Real Estate Data Interchange Standard:
Real Estate Transaction Specification
Version 2.0

RETS2 Payloads

July 31, 2006
Draft 3

Table of Contents

1 RETS2 Resources and Payloads

1.1 XML Document Development Process

1.1.1 Schema Development Process

This section discusses the recommended process for the development of schemas.

1.1.1.1 Development Process Principles

Successful schema development depends upon:.

- Active participation of the stakeholders within the community.
- Collaboration between key business people and XML schema experts.
- Use cases to align schemas with business goals.
- A software implementation that validates the XML schema and demonstrates the business benefits.

1.1.1.2 Development Process Overview

Schema development projects should follow these steps:

1. Key business people identify documents that are heavily reused. Engineering resources identify relevant existing industry standards to drawn upon.
2. Key business people identify use cases that demonstrate business value. Existing roles, documents and processes are also identified. The results are presented in an information flow diagram.
3. Key business people identify the data items on the document(s) that support the use cases and, at the same time, define the minimum required data to support each use case/transaction.
4. Key Business people develop a vocabulary for the data items in the documents in the use cases. Schema developers collaborate to designate data type definitions and roles for vocabulary items. This vocabulary should be presented in a spreadsheet.
5. Key Business people collaborate with Schema experts to arrange the vocabulary data into a dictionary with logical groupings. This task includes designating which groupings are common to multiple documents and across multiple use cases.
6. Schema experts develop the XML schema definition from these dictionaries. The result is an object-oriented XML Schema that may be readily implemented in software data mappings. Key business people and software developers must review the XML schema to ensure that it supports the use cases and can be used in software implementations.
7. Software developers validate the proposed XML schema by building example implementations using the XML schema. The schema may be revised as needed to address any software implementation issues that arise.

8. The XML schema and associated documentation is published as an initial version and made available to the community as a standard schema.
9. The standard schemas may be reviewed and revised publishing new versions over time, as approved by the community following the schema development process outlined here.

1.2 XML Schema Guidelines

1.2.1 Introduction

This document suggests best practices and guidelines that should be followed by RETS XML Schema designers when defining XML-based specifications.

1.2.1.1 Background

The RETS2 Data Interchange Standard uses XML and Web Services as the basis for this specification to promote interoperability between data providers and consumers. Unfortunately, the W3C specification for XML Schema is large and complex leaving it nearly impossible for anyone to completely understand its breadth and depth. Furthermore, the W3C offers no guidance with respect to best practices or guidelines for implementing XML Schemas within the enterprise.

This document lays out the best practices and guidelines that should be followed by RETS2 implementers to ensure consistency with respect to XML schema documents across the industry. These schemas are to describe the way that data will be exchanged between different implementers of the RETS2 standard. This document addresses many of the common features and issues pertaining to XML Schema; obscure aspects of XML Schema will not be addressed.

As the W3C XML Schema specification continues to evolve and mature, this document may be modified to keep pace with industry standard best practices and guidelines.

1.2.1.2 Resources

This document draws upon several Web-based and non-web resources that offer opinions and contributions to the data exchange (XML) and software industry's general understanding of XML Schema document Best Practices and guidelines. These resources will be referenced liberally throughout this draft version of the document. Where there has been any significant disagreement about the right approach to that particular area of using a Schema, this document will attempt to present both contrasting opinions and allow the community member to decide which approach to take.

