

MTOSI XML Implementation User Guide

Abstract

This document provides further XML structural, behavior details and usage guidelines for TMF 854. In general, items that are difficult to document in the comments of the XML are included in this supporting document.

For example, SD2-2 includes behavior descriptions and usage guidelines that:

1. require a rather long description
2. require supporting diagrams
3. appear repeatedly throughout the XML (and are easier to document in just one place, i.e., SD2-2).

This document bridges the gap between the Business Agreement and the Implementation Statement.

Table of Content

1	MTOSI XML Definitions	2
2	MTOSI XML Message Exchange Patterns (MEPs)	3
3	Top level XML Message Structure	4
4	Anatomy of an MTOSI XML Message.....	4
4.1	SOAP Header in MTOSI.....	5
4.1.1	Mandatory Header fields applicable to all MEPs	7
4.1.2	Correlation of asynchronous messages in the MTOSI message style.....	8
4.1.3	Using the MTOSI Header with respect to an MEP	9
4.2	SOAP Body in MTOSI	11
4.2.1	MTOSI Name syntax.....	11
4.2.2	Missing or empty XML elements in a response	12
4.2.3	Missing or empty XML elements in a request message (Filters).....	13
5	MTOSI Extensibility	14
5.1	Managed Object Extensions.....	14
5.2	Attribute extensions mechanism.....	14
5.2.1	Extensibility Example: Resource State	14
5.2.2	Extensibility Example: Proprietary Layer Rates	15
6	Revision History.....	18
7	Acknowledgements	18
8	How to comment on the document.....	18

1 MTOSI XML Definitions

A significant aspect of the TMF 608 MTMN/MTOSI information model is the definition of the network objects along with their containment relationships. TMF 854 has a formal definition of each network object in terms of an XML schema (XSD). These XSD definitions are the foundation of the formal arguments and the result sets of the MTOSI messages. The top level messages (both request, reply, and notification) contain a set of Network Objects (from TMF 608) as formal argument and the result set is arranged in an XML document style to accomplish a business activity. See SD2-12, Inventory Layout Description, for specific details on how the Network Object are organized into a comprehensive XML document for the getInventory() operation. The following Figure 1 illustrates the high level XML definition layers.

MTOSI XML Definition Layers

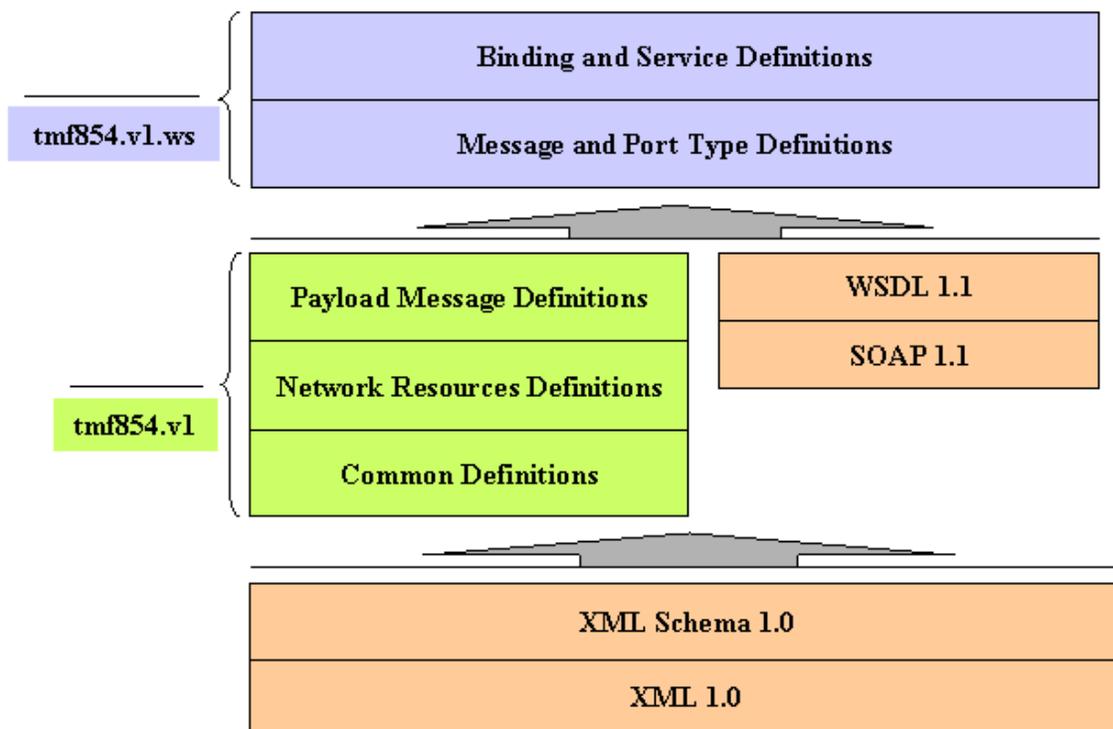


Figure 1 - MTOSI v1 XML definition layers

Since the Network Objects XSDs (defined in the Network Resource Definitions) are used in a variety of contexts, (e.g. query filter template, response set, event notification) the definition of the objects are necessarily loose (all elements are optional) and has to be intended as a skeleton to support optional information.

