SunGuide^{TM}:

General Interface Control Document

SunGuide-General-ICD-6.2

Prepared for:

Florida Department of Transportation
Traffic Engineering and Operations Office
605 Suwannee Street, M.S. 90
Tallahassee, Florida 32399-0450
(850) 410-5600

March 23, 2016

Document Control Panel			
File Name:	SunGuide-General-ICD-6.2.doc		
File Location:	SunGuide CM Repository		
CDRL:	6-1		
	Name	**Initial**	**Date**
Created By:	Lynne Randolph, SwRI	LAR	11/15/04
Reviewed By:	Steve Dellenback, SwRI	SWD	11/15/04
	Steve Novosad, SwRI	SEN	11/15/04
	Steven W. Dellenback, SwRI	SWD	10/16/07
	Steve Dellenback, SwRI	SWD	11/14/07
Modified By:	Meredith Moczygemba, SwRI	MRM	11/17/05
	Meredith Moczygemba, SwRI	MRM	09/12/07
	Steve Dellenback, SwRI	SWD	11/14/07
	Adam Hoffman, SwRI	AGH	3/23/16
Completed By:			

Table of Contents

List of Figures

List of Acronyms

ATMS	Advanced Traffic Management System
DOT	Department of Transportation
FDOT	Florida Department of Transportation
ITS	Intelligent Transportation Systems
ITN	Invitation to Negotiate
SwRI	Southwest Research Institute
TMC	Traffic Management Center
XML	Extensible Markup Language

REVISION HISTORY

Revision	Date	Changes
1.0.0	November 4, 2002	Initial Release
1.0.2	November 17, 2005	Updated for Release 2 software
3.0.0	October 16, 2007	Updated for Release 3 software
3.0.1	November 14, 2007	Added “how to use this document” section
6.2	March 23, 2016	Updated for Release 6.2

1. Scope

1.1 Document Identification

This Interface Control Document (ICD) describes the interface between individual SunGuide^{TM} clients and the various subsystems and between the subsystems and the associated drivers. The general base architecture of the XML communications including connection information, byte order and base transaction classes is delineated in this document. This ICD defines Extensible Markup Language (XML) schemas upon which XML requests shall be based in communicating amongst the various processes

1.2 Project Overview

The Florida Department of Transportation (FDOT) is conducting a program that is developing SunGuide software. The SunGuide software is a set of Intelligent Transportation System (ITS) software that allows the control of roadway devices as well as information exchange across a variety of transportation agencies. The goal of the SunGuide software is to have a common software base that can be deployed throughout the state of Florida. The SunGuide software development effort is based on ITS software available from the state of Texas; significant customization of the software is being performed as well as the development of new software modules. The following figure provides a graphical view of the software to be developed:

