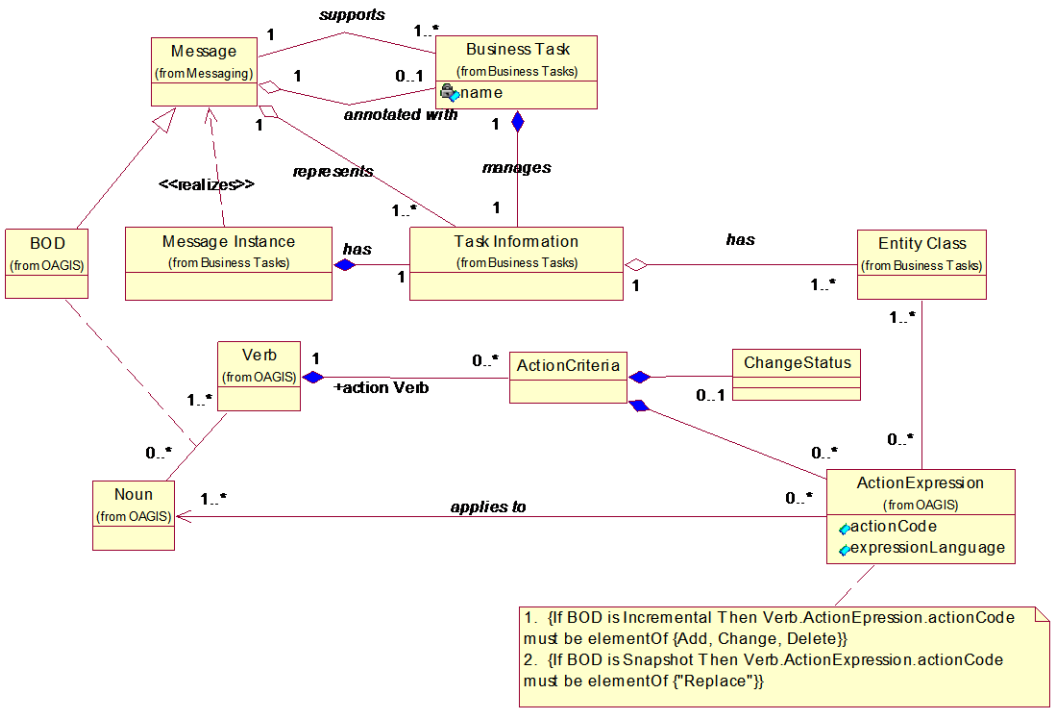

Figure 6: BOD Data Management Model for Create, Update, and Delete Operations

Table 2 relates the OAGi action verb and action code combinations to the data management approach (Snapshot and Incremental). Each combination is annotated with the Create, Update, and Delete operations that is provided through the combination. It serves to identify the "universe" of possibilities or feasible combinations of OAGIS verbs and action codes.

Verb	Action Code			
	Add	Change	Replace	Delete
Cancel	--	--	--	Incremental - Delete
Change	Incremental - Create	Incremental - Update	Snapshot[10] - Create Update, Delete	Incremental - Delete
Load	Incremental - Create	--	Snapshot - Create, Update, Delete	--
Notify	Incremental - Create	Incremental - Update	Snapshot - Create, Update, Delete	Incremental - Delete
Post	Incremental - Create	Incremental - Update	Snapshot - Create, Update, Delete	Incremental - Delete
Process	Incremental - Create	Incremental - Update	Snapshot - Create, Update, Delete	Incremental - Delete
Sync	Incremental - Create	Incremental - Update	Snapshot - Create, Update, Delete	Incremental - Delete
Update	Incremental - Create	Incremental - Update	Snapshot - Create, Update, Delete	Incremental - Delete

Table 2: Relationship of the OAGIS Action Verb and Action Code to the Data management approach and Create, Update, and Delete operations

The OAGIS definition of each verb can be found in the OAGIS library documentation.

For any "update" operation, the following rules (R) apply:

R24: The message instance *must* contain an ActionExpression with an actionCode of "Change".

R12.3: The expression of the ActionExpression *must* specify the element of the noun instance that represents the changed entity.

