Integration Development Group

Product Adoption

Third Party Provider Behaviors Guidelines

jack henry Banking®

A DIVISION OF JACK HENRY & ASSOCIATES INC®

jXchange Header Behavior

- Tracking Utilization
- Lives in Request and Response Message
- Specific Behavior

Request Structure

XSD Element Path	Required/Optional/Conditional	Comment
jXchangeHdr.JXVer	O	jXchange will return the current version deployed regardless of the value in the request header
jXchangeHdr.AuditUserId	R	This is the User Id which the consumer would like written to the audit as performing the requested service. It will vary but could be down to the user id. It will not be used to authenticate however, just audit.
jXchangeHdr.AuditWsId	R	This is the Workstation Id which the consumer would like written to the audit as performing the requested service
jXchangeHdr.ConsumerName	O	The name of the consumer that is consuming the service
jXchangeHdr.ConsumerProd	O	The name of the product which is consuming the service (Business Product Name)
jXchangeHdr.jXLogTrackingId	O	jXchange could create Id when not submitted by Consumer
jXchangeHdr.InstRtId	C	The identification of the entity of submitted message. A financial institution will utilize the routing transit or ABA nine (9) digit number assigned to financial institutions for the purpose of routing as assigned by the American Bankers Association. Any leading zeros must be provided for a complete routing and transit number. A non-financial institution entity will use a mutually agreed upon identification that must contain at least one non-integer character. When a record is directed to a specific Financial Institution within a holding company, the institution routing identification is specific Financial Institution routing identification and not the holding company identification
jXchangeHdr.InstEnv	O	An identification provided by the consumer that defines the environment in which the institution is operating. Production (PROD) is the default value

jXchangeHdr.BusCorrelId	O	The correlation identification as related to business functions and activities
jXchangeHdr.WorkFlowCorelId	O	The correlation identification as related to workflow functions and activities
jXchangeHdr.ValidConsmName	O	The consumer name that can be validated by Enterprise goverance. The consumer product name that can be validated by Enterprise goverance. **Behavior:** The combination of the valid consumer name and valid consumer product would provide the type of device being used by the consumer based on settings for the transformation layer.
jXchangeHdr.ValidConsmProd	O	The consumer product name that can be validated by Enterprise goverance.

Responses should echo back the information provided in the jXchange Header with the Consumer request.

Institution Routing Identification Behavior

- The Service Provider will need to map to the submitted Institution Routing number <InstRtId> element if their application maintains a different institution identifier
- The Institution Routing Number <InstRtId> can be the nine (9) digit assigned by the American Bankers Association (ABA).
    - All leading zeros must be included to be a complete routing and transit number
- The absence of the Institution Environment <InstEnv> element will equate to “PROD” = Production Environment.
    - It will be the responsibility of the sender to send the appropriate data to identify the environment if necessary

Backwards Compatibility

Backward compatibility is a relationship between two components, rather than being an attribute of just one of them. More generally, a new component is said to be backward compatible if it provides all of the functionality of the old component. *Backward* compatibility is the special case of compatibility in which the new component has a direct historical ancestral relationship with the old component. jXchange maintains Backwards Compatibility with the user of Version Tags

The notion of compatibility applies to messages. In the case of a message a new version of that message ("v2") is said to be backward compatible with the old version of the message ("v1") when it can both send and receive messages that work with v1. Everything that v1 could do must also be possible with v2, that can be read by v1 (which is something that v1 could do.)

XSD iteration versions are interim releases of an XML schema that contain only the changes that are backwards compatible the existing version of this schema. Changes that can be incorporated in a iterated version:

- **Backward Changes**
  - Adding new optional elements or optional attributes
  - Changing attributes cardinality from mandatory to optional
  - Adding a term to an enumerated list

- **Non-Backward Changes**
  - Changing an attribute cardinality from optional to mandatory
  - Adding a mandatory element
  - Changing an attribute or element tag name

EA Architectural Guidelines

The Enterprise Architect XSD contracts incorporate version tags within an object. The version tags represent an iteration growth of the object but will be backwards compatible. The version tag is embedded in an optional sequence. This allows the iteration to be optional based on the optional sequence but the version tag is required which conveys to the service provider that the consumer understands the preceding objects after the version tag. This structure intuitively allows the consumer to ignore the additional objects without breaking any existing application processes.

The concept of backwards compatibility extends beyond the structural design of XSD contracts to include the business services (programs) that support the XSD contracts. All service providers should adhere to maintaining backwards compatibility so not to disturb consumers' applications. Changes that are not backwards compatible:

- Creating errors due to Service Provider's data base field to element size differentials
  - Service Providers should truncate and return a warning to consumer
- Creating hard errors for previously successful message requests
  - Service Providers should establish a tenet that allows the business service to get around the error situation;
  - Regulatory changes could be an exception to this rule
- Changing an Error Code <ErrCode> from one value to another for an existing error or fault
- Removing a business service so the message can no longer be supported
  - Service Providers should adhere to the established deprecation policy
- Removing support of a previously support closed enumerated value
  - Service Providers should establish a tenet that allows the canonical value to be translated to acceptable value

XSD Contract Iteration Example

```
<xsd:complexType name="CustDetail_CType">
    <xsd:sequence>
        <xsd:element name="PersonName" type="PersonName_CType" minOccurs="0"/>
        <xsd:element name="Addr" type="Addr_CType" minOccurs="0"/>
        <xsd:element name="CustType" type="CustType_Type" minOccurs="0" nillable="true"/>
        <xsd:element name="NAICSCode" type="NAICSCode_Type" minOccurs="0" nillable="true" />
        <xsd:element name="StdIndustCode" type="StdIndustCode_Type" minOccurs="0" nillable="true" />
        <xsd:element name="Gender" type="Gender_Type" minOccurs="0" nillable="true"/>
        <xsd:element name="BirthDt" type="BirthDt_Type" minOccurs="0" nillable="true"/>
        <xsd:element name="EmplName" type="EmplName_Type" minOccurs="0" nillable="true" />
        <xsd:element name="OccType" type="OccType_Type" minOccurs="0" nillable="true" />
        <xsd:element name="CustOrgDt" type="CustOrgDt_Type" minOccurs="0" nillable="true" />
        <xsd:element name="LastMainDt" type="LastMainDt_Type" minOccurs="0" nillable="true"/>
        <xsd:element name="Deceased" type="Deceased_Type" minOccurs="0" nillable="true" />
        <xsd:element name="DeceasedDt" type="DeceasedDt_Type" minOccurs="0" nillable="true"/>
        <xsd:element name="EmailArray" type="EmailArray_AType" minOccurs="0"/>
        <xsd:element name="PhoneArray" type="PhoneArray_AType" minOccurs="0"/>
        <xsd:element name="Custom" type="Custom_CType" minOccurs="0"/>
        <xsd:sequence minOccurs="0">                                          <- Object Iteration
            <xsd:element name="Ver_1" type="Ver_1_CType"/>
            <xsd:element name="HouseHoldNum" type="HouseHoldNum_Type" minOccurs="0" nillable="true" />
            <xsd:element name="HouseHoldName" type="HouseHoldName_Type" minOccurs="0" nillable="true" />
            <xsd:element name="SpouseName" type="PersonName_CType" minOccurs="0" nillable="true" />
            <xsd:sequence minOccurs="0">
                <xsd:element name="Ver_2" type="Ver_2_CType"/>
                <xsd:any namespace="##targetNamespace" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:sequence>
    </xsd:sequence>
</xsd:complexType>
```

Error/Fault Behaviors

JXchange, error handling has been designed into the messaging structure and involves notification through use of codes, categories and descriptions, as well as options for error overrides for non-fatal errors.