Furthermore, MTOSI XSDs are occasionally under constrained to limit the complexity of XSD validation and facilitate extensibility. (e.g MTOSI name sequences, TransmissionParameters, etc)

Specific business rules may apply in a specific context and have to be enforced beyond the XSD mechanisms. The following sections list a set of rules, which in conjunction of the XSD constraints guarantees interoperability and cooperation in an MTOSI deployment.

2 MTOSI XML Message Exchange Patterns (MEPs)

MTMN and MTOSI share the same information model TMF 608. This model identifies the managed objects as well as the logical operations to manipulate the objects. In MTOSI, we call the operations at this level of abstraction “Business Activities”. In MTOSI, we identified four “communication patterns” [SD2-5 Communication Styles] to accomplish all the business activities:

- Simple Response (to perform simple request-reply interactions)
- Multiple batch response (to partition and transfer large response set)
- Bulk Response (to transfer the response set through a side channel)
- Notifications (to disseminate events)

Orthogonal to the communication patterns are the communication styles: MSG (i.e., message) and RPC (i.e., remote procedure call) describing the interaction mode. Due to resource constraints in this MTOSI release, we only address the MSG communication Style.

The combination of a communication pattern with a communication style leads to a Message Exchange Pattern (MEP). The MEP fully identifies the messages and associated choreography (sequencing and cardinality) of messages involved in a business activity. Table 1 summarizes the MEP supported in this release of MTOSI.

Table 1 - MEPs supported in MTOSI V1.0

	Communication Pattern		
Communication Style	Simple Response	Multiple Batch Response	File Bulk Response
MSG (Asynch)	(ARR) Asynchronous Request/Reply	(ABR) Asynchronous Batch Response	(AFB) Asynchronous (File) Bulk

We structured the XSD set to clearly separate the abstract interface supporting the business activities from the additional information related to the particular communication pattern used to accomplish the business activity.

SD2-1, MTOSI Implementation Statement, provides traceability between the objects and operations in TMF 608 and the associated interfaces and operations in TMF 854.

For example: `getInventory` can be traced both in the IA (as business definition) and IS (as specific XML implementation)

TMF 608 – Information Agreement (Bulk Inventory Retrieval Operations section)

```
void getInventory (InventoryFilter filter= empty, CIS GeneralizedTime dateAndTime=empty, CIS SequenceOf Structs::AVP whereAndHowToReplay = empty list), in Class OS
```

Description: This operation returns a specified portion of the inventory data from the OS to which the request is directed (i.e., the target OS). The target OS returns all objects satisfying the scope and filter constraints of the requesting OS that have a modification date equal to or later than the Diff Date and Time.

SD2-1 - Implementation Statement (Configuration Service; InventoryRetrieval section)

Operation	Support	Exception/ Error Reason	Comments
getInventory			ABR, AFB

3 Top level XML Message Structure

The MTOSI abstract interface specifies the syntax and semantics of a service. At this technology neutral level a business activity is accomplished by the exchanges of MTOSI SOAP compliant messages. MTOSI adopts the SOAP envelope as external wrapper and structure for the MTOSI messages. The following figure shows a typical MTOSI message inside a SOAP envelope.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
    ----- MTOSI header goes here -----
  </soap:Header>
  <soap:Body>
    ----- MTOSI message body goes here -----
  </soap:Body>
</soap:Envelope>
```

Refer to <http://www.w3.org/2003/05/soap-envelope> for a complete definition of the SOAP XSD.

While SOAP addresses the envelope of a message, MTOSI specifies the namespace and root elements of the MTOSI specific Header and Body.

- MTOSI namespace is “tmf854.v1” for the initial release V1.0.
- MTOSI header root element is <Header> in the tmf854.v1 namespace.
- MTOSI body root element is the name of the message (SOAP Document/literal wrapped style).

MTOSI messages, while compliant to the SOAP standard in terms of structure, do not necessarily require a SOAP stack to be processed. Beside the SOAP envelope, MTOSI does not require any other SOAP specific constraints or functionalities to be implemented by the service provider nor service consumer.

4 Anatomy of an MTOSI XML Message

The purpose of MTOSI is to identify and standardize Business Activities in terms of syntax (XSD) and behavior (WSDL + MEP) in order to achieve interoperability. Any Business Activity is fulfilled by the exchange of XML messages. An interface groups together similar business activity messages.

E.g., getTP business activity in the managedElementMgr interface has:

- getTP, the request message
- getTPResponse, the response message

All interface messages are XML instance documents with associated XSDs (XML Schema Definition) for validation.

Each MTOSI message is composed of a header and a body combined together with a SOAP [SOA] envelope. See following example:

```

<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope" xmlns="tmf854.v1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="tmf854.v1
../xsd/common/header.xsd tmf854.v1 ../xsd/interfaces/InventoryRetrieval.xsd http://www.w3.org/2003/05/soap-
envelope http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
    <header tmf854Version="1.1">
      <activityName>getInventory</activityName>
      <msgName>getInventory</msgName>
      <msgType>REQUEST</msgType>
      <senderURI>/MTOSI/InventoryOS</senderURI>
      <destinationURI>/MTOSI/EMS01</destinationURI>
      <correlationId>0001</correlationId>
      <communicationPattern>MultipleBatchResponse</communicationPattern>
      <communicationStyle>MSG</communicationStyle>
      <requestedBatchSize>0</requestedBatchSize>
      <timestamp>20051004140305</timestamp>
    </header>
  </soap:Header>
  <soap:Body>
    <getInventory tmf854Version="1.1">
      <filter></filter>
    </getInventory>
  </soap:Body>
</soap:Envelope>

```

While the body carries all the information related to the business activity (e.g., the name of the associated operation and the filter, in the case of the getInventory operation), the header is responsible for the additional information related to the particular MEP used in the interaction (e.g., correlation ID, requestedBatchSize, etc). Refer to the SD2-1, MTOSI Implementation Statement, for the exact position in the envelope and definition of the formal argument of each operation.

The next section will summarize the Header element and their use in the message.

4.1 SOAP Header in MTOSI

The following Table 2 summarizes the field included in the MTOSI header.

Header field is the tag name in the XML, O=optional, M=required (mandatory), NA=not applicable and not used in the header.

Table 2 - MTOSI Header fields

Header field	Req	Resp	Notif	Description
activityName	M	M	M	Identifies the name of the business transaction activity for the message being exchanged. The value is the name of the TMF 608 operation, e.g. getInventory. This value corresponds to the "operation" in the WSDL.
msgName	M	M	M	Identifies the name of the message (or contract) that is being exchanged as part of an operation. E.g. getInventoryResponse. This field corresponds to the message name in the WSDL.

Header field	Req	Resp	Notif	Description
msgType	M	M	M	Identifies the type of the message. One of: REQUEST, RESPONSE, NOTIFICATION, ERROR.
senderURI	M	M	M	Identifies the application sending the message.
destinationURI	M	M	M	Used to identify the <u>final</u> destination of the message (the OS that will process the content of the message). This field may point to a logical end point or abstract topic name to be resolved by the communication transport middleware in order to deliver the message.
originatorURI	O	O	O	Identifies the originator of the Business Activity.
replyToURI	O	NA	NA	Used by request messages to specify destination for response message, if not specified the SenderURI endpoint is used to send back the response.
failureToURI	O	NA	NA	Identifies the application receiving the potential error notification message. Required for all response messages and notifications which are triggered by request/response. If not specified the replyToURI is used. If the replyToURI is also not specified, senderURI endpoint is used to send back the failure response.
activityStatus	NA	M	NA	Specifies the high-level response status for an activity. Required for response messages, including error responses. One of: SUCCESS, FAILURE, WARNING
correlationId	O*	O*	NA	This field may be set by the originator of an asynchronous request that will allow it to correlate the response to the request. If this field is set, it's value must be reflected in the header of the response message. *=-mandatory for MSG style communication.