The following resources were used in the writing of this document.

www.medbiq.org	MedBiquitous XML Schema Design Guidelines document from the MedBiquitous consortioum. This document was used as an example of another industry's standards-based organization that has an implementation of an XML Schema Guidelines document.
www.w3.org/TR/xmlschema-2	W3C specification and Guidance document for Schema, Schema Part 2: Datatypes.
www.xfront.com	Roger Costello maintains this site.
www.xml.com	This site is part of the O'Reilly Network and contains an annotated version of the XML specification, created by Tim Bray.
xml.sys-con.com	XML Journal is a site published by SYS-CON Media Articles: • http://www.xml.com/pub/a/2001/08/22/easyschema.html - understanding Complex Types in XML Schema • http://xml.sys-con.com/read/40481.htm - XML Schema Best Practices • http://www.xml.com/pub/a/2001/06/06/schemasimple.html - XML Schema Made Simple
www.govtalk.gov.uk	The United Kingdom's GovTalk site has another example XML Schema guideline document that includes "mandatory requirements for XML Schema structure and content, as well as best practice recommendations for schema design"

1.2.2 Overall Document Guidance

1.2.2.1 Keep Schemas Simple

The less common facilities available with XML Schema SHOULD NOT be used where there are simpler alternatives. Schema developers SHOULD take into account the testability of their schemas.

This is perhaps the most important rule. XML Schema allows enormous power and flexibility in the way schemas are defined. In most cases, schemas can be made simple or complex while achieving the same aim. Many people who will be looking at your schemas will have little experience, **so try to keep them simple**.

1.2.3 Namespaces

The following defines the guidelines for handling namespaces in XML schema definition files.

1.2.3.1 Target Namespaces

The XML Schema definition file should define a target namespace. The namespace should be defined as a URL that uniquely qualifies this schema and its definitions.

Avoid creating XML Schema definition files with no target namespace ("chameleon"). Although chameleon schemas offer flexibility, validation performance is degraded since most parsers will not be able to cache components of the schema based on the namespace.

1.2.3.2 Default Namespaces

Any default namespace for the document MUST be the same as the target namespace. There is never a disadvantage of making the default namespace of a schema document the same as the target namespace. All other namespaces will require a prefix. This makes the usage of namespaces more explicit, and allows schema designers more flexibility in using namespaces within the schema.

XML Schema should never be the default namespace.

1.2.3.3 XML schema Namespace

The W3C XML Schema namespace MUST be qualified with a prefix with a prefix of either `xsd` or `xs`. A suitable qualifier MUST be used for other namespaces.

Example:

```
xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

1.2.3.4 Versioning

In order to promote a uniform approach to versioning schemas, the default and target namespaces defined in the XML Schema definition file must include a version identification value. The value is the year followed by the month, optionally followed by the day. Whenever the schema is changed, and the new schema is NOT backward compatible with the previous schema, the default and target namespaces versioning must be incremented.

Each XML Schema payload also includes a versionTimestamp attribute. This value, in ISO 8601 format, is UTC. It must change each time the Schema is modified.

Example

```
<xsd:schema
targetNamespace="http://www.rets.org/ns/Listing/200604"
xmlns=" http://www.rets.org/ns/Listing/200604"
xmlns:xs=http://www.w3.org/2001/XMLSchema
```