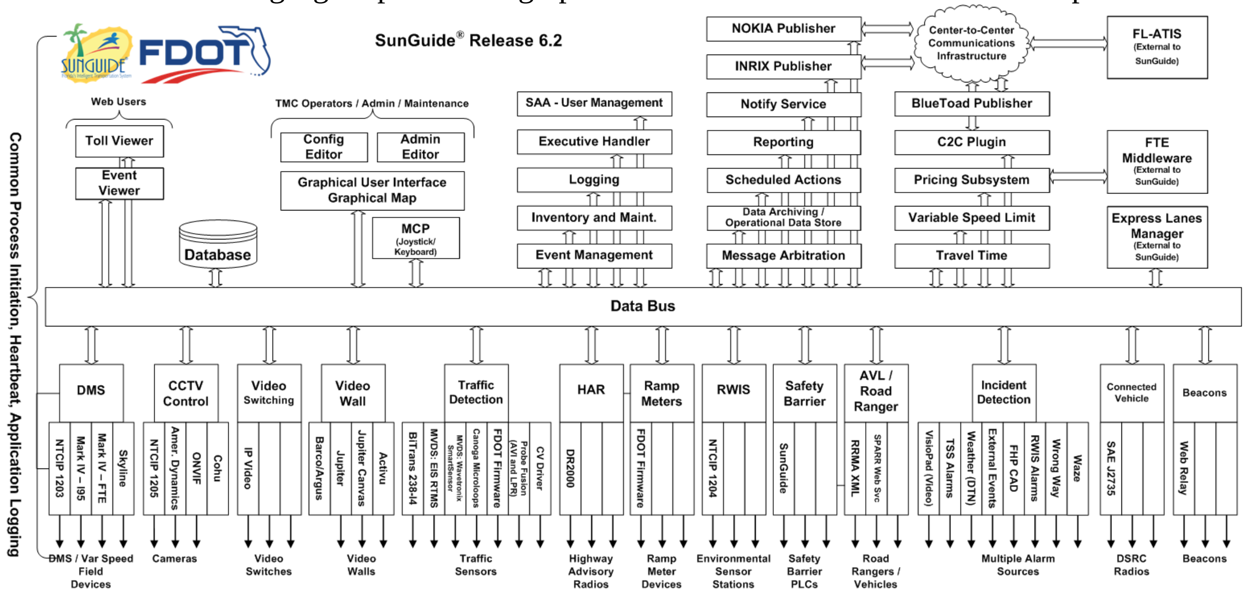


Figure 1-1 - High-Level Architectural Concept

1.3 How to Use This Document

The ICDs describe the specific interface between two SunGuide subsystems or between a SunGuide subsystem and a SunGuide driver. The relationship of appropriate documents is shown in the Figure 1-2.

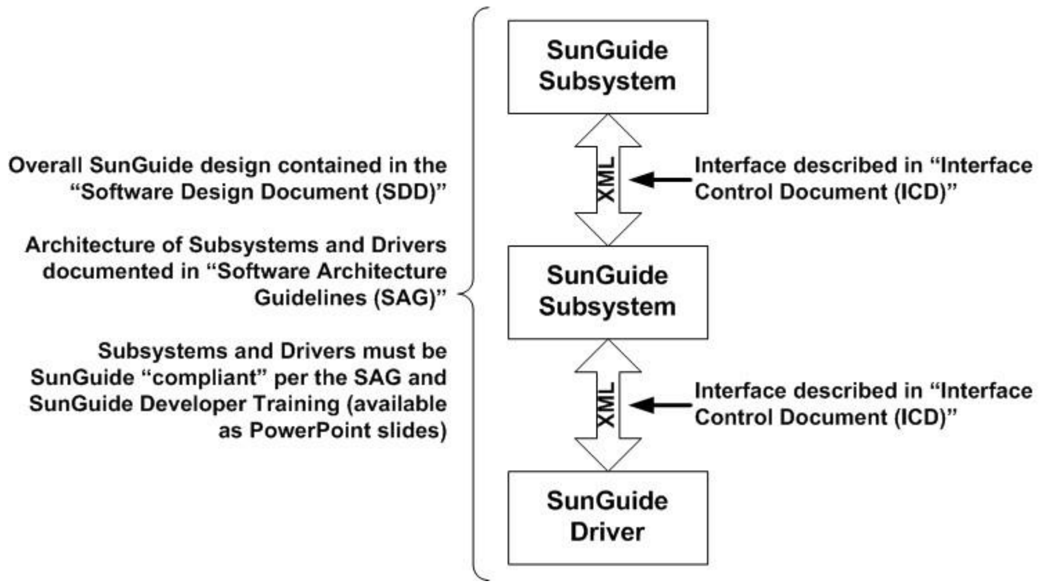

Overall SunGuide design contained in the "Software Design Document (SDD)"
Architecture of Subsystems and Drivers documented in "Software Architecture Guidelines (SAG)"
Subsystems and Drivers must be SunGuide "compliant" per the SAG and SunGuide Developer Training (available as PowerPoint slides)
SunGuide Subsystem
XML
Interface described in "Interface Control Document (ICD)"
SunGuide Subsystem
XML
Interface described in "Interface Control Document (ICD)"
SunGuide Driver