security	O	O	O	Contains credential information used to secure message processing.
securityType	O	O	O	Identifies the type of credential in the security element.
priority	O	O	O	Indicates message-handling priority for messages. It must be in the range 0-9 (lowest-highest). Default: 4
msgSpecificProperties	O	O	O	Conditionally required for request, response, and notification messages as identified by the documentation for a specific interface message. The communications infrastructure or the receiving application can use this value for routing or filtering messages.
communicationPattern	M	M	M	Communication Pattern - SimpleResponse, MultipleBatchResponse, BulkResponse, Notification
communicationStyle	M	M	M	Communication Style: RPC, MSG

Header field	Req	Resp	Notif	Description
requestedBatchSize	M*	O*	NA	logical size of the batch for a mutli-response communication pattern. (can be set to 0 to imply a single response). *NA for SimpleResponse and Notification comm. pattern
batchSequenceNumber	NA	M*	NA	Used in a multiple response Comm. Pattern to identify the batch seq number in a sequence. *NA for SimpleResponse and Notification comm.. pattern
batchSequenceEndOfReply	NA	M*	NA	Used in a multiple response Comm. Pattern, true if it is the last result batch in a sequence. *NA for SimpleResponse and Notification comm. pattern
fileLocationURI	M*	O*	NA	Used for file retrieval. Specify the base name of the file(s) to be generated and the remote destination. *Mandatory in the BulkResponse comm. Pattern, NA otherwise.
compressionType	M*	O*	NA	Used for file retrieval. Specify if compression is to be performed. *Mandatory in the BulkResponse comm. Pattern, NA otherwise.
packingType	M*	O*	NA	Used for file retrieval. Specify if the output file(s) are to be packed. *Mandatory in the BulkResponse comm. Pattern, NA otherwise.
timeStamp	O	O	O	Used to reflect the time when the message was created.
vendorExtensions	O	O	O	Additional vendor specific information.

The MTOSI Header contains generic meta information related to the MTOSI body (business payload) as well as the data related to the specific communication pattern and communication style on the implemented MEP. We will refer to these latter elements (shaded as green in Table 2) as MEP elements in the rest of this document.

4.1.1 Mandatory Header fields applicable to all MEPs

Independently from the MEP here is the list of Header element mandatory in the MTOSI messages:

Mandatory in Both request response and notification:

- **activityName** - Identifies the name of the *business transaction activity* [SD2-5] (a.k.a. operation in the WSDL) for the message being exchanged. (e.g. getInventory)
- **msgName** - Identifies the name of the message that is being exchanged as part of an operation. E.g. getInventoryResponse.
- **msgType** - Identifies the type of the message. One of: REQUEST, RESPONSE, NOTIFICATION, ERROR.

- **senderURI** - Identifies the application sending the message. (actual value depends on the specific transport adopted in the MTOSI deployment: e.g JNDI name for JMS transport)
- **destinationURI** - Identifies the final destination of the message (actual value depends on the specific transport adopted in the MTOSI deployment: e.g JNDI name for JMS transport)
- **correlationID** –see next section for detailed description.

Mandatory only in the response:

Each response message will have to specify the response status of the operation in the **activityStatus** header element. The possible values are: SUCCESS, FAILURE, WARNING

4.1.2 Correlation of asynchronous messages in the MTOSI message style

MTOSI uses the header field correlationID to correlate messages belonging to the same business activity in the context of a communication pattern instance (e.g. all the responses belonging to a getInventory request instance) See Figure 2.