R25: The message instance noun *must* provide a reference to the entity via its ID(s) (if one exists) and only include the updated properties of the entity.

[10] In a Snapshot, the Create, Update, and Delete operations are implicit and occur within the data scope of the snapshot.

4.2.2 Read Data Management

Read Data management, as referred to in this document, considers the data management instructions, needed to convey Read or query operations.

Recall, that the Get and Show verbs (Figures 3 and 5, respectively) are the OAGi language elements through which Read operations and their results are conveyed between requesting and responding systems. This section will describe how the language elements are applied for the following:

- representation and communicate read operations
- management of the records of resulting from a read operation

Both selection and filter techniques are available to Get-based BOD messages.. The techniques when applied in a Get message instance specify a read operation or query.

4.2.2.1 Techniques for Specifying Selection Criteria

Two alternative techniques are available for representation of the selection criteria in the read operation (or query). The first technique uses a reference to predefined (or canned) selection criteria. The second technique is more flexible and dynamic and allows the specification of the selection criteria in the Get request. These techniques are presented below.

For any "read" operation in a Get request, the following rules (R) apply:

R29: The selection criteria *must* be specified using either the predefined technique or the dynamic technique.

4.2.2.1.1 Predefined Selection Criteria Technique

The predefined selection criteria technique requires that the communicating systems define and agree upon the selection criteria and the name (i.e., keyword) that will be used to reference specific selection criteria. This must be done a priori to any Get requests and Show responses.

Once the predefined selection criteria are established, senders of a Get request must include a reference to the predefined selection criteria in the Expression of the Get verb.

A reference to predefined selection criteria should be named such that it describes the specific selection criteria. For example, if the selection criteria includes all of the information in a noun, then the noun name may serve as the reference.

It is likely that predefined selection criteria will evolve and change overtime. For this reason, predefined selection criteria should be versioned to ensure that requesting and responding systems are aligned to same version of predefined selection criteria. If multiple versions of predefined selection criteria are simultaneously supported by a system, then the version identifier of the predefined selection criteria should be included in its reference.

For any "read" operation, using predefined selection criteria, in a Get request, the following rules (R) apply:

R30: A reference[11] to the predefined selection criteria *must* be represented in the Expression of the Get verb.

R31: Any reference to a predefined selection criteria, that is sent in a Get request to a system, *must* be part-of the set of predefined criteria supported by that system.

R32: The expressionLanguage attribute of the Expression *must* be assigned the value "Predefined".

R33: A reference to predefined selection criteria *may* be named such that it describes the selection criteria,

R34: If multiple versions of a predefined selection criteria are simultaneously supported by a system, then the version identifier of the predefined selection criteria *should* be included in its reference.

4.2.2.1.2 Dynamic Selection Criteria Technique

This technique for representing the selection criteria is called Data Type Selection. Data Type selection enables the requesting system to identify which Data Types within the noun are requested to be returned in the response. The use of this capability is described for each corresponding Data Type for all BODs that use the Get verb. The Data Types are identified for retrieval within the Get instance of a BOD by including the name of the Data Type in the expression of the Get verb but without any filter criteria (e.g., Field Identifiers) identified within the Data Type. This will signify to the responding application that all of the data that corresponds to that Data Type is to be included in the response. If the Data Type is not requested, the Data Type identifier is not included in the Get request and this will signify to the responding component that the Data Type is not to be returned.

For any "read" operation, using dynamic selection criteria, in a Get request, the following rules (R) apply:

R35: DataType selection criteria *must* be represented in the Expression.

R36: An expression specified in the Expression element of the Get verb *must* be written in an xml expression language (i.e., XPath, XQuery).

[11] A reference to the predefined selection criteria serves to identify the selection criteria.

4.2.2.2 Technique for Specifying Filter Criteria

The filtering technique is called Field-Based Filtering. Within a Get-based Business Object Document, the first Data Type that occurs in a specific BOD structure is commonly used to provide the Field-Based Selection criteria. This is always defined within the specific BOD and is commonly the required fields for that specific Data type. The Field-Based Selection enables the requesting system to provide a value or values (in the case of multiple required Field Identifiers), in the required fields. Then the responding component uses those values to find and return the requested information to the originating business software component.