Structure

- **ErrCode** – Error Code (Required)
- **ErrCat** – Error Category (Required)
- **ErrDesc** – Error Description (Required)
- **ErrElement** – Error Element (Optional)
- **ErrElementVal** – Error Element Value (Optional)
- **ErrLoc** – Error Location (Optional)

Error Categories

- **Warning** – Successful transmission of requested response transmission, but information must be returned to the consumer describing under what conditions the successful response was able to be created
- **Error** – Failure condition that cannot be overridden and must be corrected before processing can be completed
- **Fault** – Failure condition that can be overridden
- **Override** – Specific kind of warning notifying the customer that the fault has been overridden

Behaviors

- **Override Behavior**
  - When a consumer wishes to override a fault then he must send the unique error code in the element Error Override Information array in the operation request and the service provider will understand that they should ignore the list of codes given.
  - If the <ErrOvrRd> element contains a maximum number of nines (9999999 value for a seven digit integer), the Service Provider is notified that all faults should be overridden, if possible

- **Parallel vs. Serial Error Message Handling**
  - Serial – a Service Provider will issue an error response message when it has discovered the first error in a request.
  - Parallel – The Service Provider can continue to process a request message after detecting a fault capable of being overridden in an attempt to identify all possible errors before returning an error message to the Consumer
  - Parallel Error Message Handling is the most efficient method and is suggested as the preferred error handling system for Service Providers.

Enumerated Elements

These are elements that have a pre-defined set of data values. The XSD contracts define enumerated elements by data type Closed Enumerated or Open Enumerated. The EAG defined types adopt the *string* primitive data type.

Closed Enumerated Elements

These are elements that have a pre-defined set of data values. The XSD contracts define enumerated elements by data type Closed Enumerated or Open Enumerated. The EAG defined types adopt the string primitive data type.

The Closed Enumerated values are defined in the XSD contract in the form of annotations. The fixed values are the only data set that a consumer of these elements needs to understand. A Service Provider may return a fault when a value is delivered in a message that a Service Provider does not understand. However; an XML document with values not defined in the contract will pass schema validation. This behavior allows for closed enumerated values to be forward compatible whereas the Consumer and the Service Provider understand a value but the XSD contract annotated values have not been updated.

The closed enumerated values are a much more effective means from the perspective of a SOA guideline as it reduces any ambiguity behavior of an element. The Closed enumeration values can be used by the jXchange Framework in determining how to invoke an operation. Some of the behaviors that goes along with this are; (1) if the element is not sent or sent empty, and it is required, a fault error will be returned; (2) if the element is not sent or sent empty, and it is optional, a default will be used and; (3) if the element is sent with an incorrect value, a fault

error could be returned. Generally, a name closed enumerated element will be suffixed with ~Type~ or ~Stat~.

A service provider could transform the XSD contract defined values to values that are understood by their application. For example, a service provider might accept the closed enumerated value of ~Months~ but could store that value in their application as ~M~.

Closed Enumerated Element Example

```
<xsd:complexType name="TermUnits_Type">
        <xsd:annotation>
            <xsd:documentation>
                <Jx>
                    <ElemDesc>term units Years,Months, Days,Indefinite
</ElemDesc>

    <CanonicalVal>Days,Months,Years,SemiMonthly,Indefinite,NA</CanonicalVal>
                </Jx>
            </xsd:documentation>
        </xsd:annotation>
        <xsd:simpleContent>
            <xsd:extension base="ClosedEnum_Type">
                <xsd:attribute name="JHANull" type="JHANull_Type"
use="optional"/>
                <xsd:attribute name="Rstr" type="Rstr_Type" use="optional"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>
```

Open Enumerated Elements

The Open Enumerated values are a definite set of values but those values are not represented in the contract. These enumerated values often differ from Service Provider to Service Provider as well as service provider installations sites to service provider installations sites.

Open enumeration elements are generally suffixed with Code. The element that is suffixed with Code has a mate element that is suffixed with Desc. This is because often a service provider field is represented as a code that does not convey its representation thereby the service provider returns the description of the code that is a literal value that can be understood by the consumer. Generally, an element suffixed with ~Code~ could be found in Addition and Modification business operations whereas an element suffixed with ~Desc~ would be found in Inquiry, Multi-Inquiry, and Search business operations. An example might be <IntCalcCode> would have a mate called <IntCalcDesc>.

- Service Provider conveys these canonical values by means of web service call – Service Dictionary Search (SvcDictSrch)

Open Enumerated Element Example

```
<xsd:complexType name="IntCalcCode_Type">
        <xsd:annotation>
            <xsd:documentation>
                <Jx>
                    <ElemDesc>Interest calculation method </ElemDesc>
                </Jx>
            </xsd:documentation>
        </xsd:annotation>
        <xsd:simpleContent>
            <xsd:extension base="OpenEnum_Type">
                <xsd:attribute name="JHANull" type="JHANull_Type"
use="optional"/>
                <xsd:attribute name="Rstr" type="Rstr_Type" use="optional"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>
```

Concurrency Models

- The application and database must work in concert to provide the appropriate level of data integrity and performance while minimizing user rework to address conflicts and deadlocks.
- Optimistic, Pessimistic and Chaos refer to three different types of concurrency concepts and the safeguards an application will take based on the likelihood of concurrent updates and how much rework is acceptable
    - **Optimistic** concurrency control is used when it is unlikely that different users will update the same data
    - **Pessimistic** concurrency control is used when it is likely that the same data will be updated by different users
    - **Chaos** concurrency control is used in situations when concurrent updates are not possible or “last in wins” is acceptable.

Concurrency EA Guidelines

In database systems, a consistent transaction is one that does not violate any integrity constraints during its execution and ensures that correct results for concurrent operations are generated, while getting those results as quickly as possible.

