The National Animal Identification System (NAIS)

Animal Trace Processing System

Version 1.0
ATD Technical Specification

Document Version 2.2
January 9, 2007

USDA

ATPS Version 1.0 Technical Specifications

Document History

Version	Description	Date	Author
1.0	Initial version – only web service specs documented	10/5/2006	Slush
1.1	Modifications based on suggestions by Elliott R_	10/10/2006	Slush
1.2	Modifications based on comments by Scott M_	10/11/2006	Slush
1.3	Modifications based on comments by Nigel H_	10/12/2006	Slush
1.4	Modifications based on comments by Scott Q_	10/25/2006	Slush
1.5	Added section on Security	10/26/2006	Slush
1.6	Modifications based on 10-26 conference call	10/27/2006	Slush
1.7	Exception class modifications	11/14/2006	Slush
1.8	Modifications based on 11-21 conference call	11/24/2006	Slush
1.9	Added specifications for validate prem, AIN web services	11/27/2006	Slush
2.0	Added detail to web application services	11/27/2006	Slush
2.1	Reformatted and edited	12/29/2006	GJMoore
2.2	Modification to 2.1.2.3 and ATPSMessageValidationResultWS object	1/9/2007	Slush

1 About ATPS

The Animal Trace Processing System (ATPS) is the application that satisfies NAIS phase 3 requirements. NAIS Phase 3 requirements specify the ability for the USDA to provide a complete trace back for a diseased or suspected animal in 48 hours or less. ATPS enables this requirement by providing a web-based application that allows clients who store and maintain animal trace data (Animal Trace Database providers or ATDs) to provide this information to the USDA as it is needed. Client systems include any public or private system that stores animal movement, sighting, or event data.

This document details the technical interface requirements and specifications for ATPS and all client applications that communicate with ATPS.

The ATPS client applications are known collectively and individually as Animal Trace Databases (ATD).

ATPS is a J2EE application. Every ATD communicates with ATPS via web services. ATPS also provides a web application interface that allows an ATD User to manage their Account, run reports, and manually submit responses to ATPS.

2 Service Specifications

This section describes detailed specifications for all of the services ATPS provides to enable full integration with an ATD.

A Service is defined as a function that ATPS provides to accomplish a use case. This requirements document covers both Web Services and Web Application Services.

2.1 Web Services

Functions that are accessed via web service APIs only.

ATPS provides two web services:

- Get Requests web service
- Submit Response web service.

Each ATD will call the Get Requests web service both to receive new requests from ATPS, and to check on the status of old requests.

The Get Requests web service is a synchronous web service and does not rely on message queues to deliver information.

The ATD will call the Submit Response web service to respond to outstanding requests from ATPS.

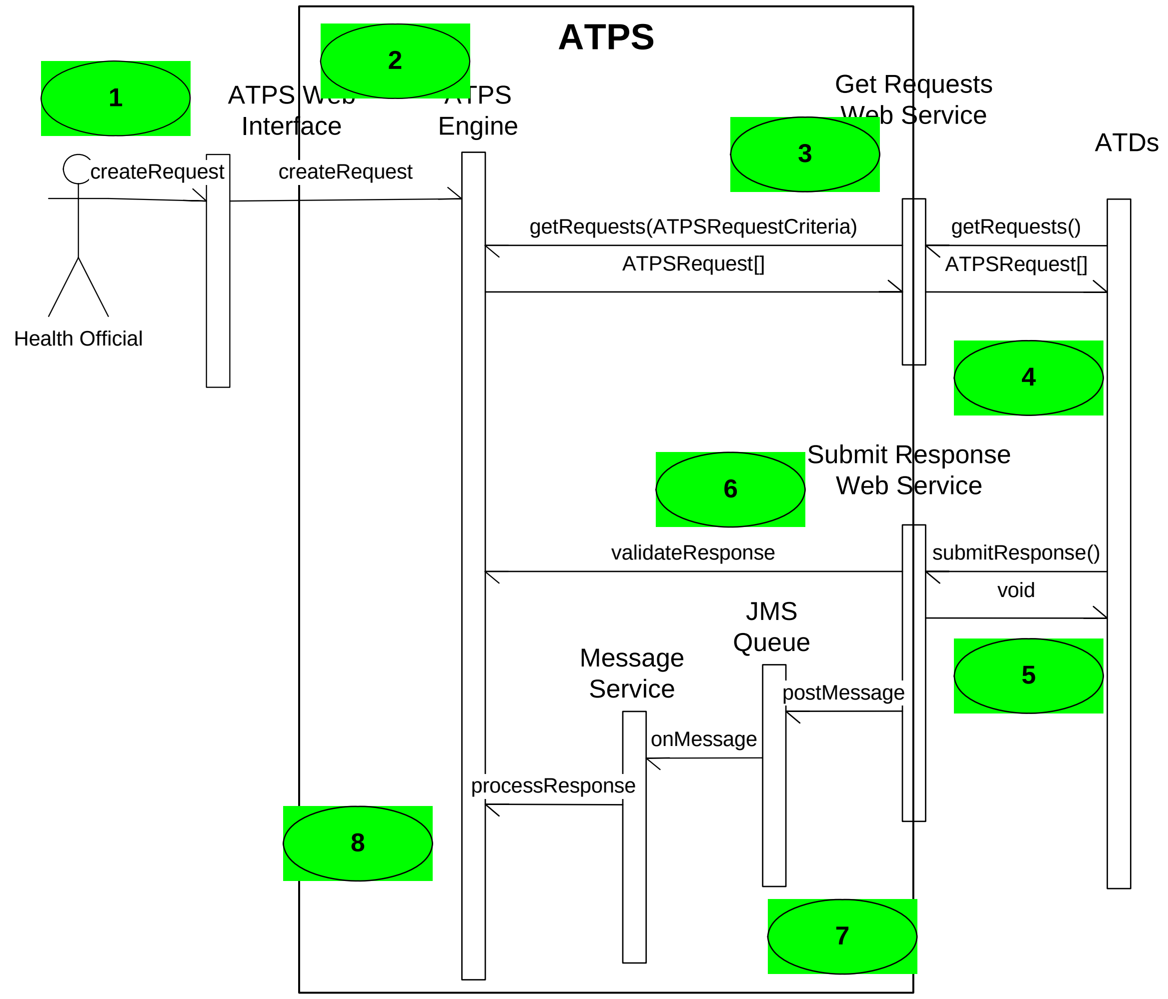
The Submit Response web service is backed by an asynchronous message queue. Therefore the ATD will not know if the message was successfully processed until they call the Get Requests web service and retrieve the original request with an updated status. Some fatal exceptions will be thrown before the response is sent to the message queue so the ATD will get immediate feedback if possible.

2.1.1 Basic Web Service Use Case

More detailed Use Cases will be located in the Use Case document. However it is instructive to outline the basic “happy path” web services Use Case when detailing the web services requirements, in order to provide a bigger context to the service requirements.

The basic “happy path” web service request/response Use Case is as follows:

1. A Heath Official creates a Request. (Note that this may also be done by ATPS itself.)
2. ATPS generates and stores a NEW request.
3. The ATD calls the Get Requests web service and retrieves the request. (Request moves to RETRIEVED.)
4. The ATD processes the request and builds a Response.
5. The ATD calls the Submit Response web service to respond to the Request.
6. ATPS validates the Response. (Request moves to RESPONDED.)
7. ATPS posts the Response to the Message Queue.
8. ATPS processes the Response. (Request moved to VALIDATED.)


ATPS
1
2
3
4
5
6
7
8
ATPS Web Interface
ATPS Engine
Get Requests Web Service
ATDs
Health Official
createRequest
createRequest
getRequests(ATPSRequestCriteria)
ATPSRequest[]
getRequests()
ATPSRequest[]
Submit Response Web Service
validateResponse
submitResponse()
void
JMS Queue
Message Service
postMessage
onMessage
processResponse

2.1.2 Get Requests Web Service

ATPS provides a service that allows an ATD to get a Request from ATPS.

A Request is how ATPS asks an ATD for information.

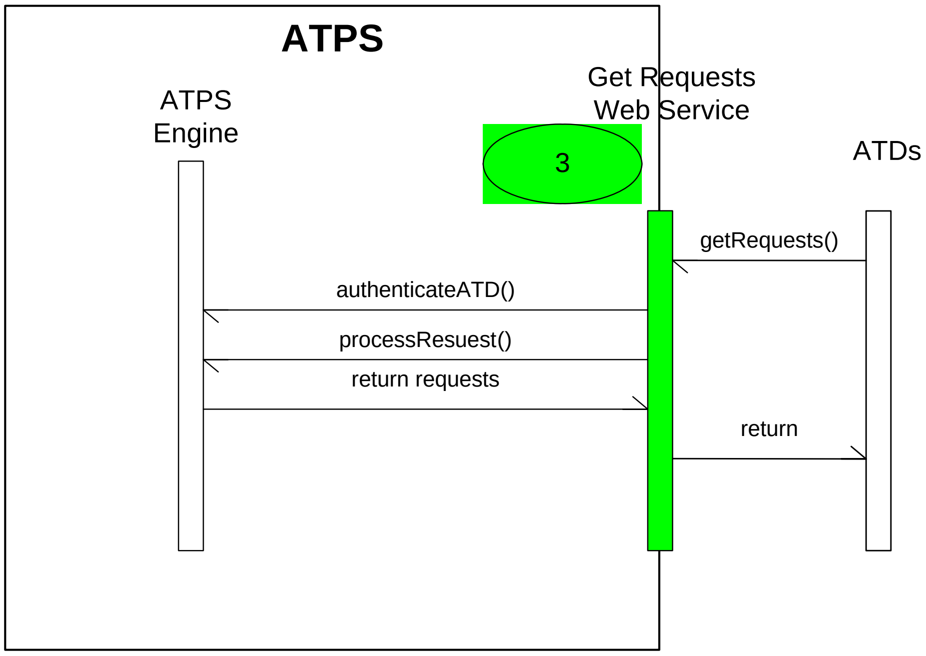
A Request is also how an ATD will get status on the progress of a Request to which they may have responded.

An ATD can Retrieve Requests from ATPS based on Request Status, Request ID, Case ID, Request Date, and other attributes.

ATPS Requests have a Status. The Request Status indicates if the Request has been received by the ATD, if it has been responded to, and if it has been successfully processed.

Each Request Status has a Category. The Status Category indicates if the Request needs to be responded to (ACTIVE), if it may be responded to (ACTIONABLE), or if it can not be responded to (STATIC).

The Get Requests web service fulfills step 3 on the basic web service use case:



- The ATD calls the Get Requests web service, passing in a criteria object describing the requests of interest.
- ATPS authenticates the ATD.
- ATPS processes the request synchronously, and returns 0-to-many Request objects that fulfill the criteria passed in.
- NEW requests are moved to a state of RETRIEVED.

2.1.2.1 Basic Requirements

ATPS does not push requests directly to the ATD. The ATD must call a web service on ATPS to retrieve requests.

All requests are specific to a particular ATD. As such, ATPS will always return only requests specific to the ATD making the service call.

ATPS recommends that every ATD check for NEW requests every 5 minutes.

An ATD may increase the frequency of request checking during an event, or technically when ATPS has an open case that is not a ping Case.

Every ATD is required to respond to all NEW Status Requests within 15 minutes of receipt.

Even if the ATD does not have any events that match the Request, the ATD is required to respond to the Request.

Every Request starts out with a Status of NEW, which is an ACTIVE Category.

When the ATD retrieves a NEW Request, the Request Status is changed to RETRIEVED. A RETRIEVED Request is still in an ACTIVE status category.

ATPS will never return a NEW request more than once.

If the ATD responds to a Request and ATPS can not process the Response, the Request Status will become ERROR. An ERROR Request is ACTIVE and must be responded to.

If the ATD responds to a Request, and ATPS can process the Response but there are data validation errors, the Request Status will become VALIDATION_ERROR. This is an ACTIONABLE Status and the ATD may respond to the Request again, but it is not required to do so.

When a Case is Closed, ATPS will send a Case Closed Request with no real request parameters for the Case to every ATD.

2.1.2.2 Request Types

ATPS will generate four different types of requests.

- Official ID Request
- Premises Request
- Ping Request
- Case Closed Request.

The ATD is required to respond to Official ID, Premises, and Ping requests. Case Closed “requests” are provided as a convenience to the ATD. This section describes the different request types.

2.1.2.2.1 Official ID Request

The Official ID Request is a request for all events pertaining to an Official ID or set of Official IDs. An Official ID is defined as a unique animal identifier or group identifier, and it may not be strictly numeric. ATPS may ask for information on up to 1,000 Official IDs in the same request. Upon receipt of such a request, the ATD is responsible for returning all events related to every Official ID in the request. Please refer to the appendix for the actual ATPS Request object definition.

Usage Rules:

If the Official ID array is populated with values, the ATD will return all Events for all Official IDs in the array.

ATPS will not populate both the array of National Premises IDs and the array of Official IDs.

ATPS will not ask for more than 1,000 Official IDs in the same request.

ATPS will not populate the Begin Request Date or the End Request Date if the Official ID array is populated. Unless the audit dates are populated, ATPS will always want all events for all Official IDs in the request.

If the Begin Audit Date is populated, ATPS only wants events for the Official IDs in the array if they have been added or modified in the ATD on or since the Audit Date. Otherwise ATPS always wants all Events for all Official IDs in the array.

The ATPS request supports multiple ID types, not just USDA "840" ID Types. Therefore the request object contains an array of Official ID "key-value" pairs, not simply an array of Official IDs. The key-value is really defined as a type-ID pair. The ID and the type will both always be populated. The type is defined as the ID type ("N","U", etc.); see the Official ID Codes Appendix for a complete list), and the ID is the actual Official ID that is of interest. A single Request may contain requests for multiple types of IDs and therefore, multiple animals.

ATPS will not put the same type-ID pair combination more than once per request, but the same animal may be inadvertently requested more than once because different tag types are allowed in the same request.

2.1.2.2.2 Premises Request

The Premises Request is a request for all events that indicate the presence of any animals at the given premises over a specified date range. A date range will always be included in a premises request. The ATD is instructed to use optimistic inventory logic when determining which events to return. Essentially, if the ATD has an event that indicated that an animal might be at the premises during the specified data range, it will return that event to ATPS. This means the ATD will frequently return events that are outside of the specified date range. ATPS may ask for information on up to 10 Premises in the same request.

Usage Rules:

ATPS will always populate the Begin Request Date or the End Request Date if the National Premises ID array is populated.

If the National Premises ID array is populated with values, the ATD will return the "inventory" of animals at the Premises in the array over the time period indicated by the Begin and End Request Dates. The ATD will return the event or events that indicate that the animal could be at the Premises during the requested timeframe. It is understood that the actual event that indicated inventory will frequently be before or after the request dates.

ATPS will not populate both the array of National Premises IDs and the array of Official IDs.

ATPS will not ask for more than 10 National Premises IDs in the same request.

If the Begin Audit Date is populated, ATPS only wants events for the National Premises IDs in the array if they have been added or modified in the ATD on or since the Audit Date, and indicate inventory given the request dates. If the Begin Audit date is not populated, ATPS always wants all Events indicating inventory during the Request dates for all National Premises IDs in the array.

2.1.2.2.3 Ping Request

ATPS will create a "ping" Request for each ATD once per hour. The purpose of the Ping Request is to ensure that every ATD is available in the event of an actual disease event.

The Ping Request will have the same format as a Premises Request. In fact it will be a premises request for real premises, albeit a premises that does not contain animals. This ping premises will likely be the NRCS in Ft. Collins, and the begin and end dates will be the day of the request.

Every ATD is required to respond to each Ping Request as if it were an actual premises request. It is almost certain that the ATD will never have an actual real animal event at the NRCS. In that case, a "no events" response is acceptable. If an ATD wants to create a "dummy" event in their system that would return an event to ATPS that is also acceptable.

ATPS will have a Ping Request for every ATD every hour. ATPS expects a response from each NEW Ping request returned to the ATD.

ATPS will not return more then one NEW Ping Request even if the ATD has not checked for NEW Requests in more than 2 hours. In that event, only the oldest NEW Ping Request will be returned, and the newer Ping(s) will be discarded by ATPS.

"Dummy" Ping Event:

Here are the specs for a standard "dummy" ping event that all ATD applications can put in their production database in order to return an event to ATPS upon receiving a ping request. This dummy event will be for an animal sighting event at the NRCS in Ft. Collins in 11/1/2005, for an animal with an official USDA 840 ID of "840003000000999". This ID was invalidated and will never be applied to an actual tag. Since it is a sighting event, and the ATD has no other event that proves that the ping animal left the premises, this event will continue to pop up in search results long after the "event date".

Here are the element and attribute values that will satisfy the ping request. The same values can be used in test and production environments.

Element	Value	Notes
eventType.id	9	Animal "sighting" event
eventDate	(see timestamp)	It will be 11/1/2005, the date at which NAIS invalidated this AIN ID.
rptPremId	0034P2K	National Premises ID of the NRCS in Fort Collins.
rptPremId.type	N	N = national, X = any other type.
id	840003000000999	AIN ID that was invalidated by NAIS.
id.type	N	N = official "840" ID.
remarks	Not a real event. Used for ATPS testing.	Please include this remark if possible.
timestamp.y	2005	Must be a 4-digit year.
timestamp.mo	11	1 = January, etc.
timestamp.d	1	Day of month.

An example of a ping response xml with the ping event will look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE eventSub PUBLIC
\"http://localhost/atps/dtds/eventSub.dtd\"
\"http://localhost/atps/dtds/eventSub.dtd\">
<eventSub>
<header>
```

```
<atpsRequestId>12345</atpsRequestId>
<atdResponse final="Y">
 <responseId>32543</responseId>
</atdResponse>
</header>
<animalRecords>
<animalRecord>
 <eventType code="9"/>
 <eventDate>
 <timestamp y="2005" mo="11" d="1" />
 </eventDate>
 <rptPremId type="N">0034P2K</rptPremId>
 <id type="N">840003000000999</id>
 <remarks>Not a real event. Used for ATPS testing.</remarks>
</animalRecord>
</animalRecords>
</eventSub>
```

However ATPS will accept a ping response with no events. The ATD is not required to store a ping event that will be returned. However, they must respond. An example of a ping response xml with no ping event:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE eventSub PUBLIC
\"http://localhost/atps/dtds/eventSub.dtd\"
\"http://localhost/atps/dtds/eventSub.dtd\">
<eventSub>
<header>
<atpsRequestId>12345</atpsRequestId>
<atdResponse final="Y">
 <responseId>32543</responseId>
</atdResponse>
</header>
<animalRecords>
</animalRecords>
</eventSub>
```

From the perspective of ATPS, a ping request is a simple Case with a single Request. The request will go into an ERROR state if the response fails. If there are data validation errors in the response, the request will go to DATA_VALIDATION state.

If a ping request makes it to a closed state, the case that created the request will also automatically go to a closed state, and ATPS will create a “case closed” request so the ATD can identify the ping as having been completed. Note that each ping request to each ATD is assigned to a unique case.

In a future release of ATPS, ATD Users will be able to schedule downtime for the ATD. This will allow ATPS to proactively not send ping requests to the ATD while they are in the maintenance window.

2.1.2.2.4 Case Closed Request

ATPS will generate a special Case Closed “request” when a Case is closed. This Request does not require a Response from the ATD. It will have no Official IDs and no Premises IDs in the request.

Requirements:

ATPS will notify every Enabled ATD that a Case has moved from OPEN to CLOSED by creating a New Request with no actual request parameters.

This Notification is provided as a service to the ATD so that it can manage ATPS responses.

The ATD will not respond to this Request since the caseStatus will be CLOSED (see Usage Rules above).

In a Closed Case Notification, the Request object will be populated thusly:

- requestID will be populated and unique.
- caseID will be populated with the Case that has moved from OPEN to CLOSED.
- caseDescription will be populated.
- caseStatus will be set to CLOSED.
- requestStatus will be set to CASE_CLOSED.
- requestCreatedDate will be populated with the timestamp of the notification request. This will represent the time at which the case was closed.

All other attributes will be empty or null.

2.1.2.3 Case

From the perspective of the ATPS, the ATPS Case is a container for requests. All requests will be associated to a Case. Multiple Requests for multiple ATDs will be assigned to the same Case. A Request cannot be in more than one Case.

General Rules:

Requests assigned to a CLOSED Case can not be responded to.

Requests assigned to a CLOSED Case can only be retrieved by an ATD by specifying the Case ID.

Case Types:

There are two different types of cases: Ping Case and Program Case.

2.1.2.3.1 Ping Case

A Ping Case is created by ATPS when ATPS generates a Ping Request.

Every Ping Request is assigned 1-to-1 to an individual Case.

A System Case is OPEN until the ATD successfully responds to the Ping Request. Then it is moved to CLOSED. The ATD is not notified that the Case ID CLOSED but they can check on the status of the Ping Case if they want, by specifying the Case ID in the Request Criteria.

2.1.2.3.2 Program Case

A Program Case is created by an ATPS Health Official User when they begin a new Animal Health investigation through ATPS.

Multiple requests for multiple ATDs will be assigned to the same Case.

A Program Case is OPEN until the ATPS Health Official User closes the case. At that point a Case Closed Request is made available to each ATD, so they can clear the Case off their system if the so choose.

2.1.2.4 Case Status

The Case Status indicates if the case is Open or Closed.

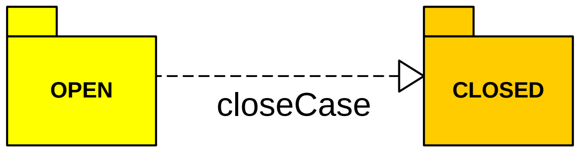
An ATD may not respond to any Request associated to a Closed Case.

Valid Values:

Case Status	*Description*
OPEN	The Case is open and ATD must respond to the message depending on the Request Status value.
CLOSED	The case is closed and no further action by the ATD is required.

Flow:

Case Status Flow



A Case will begin in a State of OPEN. It will move to a State of CLOSED when an ATPS closeCase service is executed upon the Case. A CLOSED Case can not return to OPEN.

Usage Rules:

CLOSED Cases:

An ATD can not respond to a Closed Case Request, regardless of the Request Status.

OPEN Cases:

If the Request Case is OPEN, refer to Request Status Usage Rules to determine the correct ATD action.

Case Status	*ATD Response*
OPEN	Must or May
CLOSED	May not

2.1.2.5 Request Life Cycle

ATPS Requests follow a Life Cycle from beginning to end. The Request Status indicates where the Request is in its Life Cycle.

There are technically two Request Life Cycles; Request Life Cycle and Program Case Closed Request Life Cycle.

The Program Case Closed Request Life Cycle begins and ends with a PROGRAM_CASE_CLOSED Status. It is not changeable by an ATD or ATPS.

The Request Life Cycle is more complex.

All (non-case closed) Requests start out in a NEW Status.

When the ATD receives the Request for the first time, the Status is changed to RECEIVED.

When the ATD responds to the request, the Request Status changes to RESPONDED

In the case of a Split Response, the Request Status changes to INCOMPLETE_SPLIT until all Split Responses are received. Then the Status changes to RESPONDED.

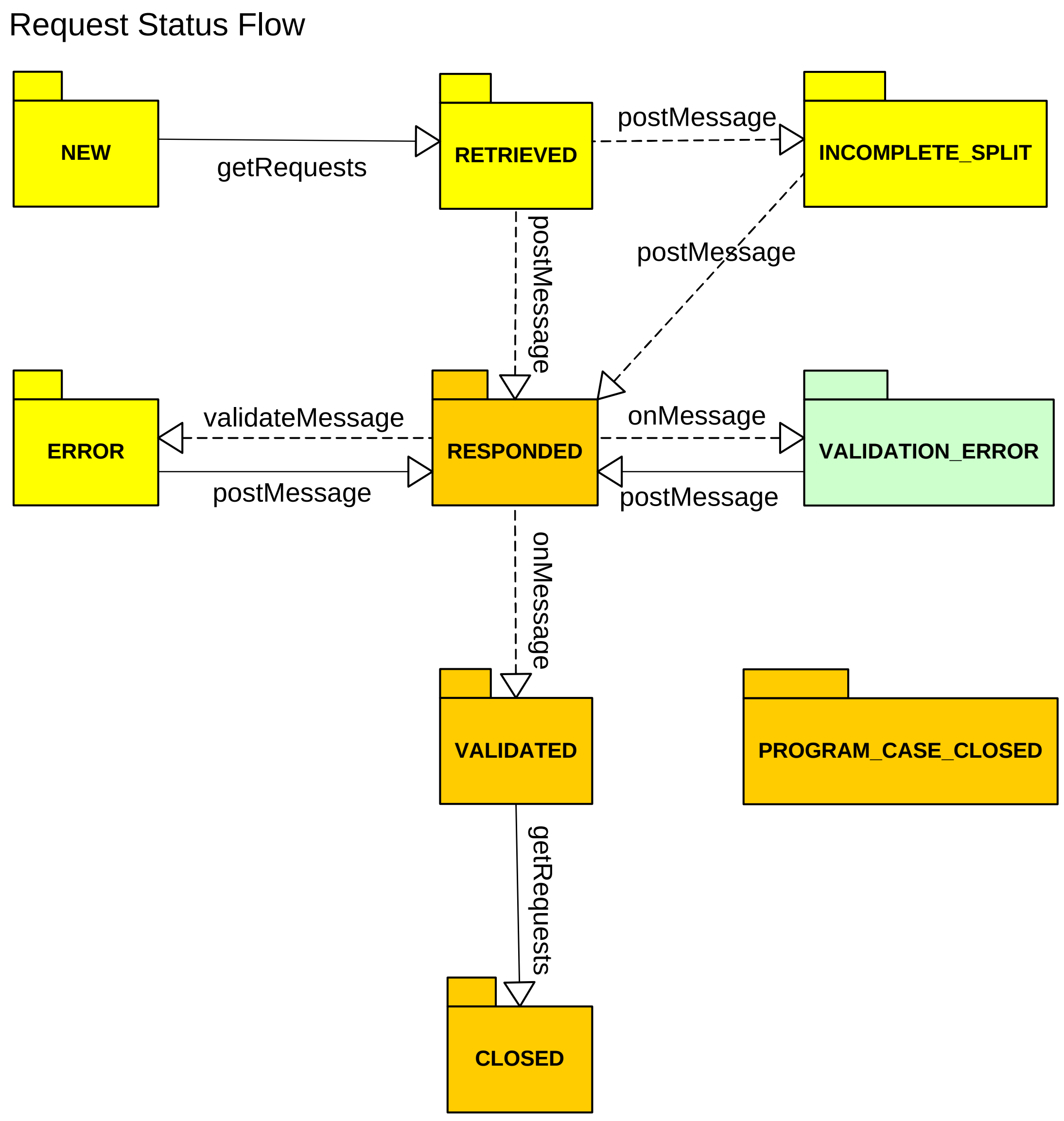
In the event that ATPS receives the Response, but the Response contains a fatal error (XML formatting error is the likely culprit), the Status will move to ERROR.

In the event that ATPS receives the Response, and one or more Response Event Data Elements have formatting or validation errors, the Status will move to VALIDATION_ERROR.

If there are no problems with the Response, the Status moves to VALIDATED.

When the ATD retrieves a VALIDATED Request, the status automatically moves to CLOSED.

The picture below illustrates the Life Cycle Flow:



Usage:

An ATPS Request contains a Status which describes where the Request is in its Life Cycle. Based on the Status, and the Status' associated Category, the ATD will know if it is required to respond to the Request.

Depending on the Request Status, the ATD either Must, May, or May Not Respond to the Request. This rule is driven by the Request Status Category of the Request. All Request Statuses resolve to one of three categories; Active, Actionable, and Static. Active Requests Must be responded to, Actionable Requests May be responded to, and Static Requests May Not be responded to. In all cases if the Case Status is Closed (see the Case Status section), the Request May Not be responded to, regardless of the Request State.

Must Respond:

An ATD is required to respond to an Open Case Request if the Request Status is New or Retrieved, or Error.

May Respond:

An ATD may respond to an Open Case Request if the Request Status is Validation Error.

May Not Respond:

An ATD may not respond to an Open Case Request if the Request Status is Responded, Verified or Closed.

ATD Response Responsibility Table:

Case Status	*Request States Category*	*Request Status*	*ATD Response*
OPEN	ACTIVE	NEW	Must
	ACTIVE	RETRIEVED	Must
	STATIC	RESPONDED	May Not
	ACTIVE	INCOMPLETE_SPLIT	Must
	ACTIVE	ERROR	Must
	ACTIONABLE	VALIDATION_ERROR	May
	STATIC	VALIDATED	May Not
	STATIC	CLOSED	May Not
CLOSED	STATIC	PROGRAM_CASE_CLOSED	May Not
	ANY	ANY	May Not

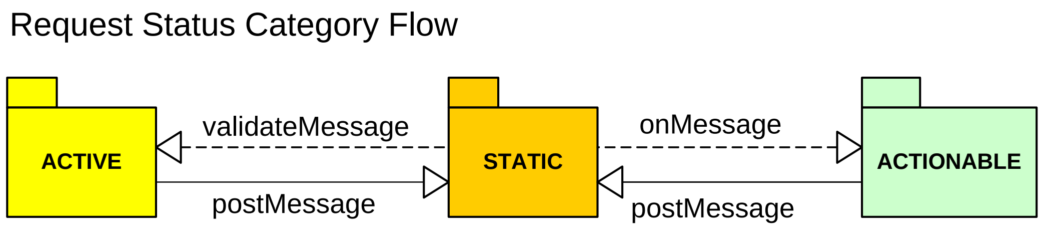
2.1.2.6 Request Status Category

The Request Status Category is a Request Status Sub-Type that indicates the responsibility of the ATD in terms of Responding to the Request. All Request Statuses will have a Status Category. Multiple Request Status Types can have the same category.