For any "read" operation, expressing filter criteria, in a Get request, the following rules (R) apply:

R37: Field-Based filter criteria *must* be represented in the noun instance.

The system responding to the Get request, communicates the results of the Read operation to the requesting system in a Show response.

4.2.2.3 Multiple-Record Handling Techniques

This section discusses two techniques for the handling of multiple records resulting from the execution of a single read operation or query when these results cannot be returned in a single Show response (message instance). This is often the case when either the requesting or responding systems have message size performance measures whose thresholds cannot be exceeded in order to maintain adequate system performance. The techniques discussed below present alternative patterns for managing the return of results in multiple Show responses.

Recall that the read operation is defined through the selection and filter techniques in the Get verb, as described above. Recall also that both the Get and Show verbs have several attributes. These attributes were previously defined and are repeated below.

Get Verb Attributes:

- **uniqueIndicator –** Indicates whether duplicates should be filtered out.
- **maxItems** – Communicates the maximum number of records of a recordSet that should be returned in a Show response..
- **recordSetSaveIndicator** – A true value indicates that the receiver of the Get request should save the record set.

- **recordSetStartNumber** – The record number identifying the first record that should be returned in the Show response. This attribute is specified on subsequent Get requests, not the initial Get request[12]. The requesting system may determine this number from the prior Show response (see the Show verb attributes for more information).

- **recordSetReferenceID** – Unique identifier of the RecordSet. It is generated by the producer of the Show response as a result of the initial Get request.

In general these attributes may be specified by a system, sending a Get request to indicate how the receiving system should respond.

Show Verb Attributes:

- **recordSetStartNumber –** The record number identifying the first record returned in the Show response. The producer of the Show response generates this number. It used by the requesting system[13] to determine the start number of the subsequent Get request

- **recordSetCount** – Number of records in the recordSet**.**

- **recordSetTotal –** Number of total records in a recordSet**.**

- **recordSetCompleteIndicator –** Indicates whether the Show response represents the end of the recordSet.

- **recordSetReferenceID –** Unique identifier of the RecordSet. It is generated by the producer of the Show response as a result of the original Get request.

In general these attributes may be specified by a system, sending a Show response, to communicate information on the results of the read operation specified in a Get request.

Both techniques for the handling of multiple records, presented below, leverage the Record Set concept. A Record Set is defined herein to represent a set of records resulting from the execution of a single read operation or query where the read operation is defined through the selection and filter techniques in the Get verb. The Record Set concept is represented by Get and Show verb attributes with a “recordSet” prefix. A Record Set is defined as a logical construct that *may* or *may not* be saved by the system that executed the read operation.

For any Get request, the following rules (R) apply:

[12] This document differentiates, as needed, initial Get requests from subsequent ones. The two types of requests are related by a single read operation (selection and filter criteria). Subsequent Get request(s) may be communicated when the initial Get request results in more records that can be returned in a single Show response.
[13] Requesting system refers to the system that sent the Get request.

R38: The requesting system *may* specify that the Record Set, representing the results of a read operation, is required to be "saved" under the following conditions[14]:

1. The Get request could result in more data than the requesting system is able to process in a single Show response.
2. The requesting system *requires* read consistency[15] for the query results.

R39: To specify that the responding system *is required to* save a Record Set, the requesting system *must* assign the recordSetSaveIndicator attribute of the Get verb to "true" in the Get message instance.

R40: The requesting system *must* specify that the Record Set, representing the results of a read operation, *is not required to* be "saved" under the following conditions:

1. The Get request could result in more data than the requesting system is able to process in a single Show response.
2. The requesting system *does not* require read consistency for the query results.

R41: To specify that the responding system *is not required* to save a Record Set, the requesting system *may* assign the recordSetSaveIndicator attribute of the Get verb to "false" in the Get message instance.