The application and database must work in concert to provide the appropriate level of data integrity and performance while minimizing user rework to address conflicts and deadlocks.

Optimistic, Pessimistic and Chaos refer to three different types of concurrency concepts and the safeguards an application will take based on the likelihood of concurrent updates and how much rework is acceptable.

Optimistic concurrency control is used when it is unlikely that different users will update the same data. In the unlikely event that the same data is updated by different users, the conflict is detected when data are saved and the second user must redo/merge changes in order to prevent overwriting the changes made by the first user. Users of a well designed optimistic concurrency application experience fast response time and are inconvenienced only in the rare case of an update conflict.

Pessimistic concurrency control is used when it is likely that the same data will be updated by different users. To prevent the need to redo or merge changes, an application serializes data access so that only one user can edit data at a time. The obvious downside is that subsequent users must wait until preceding user(s) has completed their changes and this can increase response time or data unavailability. However, overall user productivity can be better than optimistic currency control because rework is avoided.

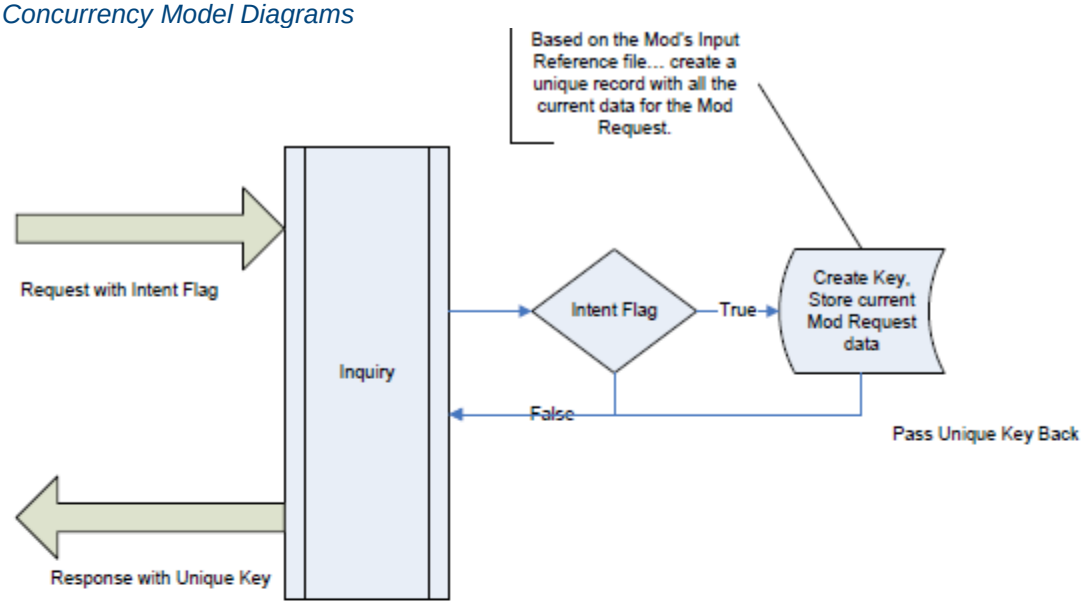
Chaos concurrency control (also known as Anarchy) is used in situations when concurrent updates are not possible or “last in wins” is acceptable. No safeguards need to be taken with chaos concurrency because there is either no chance of conflicts or overwrites are ok. Chaos concurrency is typically used in single-user applications or in multi-user applications where data are segregated in such a way that concurrent updates are either not possible (e.g. unique web session key) or so unlikely (e.g. CustomerID key) that the risk of lower concurrency level isn’t warranted

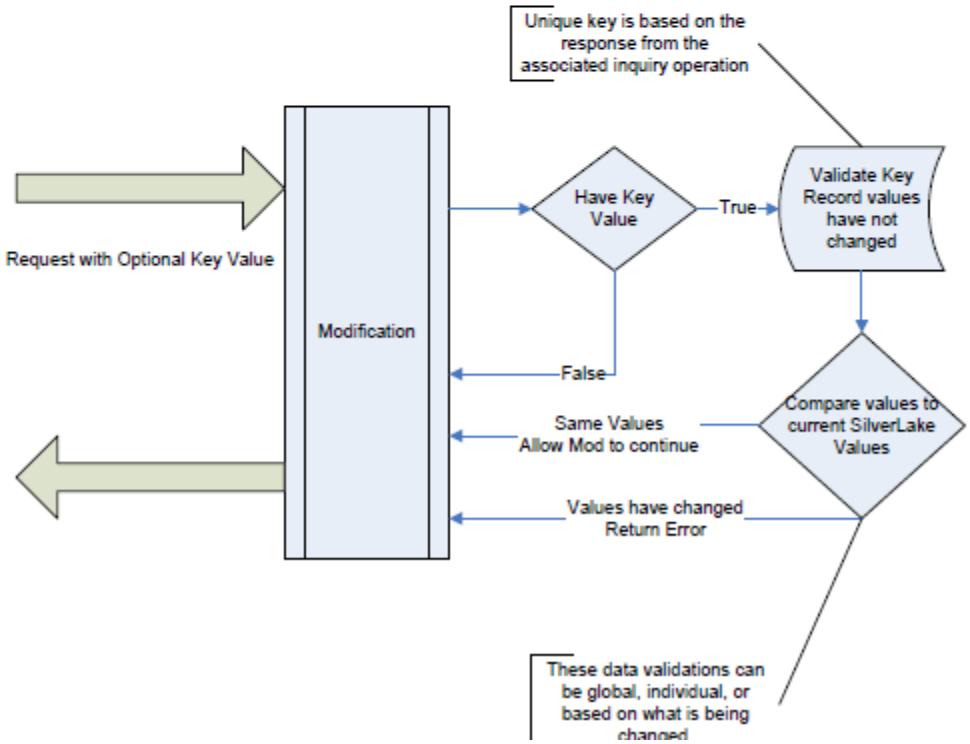
- Adoption of the Optimistic Concurrency model for modification services
    - The activity intention element will exist on all inquiry operations that support a modification mate
- The Activity Intention element allows the consumer to convey to the service provider their intention for the data returned in the response.
    - The Activity Intention element <ActIntent> supports three canonical values:
        - **Read Only (ReadOnly)** – this is the default
        - **Update (Upd)**
        - **Delete (Dlt)**
- The service provider will echo back the Activity Intention element in the response.
    - The service provider will return an Activity Intention Key when the consumer submits the Activity Intention element canonical values of Update or Delete.
    - The service provider must return a unique Activity Intention Key <ActIntentKey> for each item that is returned when the response includes an array of elements.