Figure 1-2 - SunGuide Developer Documentation

This document describes an *internal* SunGuide interface. The interface described is between two SunGuide compliant processes. The reader should review the following document to gain an understanding of how SunGuide compliant application is created (this will vary if the application is a driver or subsystem):

SunGuide Software Architecture Guidelines (SAG)

The SAG describes what needs to be included in a SunGuide application to assure that it will work cooperatively in the SunGuide environment. Once the SAG is reviewed, the following document should be reviewed:

SunGuide Software Design Document (SDD)

The SDD will provide an understanding of how individual components of SunGuide were designed. Finally the ICD, along with the associated schema should be reviewed to determine what data needs to be exchanged on the interface being defined in this document.

Additionally, a SunGuide "Developer Training" class is available that provides the students with an introduction into developing SunGuide processes. The SunGuide source code repository has a generic subsystem and a generic driver available that can be used as the basis for developing a new application.

1.4 Related Documents

The following documents were used to develop this document:

- FDOT Scope of Services: *BDQ69, Standard Written Agreement for SunGuide Software Support, Maintenance, and Development, Exhibit A: Scope of Services.* July 1, 2010.
- Notice to Proceed: Letter to Southwest Research Institute® (SwRI®) for BDQ69, July 1, 2010.
- SunGuide Project website: http://sunguidesoftware.com.

1.5 Contacts

The following are contact persons for the SunGuide software project:

- Fred Heery, ITS Section, Traffic Engineering and Operations Office Central Office, fred.heery@dot.state.fl.us, 850-410-5606
- Derek Vollmer, ITS Section, Traffic Engineering and Operations Office Central Office, Derek.Vollmer@dot.state.fl.us, 850-410-5615
- Clay Packard, Atkins Project Manager, clay.packard@dot.state.fl.us, 850-410-5623
- David Chang, Atkins Project Advisor, david.chang@dot.state.fl.us, 850-410-5622
- Tucker Brown, SwRI Project Manager, tbrown@swri.com, 210-522-3035
- Roger Strain, SwRI Software Project Manager, rstrain@swri.org, 210-522-6295

2. Data

The following sections detail the transactions that can be exchanged between client and server applications. Transactions begin with the size of the data being sent, followed by the compression type and the XML command being sent.

Transactions are sent from the client to the server and from the server to the client and may be either requests, responses or messages. The data formats expected are identical for requests, messages and responses. The actual XML being sent determines the type of command that is being transmitted.

2.1 Message Format

The data items for each data type are described in detail, including the data type and size, and a detailed description (if specific values apply to the item). All integer and bitmap data will be in big endian byte order (i.e. the least significant byte is the farthest to the right). Bits are labeled from right (#0) to left of a byte or word. Strings are not null terminated and are of variable length.