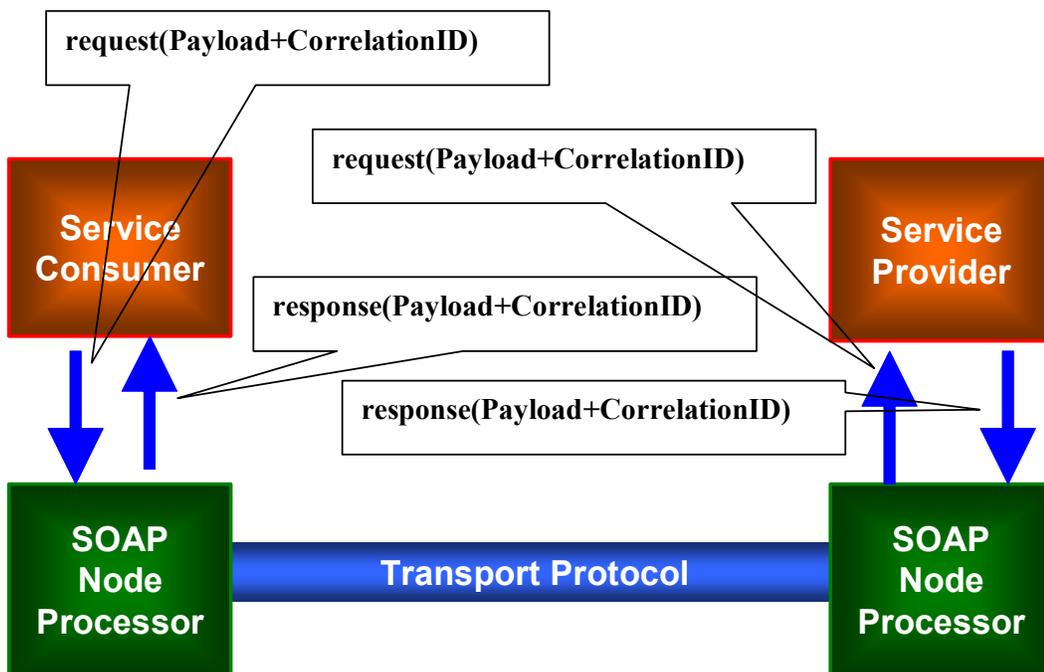


Figure 2 - use of CorrelationID in MTOSI

MTOSI mandates that the response messages related to a request MUST have the same correlationID value of the initial request. Figure 3 and illustrate the detailed behavior of the correlation mechanism.

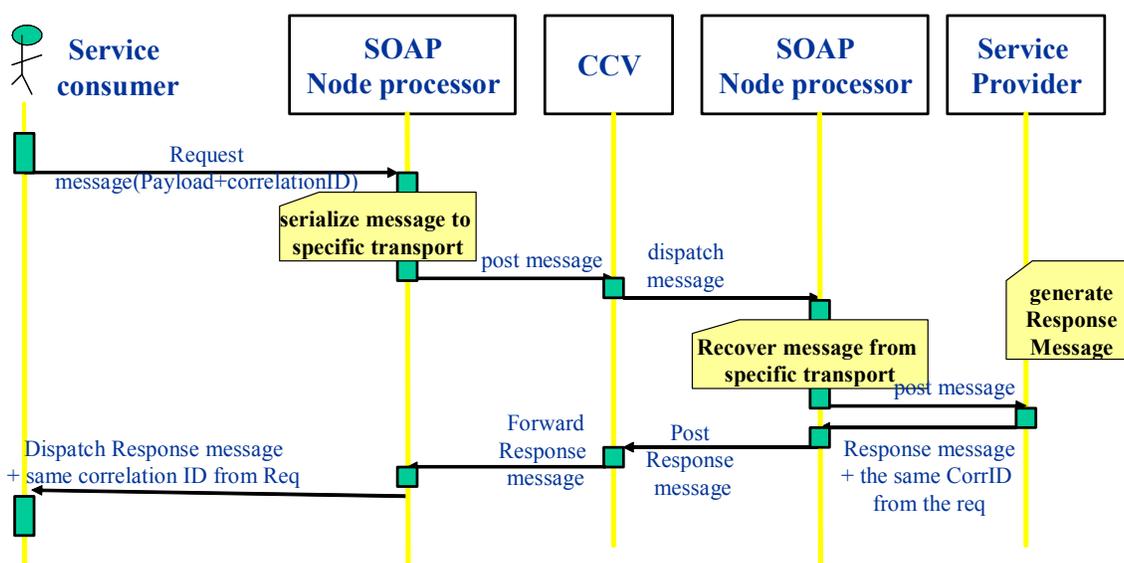


Figure 3 – MTOSI CorrelationID mechanism

While the MTOSI correlationID can be used at the application level to correlate and dispatch a message, other orthogonal correlation mechanisms can be adopted at different abstraction levels. For instance, in JMS MTOSI suggests the adoption of the fields JMS message ID and JMS Correlation ID to perform the correlation at the transport level. See SD2-9, Using JMS as an MTOSI Transport, for the details on how the MTOSI correlationID is mapped into the JMS transport stack.

4.1.3 Using the MTOSI Header with respect to an MEP

Given that in MTOSI V1.0 we only consider the Message communication style (RPC may be considered in phase 2), we have four MEPs available for the business activities:

- Asynchronous Request/Reply (ARR)
- Asynchronous Batch Response (ABR)
- Asynchronous (File) Bulk (AFB)
- Notification

The following subsections will describe the detailed behavior of each of the supported MEPs. See the SD2-1, MTOSI Implementation Statement, for a detailed list of business activities and supported MEPs.

4.1.3.1 Asynchronous Request/Reply (ARR)

This is the simplest pattern and can be seen as a degenerate case of the ABR when there is only a single response.

Mandatory MEP elements:

- **communicationPattern = SimpleResponse**
- **communicationStyle = MSG**
- **correlationID** = the field has to exist, but the actual value may be provided by the underlined transport in the implicit modality.

4.1.3.2 Asynchronous Batch Response (ABR)

This MEP allows fragmentation of the response into several logical units.

Mandatory MEP elements in both request and responses:

- **communicationPattern = MultipleBatchResponse**
- **communicationStyle = MSG**
- **correlationID** = the field has to exist, but the actual value may be provided by the underlined transport in the implicit modality

Mandatory MEP elements in request:

- **requestedBatchSize** logical size of the batch for the result fragments. For example in the getInventory it is the maximum number of Inventory top level objects to be included in an inventory XML file (batch). See TMF 517 for details on how the result set is partitioned. If set to 0 (zero) will imply a single response.

Mandatory MEP elements in responses:

- **batchSequenceNumber** - Used in a multiple response Comm. Pattern to identify the batch sequence number in a sequence. (we start from 0)
- **batchSequenceEndOfReply** - Used in a multiple response Comm. Pattern, true if it is the last result batch in a sequence.

4.1.3.3 Asynchronous (File) Bulk (AFB)

This MEP allows transferring the entire response set through a side channel (secondary to the main CCV transport). E.g. this MEP allows sending a result set using an FTP stack.

Mandatory MEP elements in both request and responses:

- **communicationPattern = BulkResponse**
- **communicationStyle=MSG**
- **correlationID** = this is used only in the request and responses (acknowledge and notifications) controlling the FTP transfer.

Mandatory MEP elements in request:

- **requestedBatchSize** logical size of the batch for the transfer fragments. For example in the getInventory it is the maximum number of Inventory top level objects to be included in an inventory XML file (batch). See TMF 517 for details on how the result set is partitioned. If set to 0 (zero) will imply a single response.