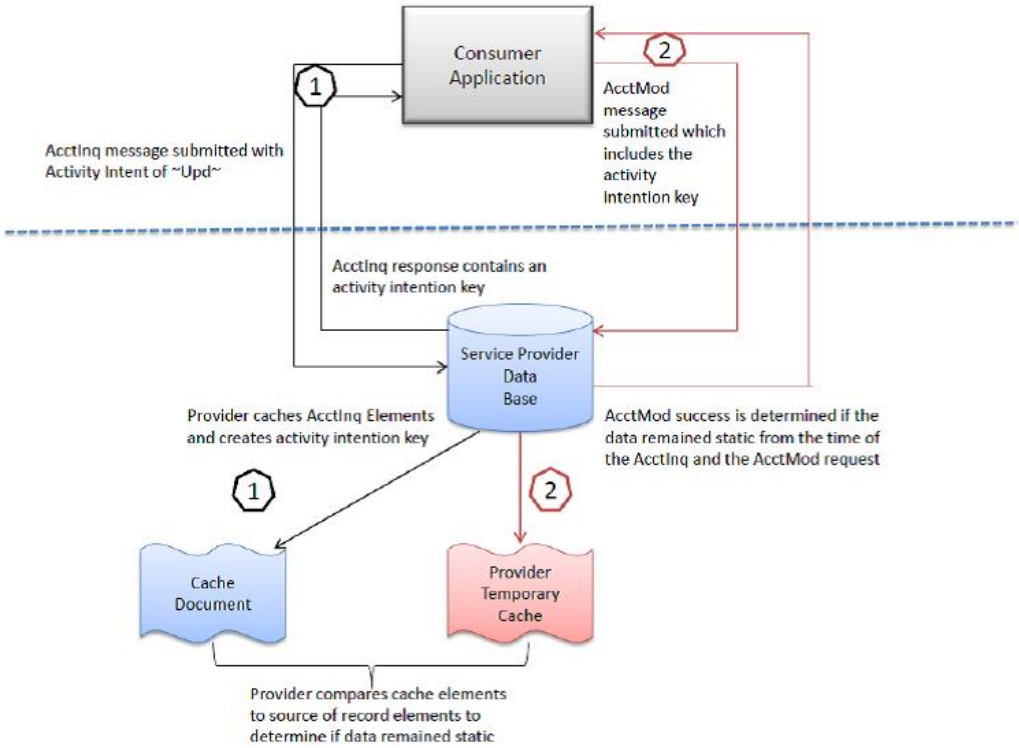
- The consumer should cache the Activity Intention Key so it can be submitted in a subsequent modification request.

- The Service provider will return a fault when the provider's saved data does not match the intended updated data when the consumer's modification request with Activity Intention Key <ActIntentKey> is submitted on the modification request.

Concurrency Model Diagrams







Addition Service Guidelines

- The named EA XSD contract will be suffixed with ~Add~
- The business function will convey to the service provider to create an entry/record in the service provider's data base for the supported business service
- The service provider would return the added entry/records keys in the response if applicable
- The majority of the child node elements will be decorated with the JHANull attribute
- Generally, the request message will have a complex object that is required but all the elements inside the complex object are optional. This should convey that at least one of the elements inside the object is required
- A consumer may send a Service Dictionary Search (SvcDictSrch) message for all addition services to obtain the service provider's element default values. This will assist the consumer in their validation process for elements without having to deliver the XML payload and receiving an error that an element is required by the service provider even though the XSD contract references the element as optional

- The EA XSD addition contracts make use of re-used complex objects. Thereby, often many of the child nodes (elements) encapsulated by a parent node (complex) are optional from an XSD contract perspective. However; service providers will have minimum requirements as to the required elements which will be conveyed to the consumer by means of message fault(s).

Addition Service XSD Contract Example

```
<xsd:complexType name="AcctBenfAddRq_MType">
  <xsd:sequence>
    <xsd:element name="MsgRqHdr" type="MsgRqHdr_CType"/>
    <xsd:element name="ErrOvrRdInfoArray" type="ErrOvrRdInfoArray_AType" minOccurs="0"
      nillable="true"/>
    <xsd:element name="AccountId" type="AccountId_CType"/>
    <xsd:element name="AcctBenf" type="AcctBenf_CType"/>
    <xsd:element name="Custom" type="Custom_CType" minOccurs="0" nillable="true"/>
    <xsd:sequence minOccurs="0">
      <xsd:element name="Ver_1" type="Ver_1_CType"/>
      <xsd:any namespace="##targetNamespace" processContents="lax" minOccurs="0"
        maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="AcctBenfAddRs_MType">
  <xsd:sequence>
    <xsd:element name="MsgRsHdr" type="MsgRsHdr_CType"/>
    <xsd:element name="BenfKey" type="BenfKey_Type" minOccurs="0" nillable="true"/>
    <xsd:element name="RsStat" type="RsStat_Type" minOccurs="0" nillable="true"/>
    <xsd:element name="Custom" type="Custom_CType" minOccurs="0" nillable="true"/>
    <xsd:sequence minOccurs="0">
      <xsd:element name="Ver_1" type="Ver_1_CType"/>
      <xsd:any namespace="##targetNamespace" processContents="lax" minOccurs="0"
        maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:sequence>
</xsd:complexType>
```

Modification Service Guidelines

The EA XSD contracts that qualify as a modification service, also, serve as a delete service. There is not, generally, an EA XSD contract service specific to a delete business requirement. The method of deletion is discussed below. The EA XSD contracts that qualify as a modification service will generally meet the following requirements:

- The named EA XSD contract will be suffixed with ~Mod~;
- The business function will convey to the service provider to update an entry/record in the service provider's data base for the supported business service;
- The service provider would return the RsStat element as success in the response if the operations request was successful otherwise, a fault is returned;