```
versionTimestamp="2006-04-07T00:00:00Z">
```

1.2.4 Global Vs. Localized (Qualified Vs. Unqualified) Elements

Two attributes of your schema file that should be specified are `elementFormDefault` and `attributeFormDefault`.

1.2.4.1 elementFormDefault and attributeFormDefault

elementFormDefault MUST be set to qualified and attributeFormDefault SHOULD beset to unqualified. This is the industry best practice and RETS2 implementers should follow this practice, as well.

The setting of these schema attributes with these values implies that all schema instance files must qualify the namespace of all elements regardless of whether the element is defined in a global or local namespace. This ensures that a developer reading or reusing a schema can rely on the visible prefixes and namespaces, instead of having to trace the detailed internal structure of a schema.

1.2.4.2 More Information

For a more detailed explanation of the difference between global (`elementFormDefault="qualified"`) versus localized (`elementFormDefault="unqualified"`) namespaces within a schema, see the following resources:

X-Front PDF: Hide Vs. Expose Namespaces

MedBiquitous PDF - See Section 5

The X-Front document provides a thorough explanation of of this issue at hand, but does not make any concrete recommendations. The MedBiquitous PDF also provides a detailed explanation of WHY it is best to use the approach set forth in section 4.1.

1.2.5 Multiple Schema Documents

Give each separate schema document a separate targetNamespace, except, perhaps only when you benefit by breaking down a very large vocabulary into multiple physical schema documents.

1.2.6 Naming Conventions

1.2.6.1 General Guidelines for Naming

A very important part of the XML grammar within your schema documents is consistent naming conventions for tags that represent the infrastructure and business related

elements. Abbreviations SHOULD NOT be used, except for the case of well known abbreviations. Well known abbreviations, including the use of initial letters only, MAY be used. However, be sure that any abbreviation used is well-known across the RETS developer community. Extremely long names SHOULD be avoided by designing concise and informative names.

1.2.6.2 Naming of Elements and Attributes

Naming conventions are based on the XML tagging from the ebXML group (ebXML Naming Conventions PDF) and the MedBiquitous schema guidelines document (see section 1.2 – Resources).

Tag name writers MUST follow these following rules **unless business requirements require other naming conventions**.

159 Naming Conventions

Rule	Description	Example
Element and Type Case	Elements and types should be defined using upper camel case (UCC).	`<PostalCode>`
Attribute Case	Attributes should be defined using lower camel case (LCC).	`<Degree discipline=”Chemistry”>`
Acronyms	Acronyms are discouraged, but where needed, use all upper case.	`<UserID>`

Illegal Characters	Illegal characters cannot be used (e.g.: forward slash, etc.). Recommended characters in a tag name are basically limited to letters and underscores.	NOT allowed:
		`<Date/Time>`
		Allowed:
		`<DateTime>`

Similar Names	Use the similar tag names with elements in a similar child structure.	`<ContactAddress>`
		`<HomeAddress>`
		`<WorkAddress>`

Plural Names	Use plural tag names only for collections.	`<CreditCards>` `<CreditCard>`
Name Size	Element and attribute name size have no limitation. The names must be meaningful. Very long names are discouraged. Design concise, descriptive names.	`<CustomerRelationshipInformation>`
Suffixes	Element and attribute names should incorporate suffixes from the proposed list of representation types (adapted from ebXML) when appropriate.	`<StartDate>` `<BilledAmount>`

160

161 Tag Suffixes Table

Representation Type	Description
Amount	A number of monetary units specified in a currency where the unit of currency is explicit or it may be implied.
Code	A character string that represents a member of a set of values.
Boolean (Flag)	An enumerated list of two, and only two, values which indicates a Condition such as on/off; true/false etc. (It was the general consensus to use 'Flag' as a term to indicate a Boolean value.)
Date	A day within a particular calendar year. Note: Reference ISO 8601.
Time	The time within any day in public use locally, independent of a particular day. Reference ISO 8601:1988.
DateTime	A particular point in the progression of time. Note: This may incorporate dependent on the level of precision, the concept of date.
Identifier	(standard abbreviation ID, meaning a unique identifier) A character string used to identify and distinguish uniquely, one instance of an object within an identification scheme.

162

163 1.2.7 Element Vs. Type

164 Generally, Named types should be used instead of anonymous types.

165 In many cases, there is a choice of defining a re-usable component as either a data type or as an element. A
