Proposed FireLight Commissions Process

General Requirements

1. Provide the ability for each FireLight implementation to OPTIONALLY calculate the commission amount on every order, by leveraging the carrier managed/loaded 1204 file and calculating the net vs gross commissions from the contracts defined premium amount.
2. Support the industry's current commission rate management process of XTbML commission rate management via the ACORD 1204 transaction. (samples provided)
3. Provide the ability for the implementers to load, test and promote 1204 upload(s) through FireLight regions.
4. ***OPTIONAL BUT HIGHLY DESIRED*** - Provide a web service based security and inbound messaging to support an implementer's automated push from the Vertex, VTXml tool that would push a 1204 from the VTXml tool to an implementers FireLight subscription eliminating their need to upload the 1204 file.
5. Provide an upload utility to enable our implementers to upload a 1204 transaction into their FireLight subscription. REQUIRED with or without web service push from VTXml tool.
6. Leverage the loaded and promoted 1204 transaction to calculate the netting amount for each order based on funding types.
7. Provide the ability for each implementation to enable/disable commission netting across their implementation with DISABLED being the default setting.
8. Provide carriers with eLabel's to be used for premium amount, commission options and funding types to enable multi-carrier commission netting. (field aliasing would be a nice to have to meet this requirement)

Details Regarding Carrier Commission Rate Management

Carrier Manages Commission Rate File

1. Via the FireLight admin portal or via web service with Vertx, a carrier will upload or push an ACORD 1204 XML file which defines the rate tables for their distribution channel(s). This file could be a 1..1 carrier to distribution channel or a 1..n carrier to distribution channels.
    a. There may be instances where the NY carrier, (or one like it) has a unique carrier code, so possibly a n..n carrier to distribution channel model…
2. There will be a unique <TXLifeRequest> aggregate for each Carrier/Producer sales relationship/combination even if there are multiple <TXLifeRequest> instances defining multiple relationships within the singular 1204 uploaded/transmitted.
3. Carrier will rely on DTCC app/sub files in test region to test/validate the FireLight netting functions are working properly.
4. The 1204 transaction defines the specific product(s) that are in scope via industry defined entity recognition of a combination of <ProductCode> and <CarrierCode> properties. We already have "Carrier Code" defined as a property for each product, it's highly unlikely that we can substitute CUSIP ID for <ProductCode> and we will need to add a new field within the admin tool for each product to have its own <ProductCode> property to align enabling us to match up.

5. Updates and ongoing rate file management will require management in the context of a distribution channel and ONLY a single file, (TXLifeRequest instance) per distribution channel/implementation will ever be allowed to be active in a given environment.
6. We will need to provide a process to manage multiple uploads/updates identifying the unique <TXLifeRequest> <party> instances via entity recognition from a single or multiple 1204 uploads. Communicated a different way, first load may be for 2 distribution channels in a single 1204, next upload may be for only a single distribution channel previously loaded and we will need to know which to update based on the entity recognition within the XML itself.
    a. Entity Recognition for a carrier is //Party/Carrier/CarrierCode
    b. Entity Recognition for a distributor is //Party/Organization/OrgCode (this may require a new property)
    c. It will require the combination to define unique instances.

Requirements for interpreting the 1204

1. Once the combination of Carrier/Distribution channel is defined, we will follow the //Party/Producer/CarrierAppointment/DistributionAgreementInfo aggregates reference to the corresponding //DistributionAgreement aggregate.
    a. Inside this //DistributionAgreement aggregate, the //PolicyProductInfo aggregate repeats referencing each product as part of that distribution agreement.
    b. Then within each //PolicyProductInfo aggregate, in the context of that specific Product, the //CommSchedule is referenced by //PolicyProductInfo/CommScheduleCode as the entitiy recognition to the correct commission schedule to be used in this calculation.
    c. Then within the //CommSchedule aggregate there is a //CommFormula aggregate which repeats for each commissionable event (a.k.a. funding type)
    d. Now that we know the Product and the Commission Schedule and the commissionable event (a.k.a. funding type), we can now find the rate table via the //CommSchedule/CommFormula/TableRef/TableIdentity and ProviderDomain (with the two properties of TableIdentity and ProviderDomain being the entity recognition for a given rate table.
    e. Now we find the correct rate table via //XTbML/ContentClassification/TableIdentity and ProviderDomain, we then identitfy which order elements are to be considered in this calculation via the //XTbML/Table/MetaData/KeyDef and AxisDef combintations
    f. Rate tables can vary by Jurisdiction as defined in the //XTbML/Table/MetaData/JurisdictionCC

Permissible AxisDef and KeyDef order data references within the XTbML (what can commission rates vary by?)

1. KeyDef – A KeyDef is a NON NUMERIC reference to new business data (specifically the way it's formatted in the 103 transaction) that is NON NUMERIC.
2. AxisDef – An AxisDef is a NUMERIC reference to new business data (specifically the way it's formatted in the 103 transaction) that is NUMERIC
    a. KeyDef - , A //KeyDef is made unique by //KeyDef/KeyType and //KeyDef/KeySubType, there are two different flavors of KeyType, defined via KeyType=1 (code) OR KeyType=2 (string)

i. The differentiator between these CODE and STRING is a CODE will reference a defined set of enumeration lists, the permissible list of enumeration lists and a STRING will reference an objects property.

b. Within the KeyDef aggregate the properties below are leveraged to communicate;
   i. KeyType – String or code (see above)
   ii. KeySubType – ONLY applicable when KeyType "2-string", can only be 20-Interest Rate Class OR 21-ProductCode/RiderCode(for rider/arrangement selections)
   iii. KeySubClassType – Referrences to the object with in the ACORD model that contains the string value property which impacts rates will reside.
      1. In scope for our implementation, we would need to support;
         a. 32-Sub Account (fund)
         b. 86-Ann Rider
         c. 89-Arrangement
         d. 26-Payout
         e. Enumerations are referenced via the KeyCodeType property further down.
   iv. KeyName – Simply a text name not for our use
   v. Dimension Sequence – When there are multiple KeyDef instances, this defines the order of the dimensions.
   vi. EnumeratedTypeCodeValue – Applicable when KeyType=1, KeySubClassType is used, this property is used to communicate the actual TC value for the specific enumeration referenced in the KeySubClassType property
   vii. EnumeratedStringValue – Applicable when KeyType=2, this property is used to communicate the actual text value of the object referenced in the KeySubClassType reference. We will need to support these KeySubClassType instances;
      1. OLI_LU_RIDERTYPE
      2. OLI_LU_ARRTYPE
      3. OLI_LU_ANNPAYOUT (used less frequently, typically for immediate annuities ONLY)
   viii. DimensionSequence – Used to order the sequence of interrogation across all KeyDef and AxisDef instances. Defines order.
   ix. KeyCodeType – ONLY applicable when KeyType=1, Will contain an TC integer reference to the enumeration list being used from the OLI_LU_LOOKUP table, which is the lookup of lookups within the ACORD model.

c. AxisDef – Numeric by default, these are defined by //AxisDef/ScaleType and //AxisDef/ScaleSubType