- The service provider should return a fault when any of the modified elements are invalid on the request and will never return a partial success response;
- The child node elements will be decorated with the JHANull attribute. The behavioral aspect of the attribute is addressed elsewhere in this document;
- The request message will have required element(s) representative of the keys as related to the specific business service;
- The request message will contain a delete element <Dlt> that permits a consumer to submit the key(s) with a Delete=true without a complex object to convey a delete request to the Service Provider. However; the service provider reserves the right to either mark the record for deletion or perform a data base delete request.

```
<xsd:complexType name="AcctBenfModRq_MType">
  <xsd:sequence>
    <xsd:element name="MsgRqHdr" type="MsgRqHdr_CType"/>
    <xsd:element name="ErrOvrRdInfoArray" type="ErrOvrRdInfoArray_AType" minOccurs="0"
      nillable="true"/>
    <xsd:element name="AccountId" type="AccountId_CType"/>
    <xsd:element name="BenfKey" type="BenfKey_Type"/>
    <xsd:element name="AcctBenf" type="AcctBenf_CType" minOccurs="0" nillable="true"/>
    <xsd:element name="Dlt" type="Dlt_Type" minOccurs="0" nillable="true"/>
    <xsd:element name="Custom" type="Custom_CType" minOccurs="0" nillable="true"/>
    <xsd:sequence minOccurs="0">
      <xsd:element name="Ver_1" type="Ver_1_CType"/>
      <xsd:any namespace="##targetNamespace" processContents="lax" minOccurs="0"
        maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="AcctBenfModRs_MType">
  <xsd:sequence>
    <xsd:element name="MsgRsHdr" type="MsgRsHdr_CType"/>
    <xsd:element name="RsStat" type="RsStat_Type" minOccurs="0" nillable="true"/>
    <xsd:element name="Custom" type="Custom_CType" minOccurs="0" nillable="true"/>
    <xsd:sequence minOccurs="0">
      <xsd:element name="Ver_1" type="Ver_1_CType"/>
      <xsd:any namespace="##targetNamespace" processContents="lax" minOccurs="0"
        maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:sequence>
</xsd:complexType>
```

Inquiry Service Guidelines

- The named EA XSD contract will be suffixed with ~Inq~
- The request message will have required key(s) however; the key(s) may be presented in the request as a documented choice statement
- The service provider will return a fault when an invalid key(s) is submitted on the request

- The service provider would return the elements as related to the requested key in the response
- Some of the child or parent node elements could be decorated with the Restriction ~Rstr~ attribute
- The response message may be a multi-occurrence object depending upon the business activity of the inquiry message
- The request message may contain the Include Extended Element array that may require the consumer to submit a list of “x_” named objects to be returned in the response
    - A request made with a valid key but does not include any of the “x_” named objects could be returned a response with no complex objects.
- The RsStat element is **NOT** included in the response because successful response package is an implied <RsStat>Success</RsStat> element value

Search Service Guidelines

- The names EA XSD contract will be suffixed with ~Srch~
- The request message may have required key(s) however; the key(s) may be presented in the request as a documented choice statement
- The request message could have a number of elements that can serve as filters that allows a consumer to reduce the response set being returned
- The request filters will be echoed back at the root response message
- Some of the child or parent node elements could be decorated with the Restriction ~Rstr~ attribute. The behavioral aspect of the attribute is addressed elsewhere in this document.
- The response message will always be a multi-occurrence object
- The request message may contain the Include Extended Element array that allows a consumer to include a list of “x_” named objects to be included with the default response set.
- The search message has a specific message header that differs from other message headers to allow the service providers to provide a pseudo state in a stateless message environment.
    - The Search Message headers differ from other message headers so that a Service Provider can convey a data set state within a message paradigm that maintains a stateless environment. The elements encapsulated in the Search Message Header are used to control the records and the number of those records that returned to the consumer. A Service Provider establishes the number of maximum records that is returned to a consumer. A Service Provider will default a value for the maximum records when the consumer’s maximum records values in the operation request exceeds the Service Provider’s established thresholds for an operation.

```
<xsd:complexType name="SrchMsgRqHdr_CType">
  <xsd:annotation>
    <xsd:documentation>Search specific message request headers</xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:element name="jXchangeHdr" type="jXchangeHdr_CType"/>
    <xsd:element name="MaxRec" type="MaxRec_Type"/>
    <xsd:element name="Cursor" type="Cursor_Type" minOccurs="0"/>
    <xsd:sequence minOccurs="0">
      <xsd:element name="Ver_1" type="Ver_1_CType"/>
      <xsd:element name="AuthenUsrCred" type="AuthenUsrCred_CType" minOccurs="0"/>
      <xsd:sequence minOccurs="0">
        <xsd:element name="Ver_2" type="Ver_2_CType"/>
        <xsd:any namespace="##targetNamespace" processContents="lax" minOccurs="0"
          maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:sequence>
  </xsd:sequence>
</xsd:complexType>
```

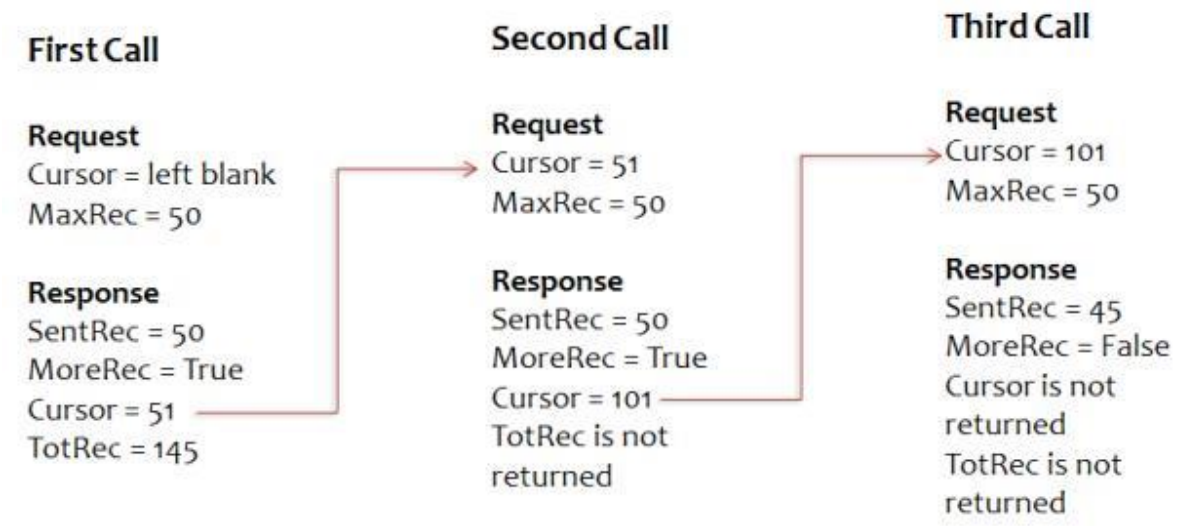
MaxRec (Request) - Tells the provider the maximum number of records to return in the response

Cursor (Request) - Tells the provider at which record to begin returning the results

```
<xsd:complexType name="SrchMsgRsHdr_CType">
  <xsd:annotation>
    <xsd:documentation>Search specific message response headers</xsd:documentation>
  </xsd:annotation>
  <xsd:sequence>
    <xsd:element name="jXchangeHdr" type="jXchangeHdr_CType"/>
    <xsd:element name="SentRec" type="SentRec_Type" minOccurs="0"/>
    <xsd:element name="MoreRec" type="MoreRec_Type" minOccurs="0"/>
    <xsd:element name="Cursor" type="Cursor_Type" minOccurs="0"/>
    <xsd:element name="TotRec" type="TotRec_Type" minOccurs="0"/>
    <xsd:element name="MsgRecInfoArray" type="MsgRecInfoArray_AType" minOccurs="0"/>
    <xsd:sequence minOccurs="0">
      <xsd:element name="Ver_1" type="Ver_1_CType"/>
      <xsd:any namespace="##targetNamespace" processContents="lax" minOccurs="0"
        maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:sequence>
</xsd:complexType>
```