166 component MUST be defined as a data type if either: it is to be used with different element names in different
167 contexts; or it is expected that further data types will be derived from it.

168 A component SHOULD be defined as an element if there is no intention to derive new components from it; and
169 the element is to be used with its name unchanged

170 There are many circumstances in which an element should be used with its name unchanged. It is therefore
171 possible to build a dictionary of element names with known interoperable semantics. For example, an address
172 could have several meanings and so be used with different names. An address should therefore be defined as a
173 global data type

174 The other reason to choose between an element and a data type to define a component is if there is an intention to
175 derive other components from it. By only using data types in this case, we simplify understanding of schemas by
176 only having a single inheritance mechanism and avoiding use of `xs:redefine` for this purpose.

177 1.2.8 Element Vs. Attribute

178 Schemas SHOULD be designed so that elements are the main holders of information
179 content in the XML instances. Attributes are more suited to holding ancillary metadata –
180 simple items providing more information about the element content. Attributes MUST
181 NOT be used to qualify other attributes where this could cause ambiguity.

Unlike elements, attributes cannot hold structured data. For this reason, elements are preferred as the principal holders of information content. However, allowing the use of attributes to hold metadata about an element's content (for example, the format of a date, a unit of measure or the identification of a value set) can make an instance document simpler and easier to understand.

1.2.9 Creating Extensible Content Models

If an existing type definition does not meet your exact requirements, you MAY use the XML Schema inheritance mechanism to define a new data type based largely on an existing one.

In some cases a data type enumerates all permitted values, or defines a standardized data format such as an address whose importance for interoperability goes beyond XML messages. In this instance, make sure the modified definition still complies with the underlying data standard.

1.2.9.1 Extensibility Using Type Inheritance in XML Schema

There are four types of inheritance available using `extension` and `restriction`. These are:

restriction of a simple data type

extension of a simple data type (to form a complex type)

restriction of a complex data type

extension of a complex data type

The guidelines about which type of inheritance can be used are detailed below in sections 8.2 and 8.3.

1.2.9.2 Simple Types

When appropriate, simple data types defined in the XML Schema data model should be used (and potentially restricted or extended) rather than creating a user defined complex data type. Restriction of a simple type reduces the possible values of the type while extension allows one to create a complex type with simple content that has attributes. Schema designers will want to keep both of these forms of content extensibility in mind as they create their RETS schemas.

1.2.9.2.1 *Example - this example derives a new Simple Type Declaration:*

<simpleType name="myRETSNameType"><restriction base="string">

<enumeration value="Paula O'Brien" /></restriction></simpleType>

Associating myRETSNameType with an element, and then I can use the type in an XML document. So the element declaration:

<element name="RETSMeetingAttendee" type="my:myRETSNameType"/>

allows me to use:

<my:RETSMeetingAttendee>Paula O'Brien</my:RETSMeetingAttendee>

The designer should also keep in mind that the correct simple type defined in the XML Schema data model should be used.

Example – Simple Data Types:

If a date value is needed, use <xsd:element name="Date" type="xsd:date"/> instead of <xsd:element name="Date" type="xsd:string"/>

Please refer to the section in the W3C Schema DataTypes document (http://www.w3.org/TR/xmlschema-2/#built-in-datatypes) that lists the hierarchy of Built-in datatypes (Section 3) for more information about the type hierarchy for more guidance on which built-in types a Simple Type should derive from or may derive from.

1.2.9.3 Complex Types

Complex types should be extended but not restricted. Extension involves adding extra attributes or elements to a derived type. Derivation by restriction of complex types should be avoided.

Schema designers should use caution adding complex types by way of extension, since they may derive new complex types that can extend any simple or complex type from within the current schema or any imported schema. This feature, while powerful may add additional complexity that makes it difficult for creators of instance documents to understand the nature of the inheritance. However, this feature is an important tool for schema designers in creating an extensible content model.

Example:

Base Type