For any initial Get request, the following rules (R) apply:

R42: If the requesting system has not specified value assignments to the Get verb attributes then the responding system must default the attribute values as follows:

- uniqueIndicator – "true"
- maxItems – "unbounded"
- recordSetSaveIndicator – "false"
- recordSetStartNumber - *Not Applicable (Ignore)*
- recordSetReferenceID - *Not Applicable (Ignore)*

For any Get request, leveraging a Record Set (saved or unsaved), the following rules (R) apply:

R43: The responding system *must* assign values to the following Show verb attributes that describe the number of records being returned in the Show response. These attributes are:

- recordSetStartNumber
- recordSetCount
- recordSetCompleteIndicator

[14] Alternatves to a Record Set-based solution may be used to satisfy these conditions (e.g. message segmentation at the transport layer).

[15] Read consistency ensures that all the data returned by a single query comes from a single point in time.

R43.1: The responding system *may* assign values to the following Show verb attributes:

- recordSetTotal

For any subsequent Get request, the following rules (R) apply:

R44: The requesting system *must* assign values to all of the Get verb attributes that describe the number of records being requested in the Get message instance. These attributes are:

- recordSetStartNumber

The recordSetStartNumber must be calculated using the following equation: GetMessage $_{i+1}$.recordSetStartNumber = ShowMessage $_{i}$.recordSetCount + 1 where i represents a Get/Show message instance (request/response) pair.

Notice that the number of records returned is always limited by the maximum number of items (records) specified in the maxItems attribute by the sending system in the Get message. This attribute is set per the message size performance measure of the sending system with respect to message consumption.

For any Get request, the following rules (R) apply:

R45: The requesting system *may* specify in a Get request the maximum number of items (records) to be returned in a Show response using the maxItems attribute of the Get verb.

Similarly, the receiving systems may have a message size performance measure with respect to the message production. Therefore the number of records in the Show message instance should always correspond to the more restrictive performance measure among the sending and receiving systems. In other words, the record count in the Show message instance should equal the minimum of the sending system's maximum number of items (records) and the receiving system's maximum number of items (records).

Recall that all BOD definitions restrict a given BOD instance to exactly one verb instance and one to many noun instances. In addition, the definition of the Get verb allows one to many Expression instances. As a result, it is possible that a single Get verb-based BOD instance could communicate multiple read operations (Expression instance and Noun instance combinations). However, such use is limited by a single set of attributes on the Get and Show verbs for managing the results of the read operation. Since a Get verb-based or Show verb-based BOD instance may have at most a single Get or Show verb instance, it is not possible to separately manage the results of multiple read operations. Therefore the following rule is defined.

For any Get request the following rules (R) apply:

R46: Although the schema supports a many-to-many relationship between the verb's Expression and the Noun, the following constraints *must* be applied:

1. Exactly one read operation must be represented.

2. A single read operation must be comprised of the following:
   - One or more Expression instance(s)
   - No more than one Noun instance

As a result of this rule, all Expression instance(s) may be associated with at most one Noun instance.

4.2.2.3.1 A Single Show Response to a Single Get Request

The first technique or pattern uses multiple pairs of Get request and Show responses to request and return the complete results of the read operation or query. This technique leverages the Record Set concept, represented by attributes, prefixed with "recordSet" in the Get and Show verbs (see above).

This technique relies on applying the Record Set concept in both the Get requests and Show responses. Since the requesting system may specify that a Record Set be "saved" by the responding system, two alternatives exist in using Record Set: Saved Record Set and Unsaved Record Set.

Using a Saved Record Set

In this alternative, the initial Get request specifies that the responding system *must* save a Record Set, by having assigned the recordsetSaveIndicator to "true".

The responding system must uniquely identify the RecordSet (using the recordSetReferenceID) and return its identifier in the Show response along with additional information on the records, such as the number of records (recordSetCount) of the Record Set being returned. The Record Set identifier must be then specified on any subsequent Get requests where additional records of the Record Set are requested.

For any Get request, leveraging a Saved Record Set, the following rules (R) apply:

R47: The responding system *must* create a unique identifier of the Record Set and assign its value to the recordSetReferenceID attribute of the Show verb in the corresponding Show response.