SentRec (Response) - Tells the user how many records were returned; may or may not be equal to the maximum number requested
MoreRec (Response) - Tells the user if there are more records available than what was returned
Cursor (Response) -Tells the user which record (as a number) would be next to be returned

Search Message Header Behavior Example



- The following are those elements and their respective role in the search message header;The RsStat element is NOT included int the response because successful response package is an implied element value
- A request made with a valid key, if applicable, but due to filters submitted in the request returned a zero set of response does not constitute a fault. The service provider will provide a warning to convey that no records matched the consumer's search criteria
- The response set is typically a limited element response set but could return some key(s) that allows a consumer to capture to make an inquiry message service call if applicable for the search business activity

```
<xsd:complexType name="AcctHistSrchRq_MType">
        <xsd:sequence>
                <xsd:element name="SrchMsgRqHdr" type="SrchMsgRqHdr_CType"/>
                <xsd:element name="InAcctId" type="AccountId_CType"/>
                <!-- This is a documented filter statement - Any or All of the following can be sent -->
                <xsd:element name="ChkNumStart" type="ChkNumStart_Type" minOccurs="0" nillable="true"/>
                <xsd:element name="ChkNumEnd" type="ChkNumEnd_Type" minOccurs="0" nillable="true"/>
                <xsd:element name="StartDt" type="StartDt_Type" minOccurs="0" nillable="true"/>
                <xsd:element name="EndDt" type="EndDt_Type" minOccurs="0" nillable="true"/>
                <xsd:element name="LowAmt" type="LowAmt_Type" minOccurs="0" nillable="true"/>
                <xsd:element name="HighAmt" type="HighAmt_Type" minOccurs="0" nillable="true"/>
                <xsd:element name="SrtMthd" type="SrtMthd_Type" minOccurs="0" nillable="true"/>
                <xsd:element name="TmType" type="TmType_Type" minOccurs="0" nillable="true"/>
                <xsd:element name="EFTOnly" type="EFTOnly_Type" minOccurs="0" nillable="true"/>
                <xsd:element name="MemoPostInc" type="MemoPostInc_Type" minOccurs="0" nillable="true"/>
                <!-- End documented filter statement -->
                <xsd:element name="Custom" type="Custom_CType" minOccurs="0" nillable="true"/>
                <xsd:sequence minOccurs="0">
                        <xsd:element name="Ver_1" type="Ver_1_CType"/>
                        <xsd:element name="XferKey" type="XferKey_Type" minOccurs="0" nillable="true"/>
                        <xsd:sequence minOccurs="0">
                                <xsd:element name="Ver_2" type="Ver_2_CType"/>
                                <xsd:any namespace="##targetNamespace" processContents="lax" minOccurs="0"
                                        maxOccurs="unbounded"/>
                </xsd:sequence>
        </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="AcctHistSrchRs_MType">
        <xsd:sequence>
                <xsd:element name="SrchMsgRsHdr" type="SrchMsgRsHdr_CType"/>
                <xsd:element name="AcctHistSrchRecArray" type="AcctHistSrchRecArray_AType" minOccurs="0"
                        nillable="true"/>
                <xsd:element name="SvcPrvdInfo" type="AcctHistSrchRs_EType" minOccurs="0"  nillable="true"/>
                <xsd:element name="Custom" type="Custom_CType" minOccurs="0" nillable="true"/>
                <xsd:sequence minOccurs="0">
                        <xsd:element name="Ver_1" type="Ver_1_CType"/>
                        <xsd:any namespace="##targetNamespace" processContents="lax" minOccurs="0"
                                maxOccurs="unbounded"/>
                </xsd:sequence>
        </xsd:sequence>
</xsd:complexType>
```

Search Record Request / Response Behavior

There are elements that pertain strictly to search operations that are used to control the records and the number of those records that are returned to the user.

- **Cursor** (Request) – Tells the provider at which record to begin returning the results
- **MaxRec** (Request) – Tells the provider the maximum number of records to return in the response
- **SentRec** (Response) – Tells the consumer how many records were returned; may or may not be equal to the maximum number requested
- **MoreRec**(Response) – Tells the user if there are more records available than what was returned
- Cursor (Response) – Tells the user which record (as a number) would be next to be returned
- **TotRec**(Response) – Tells the user the total number of records that exist in the file that meet the criteria of the request; is not returned if Cursor was included with the request
- An example can be found at the above EA link.

Wildcard Search

A wildcard search allows a consumer to get a response set that is dynamic based on a message request elements. Many elements, conveyed clearly by their name, convey to a consumer a key that then conveys to the service provider that the consumer is interested in the data that matches that key. However; many search filter elements need to provide a consumer a means to search for the matches. This can be referred to as a wildcard search because the consumer does not have enough information to do an exact match search. These types of searches can be based on partial data value. The consumer needs to convey to the service provider as to the type of partial search such as but not limited to Exact Match, Contains within, Starts With, and Ends With.

- A wildcard search allows a consumer to get a response set that is dynamic based on a message request elements
- Many search filter elements need to provide a consumer a means to search for the matches
- This can be referred to as a wildcard search because the consumer does not have enough information to do an exact match search
- These types of searches can be based on partial data value
- The consumer needs to convey to the service provider as to the type of partial search such as but not limited to:
    - **Exact Match**
    - **Contains within**
    - **Starts With**
    - **Ends With**
- A service provider cannot require exact case matches
    - For example, if a consumer submits a last name data value of ~mcgrath~ then the service provider should respond back with any data values that match

regardless of case which could include but not limited to ~McGrath~, ~Mcgrath~, ~mcgrath~, ~MCGRATH~ or any combination of case difference thereafter

Exact Match Search Response Example

Request: <LastName SrchType="Exact">Smi</LastName>

Response: <LastName>Smi</LastName>
<FirstName>John</FirstName>
<LastName>Smi</LastName>
<FirstName>Joe</FirstName>

Contains Match Search Response Example

Request: <LastName SrchType="Contains">Smi</LastName>

Response: <LastName>Smi</LastName>
<FirstName>John</FirstName>
<LastName>Smi</LastName>
<FirstName>Joe</FirstName>
<LastName>Gunsmith</LastName>
<FirstName>Jane</FirstName>

Ends With Match Search Response Example

Request: <LastName SrchType="EndsWith">Smi</LastName>

Response: <LastName>Dritiesmi</LastName>
<FirstName>Lucas</FirstName>

Starts With Match Search Response Example

Request:
<LastName SrchType="StartsWith">Smi</LastName>

Response:
<LastName>Smith</LastName>
<FirstName>Mike</FirstName>
<LastName>Smitherrens</LastName>
<FirstName>Jason</FirstName>
<LastName>Smitty</LastName>
<FirstName>James</FirstName>

Misc Information

Authentication User Credentials

- Authentication of end user credentials in the form of a WS Security Element that contains a single SAML V2.0 assertion

  http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf

Custom Elements

The EA XSD contracts *wildcard* schema is provided by means of a *custom* complex. The custom complex (Custom_CType) provides a mean for institution specific data as an extension point for custom variations of the XSD common dictionary. The *##other namespace* property is leveraged as this allows all elements which are from namespace other than targetNamespace.