Valid Values:

Request Status Category	*Description*
ACTIVE	A Request that still requires a response from an ATD is an ACTIVE Request.
ACTIONABLE	An ACTIONABLE Request allows a response from and ATD but does not require a response from an ATD.
STATIC	A STATIC Request does not allow a response from an ATD.

Flow:



Usage:

An ATD must respond to a Request if it contains an ACTIVE Request Status Category. Currently there are 3 Active States: NEW, RETRIEVED, and ERROR. They are described in detail below.

Requests with an ACTIONABLE Request Status Category permit an ATD to respond, but do not require an ATD to respond. In other words, the ATD May respond to an ACTIONABLE Request. Currently there is 1 Actionable Request State: VALIDATION_ERROR.

At various points in a Request life cycle, ATPS will not allow the ATD to Respond. These states are either when the Request is being processed by ATPS, or when the Request is completed. These are known as STATIC Requests States. Currently ATPS has 3 Static Request States: RESPONDED, VALIDATED, and CLOSED.

ATD Responsibility Table:

Case Status	*Request Status Category*	*ATD Response*
OPEN	ACTIVE	Must
	ACTIONABLE	May
	STATIC	May Not
CLOSED	ANY	May Not

2.1.2.7 Request Status

The Request Status describes the state of the Request. A Request will go through several states while it is being processed, before it is set to a Closed State. The various states are achieved via ATD service calls and internal ATPS service calls.

Valid Values:

Request Status	*Category*	*Description*
NEW	ACTIVE	Request that has not been retrieved by an ATD.
RETRIEVED	ACTIVE	Request that has been retrieved by an ATD but no response has been processed.
RESPONDED	STATIC	Request that has been responded and validated, but not processed.
INCOMPLETE_SPLIT	ACTIVE	Request that has been partially responded to.
ERROR	ACTIVE	Request that has been responded but did not pass validation and has not been processed.
VALIDATION_ERROR	ACTIONABLE	Request that has been responded and passed validation but has data validation errors.
VALIDATED	STATIC	Request that has been responded, validated, and processed successfully.
CLOSED	STATIC	Request that has been validated and subsequently retrieved by the ATD.

PROGRAM_CASE_CLOSED	STATIC	Special status for notifying the ATD that a case is closed.

Status Rules:

2.1.2.7.1 NEW Request Status

A NEW Request is a Request that has not been retrieved by the ATD. It is created by ATPS and will remain NEW until the ATD retrieves it by calling the getRequests web service. NEW is an ACTIVE Request Status Category, so it must be responded to by an ATD.

2.1.2.7.2 RETRIEVED Request Status

A RETRIEVED Request is a Request that has been returned to an ATD as a result of its calling the getRequests web service. The ATD does not actively set the Request to RETRIEVED; simply by calling getRequests with parameters that result in a NEW Request being returned, the Status of the Request will be updated to RETRIEVED.

For example, if an ATD calls getRequests and ATPS returns a single NEW Request, and the ATD does no other action, if they call getRequests again with parameters that will return the same Request, it will now have a Status of RETRIEVED.

Likewise, an ATPS will only return a NEW Request once. After that the Request will have a state of RETRIEVED.

It is possible that a Request is only in a RETRIEVED Status because the ATD is currently working on its Response, in which case the ATD does not need to respond to the Requests again. However a Request might also have a RETRIEVED Status because the ATD attempted a Respond, but ATPS could not identify to which Request the ATD was attempting to respond. For example if an ATD does not supply the Request ID in its Response, or if the Response Request ID is not valid, then ATPS can not update the Status of the Request (it doesn’t know which Request to update!), and the Request will remain RETRIEVED.

In this scenario, ATPS will throw an exception to the ATD. The ATD is expected to catch this exception, analyze the error, make the appropriate changes, and re-submit the response. The status of the request, however, will remain RETRIEVED, and the exception will not be stored.

Therefore RETRIEVED is actually an ACTIVE Request Status Category so it must be responded to by an ATD.

2.1.2.7.3 RESPONDED Request Status

A RESPONDED Request is a Request that has been responded to by an ATD. RESPONDED indicates that ATPS has received the Response from the ATD, but has not completed processing it. The implication is that ATPS is responsible for completing the processing of the response and the ATD does not need to take any action on the Request, as long at it remains in a RESPONDED state.

RESPONDED is a STATIC Status Category so it can not be responded to by an ATD.

A RESPONDED Request will likely not remain RESPONDED long; it will go to ERROR, VALIDATION_ERROR, INCOMPLETE_SPLIT, or VALIDATED during the processing of the Response.

2.1.2.7.4 INCOMPLETE_SPLIT Request Status

In the case of a Split Response by the ATD, the Request Status will remain INCOMPLETE_SPLIT until all the split responses have been received by ATPS. See below for more information on Split Responses and how they affect Request Status.

INCOMPLETE_SPLIT is an ACTIVE Request Status Category so it must be responded to by an ATD.

Split Response Status:

In the event that the ATD has submitted multiple or Split responses to an ATPS Request, ATPS will track all of the responses with the single Request object. Since it is possible for the splits to result in different statuses for the Request, ATPS will employ a hierarchy of Statuses when providing feedback on a Split Response.

Here are the Split Response cases and the Request Status for each scenario:

- Scenario 1: Not all Splits received.

Request Status: INCOMPLETE_SPLIT.

For example, if an ATD has split the Response into 2 parts, and ATPS has received one Split but not the other, then the status of the Request will be INCOMPLETE_SPLIT. The ATD will not know if the first Split was successfully processed until ATPS receives all of the splits. If the ATD receives an exception from the postMessage web service when it submits the first of two Splits, which would result in an ERROR if there was no second split forthcoming, the Request will remain at INCOMPLETE_SPLIT until the second (last) split is received. Then it will move to ERROR (see scenario 2 below).

- Scenario 2: All Splits received. At least one Split results in an ERROR.

Request Status: ERROR

No matter what else happens, if all of the Splits are received and at least one of them results in an ERROR status, the Response will have a status of ERROR, and the first ERROR exception will be returned. If another Split was validated or had validation errors, this information will not be reflected in the status of the Request.

- Scenario 3: All Splits received. No splits result in ERROR. At least one Split results in VALIDATION_ERROR.

Request Status: VALIDATION_ERROR

If all the splits are received and none of them resulted in an ERROR status, but one or more did result in a VALIDATION_ERROR, then the Response will be VALIDATION_ERROR, and the first 100 Invalid Items will be returned, even if they span multiple splits. Note that ATPS will only store the first 100 Invalid Items found, even if the ATD Response contains more than 100 Invalid Items.

- Scenario 4: All Splits Received, all splits Validated.

Request Status: VALIDATED

In this case (“happy path”), the Request status will be VALIDATED (and will move to CLOSED when retrieved by the ATD).

- Scenario 5: Duplicate Split number received.

Request Status: INCOMPLETE_SPLIT

If ATPS receives a duplicate Split number for a Request from an ATD, it will assume that the ATD is resubmitting the Response and will inactivate the original response.

2.1.2.7.5 ERROR Request Status

An ERROR Request is a Request that has been responded to by an ATD, but that cannot be processed at all by ATPS due to an error on the part of the ATD. ATPS requires the ATD resolve the error and resubmit the Response.

Generally ERROR Requests can be divided into two categories; XML errors and fatal data validation errors.

XML errors happen when the XML sent by the ATD has formatting flaws that prevent the XML from passing the Response DTD specifications.

Fatal data validation errors occur when the XML format passes the DTD specifications, but the content of the Response data prevents ATPS from processing the message.

However, there is a third category of "error" which will not update the Request to an ERROR Status. Any time ATPS can not determine the Request ID that is being responded to, or if the ATD is responding to an invalid Request ID, the Status of the Request will remain what it was before, and ATPS will throw an ATPSException to the ATD. In this case, ATPS will not know which request to update, so it will not update any Request.

An Invalid Request ID is any Request ID that is not assigned to the ATD responding to it, or has a Status Category of STATIC, or has a Case Status of CLOSED.

In all scenarios, ATPS will throw an ATPSException with a "detailed reason" why the Response failed.

If the ATD calls getRequests and receives a Request that has an ERROR Status, the Request object will have the ATPSExceptionInfoWS attribute populated. This will allow the ATD to retrieve the cause of the ERROR status.

If the ATD responds to the same Request two times, and both Responses result in an ERROR, ATPS will only store the latest ATPSExceptionInfoWS in the Request.

ERROR is an ACTIVE Request Status Category so it must be responded to by an ATD.

2.1.2.7.6 VALIDATION_ERROR Request Status

A VALIDATION_ERROR Request is a Request that has been successfully responded to, but not fully validated. Typically this means one or more Response Events could not be processed by ATPS due to either Data Format or Data Integrity errors. A Request in this state may be responded to by the ATD again, but it is not required.

If an ATD responds to a Request that has a state of VALIDATION_ERROR, they must respond with the entire Response. The ATD is not allowed to respond with only the Response Events that are invalid. This is because ATPS will not know exactly which items are supposed to be replaced, since ATPS is not required to store the ATD Event ID. This is also due to the fact that the ATD source data may have changed between the initial Response that resulted in the VALIDATION_ERROR, and the new Response.

There are two categories of VALIDATION_ERROR issues: Data Format, and Data Integrity.

A Data Format error results from a Response Event that has one or more elements that can not be stored by ATPS due to basic formatting errors. An example of this is a Date element that is of the incorrect format, or a Premises ID that is too many characters.

A Data Integrity error results from a Response Event Element that does not match an agreed-upon or standard list of possible values. For example, a National Premises ID value that does not match the National Registry, or a Species Code that does not match the agreed-upon list of Species Codes are both Data Integrity Errors.

If an ATD submits a response that results in a Status of VALIDATION_ERROR, the ATD will not receive the results immediately. This is because the result submission is asynchronous. In the event of a VALIDATION_ERROR, the ATD will receive a successful return from the call to ATPS. The "Success" means that ATPS was able to process the Response – i.e. it made it past the ERROR Status. It does not mean that all items in the Response are valid. The ATD must get the Request and check the status of the request to determine if the Response was fully valid.

When an ATD gets a Request that has a state of VALIDATION_ERROR, ATPS will populate a list of Invalid Items that describe the reasons why particular events were not validated. See below for more information on the Invalid Items list.

ATPS will only record up to 100 validation errors per ATD Response. For example if a particular Response contains 102 validation errors, the last two will not be presented back to the ATD when the Request is retrieved.

If a particular Response Event contains multiple validation errors, all of the errors will be added to the Invalid Item list. Therefore, it is possible for the same ATDEventID to be present multiple times in the Invalid Item list, but it is likely that even if the same Event Element has multiple validation errors, only the first error found will be present in the list.

Finally, ATPS will only persist the results of the last Response from an ATD. If an ATD responds to a Request two times, and both responses had validation errors, when the ATD retrieves the Request only the Invalid Items from the latest Response will be included in the Response object.

2.1.2.7.7 VALIDATED Request Status

A VALIDATED Request is a request that has been successfully processed with no errors. If the ATD submitted Split Responses, it further indicates that all the splits have been received and they are all successfully processed.

Successfully Processed means that there were no errors with the Response, and that no Response Events had validation errors of any sort. A VALIDATED Response does not indicate any level of data quality.

VALIDATED is a STATIC Status Category, so it can not be responded to by an ATD. A VALIDATED Request will be moved to CLOSED when it is retrieved by the ATD (see below).

2.1.2.7.8 CLOSED Request Status

A CLOSED Request is a Request that was VALIDATED, and subsequently returned to an ATD as a result of the ATD retrieving the Request in a VALIDATED state. A Request will remain in a VALIDATED state indefinitely until retrieved by the ATD.

CLOSED is a STATIC Status Category so it can not be responded to by an ATD.

2.1.2.7.9 PROGRAM_CASE_CLOSED Request Status

A PROGRAM_CASE_CLOSED Request Status indicates a special Request that will serves to notify every ATD that a Program Case has been moved to a CLOSED status.

This Request Status is STATIC and can not be responded to by the ATD.

2.1.2.8 Get Requests Web Service Specification

Signature:

```
ATPSRequestWS[] getRequests(
String encryptedATDId,
String pin,
ATPSRequestCriteriaWS criteria)
throws SOAPException;
```

Arguments:

2.1.2.8.1 encryptedATDId

Type: String, 50 characters max, special characters allowed

Null: No

The Encrypted ATD ID is a required argument. The ATD ID coupled with the PIN is how ATPS authenticates the ATD. The Encrypted ATD ID can only be accessed by an Application User assigned to an Account containing the ATD, and can only be accessed via the web application.

2.1.2.8.2 pin

Type: String, 10 characters max, special characters allowed

Null: No

The C is a required argument. The Eauth ID coupled with the PIN is how ATPS authenticates the ATD. The V can only be accessed by an Application User assigned to an Account containing the ATD, and can only be accessed via the web application.

2.1.2.8.3 criteria

Type: ATPSRequestCriteriaWS

Null: No

The ATPS Request Criteria is how the ATD will specify which requests it wishes to get.

2.1.2.8.4 return

Type: ATPSRequestWS[]

Null: No, but may be empty

The ATPS Request contains the Request information. Depending on the Status of the Request, the ATD may be required to respond.

2.1.2.9 ATPSRequestWS Object

Attribution:

```
ATPSRequestWS{
Long requestId;
ATPSCaseWS case;
String requestStatusCategory;
String requestStatus;
Date requestCreatedDate;
Date requestModifiedDate;
ATPSOfficialIdWS[] officialIds;
String[] nationalPremisesIds;
String species;
Date beginRequestDate;
Date endRequestDate;
Date beginAuditDate;
Date endAuditDate;
ATPSInvalidItemWS [] invalidItems;
ATPSInvalidItemWS [] exceptionItems;
}
```

Attributes:

2.1.2.9.1 ATPSRequestWS.requestId

Type: Long integer

Size: 15 digits or less

Null: No

Description:

The ATPS request ID is a unique, ATPS-generated numeric ID for the request. Every request will have a unique ID.

Usage Rules:

ATDs will include the request number on all responses relating to the request.

2.1.2.9.2 ATPSRequestWS.case

Type: ATPSCaseWS

Null: No

Description:

This object contains information about the Case to which this Request is attached. All Requests are attached to a Case.

2.1.2.9.3 ATPSRequestWS.requestStatusCategory

Type: String

Size: 20 characters max

Null: No

Valid Values:

Request Status Category	Description
ACTIVE	A Request that still requires a response from an ATD is an ACTIVE Request.
ACTIONABLE	An ACTIONABLE Request allows a response from and ATD but does not require a response from an ATD.
STATIC	A STATIC Request does not allow a response from an ATD.

Description:

There are three Request Status Categories: ACTIVE, ACTIONABLE, and STATIC. Their use is described in detail in the Request Life Cycle section.

2.1.2.9.4 ATPSRequestWS.requestStatus

Type: String

Size: 20 characters max

Null: No

Valid Values:

Request Status	Category	Description
NEW	ACTIVE	Request that has not been retrieved by an ATD.
RETRIEVED	ACTIVE	Request that has been retrieved by an ATD but no response has been processed.
RESPONDED	STATIC	Request that has been responded and validated, but not processed.
INCOMPLETE_SPLIT	ACTIVE	Request that has been partially responded to.
ERROR	ACTIVE	Request that has been responded but did not pass validation and has not been processed.
VALIDATION_ERROR	ACTIONABLE	Request that has been responded and passed validation but has data validation errors.
VALIDATED	STATIC	Request that has been responded, validated, and processed successfully.
CLOSED	STATIC	Request that has been validated and subsequently retrieved by the ATD.
PROGRAM_CASE_CLOSED	STATIC	Special status for notifying the ATD that a case is closed.

Description:

The Request Status describes where the Request is in its Life Cycle.

See the Request Life Cycle section for more information.

2.1.2.9.5 ATPSRequestWS.requestCreatedDate

Type: Date

Null: No

Description:

The ATPS Request Created Date is the date when the Request was generated by ATPS.

Usage Rules:

No Usage Rules. The created date is provided as a convenience to the ATD.

2.1.2.9.6 ATPSRequestWS.requestModifiedDate

Type: Date

Null: No

Description:

The ATPS Request Modified Date is the date when the Request was last modified. For NEW requests, the created and modified dates should be identical. As the Request moves through Statuses because of actions by the ATD, the modified date will change to reflect the last time the Request was modified.

Usage Rules:

No Usage Rules. The created date is provided as a convenience to the ATD.

2.1.2.9.7 ATPSRequestWS.officialIds

Type:	ATPSOfficialIdWS[]
Null:	No, but may be empty
Size:	1,000 maximum

Description:

The Official ID array represents the list of all Official IDs for which ATPS wants event information. The array will be empty (but not null) if the request is a premises request.

The Official ID object contains an ID and a Type attribute. The ID is the actual ID, and the Type indicates what type of ID it is. ATPS may ask for information on IDs of multiple types in the same Request. The Type is included so that the ATD does not have to “guess” at the type of ID that is being requested.

2.1.2.9.8 ATPSRequestWS.nationalPremisesIds

Type:	String[]
Null:	No, but may be empty
Size:	10 maximum

Description:

The National Premises ID array represents the list of all National Premises for which ATPS wants event information. The array may be empty if the Request contains no National Premises for which information is required.

The National Premises ID will always be a National Premises ID.

2.1.2.9.9 ATPSRequestWS.species

Type:	String
Null:	Can be null
Size:	3
Valid Values:	See Species Category Code Appendix

Description:

The Species is a qualifier for a particular species. This attribute is included as a convenience to the ATD. If this attribute is populated, ATPS is only interested in events for the species indicated in the Request. If the ATD does not store events for the indicated species, they can automatically submit a response with no events without even checking their persisted data.

If populated, the species attribute will contain a species category, and not an individual species code.

2.1.2.9.10 ATPSRequestWS.beginRequestDate

Type: Date

Precision: Day

Null: Null unless nationalPremisesID array is populated, then not null

Description:

Begin Request Date is the start date of the date range for which ATPS is requesting animal inventory for one or more premises.

Usage:

Begin Request Date is used in tandem with the National Premises ID array. The ATD will return all animal events that indicate that the animal may be at any premises in the National Premises ID array on or after the Begin Request Date specified. Even if the event that indicated inventory occurred before the Begin Request Date the ATD is expected to return that event if there is no other event that proved that the animal was not at the premises.

Begin request Date will only be populated if the National Premises ID array is not empty. Otherwise it will be null.

Begin Request Date must be populated if End Request Date is populated. End Request Date can be the same day as Begin Request Date but can not be earlier than Begin Request Date.

2.1.2.9.11 ATPSRequestWS.endRequestDate

Type: Date

Precision: Day

Null: Null unless nationalPremisesID array is populated, then not null

Description:

End Request Date is the end date of the date range for which ATPS is requesting animal inventory for one or more premises.

Usage:

End Request Date is used in tandem with the National Premises ID array. The ATD will return all animal events that indicate that the animal may be at any premises in the National Premises ID array on or before the End Request Date specified. Even if the event that indicated inventory occurred after the End Request Date the ATD is expected to return that event if there is no other event that proved that the animal was not at the premises.

End request Date will only be populated if the National Premises ID array is not empty. Otherwise it will be null.

Begin Request Date must be populated if End Request Date is populated. End Request Date can be the same day as Begin Request Date but can not be earlier than Begin Request Date.

2.1.2.9.12 ATPSRequestWS.beginAuditDate

Type: Date

Precision: Day

Null: May be null

Description:

If the Begin Audit Date is populated, this indicates that ATPS is only interested in events that match the Request criteria, and furthermore only events that were added or modified in the ATD on or later than the Begin Audit Date.

This allows ATPS to ask a “follow up” to an older request. The ATD can provide updated information only without having to respond with information it has already sent.

Begin Audit Date may be populated for any Request.

End Audit Date must be populated if Begin Audit Date is populated. End Audit Date can be the same day as Begin Audit Date but can not be earlier than Begin Audit Date. If the dates are the same, ATPS wants all events “audited” (inserted or updated) that day.

2.1.2.9.13 ATPSRequestWS.endAuditDate

Type: Date

Precision: Day

Null: May be null

Description:

If the End Audit Date is populated, this indicates that ATPS is only interested in events that match the Request criteria, and furthermore only events that were added or modified in the ATD on or earlier than the End Audit Date.

This allows ATPS to ask a “follow up” to an older request. The ATD can provide updated information only without having to respond with information it has already sent.

End Audit Date may be populated for any Request.

Begin Audit Date must be populated if End Audit Date is populated. End Audit Date can be the same day as Begin Audit Date but can not be earlier than Begin Audit Date. If the dates are the same, ATPS wants all events “audited” (inserted or updated) that day.

2.1.2.9.14 ATPSRequestWS.invalidItems

Type: ATPSInvalidItemWS[]

Null: No, will be empty unless Request Status is VALIDATION_ERROR.

Size: 100 maximum

Description:

This is an array of Invalid Items that indicate specific field-level validation errors in the ATD Response.

The Invalid Item may contain up to eight attributes that identify the invalid item: The ATD Response ID, the Split Number, the ATD Event ID (if supplied by the ATD), The Response Record Sequence, Element Name, Element Value, Cause code, and Reason or description of the invalid cause. See the description of the Invalid Item object for more information.

The array will be empty unless the Request Status is VALIDATION_ERROR. Then it will have at least one item and at most 100 items in the array.

2.1.2.9.15 ATPSRequestWS.execptionItems

Type: ATPSInvalidItemWS[]

Null: Will be null unless Request Status is ERROR

Description:

ATPS Exception Items will only be present if the Request Status is ERROR, or INCOMPLETE_SPLIT. All other times it will be null. In these cases an exception was raised when the client attempted to submit the response. This exception contained a message and a fault code. The Request itself will retain the message and fault code so that the client can retrieve this information later without having to hang on to the exception information.

2.1.2.10 ATPSCaseWS Object

Attribution:

```
ATPSCaseWS{
Long caseId;
String caseDescription;
String caseStatus;
}
```

Attributes:

2.1.2.10.1 ATPSCaseWS.caseId

Type:	Long integer
Size:	15 digits or less

Description:

The ATPS Case ID is a unique ATPS-generated numeric ID for the case. The same Case ID will exist for multiple Request IDs. The Case ID is provided as a convenience to the ATDs.

Usage Rules:

No required Usage.

2.1.2.10.2 ATPSCaseWS.caseDescription

Type:	String
Size:	256 characters max
Null:	No

Description:

The ATPS Case Description is a text description of the Case. The Case ID is provided as a convenience to the ATDs, and is provided because the ATDs have specifically asked for its inclusion in this object.

To Do: Determine what information is permissible in the Case Description. It is probable that the description will not include the disease type.

Usage Rules:

No required Usage.

2.1.2.10.3 ATPSCaseWS.caseStatus

Type:	String.
Size:	20 characters max.

Null: No

Valid Values:

Case Status	Description
OPEN	The Case is open and ATD must respond to the message depending on the Request Status value.
CLOSED	The case is closed and no further action by the ATD is required.

Description:

The ATPS Case Status describes the Case Status. Current allowed values include "OPEN" and "CLOSED".

2.1.2.11 ATPSOfficialIdWS Object

Attribution:

```
ATPSOfficialIdWS{
String officialId;
String officialIdType;
}
```

Description:

2.1.2.11.1 ATPSOfficialIdWS.officialId

Type: String

Null: No

Size: 17 characters

Description:

The Official ID is the Official Animal Identification Number for the animal. It is not required to be a USDA "840" number. The Official ID will not be null.

2.1.2.11.2 ATPSOfficialIdWS.officialIdType

Type: String

Null: No

Size: 1 character

Valid Values: See Official ID Codes in the Appendix.

Description:

The Official ID Type is a descriptor of the Official ID. This attribute is included as a convenience to the ATD. It is likely that the ATD will know by the format of the ID itself what Type it is, but the Type attribute takes all guesswork out of the Request. ATPS will only request information for Official ID Types.

2.1.2.12 ATPSInvalidItemWS Object

Attribution:

```
ATPSInvalidItemWS{
   String ATDResponseId;
   Integer split;
   String ATDEventId;
   Long recordSequence;
   String elementName;
   String elementValue;
   ATPSExceptionInfoWS exceptionInfo;
}
```

Description:

2.1.2.12.1 *ATPSInvalidItemWS.ATDResponseId*

Type: String (20 max)

Null: No

Description:

This is the ATD-supplied Response ID that contains the invalid item. It is included to allow the ATD to look up the response more quickly.

2.1.2.12.1 *ATPSInvalidItemWS.split*

Type: Integer

Null: Can be null

Description:

If the response is a split response, the split attribute will contain the split number containing the invalid item.

2.1.2.12.1 *ATPSInvalidItemWS.ATDEventId*

Type: String

Null: Can be null

Description:

If the ATD supplies an ATD Event ID with its Response, ATPS will include that ID in the Invalid Item. This should help the ATD quickly identify the Event that caused the validation error, but it is an optional Response Element so it may be null.

2.1.2.12.2 *ATPSInvalidItemWS.recordSequence*

Type: Long

Null: No

Description

The Record Sequence identifies the particular record that contains the element with the data validation error. The first record in the list of records will have a record sequence of 0. This helps the ATD identify the particular record with the validation error in the event the ATD does not supply the ATDEventId Element in its Response.

2.1.2.12.3 *ATPSInvalidItemWS.elementName*

Type: String

Null: No

Description:

The Element Name will be the element name of the element that failed. For example if the "species" element failed validation because the value was "XXXX", then the Element Name would be "species"

2.1.2.12.4 ATPSInvalidItemWS.elementValue

Type: String

Null: No

Description:

The Element Value will be the value of the element that failed. For example if the "species" element failed validation because the value was "XXXX", then the Element Value would be "XXXX"

2.1.2.12.5 ATPSInvalidItemWS.exceptionInfo

Type: ATPSExceptionInfoWS

Null: No

Valid values (subject to change):

Description:

This will contain the exception code and description relating to the invalid item. For instance, a data format invalid item will have a different cause than a validation error.

2.1.2.13 ATPSExceptionInfoWS Object

Attribution:

```
ATPSExceptionInfoWS {
String cause;
String message;
}
```

Description:

This is a generic "exception" object that stores a cause or ID, and a description of the problem. It is used by both invalid items and "real" exceptions.

2.1.2.13.1 ATPSExceptionInfoWS.cause

Type: String

Null: No

Valid values (subject to change):

Error causes:

Cause	Description
8000	Unknown Exception
8001	Incomplete response
8002	Response parsing error

8003	Connection Refused
8004	Disabled ATD failure
8005	Authorization failure

Invalid Item causes:

Cause	Description
7000	Element value data format error
7001	Element value data validation error

Description:

The cause is a code that the ATD can use to categorize and potentially automatically respond to a Request that is in an ERROR state.

The Fault Code will be present in the exception when it is thrown initially as well.

2.1.2.13.2 ATPSExceptionInfoWS.message

Type: String

Null: No

Description:

The Message is a text description of the exception. It may the description in the table of values in the Cause item above, or a more detailed description may be included instead.

The message will be a text description that details why the element failed.

For example if the "species" element failed validation because the value was "XXXX", then the message would likely be "value too long." Note that in this example the element is likely both too long and not a valid value, but ATPS does not guarantee that it will return every error for the element.

2.1.2.14 ATPSRequestCriteriaWS Object

Attribution:

```
ATPSRequestCriteriaWS{
Long requestId;
Long caseId;
String[] requestStatus;
String requestStatusCategory;
Date beginRequestCreatedDate;
Date beginRequestModifiedDate;
}
```

General Usage Rules:

At least one attribute besides Created Date and Modified Date must be populated when calling getRequests.

Requests that are in CLOSED Cases will only be returned if the ATD specifies the Case ID in the request criteria, or if the ATD specifies the individual request ID. This is to prevent ATPS from returning potentially massive numbers of Requests.

If multiple criteria are populated, the search logic acts like an AND qualifier. The exception is Request Status array element. If the request status array has multiple items in it, they will act like an OR qualifier.

Example: To get requests created after 9/25/2006, and having a status of NEW or RETRIEVED, the ATD will populate the Request Created Date with a Date of 9/25/2006, and the Status array with NEW and RETRIEVED items.

Description:

2.1.2.14.1 ATPSRequestCriteriaWS.requestId

Type: Long

Size: 15 digits or less

Null: Can be null

Description:

The Request ID is the ATPS-generated ID of the Request. The ATD can populate this attribute to get a single Request matching the Request ID.

If the Request matching the Request ID exists, and it is a Request for the ATD, ATPS will return the request.

If the Request ID does not exist, ATPS will return an empty array.

If the Request ID exists but the Request was for a different ATD, ATPS will return an empty array. Note that even if ATPS requests the same information from every ATD, each ATD is assigned a unique request ID, even though the rest of the request parameters will be identical.

2.1.2.14.2 ATPSRequestCriteriaWS.caseId

Type: Long

Size: 15 digits or less

Null: Can be null

Description:

The Case ID is the ATPS-generated ID of a Case. The ATD can populate this attribute to get all requests matching the Case ID for the ATD.

If the Case ID does not exist, ATPS will return an empty array.

If the Case ID does exist, ATPS will return all requests matching the Case ID for the ATD. ATPS will not return any requests for any other ATD.

2.1.2.14.3 ATPSRequestCriteriaWS.requestStatus

Type: String[]

Null: Can be null or empty

Valid Values: See ATPSResponse.requestStatus.

Description:

The Request Status is an array of Status values that the ATD can set in order to get Requests matching any of the Status values in the array. See above for valid values.