R48: For any subsequent Get request, the requesting system *must* specify the unique identifier of the Record Set, provided by the responding system (in the Show response to the initial Get request), in the recordSetReferenceID attribute of the Get verb.

An Example Using Saved Record Sets:

A requesting system sends a Get request for all Shipments for Company Code ABC with no more than 100 unique shipments at a time. The requesting system *does* require that the receiving system maintain a saved record set for the results of the request. Subsequent Get requests are issued for additional records (beyond those included in the initial Show response).

The Get verb attribute value assignments are:

- uniqueIndicator = “true”
- maxItems = 100
- recordSetSaveIndicator = “true”

The responding system processes the Get request, constructs, executes a query, creates a record set, and returns 1000 shipments for Company Code ABC. The system responds with the first 100 records in a Show response. The attribute value assignments are:

- recordSetStartNumber = 1
- recordSetCount = 100
- recordSetTotal = 200
- recordSetCompleteIndicator = false
- recordSetReferenceID = 253

The requesting system (having received the Show response then requests the next 100 records. It sends the Get request with the following attribute value assignments:

- maxItems = 100
- recordSetStartNumber=101
- recordSetReferenceID = 253

The responding system returns the Show response with the following attribute value assignments:

- recordSetStartNumber = 101
- recordSetCount = 100
- recordSetTotal = 200
- recordSetCompleteIndicator = true
- recordSetReferenceID = 253

When leveraging the Saved Record Set approach, Record Set timeout settings should be maintained by the responding system. Once threshold for a Record Set timeout has been met the responding system may recover the resources that were used to manage that

Record Set. Timeout settings should be agreed to between trading partners as part of the overall contract and are communicated within the Get and Show message instances.

Using an Unsaved Record Set

This alternative is very similar to the "Saved Record Set" alternative with one primary difference: the Record Set is not saved by the responding system.

In this alternative, the initial Get request specifies that the receiving system *is not required* to save a Record Set by having assigned the recordsetSaveIndicator to "false". Note that although it is not necessary to save a Record Set from the perspective of the sender, the receiver *may* still elect to save the Record Set.

The receiving system *may* uniquely identify the Record Set (using the recordSetReferenceID) and return its identifier in the Show message along with additional information, such as the number of records (recordSetCount) of the Record Set being returned. If a Record Set identifier was provided, then it *must* be specified on any subsequent Get requests where additional records of the Record Set are requested.

If the responding system has elected to not save the Record Set, then the responding system must re-execute the read operation or query upon any subsequent Get requests where additional records of the Record Set are requested.

For any Get request, leveraging an Unsaved Record Set, the following rules (R) apply:

R49: The responding system *may* create a unique identifier of the Record Set and assign its value to the recordSetReferenceID attribute of the Show verb in the corresponding Show response.

R50: For any subsequent Get requests, the sender *must* specify the unique identifier of the Record Set, *if* provided by the responding system (in the Show response to the initial Get request), in the recordSetReferenceID attribute of the Get verb.

An Example Using Unsaved Record Sets:

This is example is almost identical to the previous example, illustrating use of the Saved Record Set; the difference is in the value assignments of the recordSetSaveIndicator and recordSetReferenceID attributes.

A requesting system sends a Get request for all Shipments for Company Code ABC with no more than 100 unique shipments at a time. The requesting system *does not* require that the receiving system to maintain a record set for the results of the request. Subsequent Get requests are issued for additional records (beyond those included in the initial Show response).

The Get verb attribute value assignments are:

- uniqueIndicator = “true”
- maxItems = 100
- recordSetSaveIndicator = “false”

The responding system processes the Get request, constructs, executes a query, creates a record set, and returns 1000 shipments for Company Code ABC. The system responds with the first 100 records in a Show response. The attribute value assignments are:

- recordSetStartNumber = 1
- recordSetCount = 100
- recordSetTotal = 200
- recordSetCompleteIndicator = false

The requesting system (having received the Show response then requests the next 100 items. It sends the Get request with the following attribute value assignments:

- maxItems = 100
- recordSetStartNumber=101

The receiving system of the Get request returns the Show response with the following attribute value assignments:

- recordSetStartNumber = 101
- recordSetCount = 100
- recordSetTotal = 200
- recordSetCompleteIndicator = true

4.2.2.3.2 Multiple Show Responses to a Single Get Request

Certain request/response scenarios exist (i.e., a data load from one system to another) that are characterized by the following:

- a large number of records in the resultant Record Set

825 • all of the records satisfying the read operations must be returned to the requesting
826 system

827 In such scenarios system performance may be gained by limiting the number of Get
828 requests to a single request. The benefits are listed below:

829 • The responding system must not “save” the Record Set (in the Saved Record Set
830 case)

831 • The responding system must not re-execute the read operation or query (in the
832 Unsaved Record Set case)

833 • The overhead in issuing creating, communicating, and processing multiple Get
834 requests, for a subset of a Record Set at a time, associated with the Get request
835 and Show response pairs is avoided.

836 This technique or pattern uses a single Get and multiple Show message instances to
837 request and return, respectively, the complete results of the read operation or query. As
838 with the first technique, it leverages the Record Set concept.

839 This technique relies on applying the Record Set concept in both the Get and Show
840 message instances. However, in this case there is no need for the requesting system to
841 specify that the responding system “save” a Record Set. For this reason, several of the
842 Get verb attributes, used to identify the RecordSet and the records in the Record Set, are
843 not applicable; they are the following:

844 • recordSetSaveIndicator

845 • recordSetStartNumber

846 • recordSetReferenceID

847 As with the previous technique, all of the Show attributes, describing the Record Set, with
848 the exception of the recordSetReferenceID must be assigned values (see rule R46).

849 There is currently no mechanism in the Get verb by which a requesting system may specify
850 to the responding system that all of the records of a Record Set should be communicated
851 to the requesting system. In lieu of such a mechanism, private agreements may be
852 created between systems that outline the conditions under which this technique or pattern
853 should be applied. These conditions include:

854 • specific message(s),

855 • the threshold (number of records in the Record Set) at which point multiple Show
856 message instances will be sent

857 # APPENDIX A: REFERENCES

858 • OAGi (2005) Open Application Group Integration Specification (OAGIS) Library",
859 Version 9.0, Open Applications Group, April.

860 • W3C (2004), "XML Schema", October 28.

861

APPENDIX B: RULE TERMINOLOGY

This document uses the following terminology:

1. MUST: This word means that the requirement is absolutely REQUIRED to be implemented with no exceptions.
2. MUST NOT: This phrase means that the requirement specifies an absolute PROHIBITION and is not to be implemented.
3. SHOULD: This word means that the requirement is REQUIRED unless an exception has been granted through the exception process.
4. SHOULD NOT: This phrase means that the requirement is REQUIRED NOT to be implemented unless an exception has been granted through an exception process.
5. MAY: This word means that the requirement is OPTIONAL
6. Note: Terminology adapted from Scott O. Bradner, “Key words for use in RFC’s to Indicate Requirement Levels,” The Internet Engineering Task Force (IETF) RFC (Requests for Comments) 2119, March 1997.

APPENDIX C: EXAMPLES

The following table offers some examples on the application of the data management techniques. The examples are illustrated through the use of business scenarios.

Data Management Technique	BOD Message	TaskID (Business Event)	Verb	Verb Action Code	Verb ActionExpression	Required Entity IDs
Business Scenario: 1. A Purchase Order is created.						
***Incremental* - Create**	ProcessPurchaseOrder	New Purchase Order	Process	Add	/ProcessPurchaseOrder/DataArea/ PurchaseOrder	PurchaseOrderHeader. DocumentID.ID
Notes: 1. The message contains the complete order.						
Business Scenario: 2. The order quantity on an existing line in a Purchase Order is updated.						
Snapshot	ChangePurchaseOrder	Purchase Order Change	Change	Replace	/ProcessPurchaseOrder/DataArea/ PurchaseOrder	PurchaseOrderHeader. DocumentID.ID
Notes: 1. The message contains the complete order.						
***Incremental* - Update**	ChangePurchaseOrder	Purchase Order Change	Change	Change	/ProcessPurchaseOrder/DataArea/ PurchaseOrder/PurchaseOrderLine/	PurchaseOrderHeader. DocumentID.ID; PurchaseOrderLine LineNumber
Notes:						