The provision for a consumer (message sender) to deliver custom elements to a service provider (message recipient) that is not part of the XSD contract. The W3C defines this component as Wildcard Schema.

Wildcard is the XML-Schema insider's term for an "xs:any" or an "xs:anyAttribute" declaration inside a schema. The idea is, wherever a wildcard appears in an XML Schema content model, "any" element or "any" attribute can be allowed to appear in an actual XML instance. This declaration permits zero or more elements with "any" name inside the "targetNamespace" to appear at the end of the content model.

This approach does come with some problems even though it provides an excellent extensibility mechanism for cooperating with stakeholders between future and past versions of the same schema. There are several reasons wildcards are not a good way to achieve compatibility between versions.

1. A targetNamespace wildcard permits too much freedom for current-version message producers to insert garbage that will be incompatible with future-version message consumers;

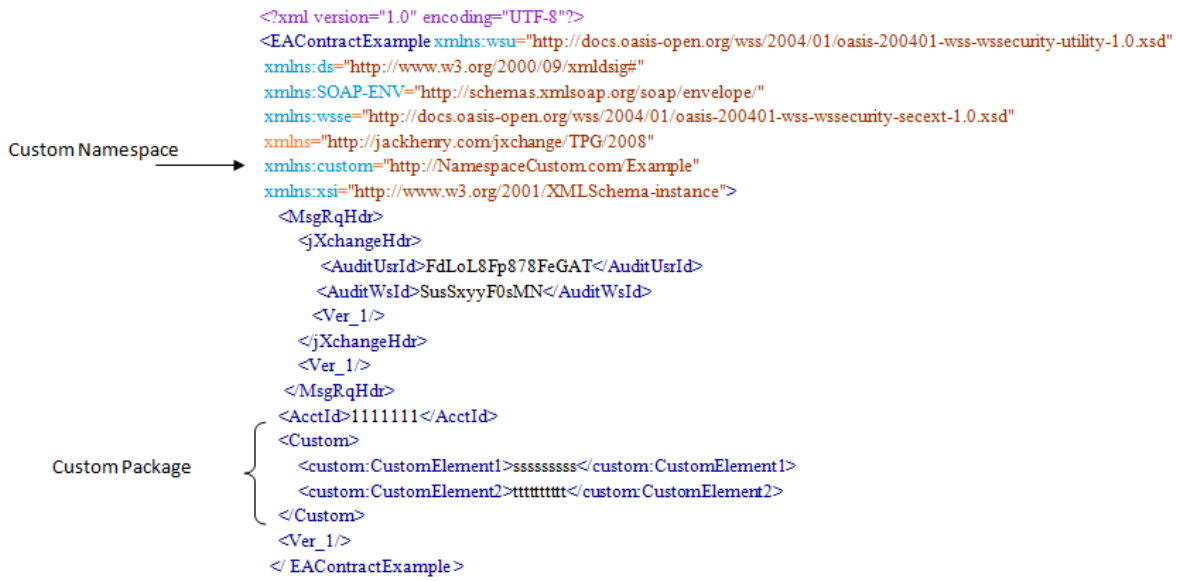
2. A wildcard in a specific place does not provide enough flexibility for natural data model evolution in the second version of the schema;

3. Technical limitations of wildcards - the element declarations consistent and unique particle attribution rules - prevent their use in "natural" ways and require awkward "wrapping" techniques;

4. And most seriously, wildcards put the onus on the designer of the original version of a schema to anticipate where and how evolution of the schema will occur. Experience shows that few people understand the future well enough to actually anticipate it.

The EA XSD contracts wildcard schema is provided by means of a custom complex. The custom complex (Custom_CType) provides a mean for institution specific data as an extension point for custom variations of the XSD common dictionary. The ##other namespace property is leveraged as this allows all elements which are from namespace other than targetNamespace.

XML Custom Example



Documented Choice Statements

- EA does not make usage of xsd:choice statements because:
  - Some programming languages do not provide an explicit mapping from xsd:choice whenever a code generator encounters xsd:choice in a type definition, it could map that type to a less than friendly API
  - XSD does not permit versioning within the choice statement which creates a challenge to maintain forward compatibility.

Choice Statement Example

```
<xs:complexType name="chadState">
 <xs:choice minOccurs="1" maxOccurs="1">
  <xs:element ref="selected"/>
  <xs:element ref="unselected"/>
  <xs:element ref="dimpled"/>
  <xs:element ref="perforated"/>
 </xs:choice>
 <xs:attribute name="candidate" type="candidateType"/>
</xs:complexType>
```

- EA makes use of documented choice statements.
    - The XSD syntax in itself does not convey a specific behavior but the comments provided via either XML comments or annotations does convey this behavior
    - The disadvantage to this method is that it requires a developer to read the XSD in order to understand behavior rather than allowing a code generator toolkit to understand based on XSD syntax

EA Document Choice Statement Example

```
<xsd:complexType name="AcctAddRq_MType">
  <xsd:sequence>
    <xsd:element name="MsgRqHdr" type="MsgRqHdr_CType"/>
    <xsd:element name="ErrOvrRdInfoArray" type="ErrOvrRdInfoArray_AType" minOccurs="0"
      nillable="true"/>
    <xsd:element name="AccountId" type="AccountId_CType"/>
    <!-- Only one of the following elements should occur - Like a choice statement -->
    <xsd:element name="DepAdd" type="DepAdd_CType" minOccurs="0" nillable="true"/>
    <xsd:element name="TimeDepAdd" type="TimeDepAdd_CType" minOccurs="0" nillable="true"/>
    <xsd:element name="LnAdd" type="LnAdd_CType" minOccurs="0" nillable="true"/>
    <!-- End "documented only" choice statement -->
   <xsd:element name="Custom" type="Custom_CType" minOccurs="0" nillable="true"/>
        <xsd:sequence minOccurs="0">
         <xsd:element name="Ver_2" type="Ver_2_CType"/>
         <xsd:any namespace="##targetNamespace" processContents="lax" minOccurs="0"
           maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:sequence>
</xsd:complexType>
```

Forced Elements

The vast majority of EA XSD message elements are optional because messages often are supported by multiple service providers. This can be the cause of some ambiguity on the part of the message receiver (consumer) when an element, understood by the service provider, but not returned in the message because the element is optional. The guidance for forced elements is as follows;

- "When a data value is zero a service provider's response will return the element whose primitive type is decimal as a value of "0".
    - For example, the element for current balance could appear as <CurBal>0</CurBal>.
    - This would explicitly convey to the consumer that the numeric element has an actual value rather than implicitly due to the element missing in the response.
- The following exceptions exist for this guidance:
    - The service provider has no knowledge of the element
    - The service provider is returning the element as part of an echo-back element from the request
    - The service provider value is known but undefined (Null)

The proceeding table provides guidance for all element types.