- **fileLocationURI** - URI provided by the requesting OS indicating the rootname of the file(s) to be produced and location of where to place the retrieved XML inventory file(s). See RFC 3986 for details on URI syntax
- **compressionType**- The type of compression to apply to the generated file(s). MTOSI v1.0 will define the following two options; NO_COMPRESSION, GZIP. Default behavior (when request parameter is omitted) is NO_COMPRESSION. Implementation of this file processing instruction by the Target OS is optional, and any incompatible request shall be handled with the appropriate exception. Vendor extension of this attribute shall be permitted

- **packingType** - The type of packing to apply to all the inventory file(s) generated from the same request. MTOSI v1.0 will define the following three options; NO_PACKING, ZIP, TAR. Default behavior (when request parameter is omitted) is NO_PACKING. Implementation of this file processing instruction by the Target OS is optional, and any incompatible request shall be handled with the appropriate exception. Vendor extension of this attribute shall be permitted

Mandatory MEP elements in response:

The XML response to an AFB request is an acknowledge of the FTP transfer and optionally a set of intermediate notification regarding the state of the FTP transfer (see use case in TMF 517).

The header element **activityStatus** should be used (values: SUCCESS, FAILURE, WARNING) in the acknowledge to notify the service consumer of the result of the activity.

No additional elements are necessary in the header.

4.1.3.4 Notification

MTOSI Notifications are based on a minimal subset of functionality and interfaces in the WS-Notification specification. See SD2-8, MTOSI Notification Service, for complete details. A Notification Producer will simply call the Notification Broker issuing a notification. The MTOSI Notification consumer will have to first subscribe to the interested MTOSI topic by calling the Notification Broker and than will be later notified when event matching the selector criteria are posted on the interest topic. As is, the MTOSI Notification can be implemented in any transport as long as the interface is compliant with the subset of the WS notification interfaces used in MTOSI. In particular SD2-9 Using JMS as an MTOSI Transport has a complete description of the JMS bindings and mechanisms for the JMS stack.

Beside the common mandatory header element described in 4.1.1 no other MEP element are necessary in the notification MEP.

4.2 SOAP Body in MTOSI

In MTOSI, we adopted the SOAP Document/literal wrapped style. In detail this means that the first element in the MTOSI body has to be the operation name (message name in MTOSI) and the rest of the body is a plain XML structure as defined in the XSD set. Note that the header contains the name of the message carried in the body. A SOAP-compatible envelope is provided in this MTOSI release to allow a designer/developer to assemble and validate MTOSI messages in isolation. Furthermore, while it is expected that an MTOSI deployment will take full advantage of the WS facilities such as SOAP parser, dispatchers and validators, nevertheless, MTOSI does not mandate nor require supporting a complete SOAP stack. In this later scenario, the WS framework will directly assemble the MTOSI header and body into a SOAP envelope.

We refer to the MTOSI body message as top level message and top level message type for the related XSD defining the message.

4.2.1 MTOSI Name syntax

MTOSI names are defined according to the network object containment hierarchy (see SD2-7, MTOSI Object Naming). In XML, MTOSI names are defined as a sequence of name components, where each component is the relative name with respect to the containing element. The following example illustrates an (Absolute) MTOSI name:

```

<name>
  <mdNm>CompanyX/EMSY</mdNm>
  <meNm>OLT1</meNm>
  <ehNm>/shelf=2/slot=3</ehNm>
  <eqNm>1</eqNm>
</name>

```

The XSD for an MTOSI Name will enforce the correct sequence of name components in a name but will not be able to prevent a wrong name component from being used. The XSD will also not be able to enforce the content of a name component. Names should be conformant to SD2-7 and enforced by the MTOSI vendor through the means of business rules or application code.

4.2.2 Missing or empty XML elements in a response

When an attribute is requested via the MTOSI, there are several possibilities concerning the availability of the value for the attribute. In the following table, we should the various attribute value availability possibilities and the associated behavior by the Target OS (i.e., the OS to which the attribute retrieval request is made. It is critical to keep in mind that MTOSI Release 1 does not allow for the retrieval of object attributes in isolation, i.e., the entire object needs to be retrieved and the Requesting OS can not specify a subset of an object's attributes for retrieval.

Attribute Availability	Expected Behavior by the Target OS	XML representation	Comment
1. The value of the requested attribute is known to the Target OS. And is not empty	The Target OS returns the attribute value.	<TAG>value here</TAG> <TAG>23</TAG>	If the attribute is a string and it is known that the string is intended to be empty, the Target OS needs to indicate this to the Requesting OS.
2. The value of the requested attribute is known to the Target OS. And the value is empty.	This is only applicable to Strings. The Target OS returns the attribute value. Example name alias <nameAlias/>	If Type is String: <TAG></TAG> Or <TAG/> We cannot use this for any types other than string	If the attribute is a string and it is known that the string is intended to be empty, the Target OS needs to indicate this to the Requesting OS.
3. This attribute is supported by the Target OS, but the current value is not known to the Target OS.	The Target OS needs to inform the Requesting OS that the attribute is supported but its value is currently unknown. In enumerations we use the value UNKNOWN	Valid for String, Numeral and Booleans: <TAG nil=true></TAG> Or <TAG nil=true />	In the case of strings, it is important to distinguish this case from the case where the string is intentionally set to be empty.
4. The value of the requested attribute is known to the Target OS, but the value is suspect.	The Target OS returns the attribute value with an indication that the value is suspect.	We use the quality indicator in the inventory Layout	

5. The attribute is not supported by the Target OS.	The Target OS needs to inform the Requesting OS that the attribute is not supported by not including it in the response.	Tag is omitted from the XML	
6. The attribute is not understood by the Target OS.	-	-	This case cannot happen (not even between major MTOSI releases) since the Requesting OS cannot ask for specific attributes with regard to a give object type. The Target OS will always be working from its understand of the object type definition and associated attributes.

In a request message a missing tag in a request can be in general interpreted as a relaxation of the request constraint. The next chapter addresses the partial specification of a filter in the context of a request message.

4.2.3 Missing or empty XML elements in a request message (Filters)

A filter in MTOSI is used to restrict a data retrieval query: e.g. `getInventory`, `getActiveAlarms`

All the XML components of a filter are to be intended to joined by a logical AND. The server OS must compute the intersection of the various filter predicates. Not specifying optional elements in the filter is to be intended as relaxing a constraint on that element (allowing all the values to be matched). E.g.

```
<getActiveAlarms tmf854Version="1.0">
  <filter>
    <acknowledgeIndication>AI_EVENT_ACKNOWLEDGED</acknowledgeIndication>
    <perceivedSeverityList>
      <perceivedSeverity>PS_CLEARED</perceivedSeverity>
    </perceivedSeverityList>
  </filter>
</getActiveAlarms>
```

This filter will retrieve all activeAlarms matching all alarms with `acknowledgeIndication=AI_EVENT_ACKNOWLEDGED` AND `perceivedSeverity= PS_CLEARED`

Removing the `perceivedSeverityList` from the filter will cause the query to return alarm matching any kind of `perceivedSeverity`.

The same mechanism can be generalized and extended to the other elements of a request message that concur to the definition of a scope of a result set. For example the `diffDateAndTime` element in the `getInventory()` operation if not specified will be interpreted by the service provider as "retrieve all the known NE regardless of the change date".

5 MTOSI Extensibility

MTOSI can be extended by the vendor (or by the MTOSI architects) to tailor the solution to a specific context.

There are three main mechanism to extend the XSDs:

- Managed Object Extensions
- Attribute Extensions
- Vendor Object and Vendor Notifications

5.1 Managed Object Extensions

All Managed Object as well as the most critical XML structure have a specific placeholder (VendorExtensions) to allow a vendor to extend a message and still enforce validation on the vendor extension while maintaining backward and forward compatibility with the rest of the MTOSI release on the same major number. Refer to SD2-6 for a detailed description of the VendorExtension structure and examples.

example of XML with vendor extensions