1. Only the DocumentID is provided in the OrderHeader.
2. All elements of the OrderLine are provided.

Data Management Technique	BOD Message	TaskID (Business Event)	Verb	Verb Action Code	Verb ActionExpression	Required Entity IDs
Business Scenario: 3. A line item on an existing Purchase Order is removed.						
Snapshot	ChangePurchaseOrder	Purchase Order Change	Change	Replace	/ProcessPurchaseOrder/DataArea/ PurchaseOrder	PurchaseOrderHeader. DocumentID.ID; PurchaseOrderLine LineNumber
Notes: 1. The complete order is provided						
Incremental **- Delete**	ChangePurchaseOrder	Purchase Order Change	Change	Delete	/ProcessPurchaseOrder/DataArea/ PurchaseOrder/PurchaseOrderLine/	PurchaseOrderHeader. DocumentID.ID; PurchaseOrderLine LineNumber
Notes: 1. Only the DocumentID is provided in the OrderHeader. 2. Only the LineNumber is provided in the OrderLine.						
Business Scenario: 4. A Purchase Order is cancelled						
Incremental **- Delete**	CancelPurchaseOrder	Purchase Order Cancell-ation	Cancel	Delete	/CancelPurchaseOrder/DataArea/ PurchaseOrder	PurchaseOrderHeader. DocumentID.ID;
Notes: 1. Only the DocumentID is provided in the OrderHeader. 2. Application of the action code in this scenario is subject to business policies. For example, a Cancel request by way of a Delete action code may result in a “logical” deletion versus. “physical” deletion of the order.						

Data Management Technique	BOD Message	TaskID (Business Event)	Verb	Verb Action Code	Verb ActionExpression	Required Entity IDs
Business Scenario: 5. Two Purchase Orders are changed. The order quantity on an existing line in the first Purchase Order is updated. A line item on the second Purchase Order is removed.						
Snapshot	ChangePurchaseOrder	Purchase Order Change	Change	Replace	/ProcessPurchaseOrder/DataArea/ PurchaseOrder	PurchaseOrderHeader. DocumentID.ID
Notes: 1. The message contains two complete orders. 2. This is an example of a single ActionExpression applying to multiple noun instances.						
Incremental **- Update** **- Delete**	ChangePurchaseOrder	Purchase Order Change	Change	Change	/ProcessPurchaseOrder/DataArea/ PurchaseOrder/ PurchaseOrderHeader/DocumentID ="111" and PurchaseOrderLine/LineNumber ="1"	PurchaseOrderHeader. DocumentID.ID; PurchaseOrderLine LineNumber
				Delete	/ProcessPurchaseOrder/DataArea/ PurchaseOrder/ PurchaseOrderHeader/DocumentID ="222" and PurchaseOrderLine/LineNumber ="1"	
Notes: 1. Only the DocumentID is provided in the OrderHeader. 2. For the Delete, only the LineNumber is provided in the OrderLine. 3. This is an example of a multiple unique ActionExpressions applying to different noun instances; note that the ActionExpression identifies the data element instance being managed.						

880 Table 3: Application of Data Management Techniques using OAGIS BODs[16]

881

[16] Business scenarios 1 through 4 assume a single noun instance in the message instance.

Note: The above examples collectively rely on a set of Business Events that were defined at a level of granularity consistent with managing the Purchase Order as a whole: New Purchase Order, Purchase Order Change, and Purchase Order Cancellation. Finer-grain Business Event definition is possible if it is deemed desirable to manage message routing across systems at finer-levels of control (i.e., Purchase Order Line Item Change).