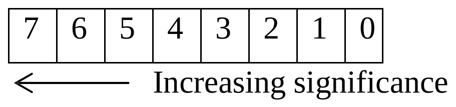


Figure 2-1 – Illustration of Byte Ordering

Table 2.1 – Transaction Parameters

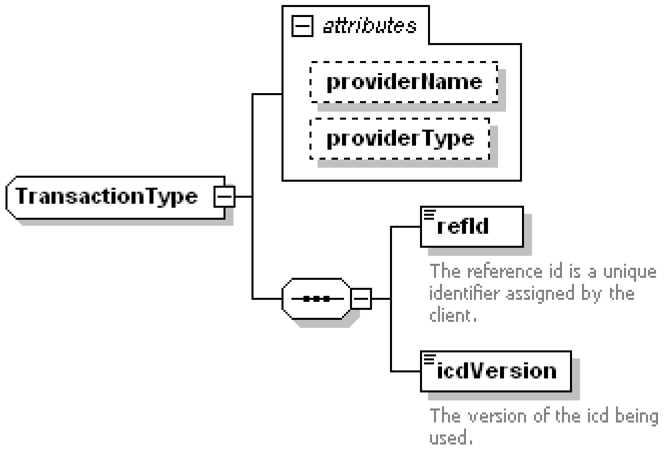
Data Item Description	Data Type and Size	Detailed Data Description
Transmitted Size	Integer – 4 bytes	The size of the XML being transmitted (if compression is being used, then this is the size of the data after compression is performed).
Decompressed Size	Integer – 4 bytes	This is the size of the original message before compression. ISO-8859-1 is used to compress/decompress. If original message size is equal to zero, then compression is not used. Note: ZLIB is the only supported compression algorithm. More information on ZLIB is available at: http://www.gzip.org/zlib/.
XML	String - Varied	Variable, see schema information for specific requests, responses, and messages.

XML schemas, sample XML and documentation for the requests and responses are shown in the following sections.

2.2 XML Transactions

The diagram in Figure 2-2 shows the transaction type used by all requests, messages, and responses sent to or from the subsystem. The TransactionType is an abstract type that each request, message, or response sent to the subsystem must extend. The *refId* is an identifier assigned by the client to uniquely identify this transaction. Responses generated by the subsystem will return the *refId* that was designated in the request. The *icdVersion* will verify that the client is using the appropriate ICD and XML schemas. The *providerName* attribute stores the name of the data provider (i.e. “cctv1,” “tampaDms,” etc.) as specified in the configuration file, while the *providerType* attribute stores the type of data provider (i.e. “cctv,” “dms,” etc.).

diagram


attributes
providerName
providerType
TransactionType
refId
The reference id is a unique identifier assigned by the client.
icdVersion
The version of the icd being used.

children **refId icdVersion**

used by complexTypes **MessageType RequestType ResponseType**

attributes

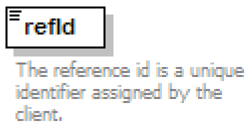
Name	Type	Use	Default	Fixed	Annotation
providerName	identifier	optional			
providerType	identifier	optional			

source

```
<xs:complexType name="TransactionType" abstract="true">
  <xs:sequence>
    <xs:element name="refId" type="xs:string">
      <xs:annotation>
        <xs:documentation>The reference id is a unique identifier assigned by the client.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="icdVersion" type="xs:string">
      <xs:annotation>
        <xs:documentation>The version of the icd being used.</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="providerName" type="identifier" use="optional"/>
  <xs:attribute name="providerType" type="identifier" use="optional"/>
</xs:complexType>
```

Figure 2-2 – TransactionType Abstract Type

diagram


refId
The reference id is a unique identifier assigned by the client.

type **xs:string**

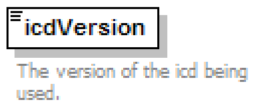
annotation documentation The reference id is a unique identifier assigned by the client.

source

```
<xs:element name="refId" type="xs:string">
```

```
  <xs:annotation>
    <xs:documentation>The reference id is a unique identifier assigned by the client.</xs:documentation>
  </xs:annotation>
</xs:element>
```

Figure 2-3 – Required refId Element

diagram	
type	**xs:string**
annotation	documentation The version of the icd being used.
source	`<xs:element name="icdVersion" type="xs:string">`