```
<xsd:complexType name=”BaseAddress”>
      <xsd:sequence>
            <xsd:element name=”State” type=”xsd:string”/>
```

```
      </xsd:sequence>
</xsd:complexType>
```

Derived Type

```
<xsd:complexType name="RETSAddress">
      <xsd:extension base="BaseAddress">
            <xsd:sequence>
                  <xsd:element name="City" type="xsd:string"/>
            </xsd:sequence>
      </xsd:extension>
</xsd:complexType>
```

Note that we can also extend the content model of one schema by using a datatype from another, imported schema.

In another ***example***, If the following text is the content of a schema named **RETSDataTypes.xsd**:

```
xmlns="http://www.RETS.org/RETS2/RETSDataTypes"
<xsd:complexType name="RETSAddress">
      <xsd:sequence>
            <xsd:element name="StreetNamePrefix"
type="xsd:string"/>
            <xsd:element name="StreetNumber"
type="xsd:string"/>
            <xsd:element name="StreetName"
type="xsd:string"/>
            <xsd:element name="StreetSuffix"
type="xsd:string"/>
            <xsd:element name="AppartmentNumber"
type="xsd:string"/>
```

```
            <xsd:element name="City" type="xsd:string"/>
            <xsd:element name="State" type="xsd:string"/>
            <xsd:element name="PostalCode"
type="xsd:string"/>
      </xsd:sequence>
</xsd:complexType>
```

Next, this complex type `RETSAddress` is extended in a new schema document MyRETSTypeDefinitions.xsd:

```
xmlns=" http://www.RETS.org/rets2/RETSDataTypes"
<xsd:include schemaLocation="RETSDataTypes.xsd"/>
<xsd:complexType name="RETSInternationalAddress">
      <xsd:complexContent>
            <xsd:extension base="RETSAddress" >
                  <xsd:sequence>
                        <xsd:element name="Country"
type="xsd:string"/>
                        <xsd:element name="CountryCode"
type="xsd:string"/>
                  </xsd:sequence>
            </xsd:extension>
      </xsd:complexContent>
</xsd:complexType>
```

An instance document will look like the following:

```
xmlns="http://www.RETS.org/rets2/RETSDataTypes"
xsi:schemaLocation="http://www.RETS.org/rets2/RETSDataTypes
/MyRETSTypeDefinitions.xsd
```

```
<Address xsi:type="RETSInternationalAddress">
      <StreetNamePrefix></StreetNamePrefix>
      <StreetNumber>2665</StreetNumber>
      <StreetName>North Moreland Blvd.</StreetName>
      <StreetSuffix></StreetSuffix>
      <City>Cleveland</City>
      <State>OH</State>
      <PostalCode>44240</PostalCode>
      <Country>United States</Country>
      <CountryCode>US</CountryCode>
</Address>
```

Note that a possible drawback to the extensibility done here is that it is location restricted. The extensibility is restricted to appending elements onto the end of the content model (in this case, after the <PostalCode> element). What if we wanted to extend <Address> by adding elements to the beginning (before <StreetNamePrefix>), or in the middle, etc? We can't do it with this mechanism. This is where the <Any> element comes in.

1.2.9.4 Extensibility via the <any> Element

An <any> element may be inserted into a content model to enable instance documents to contain additional elements.

An **Example** showing an <any> element:

```
xmlns="http://www.RETS.org/rets2/RETSDataTypes"
<xsd:complexType name="RETSAddress">
      <xsd:sequence>
            <xsd:element name="StreetNumber"
type="xsd:string"/>
```

```
            <xsd:element name="StreetName"
type="xsd:string"/>

            <xsd:element name="StreetSuffix"
type="xsd:string"/>

            <xsd:element name="City" type="xsd:string"/>

            <xsd:element name="State" type="xsd:string"/>

            <xsd:element name="PostalCode"
type="xsd:string"/>

            <xsd:any namespace="##any" minOccurs="0"
maxOccurs="unbounded"/>

      </xsd:sequence>

</xsd:complexType>
```

This says "The content of RETSAddress is StreetNumber, StreetName, StreetSuffix, City, State, PostalCode and then (optionally) any well-formed element. The new element may come from any namespace."

Note the `<any>` element may be inserted at any point, e.g., it could be inserted at the top, in the middle, etc. In this version of the schema it has been explicitly specified that after the `<PostalCode>` element any well-formed XML element may occur and that XML element may come from any namespace.

1.2.9.5 General extensibility Guidelines – Avoiding Non-Determinism

RETS Schema designers may choose whether to include a single point of extensibility or to allow extensibility at multiple points in the schema. Where to put extensibility points is highly dependent on the domain being modeled by the schema. A few simple guidelines should be followed when making these decisions:

Schemas or sections of schemas that define well-established or fundamental content need not contain extensibility points.

Schemas or sections of schemas that attempt to codify a new or dynamic area of content should use extensibility points throughout.

Only elements from a namespace different from the document namespace should be allowed in the extension. This restriction is specified in XML Schema as:

```
<xsd:any namespace="##other"/>
```

A namespace constraint set to ##other avoids content collision and non-deterministic content models.

1.2.10 Schema Design Patterns – Styles of Schema Design

As with software design, there are design patterns associated with XML Schema design. The most popular XML Schema design patterns are Russian Doll, Salami, Bologna, Venetian Blind, and Garden of Eden.

To understand any of these design patterns, it is necessary to differentiate between a global component (element or type) and a local component (element or type). A global component is an immediate child of the `<schema>` element in the XML Schema definition file. A local component is not an immediate child of the `<schema>` element in the XML Schema definition file. Global components are associated with the target namespace of the schema and may be reused in other schema.

It is also important to understand that any element defined in the global namespace can be the root for a valid XML instance document adhering to the schema defined for that namespace.

1.2.10.1 Recommended Design Pattern – Venetian Blind pattern

The recommendation for a design pattern choice is the ***Venetian Blind*** pattern. The Venetian Blind design corresponds to having a single global element that nests local elements (that nest further local elements). Only one element, considered the root, is defined within the global namespace. However, the local elements use types (simple or complex) that are defined within the global namespace.

The benefits of a reusable type definitions coupled with a single 'root' element in the global namespace provide a schema designer with the control and reusability necessary in the messaging interface definitions. But, there may be exceptions to this rule. It is up to the schema designer to determine if she her situation is viable for using another design strategy. Please note that for the sake of brevity our example ONLY defines Simple Types.

Example:

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

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"

     elementFormDefault="qualified"
attributeFormDefault="unqualified">

     <xs:element name="LotSize">
```

```
386            <xs:complexType>
387                 <xs:sequence>
388   <xs:element name="Area" type="AreaType"
389   maxOccurs="unbounded"/>
390                       <xs:element name="Dimensions"
391   type="DimensionsType"/>
392                       <xs:element name="Length"
393   type="LengthType"/>
394                       <xs:element name="Width"
395   type="WidthType"/>
396                 </xs:sequence>
397            </xs:complexType>
398       </xs:element>
399       <xs:simpleType name="AreaType">
400            <xs:restriction base="xs:string">
401                 <xs:minLength value="1"/>
402                 <xs:maxLength value="35"/>
403            </xs:restriction>
404       </xs:simpleType>
405       <xs:simpleType name="DimensionsType">
406            <xs:restriction base="xs:string">
407                 <xs:minLength value="1"/>
408                 <xs:maxLength value="35"/>
409            </xs:restriction>
410       </xs:simpleType>
411       <xs:simpleType name="LengthType">
```

```
            <xs:restriction base="xs:string">
                <xs:minLength value="1"/>
                <xs:maxLength value="35"/>
            </xs:restriction>
        </xs:simpleType>
        <xs:simpleType name="WidthType">
            <xs:restriction base="xs:string">
                <xs:minLength value="1"/>
                <xs:maxLength value="35"/>
            </xs:restriction>
        </xs:simpleType>
</xs:schema>
```

Please see the following resources for a more thorough comparative discussion of the other four types of design patterns:

MedBiquitous XML Schema Design Guidelines PDF – Section 4

UK GovTalk Schema Guidelines Document PDF – Appendix A

1.3 RETS2 Common Schemas

RETS2 Common schemas are reusable building blocks used to build the standard RETS2 schemas and are also expected to be used by the localized payload.

1.4 RETS2 Payloads

1.4.1 Standard Payloads

1.4.1.1.1 *Metadata Payloads*

The RETS2 Metadata supports a Separation of Concerns, such as:

- Basic Data and Query
- User Interface
- Validation (future)
- Security (future)

Current RETS2 WellKnown Metadata payloads are: ResourceList, LookupList, Vocabulary, DataDictionary, and UserInformation. Additional details about these documents and examples of the role they play in the RETS2 Service can be found within the Discovery section of the RETS2 Service document.