886

Data Management Technique	BOD Message	Verb	Verb Attributes	Verb ActionExpression (Data Type Expression)	Noun Elements (Field-Based Selection)
Business Scenario: 1. Get up to 10 purchase orders for a given customer whose order status is "Shipped".					
- Read	GetPurchaseOrder	Get	uniqueIndicator = "True" maxItems = 10 recordSetSaveIndicator = "False"	expressionLanguage = "XPath" /GetPurchaseOrder/DataArea/PurchaseOrder	PurchaseOrderHeader/CustomerParty/PartyIDs/ID = "0001" PurchaseOrderHeader/Status = "Shipped"
Notes: 1. This example uses the dynamic technique for specifying the selection criteria. 2. The query expression uses the XPath language					
Business Scenario: 2. Get up to 100 customer ids that have orders whose status is "Pending".					
- Read	GetPurchaseOrder	Get	uniqueIndicator = "True" maxItems = 100 recordSetSaveIndicator = "False"	expressionLanguage = "XPath" /GetPurchaseOrder/DataArea/PurchaseOrder/PurchaseOrderHeader/CustomerParty/PartyIDs/ID	PurchaseOrderHeader/Status = "Pending"
1. Notes:This example uses the dynamic technique for specifying the selection criteria. 2. The query expression uses the XPath language.					

887

888

Data Management Technique	BOD Message	Verb	Verb Attributes	Verb ActionExpression (Data Type Expression)	Noun Elements (Field-Based Selection)
Business Scenario: 3. Get the entire purchase order for a given purchase order.					
- Read	GetPurchaseOrder	Get		expressionLanguage = "Predefined" PurchaseOrder	PurchaseOrderHeader/ DocumentID/ID = "PO123"
Notes: 1. This example uses the predefined technique for specifying the selection criteria. 2. This example is a request for all of the information of a purchase order defined in the noun. 3. The name of the noun describing the information being selected (PurchaseOrder) is used as the reference for the predefined selection criteria.					
Business Scenario: 4. Get the summary information for a given purchase order.					
- Read	GetPurchaseOrder	Get		expressionLanguage = "Predefined" PurchaseOrderSummary	PurchaseOrderHeader/ DocumentID/ID = "PO123"
Notes: 1. This example uses the predefined technique for specifying the selection criteria. 2. This example is a request for a subset of the information of a purchase order defined in the noun, specifically the summary information of a purchase order. 3. The name of the noun in conjunction with a name describing the subset of the information being selected (PurchaseOrderSummary) is used as the reference for the predefined selection criteria.					

889

890

Data Management Technique	BOD Message	Verb	Verb Attributes	Verb ActionExpression (Data Type Expression)	Noun Elements (Field-Based Selection)
Business Scenario: 5. Get the order line information for a given purchase order.					
- Read	GetPurchaseOrder	Get		expressionLanguage = “Predefined” PurchaseOrderLine	PurchaseOrderHeader/ DocumentID/ID = “PO123”

Notes:

1. This example uses the predefined technique for specifying the selection criteria.
2. This example is a request for a subset of the information of a purchase order defined in the noun, specifically the order line information of a purchase order.
3. The name of the noun in conjunction with a name describing the subset of the information being selected (PurchaseOrderSummary) is used as the reference for the predefined selection criteria.

891

892

893

Data Management Technique	BOD Message	Verb	Verb Attributes	Verb ActionExpression (Data Type Expression)	Noun Elements (Field-Based Selection)
Business Scenario: 6. Get the order summary and order line ship to party information for a given purchase order.					
- Read	GetPurchaseOrder	Get		expressionLanguage = "Predefined" PurchaseOrderHeader	PurchaseOrderHeader/ DocumentID/ID = "PO123" PurchaseOrderLine/ShipToParty/PartyIDs/ID = "C155"
				expressionLanguage = "Predefined" PurchaseOrderLineShipToParty	

Notes:

1. This example uses the predefined technique for specifying the selection criteria.
2. This example uses two expressions.
3. The name of the noun in conjunction with the name describing the subset of the information being selected (PurchaseOrderHeader and PurchaseOrderLineShipToParty) is used as the reference for both predefined selection criteria.
4. This example is a request for a subset of the information of a purchase order defined in the noun, specifically the order header and order line ship to party information of a purchase order.

894 Table 4: Read Operations using OAGIS BODs