```
<?xml version="1.0" encoding="UTF-8"?>
<Equipment_T xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="..\schema\MTOSI\V1\Common\Equipment_T.xsd">
  <name>
    <mdNm>md1</mdNm>
    <meNm>eq1</meNm>
    <ehNm>eh1</ehNm>
    <eqNm>eq1</eqNm>
  </name>
  <vendorExtensions extAuthor="Acme" tmf854version="1.0" extVersion="2.0">
    <CommonLanguageName>12123434355trfd</CommonLanguageName>
  </vendorExtensions>
</Equipment_T>
```

5.2 Attribute extensions mechanism

All the critical MTOSI attributes can be extended by the vendor by leveraging on the extensibility placeholders in the Managed Element structure. For example, some of the simple MTOSI elements are identified extensible and can be extended by the vendor by simply introducing new values prefixed by PROP_.

The forms of manageable extensions are:

- **Open** - specific values for this attribute are not specified
- **Closed** - a discrete set of possible values for this attribute are defined
- **Qualifiable** - this attribute may need further qualification by the vendor
- **Extendable** - the value set for this attribute can be extended by a vendor
- **Overlap** - the value set for this attribute can be extended by a vendor.

See SD2-6 Versioning and Extensibility for the details and SD2-13 for a map of what is extendable.

5.2.1 Extensibility Example: Resource State

SD2-13 defines the following forms of attribute extensibility in the Overview sheet: Qualifiable, Extendable-Vendor, Extendable-MTOSI and Overlap. We use the resourceState attribute to provide examples for these four types of attribute extensibility.

First, we note that the resourceState has the following possible values:

- planned (without any supporting network resources)

- installed (supported by network resources)
- retired (no more supported by network resources)
- unknown is available for resources when other values are not applicable.

As that a service provider would like to extend these values as follows:

1. The state “retired” is further qualified as retired-removed and retired-storedAway.
2. New states “present” and “notPresent” are used instead of “planned” and “installed” but the semantics of the values are not the same. For example, let’s say the present means the equipment is in the service provider’s possession and notPresent has the opposite meaning.
3. A new state “sold” is used to indicate the equipment has been sold. For the sake of argument, assume that the meaning of “sold” does not overlap any of the other MTOSI-defined values for resourceState.

The “retired” state is extended via qualification. The resourceStateQualifier would be set to retired-removed or retired-storedAway whenever the resourceState has the value “retired”.

The “planned” and “installed” states are extended (actually overlapped) by the present and notPresent states. In such cases, the resourceState would typically be populated with the value “unknown” and the resourceStateOverlap attribute would be set to present or notPresent.

The new state “sold” is used as a value of resourceState. The vendor uses the prefix PROP_ to denote that the value is vendor-defined. In particular, the value PROP_sold would be used. Now if the vendor took their new state to the MTOSI team and the team approved, the set of valid states for resourceState would be expanded. This can happen in one of two ways:

1. The MTOSI team may only want to make a minor update (backward compatible). In this case the new value would be listed as MINOR_sold.
2. On the other hand, the MTOSI team may decide to make a major update (not backward compatible). In this case, the new value would be just “sold”.

It should be noted that the difference between the two above mentioned options is that while in the first case, there is no “type checking” on the value, i.e., MINOR_xxx could be anything and message processor would pass it, in the second case, there is type checking and a valid value must be used or a validation exception will be raised.

5.2.2 Extensibility Example: Proprietary Layer Rates

This section show a particular example of a vendor extensible element: layerRates

It is expected that this will be the required behavior if and when the vendor decides to support their own layer rates.

The MTNM v3.0 model uses integers to represent layerRates. For MTNM, proprietary layer rates (i.e., vendor-specific layer rates) are assigned by the vendor using values greater than or equal to 10,000 (this is explained further in TMF 814A v3.0). For MTOSI, it was agreed to use a string instead of an integer to represent each layerRate. In particular, it was decided to use the full name of the layerRate. For example, MTOSI will use LR_Line_OC768_STS768_and_MS_STM256 instead of the integer “91” whereas MTNM just passes the value “91” to represent the previously mentioned layerRate. For MTOSI, proprietary layerRates will not use the “LR_” prefix, but rather “PROP_”. All other prefixes are illegal.

5.3 Vendor Object and Vendor Notifications

MTOSI allows a vendor to introduce additional objects and notification to the messages. In order to maintain backward and forward compatibility in a minor version, the vendor objects and vendor notification must obey to the specification detailed in SD2-6 MTOSI Versioning and Extensibility.

References

- [SOA] SOAP Version 1.2, W3C Recommendation 24 June 2003, <http://www.w3.org/2000/xml/Group/>
- [TMF517] MTOSI Business Agreement
- [TMF608] MTNM Information Agreement
- [TMF854] MTOSI XML Solution Set
- [SD2-1] MTOSI Implementation Statement
- [SD2-5] Communication Styles
- [SD2-6] Versioning and Extensibility
- [SD2-7] MTOSI Object Naming
- [SD2-8] MTOSI Notification Service
- [SD2-9] Using JMS as an MTOSI Transport
- [SD2-11] MTOSI IA-SS Mapping
- [SD2-12] Inventory Layout Description
- [SD2-13] Attribute Extensibility

6 Revision History

Version	Date	Description of Change
1.0	May 2005	This is the first version and as such, there are no changes to report.

7 Acknowledgements

<FirstName>	<LastName>	<Company>
Michel	Besson	Cramer
Francesco	Caruso	Telcordia Technologies Inc.
Shlomo	Cwang	TTI Telecom
Steve	Fratini	Telcordia Technologies Inc.
Elisabetta	Gardelli	Siemens
Jérôme	Magnet	Nortel Networks

8 How to comment on the document

Comments and requests for information must be in written form and addressed to the contact identified below:

Francesco Caruso Telcordia Technologies Inc.
Phone: +1 732 699 3072
Fax: +1 732 699 7015
e-mail: caruso@research.telcordia.com

Please be specific, since your comments will be dealt with by the team evaluating numerous inputs and trying to produce a single text. Thus we appreciate significant specific input. We are looking for more input than "wordsmith" items, however editing and structural help are greatly appreciated where better clarity is the result.