Document	Schema Location
Resource List The ResourceList Metadata provides a Requestor with basic information about all of the Resources supported by a RETS2 Provider.	http://retsserver.realtors.org:8080/xsd/ResourceList.xsd
Vocabulary Vocabulary Metadata supplies all of the necessary information for a RETS Requestor to construct a Query for a RETS Search action.	http://retsserver.realtors.org:8080/xsd/Vocabulary.xsd
DataDictionary The DataDictionary Metadata contains detailed information for all fields (searchable and non-searchable) within a Resource.	http://retsserver.realtors.org:8080/xsd/DataDictionary.xsd
LookupList The LookupList Metadata contains Maps that can be used across Resources to define Lookup types.	http://retsserver.realtors.org:8080/xsd/LookupList.xsd
UserInformation The UserInformation Metadata document contains system account information about a user's system account for a given user.	http://retsserver.realtors.org:8080/xsd/UserInformation.xsd

1.4.1.2 MLS Payloads

Document	Schema Location
Activity The Activity schema provides a general description for any activity, with dates and roles and a categorization. This schema is used both as a MLS and as a TMS payload.	http://retsserver.realtors.org:8080/xsd/Activity.xsd
Agency Agency is a schema that contains information describing an agent	http://retsserver.realtors.org:8080/xsd/Agency.xsd

involved in a real estate transaction. Its complexTypes for both Agent and Agency are used in other RETS payloads.	
Listing Listing is a schema that combines types from RETSCommons into a Listing payload (it contains elements that relate to the listing and sale of the property). This Listing complexType is also used in other RETS payloads.	http://retsserver.realtors.org:8080/xsd/Listing.xsd
ListingHistory ListingHistory schema describes changes and change types, by date, to ListingProperty records.	http://retsserver.realtors.org:8080/xsd/ListingHistory.xsd
ListingProperty ListingProperty is a schema that uses the Listing and Property complexTypes to create a payload that contains both Listing and Property Information for a real property.	http://retsserver.realtors.org:8080/xsd/ListingProperty.xsd
MessageOfDay MessageOfDay is a schema that contains text and timestamps for a Provider's broadcast system messages.	http://retsserver.realtors.org:8080/xsd/MessageOfDay.xsd
ObjectReferenceList The ObjectReferenceList schema provides references between Resources and Object Resources, and describes object types for a given resource, including size, caption, URL, and id.	http://retsserver.realtors.org:8080/xsd/ObjectReferenceList.xsd
OfficeRoster The OfficeRoster schema describes a list of Offices for a Brokerage or Franchise and the Agents that are assigned to those Offices.	http://retsserver.realtors.org:8080/xsd/OfficeRoster.xsd
Offices Offices is a schema that contains information describing a broker/sales office. Its	http://retsserver.realtors.org:8080/xsd/Offices.xsd

complexTypes for both Office and Offices are used in other RETS payloads.	
Property Property is schema that combines types from RETSCommons to create a payload describing a property (it contains elements that relate to the physical property and are not subject to change when listed). The property complexType is also used in other RETS payloads. The PropertyType and PropertySubType elements may be used to define a property record as Residential Single Family, MultiFamily, CommonInterest, etc.	http://retsserver.realtors.org:8080/xsd/Property.xsd
Prospect The Prospect schema describes queries created for prospecting and saved by an Agent. It contains details about the Agent and Office that generated the query.	http://retsserver.realtors.org:8080/xsd/Prospect.xsd
PublicRecord The PublicRecord schema describes a property's parcel, tax, zoning, and valuation information.	http://retsserver.realtors.org:8080/xsd/PublicRecord.xsd
RETSCommons RETSCommons is an abstract schema with simpleTypes, complexTypes and attributeGroups that are used to compose the other RETS payloads. It is based on data from the RETS 1.7 REData DTD. This "library" is used by MLS Payloads, TMS Payloads, and NRDS Payloads.	http://retsserver.realtors.org:8080/xsd/RETSCommons.xsd
SystemInformation SystemInformation provides information about a Provider	http://retsserver.realtors.org:8080/xsd/SystemInformation.xsd

implementation's parameters.	