If there are no requests matching any of the Statuses, ATPS will return an empty array.

2.1.2.14.4 ATPSRequestCriteriaWS.requestStatusCategory

Type: String[]

Null: Can be null

Valid Values: See ATPSResponse.requestStatusCategory.

Description:

The Request Status Category is a string that the ATD can set in order to get Requests matching the Status Category. See above for valid Status Category values.

If there are no requests matching the Status Category, ATPS will return an empty array.

2.1.2.14.5 ATPSRequestCriteriaWS.beginRequestCreatedDate

Type: Date

Precision: Day

Null: Can be null

Description:

The Begin Request Created Date is the initial date when the Request was created and set to NEW. The ATPS can set this attribute to instruct ATPS to return Requests that were created on the day specified or later.

If no Requests were created on the date specified or later, ATPS will return an empty array.

2.1.2.14.6 ATPSRequestCriteriaWS.beginRequestModifiedDate

Type: Date

Precision: Day

Null: Can be null

Description:

The Begin Request Modified Date is the initial date when the Request was last modified. A request is typically modified when the ATPS responds to the Request. The ATPS can set this attribute to instruct ATPS to return Requests that were modified on the day specified or later.

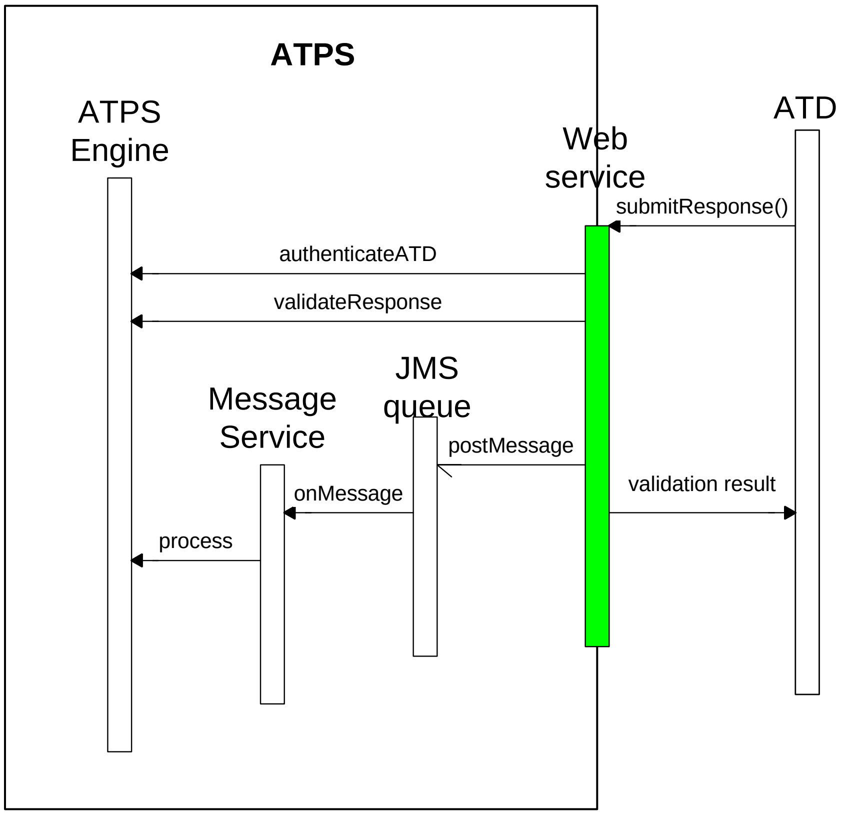
If no Requests were modified on the date specified or later, ATPS will return an empty array.

2.1.3 Submit Response Web Service

A Response is how an ATD returns requested information to ATPS.

More specifically this is how an ATD will submit a Response to a Request.

The Get Requests web service fulfills step 5 on the basic web service use case:



The "happy path" steps to processing a response are outlined below:

- The ATD calls the Submit Response web service to respond to a message.
- ATPS authenticates the ATD.
- ATPS validates the message for format errors.
- Request moves to RESPONDED if validation is successful.
- Request will usually move to ERROR if validation is not successful, and an exception will be thrown.
- ATPS posts the response to the internal message queue, where it is processed internally and asynchronously.
- Without waiting for the response to be processed, ATPS completes the web service transaction and returns control to the ATD.

2.1.3.1 Basic Requirements

All responses are submitted as an XML String.

The response XML has a DTD against which all Responses are validated when they are submitted.

Additionally ATPS will ensure the validity of the Request ID before processing the Response.

2.1.3.1.1 Required Response Data

ATPS has 4 required data fields in each response. The four fields are: Official ID, Premises ID, Event Type, and Event Date. These required elements are discussed in more detail below but some general explanation is provided in this section.

- Official ID

An Official ID is an animal ID that uniquely identifies an animal. ATPS allows several types of official IDs. The list is included in the specifications for the response XML.

- Premises ID

A Premises ID uniquely identifies a premise. ATPS allows NAIS National Premises IDs and other types of premises IDs as well.

- Event Type

The event type is a code describing the event that occurred. The list of codes and events is described in the XML specification below.

- Event Date

The Event Date is the date upon which the event occurred.

2.1.3.1.2 Multiple IDs and Official IDs

Often times, animals are identified by multiple IDs. Here are some guidelines for processing animal events with multiple IDs.

The animal event specification contains a required “official” ID and an (optional) list of “optional” IDs. An Official ID is an animal ID that uniquely identifies an animal. Optional IDs may or may not uniquely identify an animal.

For an animal with multiple IDs, ATPS would like to receive as many optional IDs as possible, so ATDs are requested to fill in the list of optional IDs with all IDs other than the official ID.

Given a choice of official IDs, the ATD must put one in the official ID element, and may put the others in the optional IDs element. The Official ID may be repeated in the Optional IDs element but this is not preferred.

The ATD may put any of the Official IDs in the official ID element, but ATPS prefers that ATDs use the following logic when determining which ID to put in the Official ID element:

- When responding to an Official ID Request: The ATD is requested to put the requested ID in the official ID element, and any other ID in the optional ID element.
- When responding to a Premises ID Request: Any Official ID can be put in the official ID element, but ATPS would prefer that the ATD rank official IDs in the following order of ID Types: N, U, A, R, F, B, T, G (see below for code descriptions).

2.1.3.2 Response XML

This section describes the format of the response XML. Also the DTD and the Strict DTD are presented here.

2.1.3.2.1 Response DTD

ATPS provides the DTD against which the Response XML is validated. This DTD is publicly available on the internet. The ATD should validate their response against the DTD before submitting it to ATPS, but ATPS can not enforce this. ATPS will validate all responses against this DTD. Any response that does not validate will not get processed. An exception will be thrown to the ATD, and if possible the Request itself will get set to ERROR and the exception details will be stored with the Request.

Here is a copy of the DTD:

```
<?xml version="1.0" encoding="UTF-8"?>
<!—DTD for submission of Animal and Group Events -->
<!ELEMENT eventSub (header,(animalRecords|groupRecords))>
<!ELEMENT header (atpsRequestId,atdResponse)>
<!ELEMENT atpsRequestId (#PCDATA)>
<!ELEMENT atdResponse (responseId)>
 <!ATTLIST atdResponse
 final (Y|N) #REQUIRED
 split CDATA #IMPLIED
 >
<!ELEMENT responseId (#PCDATA)>
<!ELEMENT animalRecords (animalRecord*)>
<!ELEMENT animalRecord (ATDEventId*, eventType, eventDate, rptPre-
mId, id, srcDestPremId*, animal*, remarks*, reTagId*, optIds*)>
 <!ATTLIST animalRecord
 elecRead CDATA #IMPLIED
 status CDATA #IMPLIED
 >
<!ELEMENT groupRecords (groupRecord*)>
<!ELEMENT groupRecord (ATDEventId*, eventType, eventDate, rptPremId,
id, srcDestPremId*, group*, remarks*)>
 <!ATTLIST groupRecord
 elecRead CDATA #IMPLIED
 status CDATA #IMPLIED
 >
<!ELEMENT ATDEventId (#PCDATA)>
<!ELEMENT eventType EMPTY>
 <!ATTLIST eventType
 code CDATA #REQUIRED >
<!ELEMENT eventDate (timestamp)>
<!ELEMENT rptPremId (#PCDATA)>
 <!ATTLIST rptPremId
 type CDATA #IMPLIED>
<!ELEMENT id (#PCDATA)>
 <!ATTLIST id
 type CDATA #IMPLIED>
<!ELEMENT srcDestPremId (#PCDATA)>
 <!ATTLIST srcDestPremId
 type CDATA #IMPLIED>
<!ELEMENT animal (DOB*,age*)>
 <!ATTLIST animal
 species CDATA #IMPLIED
 gender CDATA #IMPLIED
 breed CDATA #IMPLIED
 >
<!ELEMENT DOB (timestamp)>
 <!ATTLIST DOB
 est CDATA #REQUIRED>
<!ELEMENT age (#PCDATA)>
 <!ATTLIST age
 scale (D|M|Y) #REQUIRED>
<!ELEMENT remarks (#PCDATA)>
<!ELEMENT reTagId (#PCDATA)>
 <!ATTLIST reTagId
 type CDATA #IMPLIED>
<!ELEMENT optIds (optId*)>
<!ELEMENT optId (#PCDATA)>
 <!ATTLIST optId
 type CDATA #IMPLIED>
<!ELEMENT group (groupSubsetId*, groupCount*)>
 <!ATTLIST group
 groupType CDATA #IMPLIED
 species CDATA #IMPLIED
 breed CDATA #IMPLIED
 >
<!ELEMENT groupSubsetId (#PCDATA)>
<!ELEMENT groupCount (#PCDATA)>
<!ELEMENT timestamp EMPTY>
 <!ATTLIST timestamp
 y CDATA #REQUIRED
 mo CDATA #REQUIRED
 d CDATA #REQUIRED
 h24 CDATA "0"
 mi CDATA "0"
 s CDATA "0"
 tz CDATA #IMPLIED
 >
```

2.1.3.2.2 Strict DTD:

ATPS will also provide a “strict” DTD that the ATD can use to validate their Response. This DTD provides some data validation checks. An ATD that validates against this DTD will be ensured that most data validation errors will be avoided during processing.

Here is a copy of the Strict DTD:

```
<?xml version="1.0" encoding="UTF-8"?>
<!–Strict DTD for submission of Animal and Group Events -->
<!ELEMENT eventSub (header,(animalRecords|groupRecords))>
<!ELEMENT header (atpsRequestId,atdResponse)>
<!ELEMENT atpsRequestId (#PCDATA)>
<!ELEMENT atdResponse (responseId)>
 <!ATTLIST atdResponse
 final (Y|N) #REQUIRED
 split CDATA #IMPLIED
 >
<!ELEMENT responseId (#PCDATA)>
<!ELEMENT animalRecords (animalRecord*)>
<!ELEMENT animalRecord (ATDEventId*, eventType, eventDate, rptPre-
mId, id, srcDestPremId*, animal*, remarks*, reTagId*, optIds*)>
```

```
 <!ATTLIST animalRecord
 elecRead (Y|N) #IMPLIED
 status (C) #IMPLIED
 >
<!ELEMENT groupRecords (groupRecord*)>
<!ELEMENT groupRecord (ATDEventId*, eventType, eventDate, rptPremId,
id, srcDestPremId*, group*, remarks*)>
 <!ATTLIST groupRecord
 elecRead (Y|N) #IMPLIED
 status (C) #IMPLIED
 >
<!ELEMENT ATDEventId (#PCDATA)>
<!ELEMENT eventType EMPTY>
 <!ATTLIST eventType
 code (0|1|2|3|4|5|6|7|8|9|10|11|12|13) #REQUIRED >
<!ELEMENT eventDate (timestamp)>
<!ELEMENT rptPremId (#PCDATA)>
 <!ATTLIST rptPremId
 type (N|X) #REQUIRED>
<!ELEMENT id (#PCDATA)>
 <!ATTLIST id
 type (A|U|R|F|N|B|G|T|X) #REQUIRED>
<!ELEMENT srcDestPremId (#PCDATA)>
 <!ATTLIST srcDestPremId
 type (N|X) #REQUIRED>
<!ELEMENT animal (DOB*,age*)>
 <!ATTLIST animal
 species (ACQ|BOV|CAM|CAP|CER|EQU|OVI|AVI|POR) #IMPLIED
 gender (M|F|C|S|X) #IMPLIED
 breed CDATA #IMPLIED
 >
<!ELEMENT DOB (timestamp)>
 <!ATTLIST DOB
 est (Y|N) #REQUIRED>
<!ELEMENT age (#PCDATA)>
 <!ATTLIST age
 scale (D|M|Y) #REQUIRED>
<!ELEMENT remarks (#PCDATA)>
<!ELEMENT reTagId (#PCDATA)>
 <!ATTLIST reTagId
 type (A|U|R|F|N|B|G|T|X) #REQUIRED>
<!ELEMENT optIds (optId*)>
<!ELEMENT optId (#PCDATA)>
 <!ATTLIST optId
 type (A|U|R|F|N|B|G|T|X) #REQUIRED>
<!ELEMENT group (groupSubsetId*, groupCount*)>
 <!ATTLIST group
 groupType CDATA #IMPLIED
 species (AQU|CLM|CRA|CTF|MSL|OYS|SAL|SBA|SHR|SLP|TIL|TRO|
 BOV|BIS|BEF|DAI|CAM|CAP|CER|DEE|ELK|EQU|OVI|
 AVI|CHI|DUC|GEE|GUI|PGN|PHE|QUA|TUR|OTH|POR) #IMPLIED
 breed CDATA #IMPLIED
 >
<!ELEMENT groupSubsetId (#PCDATA)>
<!ELEMENT groupCount (#PCDATA)>
<!ELEMENT timestamp EMPTY>
 <!ATTLIST timestamp
 y CDATA #REQUIRED
 mo (1|2|3|4|5|6|7|8|9|10|11|12) #REQUIRED
 d (1|2|3|4|5|6|7|8|9|
 10|11|12|13|14|15|16|17|18|19|
 20|21|22|23|24|25|26|27|28|29|30|31) #REQUIRED
 h24 (0|1|2|3|4|5|6|7|8|9|10|11|12|
 13|14|15|16|17|18|19|20|21|22|23) "0"
 mi (0|1|2|3|4|5|6|7|8|9|
 10|11|12|13|14|15|16|17|18|19|
 20|21|22|23|24|25|26|27|28|29|
 30|31|32|33|34|35|36|37|38|39|
 40|41|42|43|44|45|46|47|48|49|
 50|51|52|53|54|55|56|57|58|59) "0"
 s (0|1|2|3|4|5|6|7|8|9|
 10|11|12|13|14|15|16|17|18|19|
 20|21|22|23|24|25|26|27|28|29|
 30|31|32|33|34|35|36|37|38|39|
 40|41|42|43|44|45|46|47|48|49|
 50|51|52|53|54|55|56|57|58|59) "0"
 tz (GMT|GMT-1|GMT-2|GMT-3|GMT-4|GMT-5|GMT-6|
 GMT-7|GMT-8|GMT-9|GMT-10|GMT-11|GMT-12|
 GMT12|GMT11|GMT10|GMT9|GMT8|GMT7|GMT6|
 GMT5|GMT4|GMT3|GMT2|GMT1) #IMPLIED
 >
```

2.1.3.2.3 ATD Response XML Examples

- Example 1: Response indicating no records found

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE eventSub PUBLIC
\"http://localhost/atps/dtds/eventSub.dtd\"
\"http://localhost/atps/dtds/eventSub.dtd\">
<eventSub>
<header>
<atpsRequestId>12345</atpsRequestId>
<atdResponse final="Y">
 <responseId>32543</responseId>
</atdResponse>
</header>
<animalRecords>
</animalRecords>
</eventSub>
```

- Example 2: Response with one record, minimally populated

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE eventSub PUBLIC
\"http://localhost/atps/dtds/eventSub.dtd\"
\"http://localhost/atps/dtds/eventSub.dtd\">
<eventSub>
<header>
<atpsRequestId>12345</atpsRequestId>
<atdResponse final="Y">
 <responseId>32543</responseId>
</atdResponse>
</header>
<animalRecords>
```

```
<animalRecord>
 <eventType code="4"/>
 <eventDate>
 <timestamp y="2006" mo="9" d="25"/>
 </eventDate>
<rptPremId type="N">002GCNK</rptPremId>
 <id type="N">840002123456789</id>
</animalRecord>
</animalRecords>
</eventSub>
```

- Example 3: Response with one “fully” populated record. This is a Corrected moved out record for a cow. The cow has an official 840 ID, and also has an optional breed registry ID.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE eventSub PUBLIC
\"http://localhost/atps/dtds/eventSub.dtd\"
\"http://localhost/atps/dtds/eventSub.dtd\">
<eventSub>
<header>
<atpsRequestId>12345</atpsRequestId>
<atdResponse final="Y">
 <responseId>32543</responseId>
</atdResponse>
</header>
<animalRecords>
<animalRecord elecRead="Y" status="C">
 <eventType code="4"/>
<eventDate>
 <timestamp y="2006" mo="9" d="25"/>
 </eventDate>
 <rptPremId type="N">002GCNK</rptPremId>
 <id type="N">840002123456789</id>
 <srcDestPremId type="N">003FY38</srcDestPremId>
 <animal species="BOV" gender="M" breed="HB">
 <DOB est="Y">
 <timestamp y="2006" mo="9" d="25"/>
 </DOB>
 <age scale="M">5</age>
 </animal>
 <remarks>recorded by Joe</remarks>
 <optIds>
 <optId type="B">00T1234001</optId>
 </optIds>
</animalRecord>
</animalRecords>
</eventSub>
```

- Example 4: Response with two records, minimally populated

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE eventSub PUBLIC
\"http://localhost/atps/dtds/eventSub.dtd\"
\"http://localhost/atps/dtds/eventSub.dtd\">
<eventSub>
<header>
```

```
<atpsRequestId>12345</atpsRequestId>
<atdResponse final="Y">
 <responseId>32543</responseId>
</atdResponse>
</header>
<animalRecords>
<animalRecord>
 <eventType code="4"/>
 <eventDate>
 <timestamp y="2006" mo="9" d="25"/>
 </eventDate>
 <rptPremId type="N">002GCNK</rptPremId>
 <id type="N">840002123456789</id>
</animalRecord>
<animalRecord>
 <eventType code="4"/>
 <eventDate>
 <timestamp y="2006" mo="9" d="25"/>
 </eventDate>
 <rptPremId type="N">002GCNK</rptPremId>
 <id type="N">840002123456790</id>
</animalRecord>
</animalRecords>
</eventSub>
```

- Example 5a: Split response with one record, minimally populated, first split of 3. (Note, normally a split response that is not the final split contains 5,000 events.)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE eventSub PUBLIC
\"http://localhost/atps/dtds/eventSub.dtd\"
\"http://localhost/atps/dtds/eventSub.dtd\">
<eventSub>
<header>
<atpsRequestId>12345</atpsRequestId>
<atdResponse final="N" split="1">
 <responseId>32543</responseId>
</atdResponse>
</header>
<animalRecords>
<animalRecord>
 <eventType code="4"/>
 <eventDate>
 <timestamp y="2006" mo="9" d="25"/>
 </eventDate>
 <rptPremId type="N">002GCNK</rptPremId>
 <id type="N">840002123456789</id>
</animalRecord>
</animalRecords>
</eventSub>
```

- Example 5b: Split response with one record, minimally populated, second split of 3. (Note, normally a split response that is not the final split contains 5,000 events.)

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!DOCTYPE eventSub PUBLIC
\"http://localhost/atps/dtds/eventSub.dtd\"
\"http://localhost/atps/dtds/eventSub.dtd\">
<eventSub>
<header>
<atpsRequestId>12345</atpsRequestId>
<atdResponse final="N" split="2">
 <responseId>32543</responseId>
</atdResponse>
</header>
<animalRecords>
<animalRecord>
 <eventType code="4"/>
 <eventDate>
 <timestamp y="2006" mo="9" d="25"/>
 </eventDate>
 <rptPremId type="N">002GCNK</rptPremId>
 <id type="N">840002123456789</id>
</animalRecord>
</animalRecords>
</eventSub>
```

- Example 5c: Split response with one record, minimally populated, final split of 3

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE eventSub PUBLIC
\"http://localhost/atps/dtds/eventSub.dtd\"
\"http://localhost/atps/dtds/eventSub.dtd\">
<eventSub>
<header>
<atpsRequestId>12345</atpsRequestId>
<atdResponse final="Y" split="3">
 <responseId>32543</responseId>
</atdResponse>
</header>
<animalRecords>
<animalRecord>
 <eventType code="4"/>
 <eventDate>
 <timestamp y="2006" mo="9" d="25"/>
 </eventDate>
 <rptPremId type="N">002GCNK</rptPremId>
 <id type="N">840002123456789</id>
</animalRecord>
</animalRecords>
</eventSub>
```

2.1.3.3 Response Elements and Attributes

This section describes all of the elements and attributes in the response. Element/Attribute requirements and behavior are detailed as well.

Summary:

A Response DTD Element and Attribute summary chart is provided here. Each element and attribute is detailed at length below.

Element and Attribute Summary chart:

element	field required	format	size	description	notes
eventSub	Y	Elements	na	Root element	Contains header plus animal-Records or groupRecords elements
header	Y	Elements	na	Header root element	Conatins atpsRequestId and atdResponse elements
atpsRequestId	Y	Numeric	15	ATPS ID of the original request	ATPS will create a unique ID for each request. Clients must include this ID when submitting this response
atdResponse	Y	Elements	na	ATD response root element	
atdResponseId	Y	Alpha-numeric	20	ATD unique ID for the response	Value supplied by client; must be unique for all messages sent by a client, except for a split response.
atdResponse. final	Y	(Y|N)	1	Indicates if this is the final response	Always "Y" for non-split responses. Only "Y" for the last split response.
atdResponse. split	See notes	Numeric	9	Split response number	Starts with 1, increment by 1's. Only required if split response.
animalRecords	See notes	Elements	na	Animal records root element	animalRecords or groupRecords element must be present but may be empty. Contains 0 to many animalRecord elements
animalRecord	N	Elements	na	Single animal record root element	Represents a single animal event. Contains several elements corresponding to animal event data attributes
animalRecord. elecRead	N	(Y|N)	1	Is event electronically read	N=false, Y=true. No human intervention in recording process.
animalRecord. status	N	(C)	1	Event status	(C)orrected record.
groupRecords	See notes	Elements	na	Group records root element	animalRecords or groupRecords element must be present but may be empty. Contains 0 to many groupRecord elements
groupRecord	N	Elements	na	Single group record root element	Represents a single group event. Contains several elements corresponding to group event data attributes
groupRecord. elecRead	N	(Y|N)	1	Is event electronically read	N=false, Y=true. No human intervention in recording process.
groupRecord. status	N	(C)	1	Event status	(C)orrected record.
ATDEventId	N	Alpha-numeric	20	ATD’s unique event identifier	

eventType	Y	Empty	na	Event type element	Actual event type code is in the element's attribute so it can be validated by the DTD
eventType.code	Y	Numeric	2	Event type code	Refer to Requirements document for current list of valid event codes.
eventDate	Y	Elements	na	Date of the event	Contains a timestamp element (see below for definition of timestamp element)
rptPremId	Y	Alpha-numeric	7	Sighting/reporting national premises ID	National Premises refers to National Premises ID in registry
rptPremId.type	N	(N|X)	1	Indicated if the premises ID is a national premises ID or not	N=national, X=any other type.
id	Y	Alpha-numeric	17	Official ID	Official ID.
id.type	N	(A|U|R|F|N|B|G|T|X)	1	Official ID type	N="840" ID. See requirements for more information.
srcDestPremId	N	Alpha-numeric	7	Source/destination premises ID	Used for moved in and moved out event types
srcDestPremId.type	N	(N|X)	1	Indicated if the premises ID is a national premises ID or not	N=national, X=any other type.
animal	N	Elements	na	Contains animal information	Includes date of birth, age, species, gender, and breed
animal.species	N	Alpha-numeric	3	Animal species code	Refer to requirements document for current list of valid species codes
animal.gender	N	Alpha-numeric	1	Animal gender	Refer to requirements document for current list of valid gender codes
animal.breed	N	Alpha-numeric	2	Animal breed	Refer to requirements document for current list of valid breed codes
DOB	N	Elements	na	Animal date of birth	Contains a timestamp element (see below for definition of timestamp element)
DOB.est	See notes	(Y|N)	1	Is animal date of birth estimated?	Required if DOB element is present
age	N	Numeric	4	Animal age	Numeric qualifier for animal age. Only makes sense coupled with the scale attribute (see below). Assumed estimated.
age.scale	See notes	(D|M|Y)	1	“Scale” of the age value	D=Days, M=Months, Y=Years. Required if age element is present.
remarks	N	Alpha-numeric	50	Event remarks	Description/other comments on event. “sold to Stan Humphries” or herd management id.

reTagId	See notes	Alpha-numeric	17	Retagged animal ID	Required for tag replacement event (event ID 6). Forbidden otherwise. This is the OLD tag, not the new tag.
reTagId.type	N	Alpha-numeric	1	Retagged animal ID type	Required if re tag ID is present. Refer to requirements document for current list of valid re tag types
optIdRecords	N	Elements	na	Optional ID records root element	May be empty or missing altogether. Contains 0 to many optional official ID record elements
optIdRecord	N	Elements	na	Optional ID record single element	Represents a single optional ID (ID and type).
optId	N	Alpha-numeric	17	Optional ID	
optId.type	N	Alpha-numeric	1	Optional ID type	Required if optId element is present.
timestamp	See notes	Empty	na	Timestamp element	Used in two spots in the DTD. Required if eventDate element is present (so it's required), and also required if DOB element is present.
timestamp.y	Y	Numeric	4	Year	Must be 4 digit year
timestamp.mo	Y	Numeric 1-12	2	Month	1=January, etc.
timestamp.d	Y	Numeric 1-31	2	Day	
timestamp.h24	N	Numeric 0-23	2	Hour	Midnight=0, 1pm = 13, 11pm =23
timestamp.mi	N	Numeric 0-59	2	Minute	
timestamp.s	N	Numeric 0-59	2	Second	
timestamp.tz	See notes	GMT[1-12]|-1-12]	6	Timezone	If h24, mi, or s are populated it is optional. Value relative to GMT: round to nearest hour if in offset timezone. Examples: Mountain Daylight Ttime=GMT-6, Mountain Standard Time=GMT-7. Normalize to EST (GMT-5) if possible.
group	N	Elements	na	Contains group information	Includes species, breed, groupType attributes, and groupSubsetId and groupCount elements
group.type	N	alpha-numeric	3	Group type	Refer to requirements document for current list of valid group type codes
group.species	N	Alpha-numeric	3	Group species group code	Refer to requirements document for current list of valid species codes
group.breed	N	Alpha-numeric	2	Group breed	Refer to requirements document for current list of valid

					breed codes
groupSubsetId	N	Alpha-numeric	20	Group sub set ID	
groupCount	N	Numeric	10	Group count	

Detail:

Each element and attribute is detailed at length in this section.

2.1.3.3.1 eventSub Element

DTD:

```
<!ELEMENT eventSub (header,(animalRecords|groupRecords))>
```

Required: Yes

Description:

The eventSub element encapsulates the entire Response from the ATD. It contains a Header element and either an animalRecords element or a groupRecords element. It has no attributes. Only one eventSub pre response is allowed.

2.1.3.3.2 header Element

DTD:

```
<!ELEMENT header (atpsRequestId,atdResponse)>
```

Required: Yes

Description:

The header element encapsulates “header” information about the Response, namely the ATPS Request ID and the ATD Response elements.

2.1.3.3.3 atpsRequestId Element

DTD:

```
<!ELEMENT atpsRequestId (#PCDATA)>
```

Required: Yes

Format: Numeric, 15 digits max.

Description:

The atpsRequestId element contains the ID if the Request to which this is a Response. If this is a Split Response, all Splits will have the same Request ID. This is a required element so if it is not present, or if it is empty, or if it does not match a Request for the ATD that is in an ACTIVE or ACTIONABLE status, the Response will not be processed and the Request will go to an ERROR state.

2.1.3.3.4 atdResponse Element

DTD:

```
<!ELEMENT atdResponse (responseId)>
 <!ATTLIST atdResponse
 final (Y|N) #REQUIRED
 split CDATA #IMPLIED
 >
```

Required: Yes

Description:

The atdResponse element contains basic information about the Response. It consists of a responseId element, and two attributes: final, and split.

2.1.3.3.4.1 atdResponse.final Attribute

DTD:

```
atdResponse final (Y|N) #REQUIRED
```

Required: Yes

Valid values:

Y

N

Description:

The atdResposne.final Attribute indicates if the Response from the ATD is the final Response from them pertaining to a specific ATPS Request ID. The valid values are Y and N. If the ATD only has one response to a Request, the atdResposne.final attribute will be set to Y. If the ATD is submitting multiple responses to a single Request ID (a split response), then the atdResposne.final attribute will be N until the final split, then it will be Y.

In chart form:

Split Response?	*Last Split?*	*Final*
N	n/a	Y
Y	N	N
	Y	Y

2.1.3.3.4.2 atdResposne.split Attribute

DTD:

```
split CDATA #IMPLIED
```

Required: Yes, if ATD is responding with a split response, otherwise no

Format: Integer

Description:

The atdResponse.split attribute describes which “split number” is being submitted by the ATD for a particular ATPS Request. This attribute is required if the ATD is submitting a split response, otherwise it is not present. The split attribute can be thought of as the “page” of the response. The split is a numeric value.

In the event of a split response, the ATD will start numbering splits with 1, and count consecutively until it reached the final split. For example if an ATD is responding to a request with three split responses, the first split will have a split value of 1. The second will have a split value of 2, and the third and final split will have a split value of 3. The following chart further illustrates:

Split	Split Attribute	Final Attribute	ATPS Request ID Element	ATD Response ID Element
1	1	N	[12345]	[54321]
2	2	N	[12345]	[54321]
3	3	Y	[12345]	[54321]

2.1.3.3.5 responseId Element

DTD:

```
<!ELEMENT responseId (#PCDATA)>
```

Required: Yes

Format: String, max length 20 characters

Description:

The responseId Element is generated by the ATD. When responding to a Request, the ATD is required to supply a unique Response ID (except for Split Responses; see below). The ID may be numeric, alpha-numeric, or alpha. The Response ID may only contain numbers and letters (no special characters). The Response ID is case sensitive.

If an ATD is submitting split responses, each Split will have the same Response ID.

If an ATD is responding to the same Request multiple times, the ATD will supply a unique Response ID for each response.

Illustrated:

Non-split response:

Response Attempt	Split	Split Attribute	Final Attribute	ATPS Request ID Element	ATD Response ID Element
First attempt	n/a	n/a	Y	[12345]	[54321]
Second attempt	n/a	n/a	Y	[12345]	[54322]

Split response:

Response Attempt	Split	Split Attribute	Final Attribute	ATPS Request ID Element	ATD Response ID Element
First attempt	1	1	N	[12345]	[54321]
	2	2	N	[12345]	[54321]
	3	3	Y	[12345]	[54321]
Second attempt	1	1	N	[12345]	[54322]
	2	2	N	[12345]	[54322]
	3	3	Y	[12345]	[54322]

2.1.3.3.6 animalRecords Element

DTD:

```
<!ELEMENT animalRecords (animalRecord*)>
```

Required: Either animalRecords or groupRecords Element is required.

Description:

The animalRecords Element is a container for 0-to-many animalRecord Elements. Note the in the event that an ATD is responding to a Request, but the response is essentially that the ATD has no events matching the request criteria, then an animalRecord Element is required, but it will be empty.

2.1.3.3.7 animalRecord Element

DTD:

```
<!ELEMENT animalRecord (ATDEventId?*, eventType, eventDate, rptPre-
mId?, id, srcDestPremId?*, animal*, remarks*, reTagId?*, optIds*)>
<!ATTLIST animalRecord
elecRead CDATA #IMPLIED
status CDATA #IMPLIED
>
```

Strict DTD:

```
<!ELEMENT animalRecord (ATDEventId?*, eventType, eventDate, rptPre-
mId?, id, srcDestPremId?*, animal*, remarks*, reTagId?*, optIds*)>
<!ATTLIST animalRecord
elecRead (Y|N) #IMPLIED
status (C) #IMPLIED
>
```

Required: No

Description:

The animalRecord Element is a representation of a single animal event. From 0-to-many animalRecord Elements may be present in a single Response. ATPS recommends that an ATD only submit 5,000 animalRecord Elements in a single response. If the ATD needs to return more than 5,000 animalRecord Elements, they need to “split” the response into multiple Split Responses, none of which contain more than 5,000 animalRecord elements.

The animalRecord Element contains several sub-elements, as well as two Attributes. These are all discussed individually below.

2.1.3.3.7.1 animalRrecord.elecRead Attribute

DTD:

```
elecRead CDATA #IMPLIED
```

Strict DTD:

```
elecRead (Y|N) #IMPLIED
```

Required: No

Valid Values:

Y

N

Description:

The animalRecord.elecRead Attribute indicates if the event described by this animalRecord Element was completely “electronically” processed. If at any point between the collection of the data and the storage of the data in the ATD database was manually processed, the elecRead Attribute must be “N”. Otherwise it can be set to “Y”.

If the ATD does not know if the event was manually processed, this attribute should not be included in the Response.

Illustrated:

Data Processing Condition	XML
Completely automated (no manual intervention required)	<animalRecord elecRead = “Y”>
All or part manual processing	<animalRecord elecRead = “N”>
Unknown	<animalRecord>

2.1.3.3.8 animalRecord.status Attribute

DTD:

```
status CDATA #IMPLIED
```

Strict DTD:

```
status (C) #IMPLIED
```

Required: No

Valid Values:

C

Description:

The animalRecord.status Attribute is meant to indicate if the event record has been corrected at some point. The only valid value for this Attribute is “C”. If this is not a corrected event, the attribute should be left out entirely.

In the event that a particular record has been corrected, depending on how the ATD stores and processes data multiple “versions” of the same event may show up in the same response. ATPS prefers that only the most recent corrected event is returned. In the event that the ATD can not discriminate and filter out older non-corrected versions of the event, ATPS requests that the ATD populate the ATDEventId with the same event ID for each event. Furthermore, in the event that an event has been corrected multiple times, if the ATD is returning all events ATPS recommends that the ATD only add the status attribute to the most recently corrected event (it is assumed that the most recent event is also the most correct).

Examples:

Event ID	Corrected Record	Latest Correction?	Status Attribute	atdEventId Element
[54321]	N	N	<animalRecord>	<ATDEventId>54321</ATDEventId>
[54321]	Y	N	<animalRecord>	<ATDEventId>54321</ATDEventId>
[54321]	Y	Y	<animalRecord status=”C”>	<ATDEventId>54321</ATDEventId>

ATPS would prefer in this example that only the last (most recently processed) event is returned.

2.1.3.3.9 ATDEventId Element

DTD:

```
<!ELEMENT ATDEventId (#PCDATA)>
```

Required: Only required in the event the ATD is returning the same event multiple times with corrections in the same response.

Format: String, max length 50 characters

Description:

This is an optional element that the ATD can include in their response events. It is a key for the Event that is internal to the ATD. ATPS has no restrictions on the format of the ATDEventId other than it can not be longer than 50 characters.

The ATDEventId is required if the ATD is responding with multiple versions of the same event, one or more of which are corrected versions of the event. In this scenario, ATPS prefers that the ATD simply return the latest corrected version of the event. In that case the ATDEventId not required. In the event the ATD can not return only the latest corrected version of the event and must return all versions of the event, ATPS requires that the ATS supply an identical ATDEventID with each event so that ATPS can determine the "most correct" version of the event.

Otherwise, ATPS will use the ATDEventId to provide more useful information about data validation errors to the ATD. For example if a particular ATD event contains an invalid species code, ATPS will include the fact that the species code for event response sequence "##" was invalid. If the ATD also includes its event ID for that particular record, then ATPS will also include the event ID as well as the sequence number. Then the ATD will be able to identify the invalid data item by the ATD Event ID, and not simply by the event sequence.

For Example, if the third response record contains a species element with a value of "ZZZ", but no ATD Event ID, then ATPS will return the following invalid item information:

InvalidItem attribute	*value*
ATDEventId	null
recordSequence	2
elementname	species
elementvalue	ZZZ
reason	[invalid value]

However if the ATD populates the event ID with its own event ID, then the response will include that value in the ATDEventID attribute:

InvalidItem attribute	*value*
ATDEventId	800005883942
recordSequence	2
elementname	species
elementvalue	ZZZ
reason	[invalid value]

2.1.3.3.10 eventType Element

DTD:

```
<!ELEMENT eventType EMPTY>
<!ATTLIST eventType
code CDATA #REQUIRED >
```

Strict DTD:

```
<!ELEMENT eventType EMPTY>
<!ATTLIST eventType
code (0|1|2|3|4|5|6|7|8|9|10|11|12|13) #REQUIRED >
```

Required: Yes

Valid values: Numbers 0 through 13

Description:

The eventType is one of the major "4" required values (the others being event date, reporting premises, and id). The eventType describes what sort of event is represented in the record. Currently 14 event types are allowed in ATPS. The event type is received as a numeric code. In order to validate the code value, the XML DTD specifies that the event type code be submitted as an attribute of the eventType. If the event type element is not present or the ID attribute is not present, ATPS will send the entire submission to an ERROR state since this XML will not validate against the standard DTD. If the event type element is present but the value is not valid, then the event will fail data validation but processing will continue.

XML Example	*ATPS Response*	*Reason*
[No eventType element]	ERROR	eventType element is required.
<eventType/>	ERROR	eventType.code attribute is required.
<eventType></eventType>	ERROR	eventType.code attribute is required.
<eventType>4</eventType>	ERROR	eventType element must be empty.
<eventType><code>4</code></eventType>	ERROR	eventType element must be empty.
<eventType code=""/>	VALIDATION ERROR	ID value not valid.
<eventType code="99"/>	VALIDATION ERROR	ID value not valid.
<eventType code="4"/>	VALIDATED	

2.1.3.3.10.1 eventType.code Attribute

DTD:

```
<!ATTLIST eventType
code CDATA #REQUIRED >
```

Strict DTD:

```
<!ATTLIST eventType
code (0|1|2|3|4|5|6|7|8|9|10|11|12|13) #REQUIRED >
```

Required: Yes

Valid values: Numeric integers between 0 and 13.

Description:

The eventType.code attribute describes the event type. This is defined as an attribute so that the Strong DTD can validate the content. ATPS will not put the entire request into ERROR if an event type code is not valid. This scenario will put the request into the VALIDATION ERROR state, and the eventType will be added to the invalid items list for the request.

The following is a list of valid ATPS event type codes and descriptions:

Event Code	Description
0	Tag Shipped – not applied to animal
1	Tag Allocated
2	Tag applied – tag is applied to an animal
3	Moved in – Animal is moved into a premises
4	Moved out – Animal is moved out of a premises
5	Lost Tag – New tag is applied to an animal that lost a tag and previous tag is unknown
6	Replaced Tag or Re-Tagged – New tag is applied to an animal that lost a tag and previous tag is known
7	Imported – Animal is imported into the U.S.
8	Exported – Animal is exported out of the U.S.
9	Sighting – Animal has a confirmed sighting at a location, no movement has occurred. (Ex: veterinarian sighting)
10	Harvested – Animal was terminated at an abattoir
11	Died – Animal died of natural causes or euthanized at the farm/ranch
12	Tag retired – Tag retired by producer, packing house, etc.
13	Animal Missing (lost stolen, etc)

2.1.3.3.11 eventDate Element

DTD:

```
<!ELEMENT eventDate (timestamp)>
```

Required: Yes

Description:

The eventDate Element describes the date when the event occurred. This element does not describe the date when the record was added to the database, although they may often be the same date. eventDate is a required element. It contains a timestamp element that describes the date and time values via attributes.

Examples:

XML Example	ATPS Response	Reason
[No eventDate element]	ERROR	eventDate element is required.
<eventDate/>	ERROR	timestamp element is required.
<eventDate></eventDate>	ERROR	timestamp element is required.
<eventDate>01-JUN-2005</eventDate>	ERROR	eventDate element must only be a timestamp element.
<eventDate>20050601</eventDate>	ERROR	eventDate element must only be a timestamp element.
<eventDate><timestamp></timestamp></eventDate>	ERROR	timestamp element is missing required attributes.
<eventDate><timestamp y="2006" mo="9" d"25" tz="GMT-6"/> </eventDate>	VALIDATED	

2.1.3.3.12 rptPremId Element

DTD:

```
<!ELEMENT rptPremId (#PCDATA)>
<!ATTLIST rptPremId
type CDATA #REQUIRED>
```

Strict DTD:

```
<!ELEMENT rptPremId (#PCDATA)>
<!ATTLIST rptPremId
type (N|X) #REQUIRED>
```

Required: Yes

Valid values: Depends on the type of premises.

Description:

rptPremId is the Reporting Premises ID for the event. ATPS allows the ATD to supply a National Premises ID or another type of premises ID (typically a state-supplied premises ID). This element is required and will cause the request to go into a status of ERROR if it is not present. If the element is empty or not valid, the Request will go to VALIDATION ERROR. If the premises is a national premises, ATPS will validate that it is a real National Premises ID, and send the Request to VALIDATION ERROR if it is not; non-national premises IDs will not be validated.

The ATD will supply a type attribute that indicates if the premises is a national premises, or some other type of premises.

2.1.3.3.12.1 rptPremId.type Attribute

DTD:

```
<!ATTLIST rptPremId
type CDATA #IMPLIED>
```

Strict DTD:

```
<!ATTLIST rptPremId
type (N|X) #REQUIRED>
```

Required: Yes

Valid values:

Reporting Premises ID Type	*Description*
N	USDA National Premises ID
X	Any Other type of premises ID

Description:

The rptPremId.type attribute describes the type of reporting premises ID that is included in the event record. This attribute is not required but will result in an VALIDATION_ERROR if it is not present. The only allowable values for the type attribute are “N” and “X”. The ATD will use “N” to indicate a National Premises ID, and an “X” for any other type of premises ID.

Examples:

XML Example	*ATPS Response*	*Reason*
[No rptPremId element]	ERROR	rptPremId element is required.
<rptPremId></rptPremId>	ERROR	rptPremId element is required—cannot be empty/null.
<rptPremId>002GNCK</rptPremId>	VALIDATION ERROR	Missing type attribute.
<rptPremId type="">002GNCK </rptPremId>	VALIDATION ERROR	Type attribute cannot be empty.
<rptPremId type="USDA">002GNCK </rptPremId>	VALIDATION ERROR	Type attribute must be "N" or "X".
<rptPremId type="N">002GNC </rptPremId>	VALIDATION ERROR	National premises ID does not validate. 7 digits are required.
<rptPremId type="N">NATLPREMID </rptPremId>	VALIDATION ERROR	National premises ID does not validate.
<rptPremId type="N">002GNCK </rptPremId>	VALIDATED	
<rptPremId type="X">002GNCK </rptPremId>	VALIDATED	ATPS does not validate non-national premises IDs ("X" type).

2.1.3.3.13 id Element

DTD:

```
<!ELEMENT id (#PCDATA)>
 <!ATTLIST id
 type CDATA #IMPLIED>
```

Strict DTD:

```
<!ELEMENT id (#PCDATA)>
 <!ATTLIST id
 type (A|U|R|F|N|B|G|T|X) #REQUIRED>
```

Required: Yes

Description:

The ID is the "official" animal identification for the event. It is recognized that a particular animal may have multiple identifications, but only one may be used as the official identifier. The ID element does not have to be a USDA "840" number only. An official ID may also include a manufacturer code, a USA tag code, or another country official code. The ATD will supply a type attribute to indicate which type of official ID it is.

The ID element is required and the Request will go into ERROR state if it is not present. If the ID element is a USDA "840" number, ATPS will validate that the tag has been shipped to a producer. If it has not, the Request will go to VALIDATION ERROR. Other ID types will not be validated.

2.1.3.3.13.1 id.type Attribute

DTD:

```
 <!ATTLIST id
 type CDATA #IMPLIED>
```

Strict DTD:

```
 <!ATTLIST id
 type (A|U|R|F|N|B|G|T|X) #REQUIRED>
```

Required: Yes

Valid Values: See the Official ID Codes Appendix for valid values.

Description:

The id.type attribute qualifies the type of Official ID included in the event. The Type attribute is required; lack of inclusion will result in the request going to a VALIDATION_ERROR status. The type must be one of the types from the list above.

Examples:

XML Example	*ATPS Response*	*Reason*
[No id element]	ERROR	id element is required.
<id></id>	ERROR	id element is required—cannot be empty/null.
<id>840002123456789</id>	VALIDATION ERROR	Missing type attribute.
<id type="">840002123456789</id>	VALIDATION ERROR	Type attribute cannot be empty.
<id type="USDA">840002123456789 </id>	VALIDATION ERROR	Type attribute must be from standard list
<id type="N">2123456789</id>	VALIDATION ERROR	NAIS "840" ID does not validate. Must be 15 digits.
<id type="N">NATLPREMID</id>	VALIDATION ERROR	NAIS "840" ID does not validate. Must be 15 digits.
<id type="N">840002123456789 </id>	VALIDATED	
<id type="X">840002123456789</id>	VALIDATED	ATPS does not validate non-national premises IDs ("X" type).

2.1.3.3.15 srcDestPremId Element

DTD:

```
<!ELEMENT srcDestPremId (#PCDATA)>
<!ATTLIST srcDestPremId
type CDATA #REQUIRED>
```

Strict DTD:

```
<!ELEMENT srcDestPremId (#PCDATA)>
<!ATTLIST srcDestPremId
type (N|X) #REQUIRED>
```

Required: Yes

Valid values: Depends on the type of premises.

Description:

srcDestPremId is the Source or Destination Premises ID for the event. Not all events or event types have a source or destination premises ID, and it is an optional element. ATPS allows the ATD to supply a national premises ID or another type of premises ID (typically a state-supplied premises ID). If the premises is a national premises, ATPS will validate that it is a real National Premises ID, and send the Request to VALIDATION ERROR if it is not; non-national premises IDs will not be validated.

The ATD will supply a type attribute that indicates if the premises is a national premises, or some other type of premises.

2.1.3.3.15.1 srcDestPremId.type Attribute

DTD:

```
<!ATTLIST srcDestPremId
type CDATA #REQUIRED>
```

Strict DTD:

```
<!ATTLIST srcDestPremId
type (N|X) #REQUIRED>
```

Required: Yes

Valid values:

Source/Destination Premises ID Type	Description
N	USDA National Premises ID
X	Any Other type of premises ID

Description:

The srcDestPremId.type attribute describes the type of reporting premises ID that is included in the event record. Even though the srcDestPremId is not required, if it is included, its type attribute is required and will result in an ERROR if it is not present. The only allowable values for the type attribute are “N” and “X”. The ATD will use “N” to indicate a National Premises ID, and an “X” for any other type of premises ID. Any other value will result in a VALIDATION_ERROR status.

Examples:

XML Example	ATPS Response	Reason
[no srcDestPremId element]	VALIDATED	srcDestPremId is optional.
<srcDestPremId></srcDestPremId>	VALIDATED	srcDestPremId is optional.
<srcDestPremId>002GNCK </srcDestPremId>	VALIDATION ERROR	Missing type attribute.
<srcDestPremId type="">002GNCK </srcDestPremId>	VALIDATION ERROR	Type attribute cannot be empty.
<srcDestPremId type="USDA">002GNCK </srcDestPremId>	VALIDATION ERROR	Type attribute must be “N” or “X”.
<srcDestPremId type="N">002GNC </srcDestPremId>	VALIDATION ERROR	National premises ID does not validate. 7 digits are required.
<srcDestPremId type="">NATLPREMID </srcDestPremId>	VALIDATION ERROR	National premises ID does not validate.
<srcDestPremId type="N">002GNCK </srcDestPremId>	VALIDATED	
<srcDestPremId type="X">002GNCK </srcDestPremId>	VALIDATED	ATPS does not validate non-national premises IDs (“X” type).

2.1.3.3.16 animal Element

DTD:

```
<!ELEMENT animal (DOB*,age*)>
 <!ATTLIST animal
 species CDATA #IMPLIED
 gender CDATA #IMPLIED
 breed CDATA #IMPLIED
 >
```

Strict DTD:

```
<!ELEMENT animal (DOB*,age*)>
 <!ATTLIST animal
 species (ACQ|BOV|CAM|CAP|CER|EQU|OVI|AVI|POR) #IMPLIED
 gender (M|F|C|S|X) #IMPLIED
 breed CDATA #IMPLIED
 >
```

Required: No

Description:

The animal element describes optional information about the actual animal in the event. The element contains two optional "sub" elements, and three optional attributes. There really is no data validation on the Animal Element, but the sub elements and attributes have some validation.

XML Example	ATPS Response	Reason
[animal element not present	VALIDATED	Animal element not required.
<animal/>	VALIDATED	Animal element can be empty.
<animal></animal>	VALIDATED	Animal element can be empty.

2.1.3.3.16.1 animal.species Attribute

DTD:

```
species CDATA #IMPLIED
```

Strict DTD:

```
 species (AQU|CLM|CRA|CTF|MSL|OYS|SAL|SBA|SHR|SLP|TIL|TRO|
 BOV|BIS|BEF|DAI|CAM|CAP|CER|DEE|ELK|EQU|OVI|
 AVI|CHI|DUC|GEE|GUI|PGN|PHE|QUA|TUR|OTH|POR) #IMPLIED
```

Required: No

Valid values:

See the Species Code Appendix for all valid Species values:

Description:

The species attribute describes the general species or species group of the animal. If the species attribute is present but empty, or is not on the list above, the Request will go to a VALIDATION ERROR state.

Examples:

XML Example	ATPS Response	Reason
<animal/>	VALIDATED	Species attribute not required.
<animal species=""/>	VALIDATION ERROR	Species attribute cannot be empty.
<animal species="BIS"></animal>	VALIDATION ERROR	Species attribute must be from the approved list of official types.
<animal species="BOV"></animal>	VALIDATED	

2.1.3.3.16.2 animal.gender Attribute

DTD:

```
gender CDATA #IMPLIED
```

Strict DTD:

```
gender (M|F|C|S|X) #IMPLIED
```

Required: No

Valid values:

Gender Code	Description
M	Male
F	Female
C	Neutered / castrated male
S	Neutered / spayed female
X	Mixed (used only in groups)

Description:

The optional gender attribute describes the gender of the animal in the event. If the gender attribute is present but empty, or is not in the list above, the Request will go to a VALIDATION ERROR status.

Examples:

XML Example	ATPS Response	Reason
<animal/>	VALIDATED	Gender attribute not required.
<animal gender=""/>	VALIDATION ERROR	Gender attribute cannot be empty.
<animal gender="MALE"></animal>	VALIDATION ERROR	Gender attribute must be from the approved list of official types.
<animal species="M"></animal>	VALIDATED	

2.1.3.3.16.3 animal.breed Attribute

DTD

```
breed CDATA #IMPLIED
```

Strict DTD:

```
breed CDATA #IMPLIED
```

Required: No

Valid values*:

(Please see the appendix for the recognized breed values.)

* Values not validated in ATPS

Description:

The breed attribute is an optional description of the animal breed. If the breed attribute is present but empty, the Request will go to a VALIDATION ERROR status. ATPS will not validate the breed value against a list of breeds because the list is subject to frequent change.

Examples:

XML Example	*ATPS Response*	*Reason*
<animal/>	VALIDATED	Breed attribute not required.
<animal breed=""/>	VALIDATION ERROR	Breed attribute cannot be empty.
<animal breed="ZZ"></animal>	VALIDATED	ATPS will not validate the breed value against the valid list.
<animal breed="PH"></animal>	VALIDATED	

2.1.3.3.17 DOB Element:

DTD

```
<!ELEMENT DOB (timestamp)>
 <!ATTLIST DOB
 est CDATA #REQUIRED>
```

Strict DTD:

```
<!ELEMENT DOB (timestamp)>
 <!ATTLIST DOB
 est (Y|N) #REQUIRED>
```

Required: No

Format: Timestamp

Description:

The "DOB" Element is the Date of Birth of the animal. It is described by a timestamp Element and an "est" attribute, which indicates if the DOB value is an estimate or not. It is an optional attribute, although the timestamp element and est attribute are required if the DOB element is present.

Examples:

XML Example	*ATPS Response*	*Reason*
[no DOB element]	VALIDATED	DOB element is optional.
<DOB></DOB> (or) <DOB/>	ERROR	est attribute is required.
<DOB est="N"></DOB>	ERROR	timestamp element is required.
<DOB><timestamp/></DOB>	ERROR	est attribute is required.
<DOB est="Y"><timestamp y="2006" mo="9" d="25" h24="13" tz="GMT-6"/></DOB>	VALIDATED	

2.1.3.3.17.1 DOB.est Attribute

DTD:

```
DOB est CDATA #REQUIRED
```

Strict DTD:

```
DOB est (Y|N) #REQUIRED
```

Required: No

Valid Values:

Y

N

Description:

The Date of Birth (DOB) estimated (est) attribute indicates if the animal date of birth is an estimated value or not. If it is not known if the DOB is an estimate, use “Y”. This is a required attribute if the DOB Element is present, so it will result in an ERROR status if it is not populated and the DOB element exists. If the value is empty or invalid, it will cause a VALIDATION ERROR status.

Examples:

XML Example	ATPS Response	Reason
<DOB><timestamp y=”2006” mo=”9” d=”25” h24=”13” tz=”GMT-6”/></DOB>	ERROR	est element is required.
<DOB est=””><timestamp y=”2006” mo=”9” d=”25” h24=”13” tz=”GMT-6”/></DOB>	VALIDATION ERROR	est attribute cannot be empty.
<DOB est=”yes”><timestamp y=”2006” mo=”9” d=”25” h24=”13” tz=”GMT-6”/></DOB>	VALIDATION ERROR	est attribute is not valid.
<DOB est=”Y”><timestamp y=”2006” mo=”9” d=”25” h24=”13” tz=”GMT-6”/></DOB>	VALIDATED	

2.1.3.3.18 age Element

Strict DTD:

```
<!ELEMENT age (#PCDATA)>
 <!ATTLIST age
 scale CDATA #REQUIRED>
```

Strict DTD:

```
<!ELEMENT age (#PCDATA)>
 <!ATTLIST age
 scale (D|M|Y) #REQUIRED>
```

Required: No

Format: Numeric

Description:

The Age element describes the age of the animal in days, months, or years. This is supposed to be the age of the animal when the event occurred, not the age of the animal when the Request is made. This element is optional, but if it is not numeric, the Request will go into VALIDATION ERROR status. The age element contains a scale attribute that indicate if the element value number represents Days, Months, or Years. It is assumed that this is an estimated value so there is not need to add an estimated attribute.

Only one scale qualifier is allowed. If the animal is known to be 3 years and 6 months, indicate that the animal is 30 months old. Day precision is not necessary; if the animal is one month old or older, even if the age of the animal is known to the day, round the age to the nearest month.

Logic Examples:

Known Age of Animal	XML
1 day	<age scale=”D”>1</age>
30 days	<age scale=”D”>30</age>
1 month	<age scale=”M”>1</age>
1 month, 1 day	<age scale=”M”>1</age>

1 month, 15 days	<age scale="M">1</age>
1 month, 16 days	<age scale="M">2</age>
11 months, 30 days	<age scale="M">12</age>
1 year	<age scale="Y">1</age>
1 year, 1 month, 1 day	<age scale="M">13</age>
1 year, 6 months	<age scale="M">18</age>
1 year, 11 months, 30 days	<age scale="M">24</age>

XML Examples:

XML Example	*ATPS Response*	*Reason*
[No age element]	VALIDATED	age element is optional.
<age></age> (or) <age/>	ERROR	scale attribute is required.
<age scale="Y"></age>	VALIDATION ERROR	age element cannot have empty value.
<age scale="Y">3 years</age>	VALIDATION ERROR	age element must be numeric.
<age scale="Y">03</age>	VALIDATED	
<age scale="Y">3</age>	VALIDATED	

2.1.3.3.18.1 age.scale Attribute

DTD:

```
age scale CDATA #REQUIRED
```

Strict DTD:

```
age scale (D|M|Y) #REQUIRED
```

Required: Yes, if age Element is present

Valid Values:

Age Code	*Description*
D	Days
M	Months
Y	Years

Description:

The age.scale attribute qualifies the age element value in terms of Days, Months, or Years. If the age element is present, the scale attribute is required, and if the attribute is not present in this case it will result in an ERROR status for the request. If the scale attribute is empty or invalid, the request will go to a VALIDATION ERROR status.

Examples:

XML Example	*ATPS Response*	*Reason*
<age>2</age>	ERROR	scale element is required.
<age scale="">2</age>	VALIDATION ERROR	scale attribute cannot be empty.
<age scale="years">2</age>	VALIDATION ERROR	scale attribute is not valid.
<age scale="Y">2</age>	VALIDATED	

2.1.3.3.19 remarks Element

DTD:

```
<!ELEMENT remarks (#PCDATA)>
```

Required: No

Format: 50 characters max.

Description:

The remarks element can contain additional information about the event. “Sold to Stan Humphries” or a herd management ID may be placed here. Any additional information that may help animal disease officials with their search can be put in this element. It is optional so the element can be left out, and it can be empty. If it is longer than 50 characters, ATPS will truncate it and put the request into VALIDATION ERROR

Examples:

XML Example	*ATPS Response*	*Reason*
[no remarks element]	VALIDATED	remarks element is optional.
<remarks></remarks>	VALIDATED	remarks element can be empty.
<remarks/>	VALIDATED	remarks element can be empty.
<remarks>Sold to Stan Hum-pries></remarks>	VALIDATED	
<remarks>Sold to Stan Humphries on Tuesday, August 17th at the McDonald Sale Barn</remarks>	VALIDATION ERROR	remarks element value must be 50 characters or less. Value will be truncated.

2.1.3.3.20 reTagId Element

DTD:

```
<!ELEMENT reTagId (#PCDATA)>
 <!ATTLIST reTagId
 type CDATA #REQUIRED>
```

Strict DTD:

```
<!ELEMENT reTagId (#PCDATA)>
 <!ATTLIST reTagId
 type (A|U|R|F|N|B|G|T) #REQUIRED>
```

Required: No

Format: 17 characters maximum

Description:

The reTagId element is only used with a retagging event type (event type ID 6). If the event type is any other type, the reTagId element should not be present, and will be ignored, and will also cause the request to go into a VALIDATION ERROR status. The reTagId is the OLD tag. The new tag should be in the ID element.

The reTagId element contains a type attribute that the ATD will use to indicate what type of tag the old tag was.

If the ID element is a USDA “840” number, ATPS will validate that the tag has been shipped to a producer. If it has not, the Request will go to VALIDATION ERROR. Other ID types will not be validated.

2.1.3.3.20.1 reTagId.type attribute

DTD:

```
reTagId type CDATA #REQUIRED>
```

Strict DTD:

```
reTagId type (A|U|R|F|N|B|G|T) #REQUIRED>
```

Required: No

Valid Values: See the Official ID Codes Appendix for valid values.

Description:

The reTagId.type attribute qualifies the type of replaced Official ID included in the event. If the reTagId element is present, the reTagId.type attribute is required; lack of inclusion will result in the request going to a VALIDATION_ERROR status. The type must be one of the types from the list above.

Examples:

XML Example (Event type ID=6 unless specified.)	*ATPS Response*	*Reason*
[no reTagId element], eventType id <>6	VALIDATED	reTypeId forbidden if event type is not “retagged”.
[no reTagId element]	VALIDATION ERROR	reTypeId required if event type is “retagged”.
<reTagId type=”N”>840002123456789 </reTagId>, eventType id <>6	VALIDATION ERROR	reTypeId forbidden if event type is not “retagged”.
<reTagId>840002123456789</reTagId>	VALIDATION ERROR	Missing type attribute.
<reTagId type=””>840002123456789 </reTagId>	VALIDATION ERROR	type attribute cannot be empty.
<reTagId type=”USDA”> 840002123456789</reTagId>,	VALIDATION ERROR	type attribute must be from standard list.
<reTagId type=”N”>2123456789 </reTagId>	VALIDATION ERROR	NAIS “840” reTagId does not validate. Must be 15 digits.
<reTagId type=”N”>NATLPREMID </reTagId>	VALIDATION ERROR	NAIS “840” reTagId does not validate. Must be 15 digits.
<reTagId type=”N”>840002123456789 </reTagId>	VALIDATED	
<reTagId type=”X”>840002123456789 </reTagId>	VALIDATED	ATPS does not validate non-national premises IDs (“X” type).

2.1.3.3.21 optIds Element

DTD:

```
<!ELEMENT optIds (optId*)>
```

Required: No

Description:

The (Optional IDs) optIds element allows the ATD to indicate other IDs that may be associated to the animal. In no scenario does the Optional ID take the place of the ID element. The ID element must always be included. The optIds element is optional so it does not have to be present. It consists of zero to many optId elements, so it can be empty.

Examples:

XML Example	*ATPS Response*	*Reason*
[No optIds element]	VALIDATED	optIds is not required.
<optIds/>	VALIDATED	optIds can be empty.
<optIds></optIds>	VALIDATED	optIds can be empty.
<optIds>840003123456789</optIds>	ERROR	optIds can only contain optId elements.

2.1.3.3.22 optId Element

DTD:

```
<!ELEMENT optId (#PCDATA)>
 <!ATTLIST optId
 type CDATA #REQUIRED>
```

Strict DTD:

```
<!ELEMENT optId (#PCDATA)>
 <!ATTLIST optId
 type (A|U|R|F|N|B|G|T) #REQUIRED>
```

Required: No

Format: 17 characters maximum

Description:

The optId element describes the optional ID that is assigned to the animal.

The optId element contains a type attribute that the ATD will use to indicate the optional tag type.

If the ID element is a USDA “840” number, ATPS will validate that the tag has been shipped to a producer. If it has not, the Request will go to VALIDATION ERROR. Other ID types will not be validated.

2.1.3.3.22.1 optId.type Attribute

DTD:

```
optId type CDATA #REQUIRED>
```

Strict DTD:

```
optId type (A|U|R|F|N|B|G|T) #REQUIRED>
```

Required: No

Valid Values: See the Official ID Codes Appendix for valid values.

Description:

The optId.type attribute qualifies the type of replaced AIN ID included in the event. If the optId element is present, the optId.type attribute is required; lack of inclusion will result in the request going to an VALIDATION_ERROR status. The type must be one of the types from the list above.

Examples:

XML Example	*ATPS Response*	*Reason*
<optId>840002123456789</optId>	VALIDATION ERROR	Missing type attribute.
<optId type="">840002123456789	VALIDATION ERROR	Type attribute cannot be empty.

</optId>		
<optId type="USDA"> 840002123456789</optId>	VALIDATION ERROR	Type attribute must be from standard list
<optId type="N">2123456789</optId>	VALIDATION ERROR	NAIS "840" ID does not validate. Must be 15 digits.
<optId type="N">NATLPREMID </optId>	VALIDATION ERROR	NAIS "840" ID does not validate. Must be 15 digits.
<optId type="N">840002123456789 </optId>	VALIDATED	
<optId type="X">840002123456789 </optId>	VALIDATED	ATPS does not validate non-national premises IDs ("X" type).

2.1.3.3.23 timestamp Element

DTD:

```
<!ELEMENT timestamp EMPTY>
<!ATTLIST timestamp
y CDATA #REQUIRED
mo CDATA #REQUIRED
d CDATA #REQUIRED
h24 CDATA "0"
mi CDATA "0"
s CDATA "0"
tz CDATA #REQUIRED
>
```

Strict DTD:

```
<!ELEMENT timestamp EMPTY>
<!ATTLIST timestamp
y CDATA #REQUIRED
mo (1|2|3|4|5|6|7|8|9|10|11|12) #REQUIRED
d (1|2|3|4|5|6|7|8|9|
10|11|12|13|14|15|16|17|18|19|
20|21|22|23|24|25|26|27|28|29|30|31) #REQUIRED
h24 (0|1|2|3|4|5|6|7|8|9|10|11|12|
13|14|15|16|17|18|19|20|21|22|23) "0"
mi (0|1|2|3|4|5|6|7|8|9|
10|11|12|13|14|15|16|17|18|19|
20|21|22|23|24|25|26|27|28|29|
30|31|32|33|34|35|36|37|38|39|
40|41|42|43|44|45|46|47|48|49|
50|51|52|53|54|55|56|57|58|59) "0"
s (0|1|2|3|4|5|6|7|8|9|
10|11|12|13|14|15|16|17|18|19|
20|21|22|23|24|25|26|27|28|29|
30|31|32|33|34|35|36|37|38|39|
40|41|42|43|44|45|46|47|48|49|
50|51|52|53|54|55|56|57|58|59) "0"
tz (GMT|GMT-1|GMT-2|GMT-3|GMT-4|GMT-5|GMT-6|
GMT-7|GMT-8|GMT-9|GMT-10|GMT-11|GMT-12|
GMT12|GMT11|GMT10|GMT9|GMT8|GMT7|GMT6|
GMT5|GMT4|GMT3|GMT2|GMT1) #REQUIRED
>
```

Required: Required as part of eventDate Element. It is also required as part of DOB Element although the DOB element itself is not required.

Description:

The timestamp Element describes a Date field down to the second, and includes provision for the time zone as well. Since XML 1.0 does not have a date data type, dates must be submitted as strings. The only way to validate a string using DTD is via element attribution. Therefore in order to ensure that valid dates are passed into ATPS, the actual date value of the timestamp is passed in solely as attributes of the element. The element itself is empty.

The attributes are divided into year (y), month (mo), day (d), hour (h24), minute (mi), second (s), and time zone (tz). The hour, minute, and second attributes are optional. The time zone attribute is not optional even though the timestamp element may not have precision below the day value.

XML Example	*ATPS Response*	*Reason*
<timestamp></timestamp>	ERROR	timestamp element has required attributes.
<timestamp/>	ERROR	timestamp element has required attributes.
<timestamp>01-JUN-2005 </timestamp>	ERROR	timestamp element must be empty.
<timestamp>20050601</timestamp>	ERROR	timestamp element must be empty.
<timestamp y=”2006” mo=”9” d=”25” tz=”GMT-6”/>	VALIDATED	

The next sections describe the various attributes in more detail, and provide some examples.

2.1.3.3.23.1 *timestamp.y Attribute*

DTD:

```
y CDATA #REQUIRED
```

Strict DTD:

```
y CDATA #REQUIRED
```

Required: Yes

Valid Values: A 4-digit year is expected. “2006” for example.

Description:

This attribute describes the Year of the timestamp. It is a 4-digit year value. Because of the enormous possible year values this attribute is not validated by the Strict DTD. If the year attribute is not present the normal DTD will fail validation and result in an ERROR request status, but an invalid value will result in a VALIDATION ERROR request status.

XML Example	*ATPS Response*	*Reason*
<timestamp y=”2006” mo=”9” d=”25” tz=”GMT-6” />	VALIDATED	
<timestamp y=”” mo=”9” d=”25” tz=”GMT-6” />	VALIDATION ERROR	(y)ear cannot be empty.
<timestamp y=”06” mo=”9” d=”25” tz=”GMT-6” />	VALIDATION ERROR	(y)ear must be 4-digit year.

<timestamp y="MMVI" mo="9" d="25" tz="GMT-6" />	VALIDATION ERROR	(y)ear is invalid format.
<timestamp mo="9" d="25" tz="GMT-6" />	ERROR	(y)ear is a required attribute.

2.1.3.3.23.2 timestamp.mo Attribute

DTD:

```
mo CDATA #REQUIRED
```

Strict DTD:

```
mo (1|2|3|4|5|6|7|8|9|10|11|12) #REQUIRED
```

Required: Yes

Valid Values: Numeric, 1-12.

Description:

The "mo" attribute describes the Month of the timestamp. It is required and will cause an ERROR request status if it is not present. If it is invalid, the Request will go to a DATA VALIDATION status. Valid values are 1-12 inclusive.

XML Example	ATPS Response	Reason
<timestamp y="2006" mo="" d="25" tz="GMT-6" />	VALIDATION ERROR	(mo)nth cannot be empty.
<timestamp y="2006" mo="15" d="25" tz="GMT-6" />	VALIDATION ERROR	(mo)nth invalid format.
<timestamp y="2006" mo="June" d="25" tz="GMT-6" />	VALIDATION ERROR	(mo)nth invalid format.
<timestamp y="2006" d="25" tz="GMT-6" />	ERROR	(mo)nth is a required attribute.

2.1.3.3.23.3 timestamp.d Attribute

DTD:

```
d CDATA #REQUIRED
```

Strict DTD:

```
d (1|2|3|4|5|6|7|8|9|
10|11|12|13|14|15|16|17|18|19|
20|21|22|23|24|25|26|27|28|29|30|31) #REQUIRED
```

Required: Yes

Valid Values: Numeric, 1-31.

Description:

The "d" attribute describes the Day of the timestamp. It is required and will cause an ERROR request status if it is not present. If it is invalid, the Request will go to a DATA VALIDATION status. Valid values are 1-31 inclusive.

XML Example	ATPS Response	Reason
<timestamp y="2006" mo="9" d="" tz="GMT-6" />	VALIDATION ERROR	(d)ay cannot be empty.
<timestamp y="2006" mo="9" d="33" tz="GMT-6" />	VALIDATION ERROR	(d)ay invalid format.
<timestamp y="2006" mo="9" d="31"	VALIDATION ERROR	(d)ay invalid format. (mo)nth

tz="GMT-6" />		"9" only has 30 days.
<timestamp y="2006" mo="9" d="25th" tz="GMT-6" />	VALIDATION ERROR	(d)ay invalid format.
<timestamp y="2006" mo="9" tz="GMT-6" />	ERROR	(d)ay is a required attribute.

2.1.3.3.23.4 timestamp.h24 Attribute

DTD:

```
h24 CDATA"0"
```

Strict DTD:

```
h24 (0|1|2|3|4|5|6|7|8|9|10|11|12|
13|14|15|16|17|18|19|20|21|22|23) "0"
```

Required: No. Defaults to "0"

Valid Values: Numeric, 0-23

Description:

The "h24" attribute describes the Hour of the timestamp, in 24 hour format. It is not required. If it is not present, ATPS will assume "0" for the hour value. If h24 is invalid, the Request will go to a DATA VALIDATION status. Valid values are 0-23 inclusive.

XML Example	*ATPS Response*	*Reason*
<timestamp y="2006" mo="9" d="25" h24="13" tz="GMT-6" />	VALIDATED	
<timestamp y="2006" mo="9" d="25" h24="24" tz="GMT-6" />	VALIDATION ERROR	(h24) hour is invalid format.
<timestamp y="2006" mo="9" d="25" h24="13:00" tz="GMT-6" />	VALIDATION ERROR	(h24) hour is invalid format.

2.1.3.3.23.5 timestamp.mi Attribute

DTD:

```
mi CDATA"0"
```

Strict DTD:

```
mi (0|1|2|3|4|5|6|7|8|9|
10|11|12|13|14|15|16|17|18|19|
20|21|22|23|24|25|26|27|28|29|
30|31|32|33|34|35|36|37|38|39|
40|41|42|43|44|45|46|47|48|49|
50|51|52|53|54|55|56|57|58|59) "0"
```

Required: No. Defaults to "0"

Valid Values: Numeric, 0-59

Description:

The "mi" attribute describes the Minute of the timestamp. It is not required. If it is not present, ATPS will assume "0" for the hour value. If mi is invalid, the Request will go to a DATA VALIDATION status. Valid values are 0-59 inclusive.

XML Example	*ATPS Response*	*Reason*
<timestamp y="2006" mo="9" d="25"	VALIDATED	

h24="13" mi="45" tz="GMT-6" />		
<timestamp y="2006" mo="9" d="25" mi="45" tz="GMT-6" />	VALIDATED	It validates. The assumption is that it is 12:45 a.m. (0:45).
<timestamp y="2006" mo="9" d="25" h24="13" mi="96" tz="GMT-6" />	VALIDATION ERROR	(mi)nute is invalid format.

2.1.3.3.23.6 timestamp.s Attribute

DTD:

```
s CDATA"0"
```

Strict DTD:

```
s (0|1|2|3|4|5|6|7|8|9|
10|11|12|13|14|15|16|17|18|19|
20|21|22|23|24|25|26|27|28|29|
30|31|32|33|34|35|36|37|38|39|
40|41|42|43|44|45|46|47|48|49|
50|51|52|53|54|55|56|57|58|59) "0"
```

Required: No. Defaults to "0"

Valid Values: Numeric, 0-59

Description:

The "s" attribute describes the Second of the timestamp. It is not required. If it is not present, ATPS will assume "0" for the hour value. If s is invalid, the Request will go to a DATA VALIDATION status. Valid values are 0-59 inclusive.

XML Example	ATPS Response	Reason
<timestamp y="2006" mo="9" d="25" h24="13" mi="45" s="59" tz="GMT-6" />	VALIDATED	
<timestamp y="2006" mo="9" d="25" s="59" tz="GMT-6" />	VALIDATED	It validates. The assumption is that it is 12:00:59 a.m.
<timestamp y="2006" mo="9" d="25" h24="13" mi="45" s="61" tz="GMT-6" />	VALIDATION ERROR	(s)econd is invalid format.

2.1.3.3.23.7 timestamp.tz Attribute

DTD:

```
tz CDATA #IMPLIED
```

Strict DTD:

```
tz (GMT|GMT-1|GMT-2|GMT-3|GMT-4|GMT-5|GMT-6|
GMT-7|GMT-8|GMT-9|GMT-10|GMT-11|GMT-12|
GMT12|GMT11|GMT10|GMT9|GMT8|GMT7|GMT6|
GMT5|GMT4|GMT3|GMT2|GMT1) #IMPLIED
```

Required: Yes, if hour, minute, or second are populated, no otherwise.

Valid Values: GMT, GMT[1-12], GMT[-1- -12]

Description:

This describes the time zone of the timestamp. It is optional but ATPS recommends that ATDs include the time zone attribute if the precision of the timestamp is at the hourly level or smaller.

The time zone always relative to GMT. If the time zone format is invalid, the Request will go to a DATA VALIDATION status. Valid values include “GMT” and GMT plus or minus 1 to 12 hours. If the time zone is not exactly one hour away from GMT, round to the nearest hour. The DTD specification does not allow plus signs “+” in attribute validations, so the absence of a minus sign between the GMT and the numeric qualifier implies a plus sign.

If possible, please normalize timestamps to the EST (GMT-5) time zone.

XML Example	ATPS Response	Reason
<timestamp y=”2006” mo=”9” d=”25” tz=””/>	VALIDATION ERROR	(tz)timezone cannot be empty.
<timestamp y=”2006” mo=”9” d=”25” tz=”CMT”/>	VALIDATION ERROR	(tz)timezone invalid format.
<timestamp y=”2006” mo=”9” d=”25” tz=”GMT+18”/>	VALIDATION ERROR	(tz)timezone invalid format.
<timestamp y=”2006” mo=”9” d=”25” tz=”GMT18”/>	VALIDATION ERROR	(tz)timezone invalid format.
<timestamp y=”2006” mo=”9” d=”25”/>	VALIDATED	tz not needed if precision is day.
<timestamp y=”2006” mo=”9” d=”25” tz=”GMT-5”/>	VALIDATED	Although it is not required.
<timestamp y=”2006” mo=”9” d=”25” h24=”13” tz=”GMT-5”/>	VALIDATED	tz is not required.
<timestamp y=”2006” mo=”9” d=”25” h24=”13”/>	VALIDATED	tz is not required.

TODO: describe group elements

2.1.3.4 Response ERROR Conditions

General:

ATPS will do some pre-checking of the ATD Response before forwarding it to the JMS queue for processing. In general it will check for two things: XML DTD Validation (this includes required elements) and a valid ATPS Request ID.

If the Response fails either case, an Exception will be thrown and the Response will not be processed. If possible (see below), ATPS will update the status of the Request for which this is a Response to ERROR. Otherwise the Exception will be the only way the ATPS will know for sure that there was an error with the Response. Indirectly the ATD will know based on the fact that they submitted a response yet the Request Status did not change.

2.1.3.4.1 XML DTD Validation Errors

If the XML fails DTD validation, the Response is not processed and an exception is thrown. ATPS will inspect the XML for the ATPS Request ID element, and attempt to validate the Request ID in order to update the status to ERROR.

If the Request ID can not be found or is not valid, no Request will be updated and the original exception will be thrown.

Special Case – XML missing ATPS Request ID element

If the atpsRequestID element is totally missing from the XML, then ATPS does not know which Request to update and will not update any Request. Of course this condition will be caught by the DTD validation since the request ID is a required XML element.

ATPS will indicate that the Request ID is missing.

2.1.3.4.2 ATPS Request ID Validation Errors

If the ATPS Request ID can not be identified, or is not a valid request ID, then the Request can not be set to ERROR – either ATPS does not know what Request this is a response to, or it is not appropriate to modify the Request since it is either STATIC, or another ATDs Request. In this case the Exception thrown by ATPS is the only way the ATD knows that the Response was not processed.

There are several ATPS Request ID error cases. In each case ATPS will not update any Request records to ERROR, it will throw an exception only.

- ATPS Request ID Error Case – Unknown ATPS Request ID element:

If the element exists but is not a known Request ID, then ATPS does not know which Request to update and will not update any Request.

ATPS will indicate only that the Request ID is not valid.

- ATPS Request ID Error Case – Other ATD ATPS Request ID element:

If the element exists but is a Request ID for another ATD ATPS will not update any Request.

ATPS will indicate only that the Request ID is not valid.

- ATPS Request ID Error Case – Closed Case ATPS Request ID element:

If the element exists but is a Request ID for Closed Case ATPS will not update any Request.

ATPS will indicate only that the Request ID is not valid.

- ATPS Request ID Error Case – Static Status ATPS Request ID element:

If the element exists but is a Request ID for a Request in a Static Status, ATPS will not update any Request.

ATPS will indicate only that the Request ID is not valid.

2.1.3.5 Split Responses

ATPS supports split responses to a single Request. The ATD is responsible for determining if a split response is necessary, and formatting the split response so that ATPS knows that the response is part of a split.

There are two response XML attributes that ATPS uses to determine if the response is split: the ATPSResponse.split attribute and the ATPSresponse.final attribute. See the documentation related to these two attributes for more information on setting them.

The ATD will order the split responses by the split attribute. This will be a numeric sequence starting at 1, and ending on the last split submitted. All of the splits will have the same ATDResponseID and ATPSRequestID values. The ATD will indicate the final split by setting the final attribute to “Y”. This attribute will be “N” for all the other splits. ATPS will know that the ATD is finished submitting splits when it receives the “Y” final split.

The splits must be consecutively numbered from 1 to n, n being the final split. For example a split response containing 4 splits will have split values of 1, 2, 3, and 4, with 4 having a final attribute values of “Y”, and the other three having a final attribute value of “N”. ATPS will accept the splits in any order.

ATPS recommends that the ATD submit only 5,000 events in a single response. So for example, if the request from ATPS results in 5,001 events being returned from the ATD, then ATPS would expect at a minimum two splits from the ATD, whose sum total of events equal 5,001 events, and non of which contain over 5,000 events.

The ATD may submit splits that are smaller than 5,000 events. The ATD may even submit a split that contains no events, if for some reason they need to do that.

2.1.3.6 National Premises ID Request Response Logic

In general, the strategy for responding to National Premises ID requests is to return events that indicate that an animal may be at the National Premises during the time period specified in the Request. The word “may” is very important; the ATD is expected to employ “optimistic” inventory logic. In other words if there is implicit evidence that an animal may have been at the premises during the requested data range, even though there is not explicit evidence proving that it was, the ATD is expected to return the Event that implies inventory. Very often this will mean that the ATD will return events that occurred before or after the requested date range.

Event Categories

It is important to understand four different Event Categories when analyzing if an event should be returned to ATPS. The Event categories are: Future Positive, Future Negative, Past Positive, and Past Negative. All Event Codes are either Future Positive or Future Negative, and either Past Positive or Past Negative.

2.1.3.6.1 Future Positive Events

A Future Positive Event Code is an event that implies presence on and after the event date. For example, a Moved In Event implies that the animal may be present after the Moved In Event, and therefore it is a Future Positive Event. It does not guarantee that the animal is at the location after the event. Future Positive Events include the following event types (codes):

Future Positive Event Codes:

Event Code	Description
0	Tag Shipped – not applied to animal
1	Tag Allocated
2	Tag applied – tag is applied to an animal
3	Moved in – Animal is moved into a premises
5	Lost Tag – New tag is applied to an animal that lost a tag and previous tag is unknown
6	Replaced Tag or Re-Tagged – New tag is applied to an animal that lost a tag and previous tag is known
7	Imported – Animal is imported into the U.S.
9	Sighting – Animal has a confirmed sighting at a location, no movement has occurred. (Ex: veterinarian sighting)

2.1.3.6.2 Future Negative Events

A Future Positive Event Code is an event that implies no presence on and after the event date. For example, a Moved Out Event implies that the animal may not be present at the location after the Moved Out Event, and therefore it is a Future Negative Event. It does not guarantee that the animal was never at the location after the event (it may move back). Future Negative Events include the following event types (codes):

Future Negative Event Codes:

Event Code	Description
4	Moved out – Animal is moved out of a premises
8	Exported – Animal is exported out of the U.S.
10	Harvested – Animal was terminated at an abattoir
11	Died – Animal died of natural causes or euthanized at the farm/ranch
12	Tag retired – Tag retired by producer, packing house, etc.
13	Animal Missing (lost stolen, etc)

2.1.3.6.3 Past Positive Events

A Past Positive Event Code is an event that implies presence before and on the event date. For example, a Moved Out Event implies that the animal may be present before the Moved Out Event, and therefore it is a Past Positive Event. It does not guarantee that the animal was always at the location before the event. Past Positive Events include the following event types (codes):

Past Positive Event Codes:

Event Code	Description
2	Tag applied – tag is applied to an animal
4	Moved out – Animal is moved out of a premises
5	Lost Tag – New tag is applied to an animal that lost a tag and previous tag is unknown
6	Replaced Tag or Re-Tagged – New tag is applied to an animal that lost a tag and previous tag is known
8	Exported – Animal is exported out of the U.S.
9	Sighting – Animal has a confirmed sighting at a location, no movement has occurred. (Ex: veterinarian sighting)
10	Harvested – Animal was terminated at an abattoir
11	Died – Animal died of natural causes or euthanized at the farm/ranch
12	Tag retired – Tag retired by producer, packing house, etc.
13	Animal Missing (lost stolen, etc)

2.1.3.6.4 Past Negative Events

A Past Negative Event Code is an event that implies no presence before the event date. For example, a Moved In Event implies that the animal was not present at the location before the Moved In Event, and therefore it is a Past Negative Event. It does not guarantee that the animal was never at the location before the event, but this event implies that the animal came from another premises. Past Negative Events include the following event types (codes):

Past Negative Event Codes:

ID	Description

0	Tag Shipped – not applied to animal
1	Tag Allocated
3	Moved in – Animal is moved into a premises
7	Imported – Animal is imported into the U.S.

All events are either future positive or negative, and either past positive or negative. This chart illustrates each event type and where it lies relative to positive/negative and past/future. Each event type appears in the chart two times, and each event appears once in the Future row, and once in the Past row:

Future		Past	
Positive	Negative	Positive	Negative
0 Shipped			0 Shipped
1 Allocated			1 Allocated
2 Applied		2 Applied	
3 Move In			3 Move In
	4 Move Out	4 Move Out	
5 Lost		5 Lost	
6 Replaced		6 Replaced	
7 Imported			7 Imported
	8 Exported	8 Exported	
9 Sighting		9 Sighting	
	10 Harvested	10 Harvested	
	11 Died	11 Died	
	12 Retired	12 Retired	
	13 Missing	13 Missing	

Usage:

There are a lot of use cases which determine if the ATD should or should not return an event, based on when the event occurred relative to the date range, and what type of event it was, and in some cases if there is another event that may provide more information about the animal. The examples below should illustrate all the possible use cases.

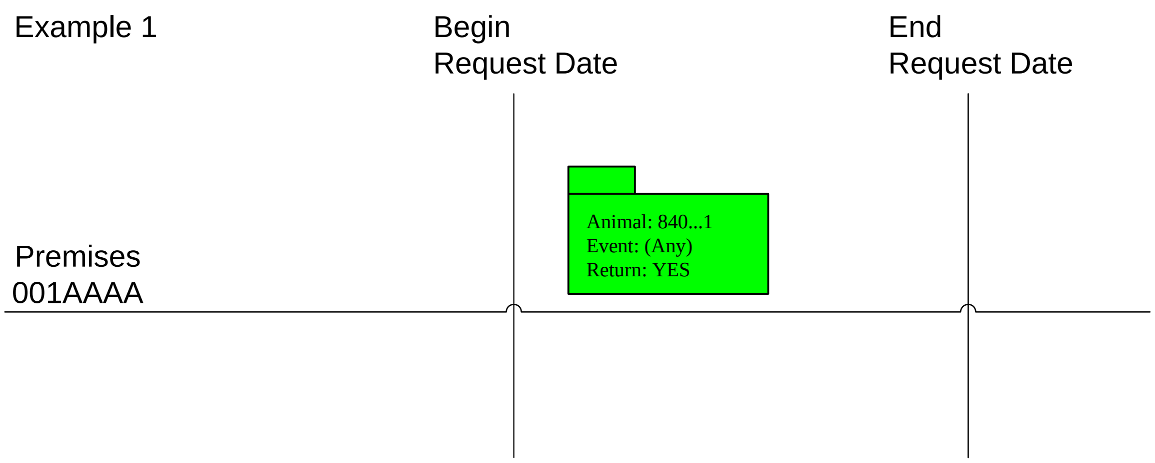
In all of these examples, ATPS is requesting information for National Premises "001AAAA" between specified beginning and ending dates. In the examples the ATD may have one or more events for a single animal "840…1" at 001AAAA. In reality, the ATD may have events for several hundred animals for the premises but one animal will suffice for the examples.

- Example 1:

ATD Scenario: Single Event; Any Event during Date Range

ATD Response: Return the event.

ATD has an event for the animal during the date range. This is the simplest example. In this example, no matter what the event type, the ATD will return the event.

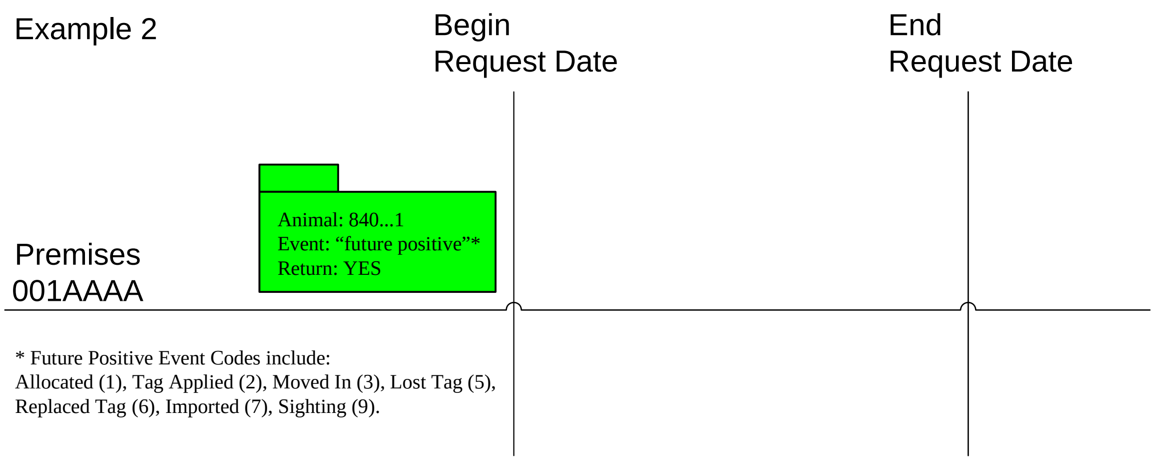


- Example 2:

ATD Scenario: Single Event; Future Positive Event before Date Range

ATD Response: Return the Event.

ATD has one event for the animal before the date range. Depending on the event type, the ATD should return the event or not return the Event. If the event type is a Positive Event, the ATD will return the Event.

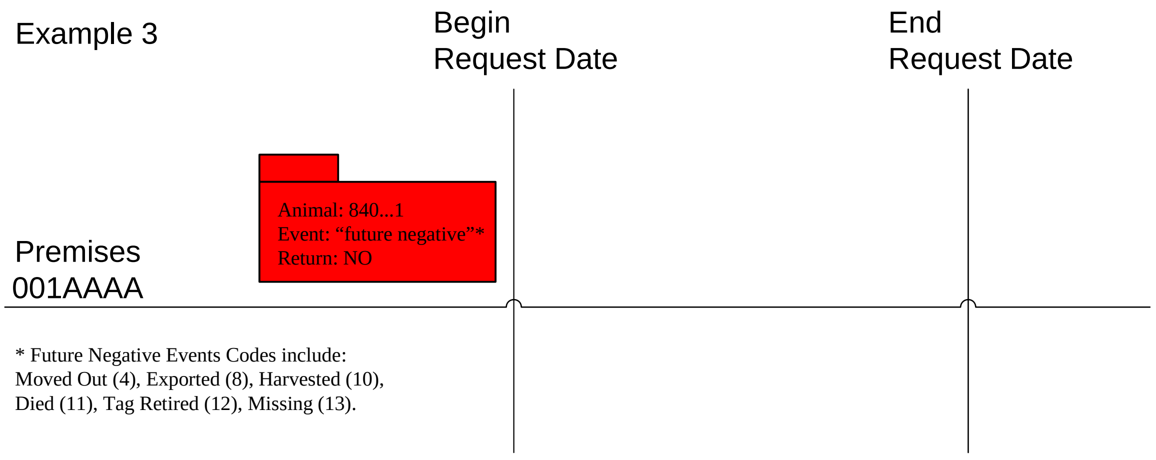

Example 2
Begin
Request Date
End
Request Date
Animal: 840...1
Event: "future positive"*
Return: YES
Premises
001AAAA
* Future Positive Event Codes include:
Allocated (1), Tag Applied (2), Moved In (3), Lost Tag (5),
Replaced Tag (6), Imported (7), Sighting (9).

- Example 3

ATD Scenario: Single Event; Future Negative Event before Date Range

ATD Response: Do not return the Event.

ATD has one event for the animal before the date range. Depending on the event type, the ATD should return the event or not return the Event. If the event type is a Negative Event, the ATD will return the Event.

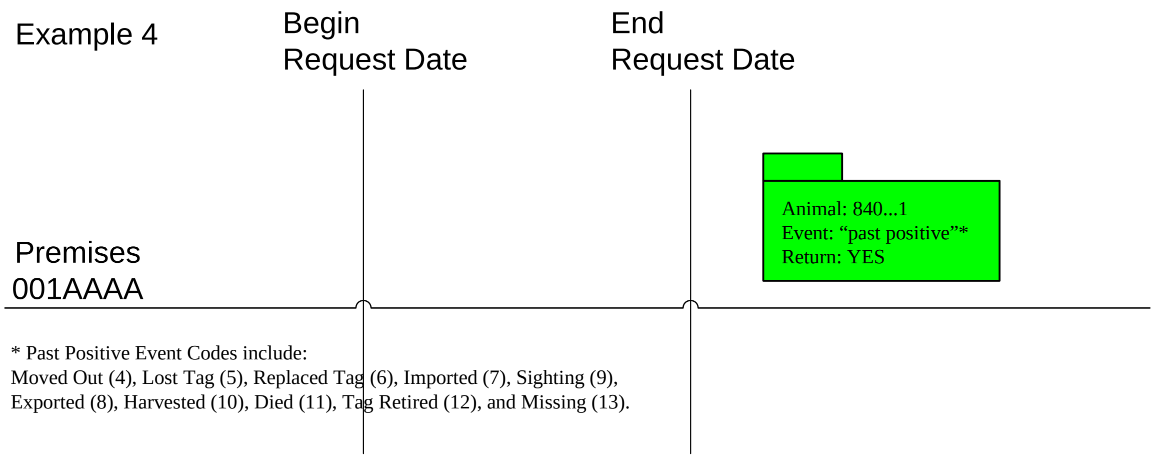

Example 3
Begin Request Date
End Request Date
Animal: 840...1
Event: “future negative”*
Return: NO
Premises 001AAAA
* Future Negative Events Codes include:
Moved Out (4), Exported (8), Harvested (10),
Died (11), Tag Retired (12), Missing (13).

- Example 4

ATD Scenario: Single Event; Past Positive Event after Date Range

ATD Response: Return the event.

ATD has one event for the animal after the date range. Depending on the event type, the ATD should return the event or not return the Event. If the event type is a “Past Positive” Event (indicating prior presence is possible), then the ATD will return the Event.

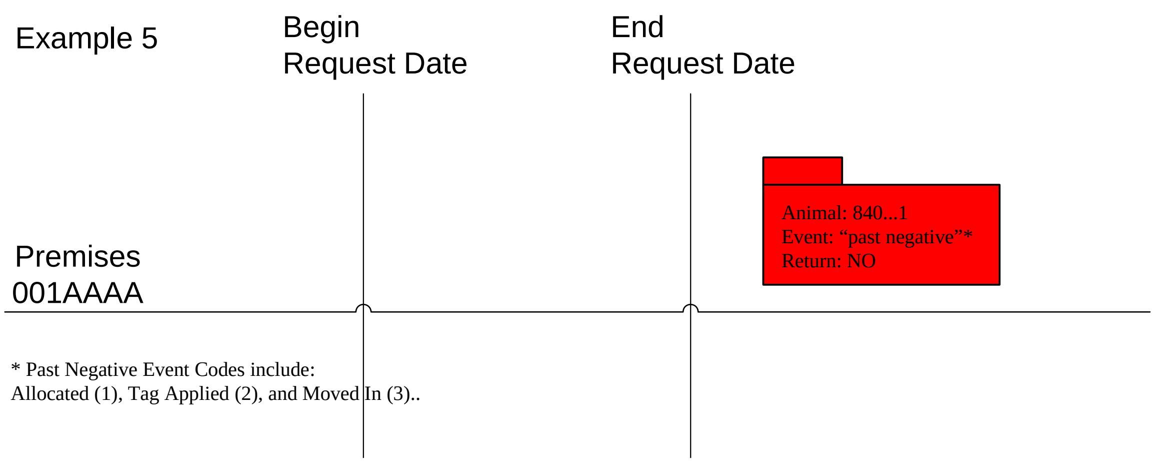

Example 4
Begin Request Date
End Request Date
Animal: 840...1
Event: “past positive”*
Return: YES
Premises 001AAAA
* Past Positive Event Codes include:
Moved Out (4), Lost Tag (5), Replaced Tag (6), Imported (7), Sighting (9),
Exported (8), Harvested (10), Died (11), Tag Retired (12), and Missing (13).

- Example 5

ATD Scenario: Single Event; Past Negative Event after Date Range

ATD Response: Do not return the event.

ATD has one event for the animal after the date range. Depending on the event type, the ATD should return the event or not return the Event. If the event type is a “Past Negative” Event (indicating prior presence is not possible), then the ATD will return the Event. Past Negative Events include the following event types (codes): Allocated (1), Tag Applied (2), and Moved In (3).



Multiple Events, Same Premises:

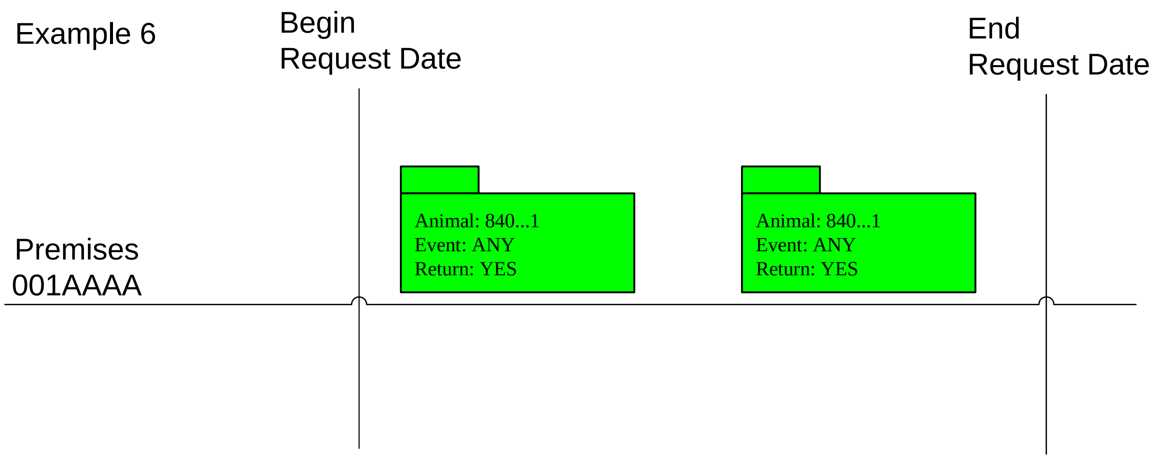
The ATD may have multiple events for the same animal at the same premises. In this case, some simple rules may be applied to determine if the ATD should or should not return the Event.

If the event is inside of the Request Date range, return the event.

- Example 6

ATD Scenario: Multiple Events inside of date range

ATD Response: Return all events.

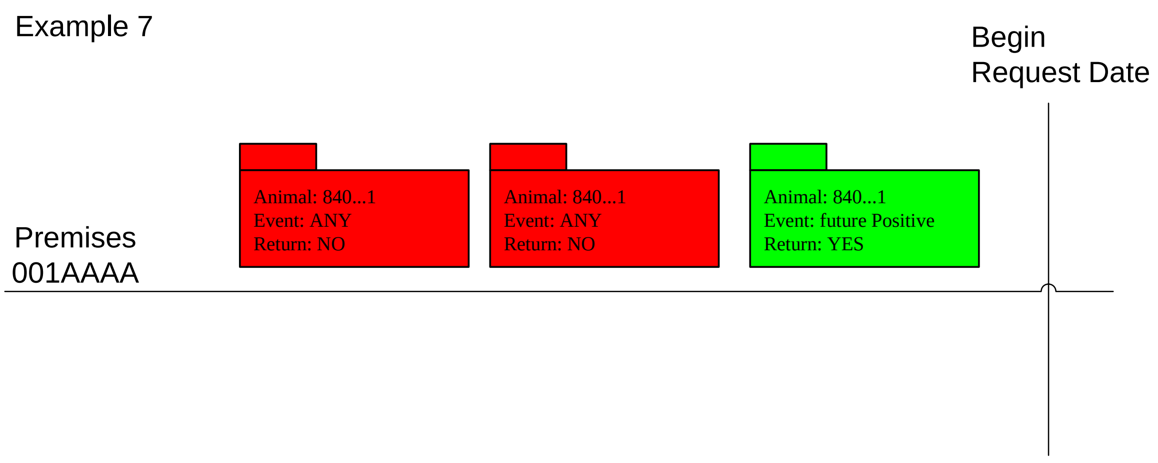


If there are multiple events before the Begin Request Date Range, sort the events most recent to oldest (i.e. start with the Event closest to the Request Begin Date and work backwards in time). If the Event closest to the Begin Request Date is a Future Positive Event, return the Event. Do not return any older events. If the most recent event is a Future Negative Event, do not return any of the “old” events.

- Example 7

ATD Scenario: Multiple Events before Begin Date. Latest Event is Future Positive.

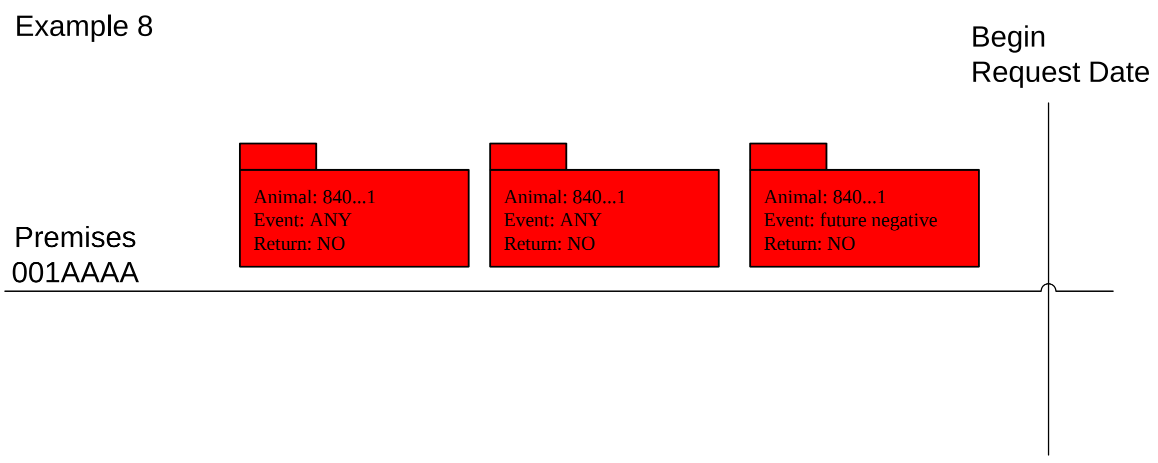
ATD Response: Return Latest Event only.



- Example 8

ATD Scenario: Multiple Events before Begin Date. Latest Event is Future Negative.

ATD Response: Return no events.

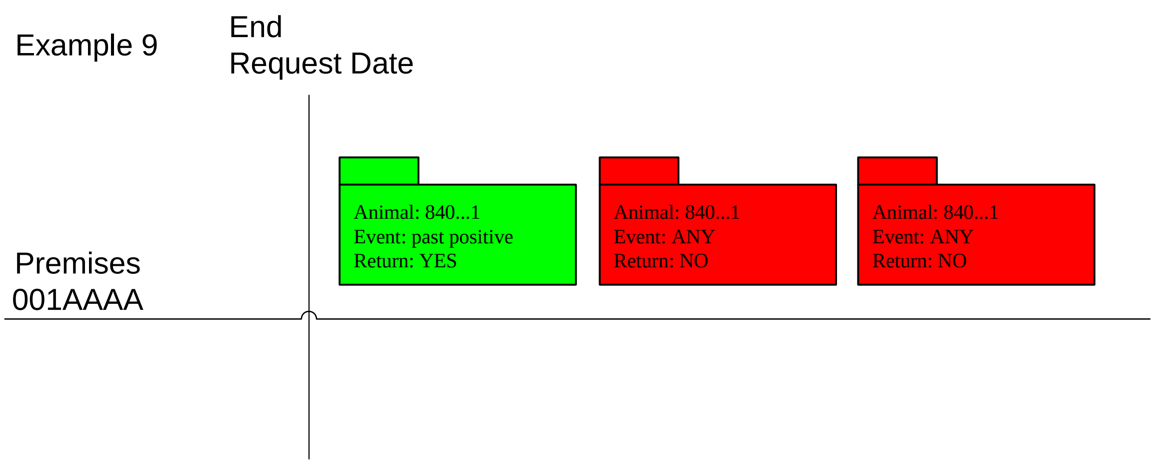


If there are multiple events after the End Request Date Range, sort the events oldest to most recent (i.e. start with the event closest to the Request End Date and work forwards in time). If the Event closest to the End Request Date is a Past Positive Event, return the event. Do not return any newer events. If the most recent event is a Past Negative Event, do not return any of the "new" events.

- Example 9

ATD Scenario: Multiple Events after End Date. Oldest Event is Past Positive.

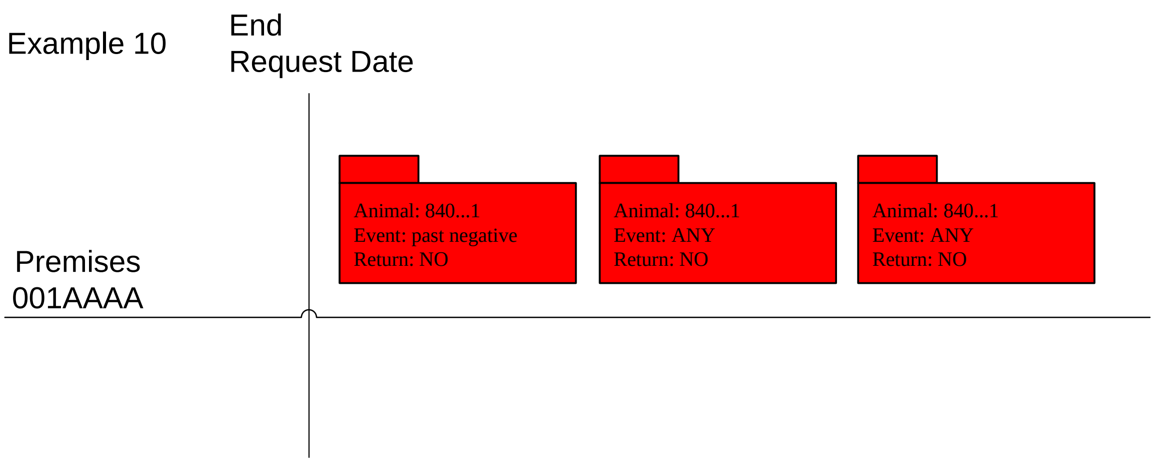
ATD Response: Return Oldest Event only.



- Example 10

ATD Scenario: Multiple Events after End Date. Oldest Event is Past Negative.

ATD Response: Do not return any events.



Multiple Events, Multiple Premises:

In certain cases an ATD may deduce that an animal was not at the requested premises by the presence of an event for that animal at other premises.

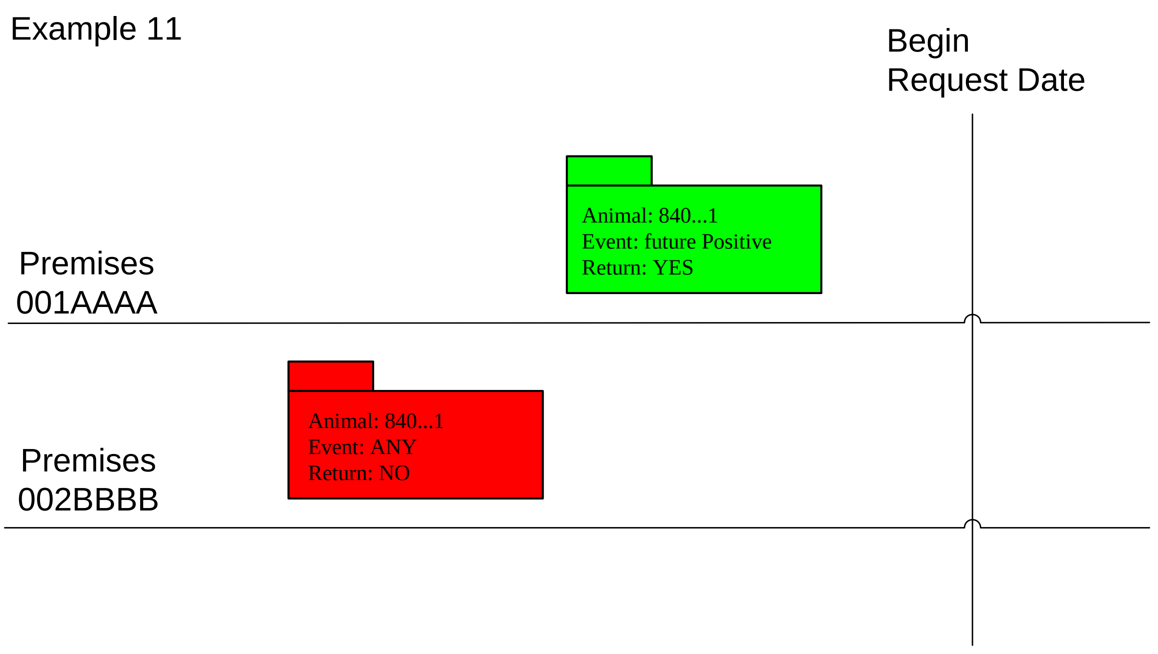
The situation can only occur if both events are either before or after the Request Date Range.

If the ATD has a Future Positive Event before the date range, normally it would return the event. However if the ATD also has ANY Event for the same animal at another Premises after the Future Positive event, but still before the date range, then the ATD will not return the Future Positive event.

- Example 11

ATD Scenario: Multiple Events before begin date. Newer Event is Future Positive in Premises of interest. Older Event is for same animal in different premises.

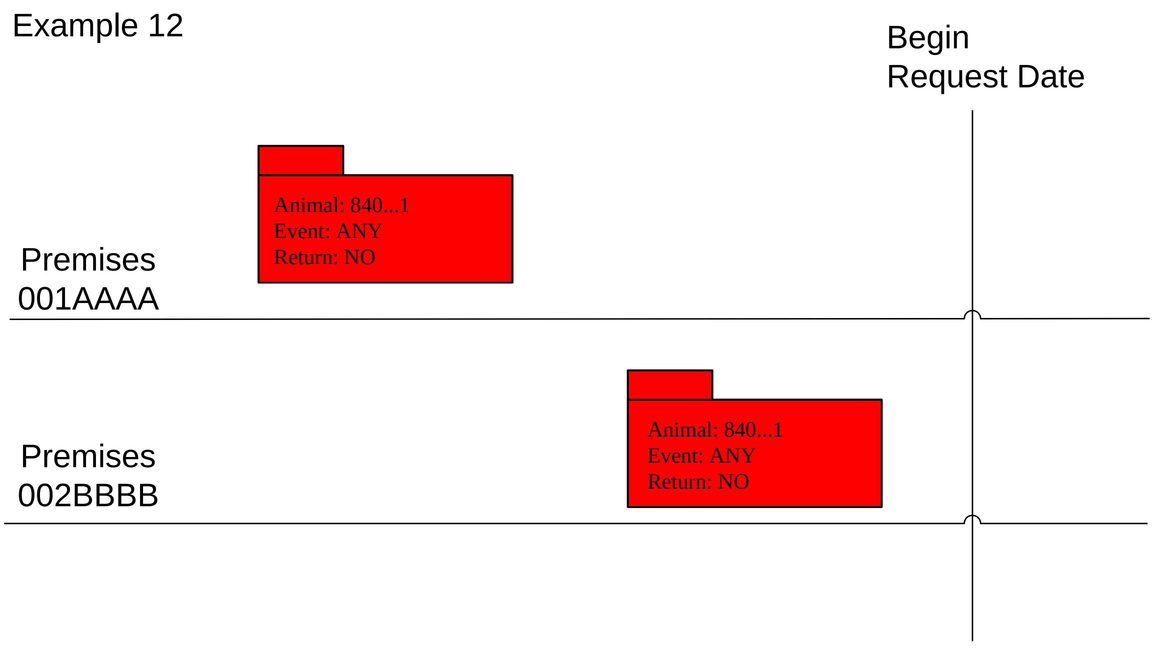
ATD Response: Return the Event.



- Example 12

ATD Scenario: Multiple Events before begin date. Older Event is Future Positive in Premises of interest. Newer Event is for same animal in different premises.

ATD Response: Do not return the Event.

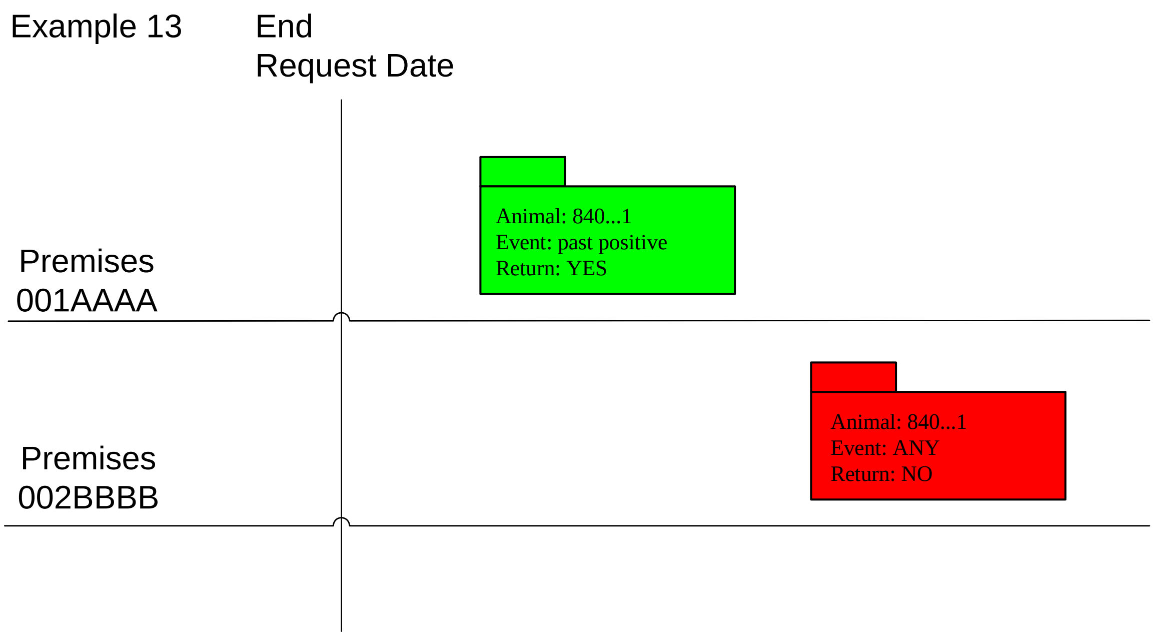


If the ATD has a Past Positive Event after the date range, normally it would return the event. However if the ATD also has ANY Event for the same animal at another Premises before the Future Positive event, but still after the date range, then the ATD will not return the Past Positive event.

- Example 13

ATD Scenario: Multiple Events after begin date. Older Event is Past Positive in Premises of interest. Later Event is for same animal in different premises.

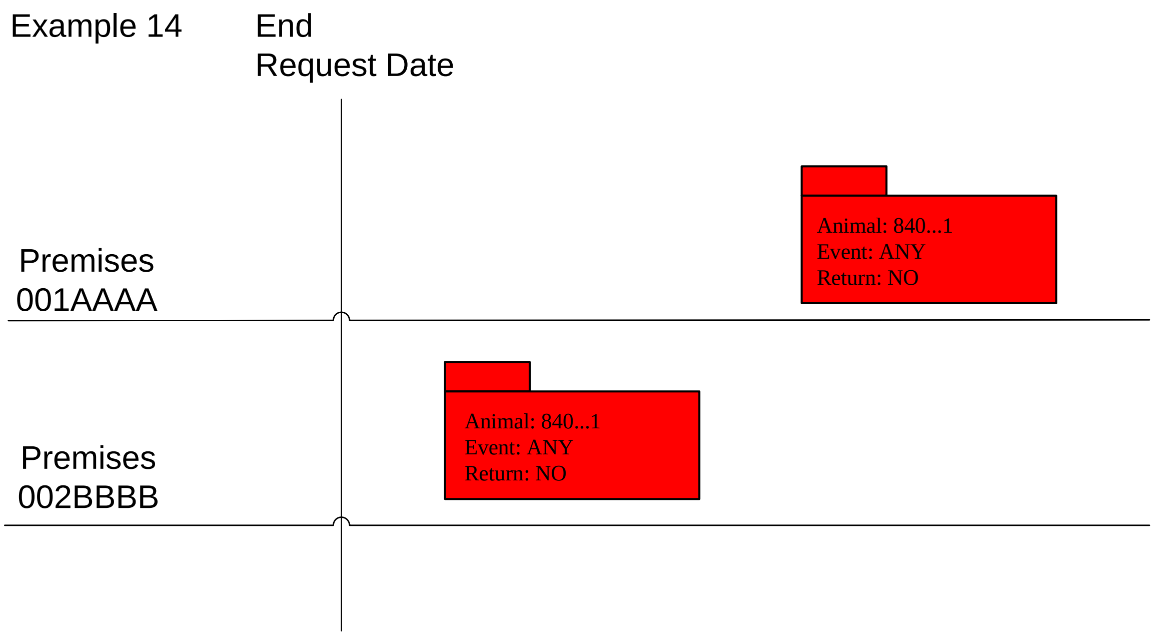
ATD Response: Return the Event.



- Example 14

ATD Scenario: Multiple Events after begin date. Later Event is Future Positive in Premises of interest. Older Event is for same animal in different premises.

ATD Response: Do not return the Event.



2.1.3.7 Submit Response Web Service Specification

Signature:

```
ATPSMessageValidationResultWS postMessage(
String encryptedATDId,
String pin,
String message)
throws SOAPException;
```

Arguments:

2.1.3.7.1 encryptedATDId

Type: String, 50 characters max, special characters allowed

Null: No

The Encrypted ATD ID is a required argument. The ATD ID coupled with the PIN is how ATPS authenticates the ATD. The Encrypted ATD ID can only be accessed by an Application User assigned to an Account containing the ATD, and can only be accessed via the web application.

2.1.3.7.2 pin

Type: String, 10 characters max, special characters allowed

Null: No

The PIN is a required argument. The Eauth ID coupled with the PIN is how ATPS authenticates the ATD. The PIN can only be accessed by an Application User assigned to an Account containing the ATD, and can only be accessed via the web application.

2.1.3.7.3 message

Type: String

Null: No

The message contains the XML that is the Response.

2.1.3.7.4 Return

Type: ATPSMessageValidationResultWS

ATPS will return a “message validation” result that describes what, if anything, went wrong when ATPS validated the message for submission. If ATPS encountered any conditions that resulted in the request being set to an ERROR state, this information will be present in the result class. Otherwise, if ATPS was able to successfully validate the message, ATPS will indicate that in the Result object as well.

ATPS may still throw a SOAP Exception. If an exception is thrown, ATPS did not process the message, and it is quite possible ATPS will leave the request in a RETRIEVED state.

If the web service returns without throwing an exception, it does not mean that the message passed validation and didn’t go to an ERROR state.

If ATPS returns a validation result without an exception in the object, then this means that the message was processed to the point where it passed basic XML format validation and the corresponding Request will not go to the ERROR status (unless this is a Split Response, then another Split could cause the Response to go to ERROR).

2.1.3.7.5 ATPSMessageValidationResultWS Object