Reference Type	Empty	Missing	Null
Int	Zero	Zero	Null
Long	Zero	Zero	Null
String	Null	Null	Null
Date	Null	Null	Null
Time	Null	Null	Null
Base64Binary	Null	Null	Null
anyURI	Null	Null	Null
ID	Null	Null	Null
Decimal	Null	Null	Null

This table summaries that

(1) if a service provider has a decimal element it will be delivered;
(2) element integers (int & long) will only be used as numbers and are therefore safe to be assumed as zero when missing or empty; and
(3) all other element types have no special behavior.

A service provider might return a zero when the contract defines an element as a :string but the service providers data store reference field is stored as an integer. In this case, the zero might need to be conveyed to the consumer due to the some expected behavior.

JHANull Attribute

The mechanism “xsi:nil= true” was added to the WC3 XML Schema standard to explicitly assign a value to null to an element rather than using an empty element to implicitly assign a value to null. Because the behavior in the jXchange framework treats all messages as literal XML and passes that XML on to the provider it allows the provider to distinguish the difference between a missing element and one that is there but has “xsi:nil = true” .

The provider is using the explicit declaration to define a behavior where the element needs to be modified to a null value and when the element is missing to mean do not modify the existing element. This behavior is legitimate and uses the capability of the XML Schema specifications. The problem with this becomes apparent when a service is implemented and the message is de-serialized into objects under the covers. All the elements that are defined by the proxy or service for messages are created as objects and null or its actual value is assigned to them. There is not a way to represent that an element is not included in the XML but the value is not null in the object world using the present parsers

- An object is null whether it doesn’t exist or the element is assigned to “xsi:nil=true”
    - This becomes a problem because most implementation of web services are deserialized on input and serialized on output
    - Any other approach is discouraged as the complexity of implementation becomes more difficult
- Clients that are using .Net or probably any tool kit will not be able to take advantage of this implementation in the jXchange framework
    - They will not have control over an element creation during serialization using the existing tools.

EA Architectural Guidelines

The behavior expectation for XML elements in a modification service for (1) absence element, (2) empty element, or (3) xsi:nil=true are the same. This should convey to the service provider to do nothing. However; jhanull=true will convey to the service provider to set the element to a null value. The jhanull=true could require different implementation per platform. It could mean null for one platform whereas another it might convey *zeros or blanks.

The behavior expectation for XML elements in an addition service for (1) absence element, (2) empty element, or (3) xsi:nil=true are the same. This should convey to the service provider to set defaults. However; jhanull=true will convey to the service provider to set the element to a null value. The jhanull=true could require different implementation per platform. It could mean null for one platform whereas another it might convey *zeros or blanks.

XML JHANull Snippet Example

The Beneficiary distribution code is being conveyed to the service provider to set the stored element value
as null.

```
<PlnCust>
   <BenfDistCode JHANull="true"/>
   <OrigOwnCustId>QYQ6RjWyTmefKv4bM</OrigOwnCustId>
   <OrigOwnBirthDt>2012-01-01</OrigOwnBirthDt>
   <OrigOwnName>oMSgTu</OrigOwnName>
</PlnCust>
```

MemoPostInc

- Element included in an AcctHistSrch message that determines the behavior searching history search for memo posted items. The default value will be Excl (Exclude). Canonical Values are:
    - **Excl**
    - **Only**
- If the MemoPostInc element has the value of “Only”, XSLT is used to transform the AcctHistSrch request to a MemoPostSrch request and then is sent to the provider. XSLT is used to transform the provider’s MemoPostSrchResponse to an AcctHistSrchResponse.
- If the MemoPostInc element has the value of “Excl”, it is just passed through.

MaskVal Attribute

The service provider will always include the clear text of the element value however; may optionally include the service provider’s masked value of the clear text. This will allow the consumer to switch from a masked value to a clear text value with requiring an additional service call.
Example: `<TaxId MaskVal="882-89-7818">882897818</TaxId>`

Rstr Behavior

The attribute Restrictions <Rstr> references the element of the same name and is a closed enumerator with the following canonical values:

- **NoAccess** = Access is denied. This attribute value may exist at the parent node as well as the element node.
- **NoAccessPart** = Access is denied as a default for all of the nodes related to the parent node and any of the related nodes could override the accessibility setting at the parent node. This attribute value can only exist at a parent node.

- **ReadWrite** = Full read and write access to any of the nodes. This is the default value.
- **ReadWritePart** = Full read and write access for all the nodes related to the parent node and any of the related nodes could override the accessibility setting at the parent node. This attribute value can only exist at a parent node.
- **ReadOnly** = Read only to any of the nodes.
- **ReadOnlyPart** = Read only as a default for all of the nodes related to the parent node and any of the related nodes could override the accessibility setting at the parent node. This attribute value can only exist at a parent node.
- **Hid** = Hide all of the nodes.

The canonical values that are prefixed with Part (Partial) will convey to the consumer that overrides can exist for any of the proceeding nodes. Therefore, these canonical values can never exist at an element node.
The elements in the XML body are decorated with an attribute that conveys restrictions for an element. However, the data pertaining to the element is delivered when the service provider needs to convey to the consumer that the data element should be masked. For example, the current balance with restrictions could appear as <CurBal Rstr = "NoAccess">1000.00</Curbal>.

X_Filter Behavior

To limit the size of responses containing automatically generated response elements, jXchange is designed to automatically return priority preferred elements from any given message pair and retain the non-preferred and less used premium complex elements until they have been specifically requested

- Operations containing the premium x_prefix complexes in the response elements, request elements (IncXtendElemArray and XtendElem) are included in the Request message structure to call for any or all premium complexes. The request elements, if needed, must contain the name of one or more premium complexes

- A Service Provider may adopt the option to return all object and allow the Service Gateway to filter out the complexes not requested.