445
446

447 ### 1.4.1.3 Transaction Payloads

Document	Schema Location
Activity The Activity schema provides a general description for any activity, with dates and roles and a categorization. This schema is used both as a MLS and as a TMS payload.	http://retsserver.realtors.org:8080/xsd/Activity.xsd
Contact Contact is a schema that describes basic contact management: the contact's information, date created and updated, and a history of messages.	http://retsserver.realtors.org:8080/xsd/Contact.xsd
Documents Documents is a schema that describes individual transaction documents and provides information about their location, key dates, senders, receivers, signers and originators.	http://retsserver.realtors.org:8080/xsd/Documents.xsd
ical ical is a schema used by Outlook and other scheduling/calendar programs to describing schedules and events.	http://retsserver.realtors.org:8080/xsd/ical.xsd
Offer Offer is a schema that describes the real estate transaction offer and counter offer process between sellers and potential buyers.	http://retsserver.realtors.org:8080/xsd/Offer.xsd
Participants Participants is a schema that describes the persons or entities participating in the transaction.	http://retsserver.realtors.org:8080/xsd/Participants.xsd
Referral Referral is a schema that describes an agent's or broker's referrals, including source, client, rate, dates and notes.	http://retsserver.realtors.org:8080/xsd/Referral.xsd
ServiceOrder ServiceOrder is a schema that describes one or more transaction service orders, including: key dates,	http://retsserver.realtors.org:8080/xsd/ServiceOrder.xsd

service providers, vendors, notes, and description.	
Transaction Transaction is a schema that describes a real estate transaction in detail, including service orders, documents, listing, property, contacts, financing, activities and participants.	http://retsserver.realtors.org:8080/xsd/Transaction.xsd
TransactionList TransactionList is a list of transaction ids, names and key dates.	http://retsserver.realtors.org:8080/xsd/TransactionList.xsd

448

449 ### 1.4.1.4 NRDs Payloads

450

Association Association schema provides information about an MLS, Local or State Association.	http://retsserver.realtors.org:8080/xsd/Association.xsd
Course Course schema provides information about an specific real estate license courses, such as test results, amounts, and sponsoring association.	http://retsserver.realtors.org:8080/xsd/Course.xsd
MemberFinancial MemberFinancial schema provides information membership and dues.	http://retsserver.realtors.org:8080/xsd/MemberFinancial.xsd
MemberRecord Provides detailed information about a member, such as status, reinstatement, preferences, and association information.	http://retsserver.realtors.org:8080/xsd/MemberRecord.xsd
MemberTransmittal MemberTransmittal schema describes a member record update.	http://retsserver.realtors.org:8080/xsd/MemberTransmittal.xsd
NRDSCommons NRDSCommons is an abstract schema that contains as a "library" of complexTypes describing common NRDS data. It is used by all other NRDS payloads.	http://retsserver.realtors.org:8080/xsd/NRDSCommons.xsd
OfficeTransmittal	http://retsserver.realtors.org:8080/xsd/OfficeTransmittal.xsd

OfficeTransmittal schema describes an office record update.	

451

452 ### 1.4.2 Local Payloads

453 In addition to well-known, or standard RETS MLS, RETS TMS and RETS NRDS
454 payloads, RETS2 supports user-defined local payloads. Local payloads are existing
455 standard schemas that have been extended, schemas developed from scratch or even
456 binary content (CSV, PDF, multi-media, etc.). These additional payload formats MUST
457 be defined in the OutputFormat section for a Resource in the ResourceList.
458