```
ATPSMessageValidationResultWS {
```

```
Boolean passedValidation;
Boolean passedException;
ATPSExceptionInfoWS [] exceptionItems;
}
```

2.1.3.7.5.1 ATPSMessageValidationResultWS.passedValidation

Type: Boolean

Null: No

Description:

The passed validation indicates if the message passed validation or not. If this Boolean is "true", then the invalid items info array will be empty. If it is false, the invalid items array will have at least one item in it. If the message did not pass validation, it is still eligible to be processed.

2.1.3.7.5.2 ATPSMessageValidationResultWS.passedException

Type: Boolean

Null: No

Description:

The passed exception indicates if the message passed all exception cases or not. If this Boolean is "true", then the exception items info array will be empty. If it is false, the exception items array will have at least one item in it. If the message did not pass exception, it is not eligible to be processed. Typically this will indicate a problem with the XML formatting.

2.1.3.7.5.3 ATPSMessageValidationResultWS.invalidItems

Type: ATPSInvalidItemWS []

Null: No

Description:

See Section 2.1.2.12 for description of the ATPSInvalidItemWS object.

2.1.3.7.5.4 ATPSMessageValidationResultWS.exceptionItems

Type: ATPSExceptionInfoWS[]

Null: No

Description:

See Section 2.1.2.13 for description of the ATPSExceptionInfoWS object.

2.1.4 Validate National Premises ID

ATPS provides a web service that allows an ATD to validate a national premises. The service accepts a fully-qualified (7 digit) premises ID, and returns an object that contains address information about the premises if it is found in the repository. An ATD can call this web service to validate that a premises ID given to them is valid, and matches the address given to them. If the ID passed in is invalid or null or does not match any national premises in the repository, ATPS will return an empty result.

The address information will contain the premises ID, and the street, city, state, and zip code for the premises. Clients will also be able to find out if the premises is an active ADDD or health official location.

2.1.4.1 Validate National Premises ID Web Service Specification

Signature:

```
ATPSPremisesWS verifyPremises(
   String encryptedATDId,
   String pin,
   String premId)
throws SOAPException;
```

Arguments:

2.1.4.1.1 encryptedATDId

Type:	String, 50 characters max, special characters allowed
Null:	No

The Encrypted ATD ID is a required argument. The ATD ID coupled with the PIN is how ATPS authenticates the ATD. The Encrypted ATD ID can only be accessed by an Application User assigned to an Account containing the ATD, and can only be accessed via the web application.

2.1.4.1.2 pin

Type:	String, 10 characters max, special characters allowed
Null:	No

The PIN is a required argument. The Eauth ID coupled with the PIN is how ATPS authenticates the ATD. The PIN can only be accessed by an Application User assigned to an Account containing the ATD, and can only be accessed via the web application.

2.1.4.1.3 premId

Type:	String
Null:	No

The premId contains the premises ID for which the client ATD wants verification / information.

2.1.4.1.4 return

Type:	ATPSPremisesWS

ATPS will return a Premises object that contains information about the requested premises ID. ATPS will return the ID itself, as well as address information, and Booleans indicating whether or not the premises is an ADDD location, or a Health Official location. If the premises ID passed in does not exist in the repository, the premises object will be empty.

2.1.4.1.5 ATPSPremisesWS Object

```
ATPSPremisesWS {
String premisesId;
String street;
String city;
String ST;
String zip5;
String zip4;
Boolean isActiveADDD;
Boolean isActiveHealthOfficial;
}
```

2.1.4.1.5.1 ATPSPremisesWS.premisesId

Type: String

Null: Yes

This contains the premises ID passed into the service call, if and only if the input premises is a valid premises. Otherwise it will be null.

2.1.4.1.5.2 ATPSPremisesWS.street

Type: String

Null: Yes

If not null, this contains the street address for the premises.

2.1.4.1.5.3 ATPSPremisesWS.city

Type: String

Null: Yes

If not null, this contains the city for the premises.

2.1.4.1.5.4 ATPSPremisesWS.ST

Type: String

Null: Yes

If not null, this contains the 2 character state code for the premises.

2.1.4.1.5.5 ATPSPremisesWS.zip5

Type: String

Null: Yes

If not null, this contains the zip 5 for the premises.

2.1.4.1.5.6 ATPSPremisesWS.zip4

Type: String

Null: Yes

If not null, this contains the zip 4 for the premises.

2.1.4.1.5.7 ATPSPremisesWS.isActiveADDD

Type: Boolean

Null: Yes

True if premises is an active Animal Device Distribution Database. False if premises is not an active ADDD. Null if the input premises ID is not valid.

2.1.4.1.5.8 ATPSPremisesWS.isActiveHealthOfficial

Type: Boolean

Null: Yes

True if premises is an active Health Official location. False if premises is not an active Health Official. Null if the input premises ID is not valid.

2.1.5 Verify USDA Animal ID (AIN ID)

ATPS provides a web service that allows an ATD to validate a USDA AIN ID. Other types of IDs will not be validated. An AIN ID will assume to have been verified if the ID has been allocated and shipped from the manufacturer. The service will return a Boolean; true of the AIN ID is verified and false otherwise.

http://csurams.cstv.com/sports/w-volley/spec-rel/112606aaa.html

2.1.5.1 Verify USDA Animal ID (AIN ID) Web Service Specification

Signature:

```
Boolean verifyAinId(
   String encryptedATDId,
   String pin,
   String ainId)
throws SOAPException;
```

Arguments:

2.1.5.1.1 encryptedATDId

Type: String, 50 characters max, special characters allowed

Null: No

The Encrypted ATD ID is a required argument. The ATD ID coupled with the PIN is how ATPS authenticates the ATD. The Encrypted ATD ID can only be accessed by an Application User assigned to an Account containing the ATD, and can only be accessed via the web application.

2.1.5.1.2 pin

Type: String, 10 characters max, special characters allowed

Null: No

The PIN is a required argument. The Eauth ID coupled with the PIN is how ATPS authenticates the ATD. The PIN can only be accessed by an Application User assigned to an Account containing the ATD, and can only be accessed via the web application.

2.1.5.1.3 ainId

Type: String

Null: No

The ainId contains the AIN ID for which the client ATD wants verification.

2.1.5.1.4 return

Type: Boolean

The service will return a Boolean; true of the AIN ID is verified and false otherwise.

2.1.6 Web Services Security

ATPS web services security is divided into three main areas; ATD authentication, ATD authorization, and SSL Transport Security. Each of these three items is discussed in detail below.

2.1.6.1 Web Services Authentication

ATPS will authenticate each ATD making a web service call by utilizing an encrypted ATD ID and an ATD "PIN". The ATD is required to supply those two authentication keys every time a web service call is made. If the PIN and ATD ID do not validate, ATPS will throw an exception. The exception will be a "connection refused" exception, and will not provide any details as to why the connection was refused.

ATPS will record connection refused exceptions and will make efforts to block clients that appear to be attempting to "hack" into the ATPS or attempting a denial of service attack. In addition, in certain cases (explained below) ATPS will automatically lock a client ATD if they supply the incorrect ATD ID three times in a row.

2.1.6.1.1 ATD PIN

The ATD PIN serves a dual purpose. It is a unique number that identifies the client ATD, and it also serves as the key to decrypt the encrypted client ID. The ATD PIN is a string and can contain special characters, and can be up to 10 characters long.

A Client User can only obtain the PIN for its ATD or ATDs via a web application service. The details of this service are explained in another section. The PIN may be reset but if it is changed it will also automatically change the encrypted ATD ID.

2.1.6.1.2 Encrypted ATD ID

The Encrypted ATD ID is essentially the password for the ATD, encrypted by the ATD PIN. The encryption serves to make the ATD ID more secure. The Encrypted ATD ID is a string and can contain special characters, and can be up to 50 characters long.

A Client can only obtain the Encrypted ATD ID for its ATD or ATDs via a web application service. The details of this service are explained in another section. The ATD ID may be reset by the client.

2.1.6.1.3 Three Strikes Rule

If the ATD supplies a correct PIN but an incorrect encrypted ATD ID, ATPS will log that as a "strike" against the ATD. If the ATD has three such unsuccessful authentication attempts without a successful authentication, ATPS will lock the ATD for 30 minutes. In all cases ATPS will return only a "connection refused" exception. This policy is in place to deter unauthorized access to ATPS.

A Web Application User can log into the application and unlock an ATD that is currently locked. The details of this service are in another section.

2.1.6.2 Web Services Authorization

ATPS will authorize each ATD every time a web services is called. The authorization will be role-based as well as utilizing an enabled/disabled flag.

2.1.6.2.1 ATD Enabled/Disabled

An ATD can be set to enabled or disabled. A disabled ATD will not be authorized to call any web services. A disabled ATD will receive a “disabled ATD failure” exception upon calling any web service. ATPS Requests will continue to be created for a disabled ATD. Only an Admin User has the ability to disable and enable an ATD.

2.1.6.2.2 ATD Role

An ATD can have one to many roles that allow them to be authorized to call a particular service. ATD roles may also be used to enforce different business logic during response processing. An ATD that fails due to role-based authorization failure will receive an “authorization failure” exception. Only an Admin User can modify the roles of an ATD.

2.1.6.3 SSL Transport Layer Security

ATPS will utilize SSL Transport Layer Security to provide secure interned communication between ATPS and its clients. ATPS will have a digital certificate that must be referenced by the trust store of the client ATD. The ATD trust store is required since web service transactions have no manual interaction. The trust store takes the place of the User verifying that the server (ATPS) is trusted.

The following steps can be followed to create a trust store for ATPS.

Create a truststore containing the CA certificate that will authorize the certificate the webserver returns when performing the SSL Handshake. Note that temporarily, ATPS is using a Thawte certificate for the server certificate. When the permanent certificate is established, detailed instructions on obtaining the CA certificate will be included in this section.

The command for creating the truststore is (note that keytool is in the [JAVA_JRE]/bin folder):

keytool -import -noprompt -trustcacerts -alias thawteCACert -v –file [fully-qualified path name to CA certificate] -keystore [path to trustStore file]/trustStore.jks -storepass [password]

Things to note:

- Even though the command is "keytool" and the destination filename flag is "keystore", we're creating a truststore because we're importing the certificate we "trust" that will authorize the certificate the server sends us when we connect to it (a "keystore" is for holding the key and certificates that a server sends out).
- The trustcacerts setting says that we trust the certificate we're importing
- The storepass defines the password that will be required to access this truststore file when we need to check some certificates.

Use the java jvm arguments when starting the webservices client:

Djavax.net.ssl.trustStore="[path to trustStore file]/trustStore.jks"

Djavax.net.ssl.trustStoreType=JKS -Djavax.net.ssl.trustStorePassword=[password set in step 1 above]

2.2 Web Application Services

Services that are accessed via a web browser-based API.

2.2.1 *Log In to Application*

Description: An Account User can log into the application. The application will authorize using USDA eAuthentication.

Services:

2.2.1.1 User getUserForEauthId(Long eauthId)

Look for a User that matches the eAuth ID passed in.

Who Can Use: This is an "open" service; no session User is required.

2.2.1.2 User[] getUsersForAccount(Long accountId)

Given an Account ID, return all application Users assigned to the Account.

Who Can Use: This is an "Account" service; an Admin User can run for any Account. An ATD or HO User can run for their own Account only.

2.2.1.3 User modifyUserForAccount(Long accountId, User user)

Modify (update) the User passed in to reflect the new attribution of the User class. The Account ID is required for authorization purposes.

Who Can Use: This is an "Account" service; an Admin User can run for any Account. An ATD or HO User can run for their own Account only.

2.2.1.4 Account getAccount(Long accountId)

Who Can Use: This is an "Account" service; an Admin or HO User can run for any Account. An ATD User can run for their own Account only.

2.2.1.5 Contact createContactForAccount(Long accountId, Contact contact)

Who Can Use: This is an "Account" service; an Admin User can create for any Account. An ATD User can run for their own Account only. An HO User can not create.

2.2.1.6 Contact[] getContactsForAccount(Long accountId)

Who Can Use: This is an "Account" service; an admin or HO User can get for any Account. An ATD User can get for their own Account only. The Account contains contacts.

Actions:

2.2.1.6 BaseAction

For all actions. If the User is not found in session, forward to the entryPage, which is protected by eAuthentication.

2.2.1.7 ProcessLoginAction

Call getUser (2.2.1.1), passing in the eAuthentication ID from the form/session. If User is not found, forward to processFirstLoginAction (2.2.1.8). If User is found, forward to home page or User request.

2.2.1.8 ProcessFirstLoginAction

Call getUser (2.2.1.1) passing in the PIN and email address from the form. If User is not found, throw exception. If User is found, modify User with eAuthentication ID (2.2.1.3). Get account for User (2.2.1.4). If account has no contacts (2.2.1.6), forward to accountContactAction (2.2.1.9) and force the User to create a contact. Otherwise, forward to home or User request.

2.2.1.9 AccountContactAction

Get a list of all contacts for the Account (2.2.1.4). Create a new Account based on the contact info entered by the User (2.2.1.5).

Logic:

User logs in using USDA eAuthentication ID (2.2.1.7).

ATPS calls service (2.2.1.1) to look up User based on eAuthentication ID.

If the User is found, ATPS puts the User object into the http session, and forwards the request to the home page or to the requested forwarding page if found.

If the User is not found, ATPS forwards to a page that asks the User for a PIN and email address (2.2.1.8).

The User inputs the PIN and email address and submits the page. PIN and email are required.

ATPS looks up a User based on PIN and email address (2.2.1.1).

If a (single) User is found, validate that the User does not already have an eAuth ID attached to it. If it does, throw an exception.

If it the User is found and the eAuth ID is empty/null, modify the User and assign the session eAuth ID to the User (2.2.1.3).

Now verify that the User’s Account has at least one active contact (2.2.1.4)

If the User’s Account does not have a contact, forward to a page that requires them to create a contact for the Account (2.2.1.9).

The User will create an Account. Required Account elements include name, phone, email, and address (street, city, st, zip5).

Once the Account is successfully created (2.2.1.5), forward to the home page.

2.2.2 Search Requests

Search for Requests from ATPS based on Request Status, Request ID, and/or Request Date.

Services:

2.2.2.1 ATPSRequest[] findRequests(Long accountId, ATPSRequestCriteria criteria)

Get all requests that match the input criteria. See 2.1.2.14 for usage rules. Note that this is the same service that the web service will call. Note also that internally, a client ID is required. The web service will always populate with the client's ID. The web application will have to get it from the User, since the User may have more than one client to choose from.

Who Can Use: Admin Users can get requests for any Account. HO Users can get requests for any Account. ATD Users can only get requests for their own Account.

2.2.2.2 Account[] getAllAccounts(); (list service)

Get list of all clients matching the Account ID passed in.

Who Can Use: Admin Users can get any Account. HO Users can any Account. ATD Users can only get their own Account.

2.2.2.3 Client[] getAllClients(Long accountId); (list service)

Get list of all clients matching the Account ID passed in.

Who Can Use: Admin Users can get clients for any Account. HO Users can get clients for any Account. ATD Users can only get cliense for their own Account.

2.2.2.4 ATPSRequestStatus[] getAllATPSRequestStatus(); (list service)

Get a list of all available request statuses

Who Can Use: public service.

2.2.2.5 ATPSRequestStatusCategory[] getAllATPSRequestStatusCategory(); (list service)

Get a list of all available request status categories

Who Can Use: public service.

Actions:

2.2.2.6 FindClientRequestsAction

2.2.2.7 ClientRequestAction

Logic:

The User can enter a variety of criteria information (2.2.2.6) to locate ATPS Requests. If the User has the Admin or HO role, the can choose from any client. An ATD User is forced to get requests from a client that is in their Account only. Get the list of all clients, and filter out all clients that do not have the same Account ID as the session User, unless the User is allowed to call this service on any client. See 2.1.2.14 for usage rules on what criteria are required. When the User submits, call the findRequests service (2.2.2.1), and display a summary of the results. The action will have to call services 2.2.2.2, 2.2.2.2.3, 2.2.2.4, and 2.2.2.5 to populate the screen. 2.2.2.3 is populated solely based on the value of 2.2.2.2.

The User can click on one of the results. This will forward to a page (2.2.2.7) that will display all information about the request. The User can then click a link to view all the exceptions and invalid items for the request, or click a link that allows them to respond to the request, provided the request is in a “respond-able” state (see 2.2.3 below).

2.2.3 Upload Manual Response

Description: “Manually” Upload and Submit a Response to a Request.

Who can use: An Account User can manually upload and submit a Response for any of the ATDs that are active in their Account.

Services:

2.2.3.1 ATPSMessageValidationResultWS validateResponse(Long accountId, String response)

This service will execute the internal response validation service to determine if the response will result in an ERROR state or not. The response string is XML. The web service will call the exact same service.

Who can use: Admin Users can call for any Account. HO Users can not call this service. ATD Users can call only for their own Account.

2.2.3.2 void submitResponse(Long accountId, String response)

This service will submit a response to a request. The response string is XML. The web service will call the exact same service.

Who can use: Admin Users can call for any Account. HO Users can not call this service. ATD Users can call only for their own Account.

Actions:

2.2.3.3 RequestResponseAction

Logic:

The User can access the RequestResponseAction action by clicking on a “respond-able” request (see 2.2.2 above). This page allows the User to upload an XML file and submit for response validation (2.2.3.1). When submitted, the validation results will be displayed. Then the User will have the option of submitting the response (2.2.3.2) or uploading a new response and re-validating (2.2.3.1). When the response is submitted, since it is an asynchronous service, ATPS will not display the results of the response. It will forward the User to the search for requests page (2.2.2)

2.2.4 Manage Account

An Account is a collection of Web Application Users, Client Contacts, and Client ATDs. Accounts can only be created by an ATPS Admin.

2.2.4.1 View Account

Services:

2.2.2.2 Account[] getAllAccounts(); (list service)

(See above for description)

2.2.1.6 Contact[] getContactsForAccount(Long accountId)

(See above for description)

2.2.1.2 User[] getUsersForAccount(Long accountId)

Get all (application) Users for the Account.

Who Can Use: This is an “Account” service; an Admin User can get for any Account. An ATD or HO User can get for their own Account only.

2.2.4.1.1 Client[] getClientsForAccount(Long accountId)

Get all (ATD) clients for the Account.

Who Can Use: This is an “Account” service; an Admin or HO User can get for any Account. An ATD User can get for their own Account only.

Actions:

2.2.4.1.2 accountAction

Logic:

An ATD User can view the Account to which they are assigned (2.2.4.1.2). An Admin or HO can view any Account. The action will contain a list of all Accounts (2.2.2.2). An Admin or HO can choose the Account and submit to view its details. The Account details will include a list of Contacts (2.2.1.6), Clients (2.2.4.1.1), and application Users (2.2.1.2) that are assigned to the Account, in addition to the Account attributes.

2.2.4.1 Create Account (ATPS Admin)

Create a new Account.

Services:

2.2.4.1.1 Account createAccount(Account newAccount)

Create a new Account

Who can use: An Admin User can create a new Account. No other User can create.

Actions:

2.2.1.4.1 accountAction

Logic:

An Admin User can create a new Account from the Account Admin page. The new Account will not have any Users or Contacts. The Account page will include a “create new” submit button that executes this service.

2.2.4.2 Modify Account

Modify Account Attributes (not including enabled/disabled).

Services:

2.2.4.2.1 Account modifyAccount(Account account)

Modify Account parameters

Who can use: An Admin User can modify any Account parameters. An ATD or HO User can modify its own Account parameters.

Actions:

2.2.1.4.1 accountAction

Logic:

An Admin User can modify Account parameters from the Account Admin page. Contacts and Users are modified using a different service; this only applies to Account attributes. The Account page will include a “modify” submit button that executes this service.

2.2.4.3 Disable/Enable Account (ATPS Admin)

Mark an entire Account as enabled or disabled

Services:

2.2.4.3.1 Account enableAccount(Long accountId)

Enables a disabled Account

Who can use: Only an Admin User can use this service

2.2.4.3.2 Account disableAccount(Long accountId)

Disables an enabled Account.

Who can use: Only an Admin User can use this service.

Actions:

2.2.1.4.1 AccountAction

Logic:

An Admin User can enable or disable an Account. The page (2.2.1.4.1) will have an enable (2.2.4.3.1) or disable (2.2.4.3.2) submit button or link, depending on the current state of the Account.

Disabling an Account effectively disables all User and Client ATDs assigned to the Account. Users will fail all service authorization attempts. Client ATDs will not receive any requests, and all responses from client ATDs will fail validation and be rejected.

2.2.5 Manage Web Application User

Web Application Users are associated to an Account. They have the ability to execute Web Application Services for any ATD associated to the Account, and also manage all contacts associated to the Account. A Web App User may only be associated to a single Account. An Account may contain multiple Web App Users.

2.2.5.1 Create Web App User for Account

Create a new Web App User and assign to an Account.

2.2.5.1.1 User createUser (Long accountId, User user)

Creates a new User assigned to an Account.

Who can use: An Admin User can create a User for any Account. An HO or ATD User can only create Users for their own Account.

Actions:

2.2.1.4.1 AccountAction

2.2.5.1.2 UserAction

Logic:

Form the Account page (2.2.1.4.1) a User can access the list of Users (including themselves), and will also have access to a submit button or link that takes them to the User page (2.2.5.1.2). From there they can create a new User Account by filling in the required fields and submitting the page. The new User will be assigned to the Account. Upon creation, ATPS will send an email to the new User informing them that their Account has been created, and how to complete the registration process. A new User required the following fields: first and last name, phone number, email address. When the User is created, a PIN will be auto-generated for the User. When the User completes the registration process by logging into the application using eAuthentication and the email/PIN, their eAuthentication ID will be assigned to their Account as well.

2.2.5.2 View/Modify Web App User

View the User details / Modify Web App User Attributes

2.2.5.2.1 User getUser (Long accountId, UserSearchCriteria cri)

Gets the User for the Account. Note that the Account ID is not necessary to get the User, it is used for service authorization purposes.

Who Can Use: Admin and HO User can call for any Account; ATD User can only call for their own Account.

2.2.5.2.2 User modifyUser (Long accountId, User user)

Modifies a User assigned to an Account.

Who can use: An Admin User can modify a User for any Account. An HO or ATD User can only modify Users for their own Account.

Actions:

2.2.1.4.1 AccountAction

2.2.5.1.2 UserAction

Logic:

Form the Account page (2.2.1.4.1) a User can access the list of Users (including themselves), and will also have access to a link (2.2.5.2.1) that takes them to the User page (2.2.5.1.2). From there the User can modify User attributes such as name, phone number, email (2.2.5.2.2). Note, the PIN and eAuth ID are not modifiable.

2.2.5.3 Disable/Enable Web App User (Admin).

Disable a Web App User to deny all access to the Web App. Enable the User to allow access.

Services:

2.2.5.3.1 User enableUser(Long accountId, Long userId)

Enables a disabled User

Who can use: Only an Admin User can use this service

2.2.5.3.2 User disableUser(Long accountId, Long userId)

Disables an enabled User.

Who can use: Only an Admin User can use this service.

Actions:

2.2.1.4.1 AccountAction

Logic:

An Admin User can enable or disable a User. The page (2.2.1.4.1) will have an enable (2.2.5.3.1) or disable (2.2.5.3.2) submit button or link, depending on the current state of the User.

A disabled User will fail all service authorization attempts.

2.2.5.4 Manage User Roles

Remove/Assign User roles to a User.

Currently there are three roles a User can have; ATD, HO (Helath Official), and Admin. Only an Admin User can assign roles to another User.

Service:

2.2.5.4.1 User modifyUserRoles(Long accountId, User user)

This service updates the roles in the User to reflect the roles in the User object passed in.

Who Can Use: Only an Admin User can update User roles.

Action:

UserAction 2.2.5.1.2

Logic:

From the User page (2.2.5.1.2), a list of enabled and disabled roles is displayed. An Admin User can modify the roles and click a submit button to modify the roles (2.2.5.4.1)

2.2.6 Manage Contact

A Contact is a client administrative entity associated to an Account, and therefore associated to all ATDs on the Account. Contacts may receive automated ATPS emails relating to important ATPS Events. Contacts may be contacted directly by ATPS production support. A contact does not have to also be a Web App User. A Contact may only be associated to a single Account. An Account may contain from one to many Contacts, but must contain at least one contact.

2.2.6.1 Create Contact

A Web App User may create a new Contact for their Account.

Services:

2.2.6.1.1 Contact createContact(Long accountId, Contact newContact)

Creates a new contact for the Account.

Who Can Use: An Admin User can create a contact for any Account. An HO or ATD User can create a contact for their own Account only.

Actions:

2.2.1.4.1 AccountAction

2.2.6.1.2 ContactAction

Logic:

From the Account page (2.2.1.4.1) the User can click a submit button or link that will take them to the contact page (2.2.6.1.2). From there the User can input the required contact information, and submit the page (2.2.6.1.1) which will create a new contact for the Account. Required fields include first/last name, phone, email address.

2.2.6.2 View/Modify/Delete Contact

A Web App User may modify, disable, or delete an Account Contact.

Services:

2.2.6.2.1 Contact getContact(Long accountId, Long contactId)

Retrieves the contact.

Who can use: Admin and HO User can get any contact. ATD User can only get a contact assigned to their Account.

2.2.6.2.2 Contact modifyContact(Long accountId, Contact contact)

Modify modifiable contact attributes.

Who can use: Admin User can modify any contact. HO and ATD User can only modify a contact assigned to their Account.

2.2.6.2.3 void deleteContact(Long accountId, Long contactId)

Delete an Account contact.

Who can use: Admin User can delete any contact. HO and ATD User can only delete a contact assigned to their Account. ATPS will not allow an Account to have 0 contacts.

Actions:

2.2.1.4.1 AccountAction

2.2.6.1.2 ContactAction

Logic:

Modify contact:

From the Account page (2.2.1.4.1) the User can click a link (2.2.6.2.1) that will take them to the contact page (2.2.6.1.2) for a particular contact. From there the User can modify contact information, and submit the page (2.2.6.2.2) which will update the contact info.

Delete Contact:

From the Account page (2.2.1.4.1) the User can click a link (2.2.6.2.3) that will delete the contact. The app will pop up a warning before actually deleting the contact.

2.2.7 Manage ATD Client

The ATD is the ATPS representation of a Web Services Client. An ATD is assigned to one and only one Account. An Account may contain multiple ATDs. Each ATD will have a unique Authentication Identifier. All enabled ATDs are expected to respond to all ATPS Requests.

2.2.7.1 Create Client ATD (ATPS Admin)

An Admin may create a new Client ATD for an Account. The ATD Authentication Identifier will be created at this time.

Services:

2.2.7.1.1 Client createClient(Long accountId, Client newClient)

Creates a new contact for the Account.

Who Can Use: Only an Admin User can create client.

Actions:

2.2.1.4.1 AccountAction

2.2.7.1.2 ClientAction

Logic:

From the Account page (2.2.1.4.1) the User can click a submit button or link that will take them to the client page (2.2.7.1.2). From there the User can input the required client information, and submit the page (2.2.7.1.1) which will create a new client for the Account. Required fields include client name. ATPS will automatically assign a PIN and authentication ID to the client upon creation.

2.2.7.2 View/Modify Client ATD

A Web App User may view the Authentication Identifier for a Client ATD in their Account. The Web App User may also modify Client ATD attribution. The ATD Authentication Identifier may be modified via the Authentication Identifier reset function (see below).

Services:

2.2.7.2.1 Client getClient(Long accountId, Long clientId)

Retrieves the client.

Who can use: Admin User can get any client. ATD User can only get a client assigned to their Account. HO can not use this service.

2.2.7.2.2 Contact modifyClient(Long accountId, Client client)

Modify modifiable contact attributes.

Who can use: Admin User can modify any client. HO and ATD User can only modify a contact assigned to their Account.

Actions:

2.2.1.4.1 AccountAction

2.2.7.1.2 ClientAction

Logic:

From the Account page (2.2.1.4.1) the User can click a submit button or link (2.2.7.2.1) that will take them to the client page (2.2.7.1.2). From there the User can update modifiable client information, and submit the page (2.2.7.2.2) which will update the modifiable client information.

2.2.7.3 Unlock ATD

A Web App User may Unlock an ATD Account that has been locked due to unsuccessful authentication attempts.

Services:

2.2.7.3.1 Client unlockClient(Long accountId, Long clientId)

Unlocks the client by setting the number of invalid login attempts to 0 for the client.

Who can use: Admin can unlock any client. ATD can unlock a client in the User's Account only. HO can not use this service.

2.2.7.1.2 ClientAction

Logic:

The client page will indicate if the client is locked. The User can click a button that will submit a request (2.2.7.3.1) to unlock the client.

2.2.7.4 Reset Client ATD Authentication Identifier

A Web App User may reset the Authentication Identifier for a Client ATD in their Account. A Reset will take effect immediately.

Services:

2.2.7.4.1 Client resetClientAuthentication(Long accountId, Long clientId)

Resets the client PIN and authentication ID

Admin can reset any client. ATD can reset a client in the User's Account only. HO can not use this service.

2.2.7.1.2 ClientAction

Logic:

The client page will display the PIN and encrypted authentication ID for the client. The User can click a button that will submit a request (2.2.7.4.1) to reset these two values.

2.2.7.5 Disable/Enable Client ATD

An Admin may Disable an ATD. A Disabled ATD does not receive Requests. An Enabled ATD receives Requests.

Services:

2.2.7.5.1 Client enableClient(Long accountId, Long clientId)

Enables a disabled client

Only an Admin User can use this service

2.2.7.5.2 Client disableClient(Long accountId, Long clientId)

Disables an enabled client.

Who can use: Only an Admin User can use this service.

Actions:

2.2.7.1.2 ClientAction

Logic:

An Admin User can enable or disable a client. The page (2.2.7.1.2) will have an enable (2.2.7.5.1) or disable (2.2.7.5.2) submit button or link, depending on the current state of the client.

A disabled Client will not receive any new requests, and all responses from a disabled client will be rejected.

2.2.7.6 Manage Client ATD Roles

Remove/Assign roles to a Client.

Currently there are three roles a User can have; ATD, HO (Helath Official), and Admin. Only an Admin User can assign roles to another User.

Service:

2.2.7.6.1 Client modifyClientRoles(Long accountId, Client client)

This service updates the roles in the client to reflect the roles in the client object passed in.

Who Can Use: Only an Admin User can update client roles.

Action:

2.2.7.1.2 ClientAction

Logic:

From the Client page (2.2.7.1.2), a list of enabled and disabled roles is displayed. An Admin User can modify the roles and click a submit button to modify the roles (2.2.7.6.1)

2.2.8 Create Case

ATPS has the ability to create several different types of “Cases”. All cases are a collection of requests, which themselves contain responses. A User can create a case, and then get details about the case. This section covers services related to creating various case types.

2.2.8.1 Create Manual Ping Case

A User can create a “ping” case that is specific to a single client ATD.

Services:

2.2.8.1.1 ATPSCase createManualPingCase(Client client, ATPSCase newCase)

Create a new case that specifically targets one ATD Client with a new ping request.

Who can use: Admin can call for any Account. HO can not call. ATD can call for assigned Account only.

2.2.2.2 Account[] getAllAccounts();

2.2.4.1.2 Client[] getClientsForAccount(Long accountId)

Actions:

2.2.8.1.2 ManualPingCaseAction

Logic:

Ping case page (2.2.8.1.2) will contain a list of Accounts (2.2.2.2) and a list of clients based on the Account chosen (2.2.4.1.2). The User will select a client, and submit the page (2.2.8.1.1) which will create a "manual" ping case specific to the chosen client.

Case creation logic:

The case "factory" will analyze the client. If the client already has a NEW ping request outstanding, ATPS will not create the case and will throw an exception. ATPS allows ATD Clients to only have one outstanding NEW ping request assigned to them.

If the ATD Client does not have a NEW ping request assigned to them, the case "factory" will recognize that this is a manual ping case, and build a request set that contains a single ping request for the requested client.

2.2.8.2 Create Manual Request Case

A User can create a "real" case that is specific to a single client ATD.

Services:

2.2.8.2.1 ATPSCase createManualRequestCase(Client client, ATPSRequest request, ATPSCase newCase)

Create a new case that specifically targets one ATD Client with a single "real" request.

Who can use: Admin can call for any Account. HO can not call. ATD can call for assigned Account only. Note, permissions may change for this service.

2.2.2.2 Account[] getAllAccounts();

2.2.4.1.2 Client[] getClientsForAccount(Long accountId)

Actions:

2.2.8.2.2 ManualRequestCaseAction

Logic:

Request case page (2.2.8.2.2) will contain a list of Accounts (2.2.2.2) and a list of clients based on the Account chosen (2.2.4.1.2). It will also allow the User to enter a list of official ID /type combos, or a list of national premises IDs, a begin/end date range, and a begin/end audit date range. The User will select a client, and submit the page (2.2.8.2.1) which will create a "manual" ping case specific to the chosen client. The use must supply either one or more official ID /type pairs, but not more than 100, or one or more national premises ID values but not more than 10. If premises IDs are populated, the begin and end date are required. Begin and end audit date are optional but if one is populated they both must be populated.

Case creation logic:

The case “factory” will recognize that this is a manual request case, and build a request set that contains a single request for the specified client.

2.2.8.3 Create “Real” Case

A User can create a “real” case. Note that requirements on this are incomplete.

Services:

2.2.8.3.1 ATPSCase createCase(ATPSRequest request, ATPSCase newCase)

Create a new case with a set of initial request parameters.

Who can use: Admin, HO can call. ATD cannot call.

Actions:

2.2.8.3.2 CaseAction

Logic:

Case page (2.2.8.3.2) will also allow the User to enter a list of official ID /type combos, or a list of national premises IDs, a begin/end date range, and a begin/end audit date range. When the User submits the page (2.2.8.2.1) ATPS will create a case utilizing the page parameters. The use must supply either one or more official ID /type pairs, but not more than 100, or one or more national premises ID values but not more than 10. If premises IDs are populated, the begin and end date are required. Begin and end audit date are optional but if one is populated they both must be populated.

Case creation logic:

The case “factory” will recognize that this is a real request case, and build a request set that contains a requests for all enabled ATPS clients.

2.2.9 Search Cases

A User can search for Cases and introspect for case status and details.

Services:

2.2.2.2 Account[] getAllAccounts();

2.2.9.1 ATPSCase[] findATPSCases(Long accountId, ATPSCaseSearchCriteria criteria)

Return cases that match the criteria elements passed in. The requestSets will not be populated.

Who can use: Admin can find cases created by any Account. HO and ATD Users can only find cases created by their own Account.

2.2.9.2 ATPSRequestSet[] getRequestSetsForCase(Long accountId, Long caseId)

Returns all the requests sets for a given case. The case ID is required so that proper service authorization can be implemented.

Who can use: Admin User can get request sets for any case. HO and ATD Users can only get request sets for case created by their Account.

2.2.9.3 ATPSRequest[] getRequestsForRequestSet(Long accountId, Long caseId, Long requestSetId)

Returns all the requests for a given request set. The case ID is required so that proper service authorization can be implemented.

Who can use: Admin Users can get requests for any case. HO and ATD Users can only get requests for case created by their Account.

2.2.9.4 ATPSResponse[] getResponsesForRequest(Long accountId, Long caseId, Long requestId)

Returns all the responses for a given request. The case ID is required so that proper service authorization can be implemented.

Who can use: Admin Users can get responses for any case. HO and ATD Users can only get responses for case created by their Account.

2.2.9.5 Message[] getMessagesForResponse (Long accountId, Long caseId, Long reponseId)

Returns all the messages for a given response. The case ID is required so that proper service authorization can be implemented.

Who can use: Admin Users can get messages for any case. HO and ATD Users can only get messages for case created by their Account.

Actions:

2.2.9.6 CaseAdminAction

Logic:

The search case action (2.2.9.6) will contain an Account drop down and a set of case search criteria, including case ID and case name (more criteria forthcoming probably). The User will find cases by submitting a search request (2.2.9.1).

A User can then drill down into the case by continuing to click on a link on the case display that will show all request sets for the case (2.2.9.2).

A User can then drill down into the request sets by continuing to click on a link on the request sets display that will show all requests for the request set (2.2.9.3).

A User can then drill down into the request by continuing to click on a link on the request display that will show all responses for the request (2.2.9.4).

A User can then drill down into the response by continuing to click on a link on the response display that will show all messages for the response (2.2.9.5).

2.2.10 Close Case

A User can close a case that is completed.

Services:

2.2.10.1 ATPSCase closeCase(Long accountId, Long caseId)

This service will move the status of the case from open to closed. It will also create a new request set with a PROGRAM_CASE_CLOSED request for all client ATDs that participated in the case (i.e. all client ATDs that have a request in this Case). Then a case is closed, if it is not a “real” case, all messages tied to the case till be deleted. If it is a “real” case, all messages tied to the case will be archived and then deleted from the production db.

Who Can Use: An Admin can close any case. A HO or ATD can only close a case that someone in their Account created.

Action:

2.2.9.6 CaseAdminAction

Logic:

In the case page (2.2.9.6), when a User has found an open case (2.2.9.1), they will have the ability to click a link that will close the case (2.2.10.1). ATPS will warn the User that this is not a reversible service.

2.3 Automated Services

ATPS has the ability to create automated services. The only automated service is the regular ping utility.

2.3.1 Create Ping Case

ATPS will attempt to ping all the enabled ATDs every hour.

Service:

2.3.1.1 ATPSCase createPingCase();

This is an automated service. ATPS will create a new ATPSCase, and will create a request set with requests for every ATD that does not currently have a NEW ping request assigned to them. Note that manual ping requests are not considered in determining which ATDs to create a request for.

Who can use: Only an Admin “User” can run this service. Even though this is tied so some automated process, the service will run as an actual User. Note that the “User” may not be tied to an actual person though.

Action:

PingAction

Logic:

An Admin User can access a page to adjust the ping frequency, or to turn on or off the automated ping service. This may utilize the existing application environment page.

Admin Users can lookup the results of ping requests via the search ping page.

3 APPENDIX

3.1 Web Service Signatures

These are provided here as a quick reference. For more detail, please see the Web Service Requirements.

3.1.1 *Get Requests Web Service Specification:*

Signature:

```
ATPSRequestWS[] getRequests(
String encryptedATDId,
String pin,
ATPSRequestCriteriaWS criteria)
throws SOAPException;
```

3.1.2 *Submit Response Web Service Specification:*

Signature:

```
ATPSMessageValidationResultWS postMessage(
String encryptedATDId,
String pin,
String message)
throws SOAPException;
```

3.1.3 *Validate National Premises ID Web Service Specification:*

Signature:

```
ATPSPremisesWS verifyPremises(
String encryptedATDId,
String pin,
String premId)
throws SOAPException;
```

3.1.4 *Verify USDA Animal ID (AIN ID) Web Service Specification:*

Signature:

```
Boolean verifyAinId(
String encryptedATDId,
String pin,
String ainId)
throws SOAPException;
```

3.2 ATPS Web Service custom class attribution

The Web Service Interface describes several custom classes. These are discussed in detail in the requirements document, and are included here for lookup purposes. It can be assumed that each attribute will have a get and set method that follow standard simple bean naming conventions. For instance, the requestStatus attribute will be accessed via a getRequestStatus() method and a setRequestStatus(String s) method.

3.2.1 ATPSRequestWS Class:

Attribution:

```
ATPSRequestWS{
Long requestId;
ATPSCaseWS case;
String requestStatusCategory;
String requestStatus;
Date requestCreatedDate;
Date requestModifiedDate;
ATPSOfficialIdWS[] officialIds;
String[] nationalPremisesIds;
String species;
Date beginRequestDate;
Date endRequestDate;
Date beginAuditDate;
Date endAuditDate;
ATPSInvalidItemWS [] invalidItems;
ATPSInvalidItemWS [] exceptionItems;
}
```

3.2.2 ATPSCaseWS Class:

Attribution:

```
ATPSCaseWS{
Long caseID;
String caseDescription;
String caseStatus;
}
```

3.2.3 ATPSOfficialIdWS Class:

Attribution:

```
ATPSOfficialIdWS{
String officialId;
String officialIdType;
}
```

3.2.4 ATPSInvalidItemWS Class:

Attribution:

```
ATPSInvalidItemWS{
String ATDResponseId;
Integer split;
String ATDEventId;
Long recordSequence;
String elementName;
String elementValue;
ATPSExceptionInfoWS exceptionInfo;
}
```

3.2.5 ATPSExceptionInfoWS Class:

Attribution:

```
ATPSExceptionInfoWS {
String cause;
String message;
}
```

3.2.6 ATPSRequestCriteriaWS Class:

Attribution:

```
ATPSRequestCriteriaWS {
Long requestId;
Long caseId;
String[] requestStatus;
String requestStatusCategory;
Date beginRequestCreatedDate;
Date beginRequestModifiedDate;
}
```

3.2.7 ATPSMessageValidationResultWS Class:

Attribution:

```
ATPSMessageValidationResultWS {
Boolean passedValidation;
Boolean passedException;
ATPSExceptionInfoWS [] invalidItems;
ATPSExceptionInfoWS [] exceptionItems;
}
```

3.2.8 ATPSPremisesWS Class:

Attribution:

```
ATPSPremisesWS {
String premisesId;
String street;
String city;
String ST;
String zip5;
String zip4;
Boolean isActiveADDD;
Boolean isActiveHealthOfficial;
```

3.3 Response DTD Specification

ATPS has defined two DTD files for the response XML. These are discussed in detail in the requirements document, and are included here for lookup purposes.

3.3.1 DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<!—DTD for submission of Animal and Group Events -->
<!ELEMENT eventSub (header,(animalRecords|groupRecords))>
<!ELEMENT header (atpsRequestId,atdResponse)>
<!ELEMENT atpsRequestId (#PCDATA)>
<!ELEMENT atdResponse (responseId)>
 <!ATTLIST atdResponse
 final (Y|N) #REQUIRED
 split CDATA #IMPLIED
 >
<!ELEMENT responseId (#PCDATA)>
<!ELEMENT animalRecords (animalRecord*)>
<!ELEMENT animalRecord (ATDEventId*, eventType, eventDate, rptPre-
mId, id, srcDestPremId*, animal*, remarks*, reTagId*, optIds*)>
 <!ATTLIST animalRecord
 elecRead CDATA #IMPLIED
 status CDATA #IMPLIED
 >
<!ELEMENT groupRecords (groupRecord*)>
<!ELEMENT groupRecord (ATDEventId*, eventType, eventDate, rptPremId,
id, srcDestPremId*, group*, remarks*)>
 <!ATTLIST groupRecord
 elecRead CDATA #IMPLIED
 status CDATA #IMPLIED
 >
<!ELEMENT ATDEventId (#PCDATA)>
<!ELEMENT eventType EMPTY>
 <!ATTLIST eventType
 code CDATA #REQUIRED >
<!ELEMENT eventDate (timestamp)>
<!ELEMENT rptPremId (#PCDATA)>
 <!ATTLIST rptPremId
 type CDATA #IMPLIED>
<!ELEMENT id (#PCDATA)>
 <!ATTLIST id
 type CDATA #IMPLIED>
<!ELEMENT srcDestPremId (#PCDATA)>
 <!ATTLIST srcDestPremId
 type CDATA #IMPLIED>
<!ELEMENT animal (DOB*,age*)>
 <!ATTLIST animal
 species CDATA #IMPLIED
 gender CDATA #IMPLIED
 breed CDATA #IMPLIED
 >
<!ELEMENT DOB (timestamp)>
 <!ATTLIST DOB
 est CDATA #REQUIRED>
```

```
<!ELEMENT age (#PCDATA)>
 <!ATTLIST age
 scale (D|M|Y) #REQUIRED>
<!ELEMENT remarks (#PCDATA)>
<!ELEMENT reTagId (#PCDATA)>
 <!ATTLIST reTagId
 type CDATA #IMPLIED>
<!ELEMENT optIds (optId*)>
<!ELEMENT optId (#PCDATA)>
 <!ATTLIST optId
 type CDATA #IMPLIED>
<!ELEMENT group (groupSubsetId*, groupCount*)>
 <!ATTLIST group
 groupType CDATA #IMPLIED
 species CDATA #IMPLIED
 breed CDATA #IMPLIED
 >
<!ELEMENT groupSubsetId (#PCDATA)>
<!ELEMENT groupCount (#PCDATA)>
<!ELEMENT timestamp EMPTY>
 <!ATTLIST timestamp
 y CDATA #REQUIRED
 mo CDATA #REQUIRED
 d CDATA #REQUIRED
 h24 CDATA "0"
 mi CDATA "0"
 s CDATA "0"
 tz CDATA #IMPLIED
 >
```

3.3.2 Strict DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<!—Strict DTD for submission of Animal and Group Events -->
<!ELEMENT eventSub (header,(animalRecords|groupRecords))>
<!ELEMENT header (atpsRequestId,atdResponse)>
<!ELEMENT atpsRequestId (#PCDATA)>
<!ELEMENT atdResponse (responseId)>
 <!ATTLIST atdResponse
 final (Y|N) #REQUIRED
 split CDATA #IMPLIED
 >
<!ELEMENT responseId (#PCDATA)>
<!ELEMENT animalRecords (animalRecord*)>
<!ELEMENT animalRecord (ATDEventId*, eventType, eventDate, rptPre-
mId, id, srcDestPremId*, animal*, remarks*, reTagId*, optIds*)>
 <!ATTLIST animalRecord
 elecRead (Y|N) #IMPLIED
 status (C) #IMPLIED
 >
<!ELEMENT groupRecords (groupRecord*)>
<!ELEMENT groupRecord (ATDEventId*, eventType, eventDate, rptPremId,
id, srcDestPremId*, group*, remarks*)>
 <!ATTLIST groupRecord
 elecRead (Y|N) #IMPLIED
 status (C) #IMPLIED
 >
```

```
<!ELEMENT ATDEventId (#PCDATA)>
<!ELEMENT eventType EMPTY>
 <!ATTLIST eventType
 code (0|1|2|3|4|5|6|7|8|9|10|11|12|13) #REQUIRED >
<!ELEMENT eventDate (timestamp)>
<!ELEMENT rptPremId (#PCDATA)>
 <!ATTLIST rptPremId
 type (N|X) #REQUIRED>
<!ELEMENT id (#PCDATA)>
 <!ATTLIST id
 type (A|U|R|F|N|B|G|T) #REQUIRED>
<!ELEMENT srcDestPremId (#PCDATA)>
 <!ATTLIST srcDestPremId
 type (N|X) #REQUIRED>
<!ELEMENT animal (DOB*,age*)>
 <!ATTLIST animal
 species (ACQ|BOV|CAM|CAP|CER|EQU|OVI|AVI|POR) #IMPLIED
 gender (M|F|C|S|X) #IMPLIED
 breed CDATA #IMPLIED
 >
<!ELEMENT DOB (timestamp)>
 <!ATTLIST DOB
 est (Y|N) #REQUIRED>
<!ELEMENT age (#PCDATA)>
 <!ATTLIST age
 scale (D|M|Y) #REQUIRED>
<!ELEMENT remarks (#PCDATA)>
<!ELEMENT reTagId (#PCDATA)>
 <!ATTLIST reTagId
 type (A|U|R|F|N|B|G|T) #REQUIRED>
<!ELEMENT optIds (optId*)>
<!ELEMENT optId (#PCDATA)>
 <!ATTLIST optId
 type (A|U|R|F|N|B|G|T) #REQUIRED>
<!ELEMENT group (groupSubsetId*, groupCount*)>
 <!ATTLIST group
 groupType CDATA #IMPLIED
 species (AQU|CLM|CRA|CTF|MSL|OYS|SAL|SBA|SHR|SLP|TIL|TRO|
 BOV|BIS|BEF|DAI|CAM|CAP|CER|DEE|ELK|EQU|OVI|
 AVI|CHI|DUC|GEE|GUI|PGN|PHE|QUA|TUR|OTH|POR) #IMPLIED
 breed CDATA #IMPLIED
 >
<!ELEMENT groupSubsetId (#PCDATA)>
<!ELEMENT groupCount (#PCDATA)>
<!ELEMENT timestamp EMPTY>
 <!ATTLIST timestamp
 y CDATA #REQUIRED
 mo (1|2|3|4|5|6|7|8|9|10|11|12) #REQUIRED
 d (1|2|3|4|5|6|7|8|9|
 10|11|12|13|14|15|16|17|18|19|
 20|21|22|23|24|25|26|27|28|29|30|31) #REQUIRED
 h24 (0|1|2|3|4|5|6|7|8|9|10|11|12|
 13|14|15|16|17|18|19|20|21|22|23) "0"
 mi (0|1|2|3|4|5|6|7|8|9|
 10|11|12|13|14|15|16|17|18|19|
 20|21|22|23|24|25|26|27|28|29|
 30|31|32|33|34|35|36|37|38|39|
40|41|42|43|44|45|46|47|48|49|
50|51|52|53|54|55|56|57|58|59) "0"
s (0|1|2|3|4|5|6|7|8|9|
10|11|12|13|14|15|16|17|18|19|
20|21|22|23|24|25|26|27|28|29|
30|31|32|33|34|35|36|37|38|39|
40|41|42|43|44|45|46|47|48|49|
50|51|52|53|54|55|56|57|58|59) "0"
tz (GMT|GMT-1|GMT-2|GMT-3|GMT-4|GMT-5|GMT-6|
GMT-7|GMT-8|GMT-9|GMT-10|GMT-11|GMT-12|
GMT12|GMT11|GMT10|GMT9|GMT8|GMT7|GMT6|
GMT5|GMT4|GMT3|GMT2|GMT1) #IMPLIED
>
```

3.4 Official ID Codes

Code	Description
A	Official ID with leading "USA"
U	USDA ID, not "840"
R	ID with lead manufacturer code
F	Foreign Country ID
N	USDA NAIS AIN ID with leading "840"
G	Group Identification Number
T	Tattoo
B	Breed Registry number
X	Unknown

3.5 Species Group Codes

Species Group Code	Description
AQU	Aquaculture
BOV	Bovine (Bison and Cattle)
CAM	Camelid (Alpaca and Llama)
CAP	Caprine (Goats)
CER	Cervids
EQU	Equine (Horses)
OVI	Ovine (Sheep)
AVI	Avian
POR	Porcine (Swine)

3.6 Species Codes

Includes Species and Species Group Codes

Code	Description
AQU	Aquaculture
CLM	Clams
CRA	Crawfish
CTF	Catfish
MSL	Mussels
OYS	Oysters
SAL	Salmon
SBA	Striped Bass
SHR	Shrimp
SLP	Scallops

Code	Description
CAM	Camelid (Alpaca and Llama)
CAP	Caprine (Goats)
CER	Cervids
DEE	Deer
ELK	Elk
EQU	Equine (Horses) [1]
OVI	Ovine (Sheep)
AVI	Avian
CHI	Chickens
DUC	Ducks

TIL	Tilapia
TRO	Trout
BOV	Bovine (Bison and Cattle)
BIS	Bison
BEF	Beef
DAI	Dairy

GEE	Geese
GUI	Guineas
PGN	Pigeon
PHE	Pheasants
QUA	Quail
TUR	Turkeys
OTH	Other
POR	Porcine (Swine)
	Feral
	Transitional
	Commercial

3.7 Breed Codes

Dairy Breeds	
Breed	Code
American Lineback	LD
Ayrshire	AY
Brown Swiss	BS
Canadian Lineback	LK
Galloway	GD
Guernsey	GU
Holstein	HO
Jersey	JE
Kerry	KY
Red Holstein	WW
Rouge Flamand	FM
Shorthorn	MS
Beef Breeds	
Breed	Code
Aberdeen Angus	AN
Abondance	AB
Africander	AF
Alpine	AL
American Breed	AE
Amerifax	AM
Ankina	AK
Ankole-Watusi	AW
Aubrac	AU
Barzona	BA
Beef Friesian	BF

Goat Breeds	
Breed	Code
Alpine	AI
Angora	AG
Boer	BZ
Cashmere	CS
La Mancha	LN
Nigerian Dwarf	ND
Nubian	NU
Oberhasli	OH
Pygmy	PY
Saanen	EN
Toggenburg	TO
Sheep Breeds	
Breed	Code
Arcott – Canadian	CD
Arcott – Outaouais	OU
Arcott – Rideau	RI
Barbados Black Belly	LY
Black Face	FB
Black Welsh Mountain	BW
Blue Faced Leister	BF
Booroola	BO
Border Cheviot	BC
Charollais	CO
Clun Forest	CF
Columbia	CL

Beefalo	BE
Beefmaster	BM
Belgian Blue	BB
Belted Galloway	BG
Blonde d'Aquitane	BD
Bonsmara	NS
Braford	BO
Brahman	BR
Brahmental	BH
Brahmousin	BI
Braler	BL
Brangus	BN
Braunveih	BU
British White	BW
Brown Swiss (beef/boeuf)	SB
Buelingo	BQ
Campine Red Pied	CP
Canadienne	CN
Charbray	CB
Charolais	CH
Chi-Angus	CG
Chianina	CA
Chi-Maine	CM
Crossbred Twinner	XT
Crossbreeds	XX
Cumberland	CU
Danish Black and White	DB
Danish Jersey	DJ
Danish Red and White	RW
Devon	DE
Dexter	DR
Dutch Belted	DL
East Flemish Red Pied	FP
Eringer	ER
Flamande	FA
Fleckvieh	FL
Florida Cracker	FC
Fribourg	FR
Friesian (Belgium)	FB
Friesian (Dutch)	DF
Galloway	GA
Gasconne	GS
Gelbray	GE
Gelbvieh	GV
Grauvieh	GI
Groningen	GR
Guzera	GZ
Gyr (or Gir)	GY

Coopworth	CP
Corriedale	CR
Cottswold	CW
Crossbred – Large	XL
Crossbred – Medium	XM
Crossbred – Small	XS
DLS	DL
Dorper	DO
Dorset – Horned	DH
Dorset – Polled	DP
Drysdale	DY
East Friesian	EF
Finnish Landrace	FN
Hampshire	HS
Hybrid	HY
Icelandic	IL
Ile de France	IF
Jacob	JA
Karakul	KK
Katahdin	KA
Kerry Hill	KH
Lacaune Dairy Sheep	CU
Leicester – Border	BL
Leicester – English	LE
Leister – Hexam	HL
Lincoln	LI
Merino	MM
Merino Polled	MP
Montadale	MT
Newfoundland Loco	NL
North Country Cheviot	NC
Oxford	OX
Perendale	PE
Polypay	PO
Rambouillet	RG
Romanov	RV
Romnelet	RM
Romney	RY
Rouge de l'Ouest	RO
Ryeland	RL
Scottish Blackface	SC
Shetland	SL
Shropshire	SR
Southdown	ST
St. Croix	SX
Suffolk	SU
Targhee	TA
Texel	TX

Hays Converter	HC
Hereford (black)	HB
Hereford (horned)	HH
Hereford (polled)	HP
Highland (Scotch Highland)	SH
Hybrid (Alberta Synthetic)	HY
Indu Brazil	IB
Kerry	KY
Kobe (Wagyu)	KB
Limousin	LM
Lincoln Red	LR
Lowline (Loala)	LO
Luing	LU
Maine-Anjou	MA
Mandalong Special	ML
Marchiginana	MR
Maremmana	ME
Mashone	MH
Meuse-Rhine-Issel	MI
Mexican Corriente	MC
Montbeliard	MO
Murrah	MU
Murray Grey	MG
Nellore	NE
Normande	NM
Norwegian Red	NR
Parthenaise	PA
Pie Rouge	PR
Piedmontese	PI
Pinzgauer	PZ
Ranger	RA
Red Angus	AR
Red Brahman	RR
Red Brangus	RB
Red Dane (Danish Red)	RD
Red Poll	RP
Romagnola	RN
Romosinuano	RS
Rotbunte	RO
Rouge du Nord	DN
Sahiwal	SW
Salers	SA
Santa Gertrudis	SG
Semepol	SL
Senapol	SE
Shaver Beef Blend	SV

Tunis	TU
Bison Breeds	
Breed	**Code**
Plains Bison	PB
Wood Bison	WO
Horse Breeds	
Breed	**Code**
Andalusian	AA
American Bashkir Curly	AC
Arabian	AD
Anglo-Arabian	AO
Appaloosa	AP
American Saddlebred	AS
Buckskin	BU
Baden-Wurttemberg	BW
Bayerisches Warmblood	BY
Canadian Horse	CI
Connemara	CM
Cleveland Bay	CV
Clydesdale	CY
Dartmoor Pony	DT
Danish Warmblood	DW
Dutch Warmblood	DW
Exmoor Pony	EX
Cheval de Selle Francais (French Saddle Horse)	FC
Fell Pony	FE
French Horse	FH
Fjord	FJ
Friesian	FR
Belgian	GI
Gelderlander	GL
Belgian Warmblood	GW
Hessen	HE
Haflinger	HF
Highland Pony	HG
Hackney Pony	HK
Hackney Horse	HN
Holsteiner	HT
Hunter (Sport Horse)	HU
Hanovarian	HV
Hungarian Warmblood	HW
Icelandic	IC
Swiss Warmblood	IW
Lipizzaner	LZ
Missouri Foxtrotting	MF
Morgan	MN

Shorthorn (polled)	SP
Shothorn (beef-scotch)	SS
Shothorn (Illawarra)	IS
Simbrah	SI
Simmental	SM
South Devon	DS
Sussex	SX
Taba-pua	TB
Tarentaise	TA
Tasmanian Grey	TG
Taurindicus	TN
Texas Longhorn	TL
Tuli	TI
Welsh Black	WB
West Flemish Red	WF
White Park	WP
Yak	YA
Swine breeds	
Breed	***Code***
Berkshire	BK
Chester White	CW
Duroc	DU
Hampshire	HA
Lacombe	LC
Landrace	LA
Large Black (British)	LB
Large White	LW
Pietrain	PE
Poland China	PC
Red Wattle	RW
Spotted	SO
Tamworth	TM
Wessex Saddleback	WS
Yorkshire	YO

Miniature Horse	MU
New Forest	NF
Noriker	NK
Anglo-Normand	NO
Oldenburg	OB
Polo Pony	OL
Paso Fino	PF
Percheron	PH
Palomino	PL
Pinto	PN
Paint	PT
Peruvian	PV
Polish Warmblood	PW
Quarter Horse	QH
Rheinlander	RH
Rustic Pony	RU
Shetland	SE
Suffolk Punch	SF
Standardbred	SN
Shire	SY
Trotteur Francais	TF
Thoroughbred	TH
Tarpan	TP
Trakehner	TR
Tennessee Walking	TW
Viking	VK
Welsh	WE
Westfalen	WF
German Warmblood	WG
Swiss Horse	WI
Wielkopolski (Polish Trakehner)	WR
Wurttemberg	WU
Swedish Warmblood	WW