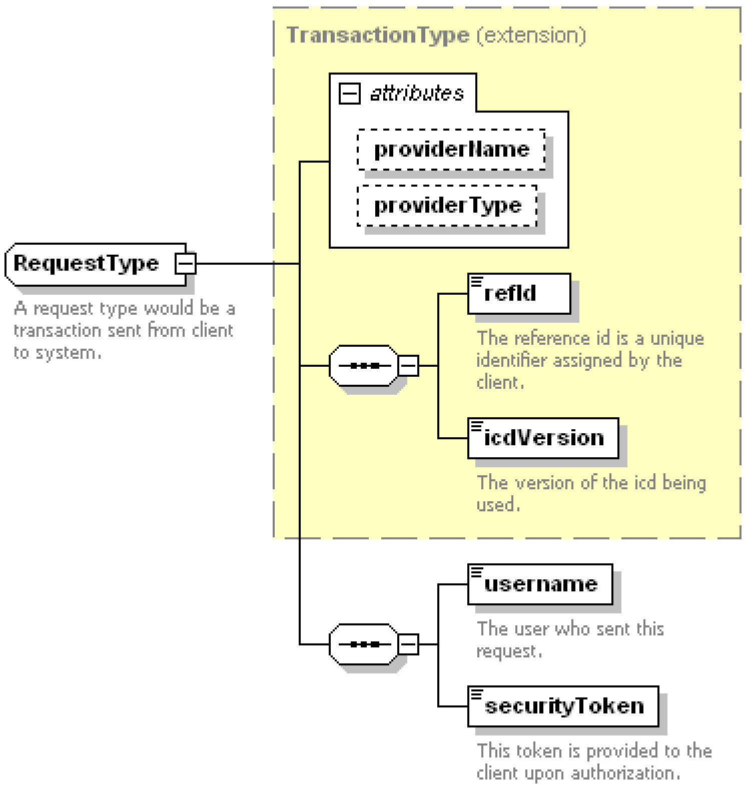
```
<xs:element name="icdVersion" type="xs:string">
  <xs:annotation>
    <xs:documentation>The version of the icd being used.</xs:documentation>
  </xs:annotation>
</xs:element>
```

Figure 2-4 – Required icdVersion Element

2.2.1 Request

A request (Figure 2-5) is a transaction that contains two additional data fields: *username* and *securityToken*. Requests that are sent from clients to subsystems must contain the *username* and *securityToken* element. These two elements are optional as requests from subsystems to drivers do not require an authenticated user. The *username* is a string representing the client who has been authenticated. The *securityToken* is a string that is returned to a client upon authentication to a subsystem.

diagram


TransactionType (extension)
attributes
providerName
providerType
RequestType
A request type would be a transaction sent from client to system.
refId
The reference id is a unique identifier assigned by the client.
icdVersion
The version of the icd being used.
username
The user who sent this request.
securityToken
This token is provided to the client upon authorization.

type extension of **TransactionType**

children **refId icdVersion username securityToken**

attributes

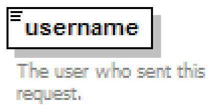
Name	Type	Use	Default	Fixed	Annotation
providerName	identifier	optional			
providerType	identifier	optional			

annotation documentation A request type would be a transaction sent from client to system.

source

```
<xs:complexType name="RequestType" abstract="true">
  <xs:annotation>
    <xs:documentation>A request type would be a transaction sent from client to system.</xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="TransactionType">
      <xs:sequence>
        <xs:element name="username" type="identifier">
          <xs:annotation>
            <xs:documentation>The user who sent this request.</xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="securityToken">
          <xs:annotation>
            <xs:documentation>This token is provided to the client upon authorization.</xs:documentation>
          </xs:annotation>
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:minLength value="6"/>
              <xs:maxLength value="30"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 2-5 – RequestType Abstract Type

diagram	
type	identifier
facets	minLength 1 maxLength 30 whiteSpace preserve
annotation	documentation The user who sent this request.
source	see below

```
<xs:element name="username" type="identifier" minOccurs="0">
  <xs:annotation>
    <xs:documentation>The user who sent this request.</xs:documentation>
  </xs:annotation>
</xs:element>
```

Figure 2-6 – Required username Element

diagram	securityToken This token is provided to the client upon authorization.
type	restriction of **xs:string**
facets	minLength 6 maxLength 30
annotation	documentation This token is provided to the client upon authorization.
source	see below

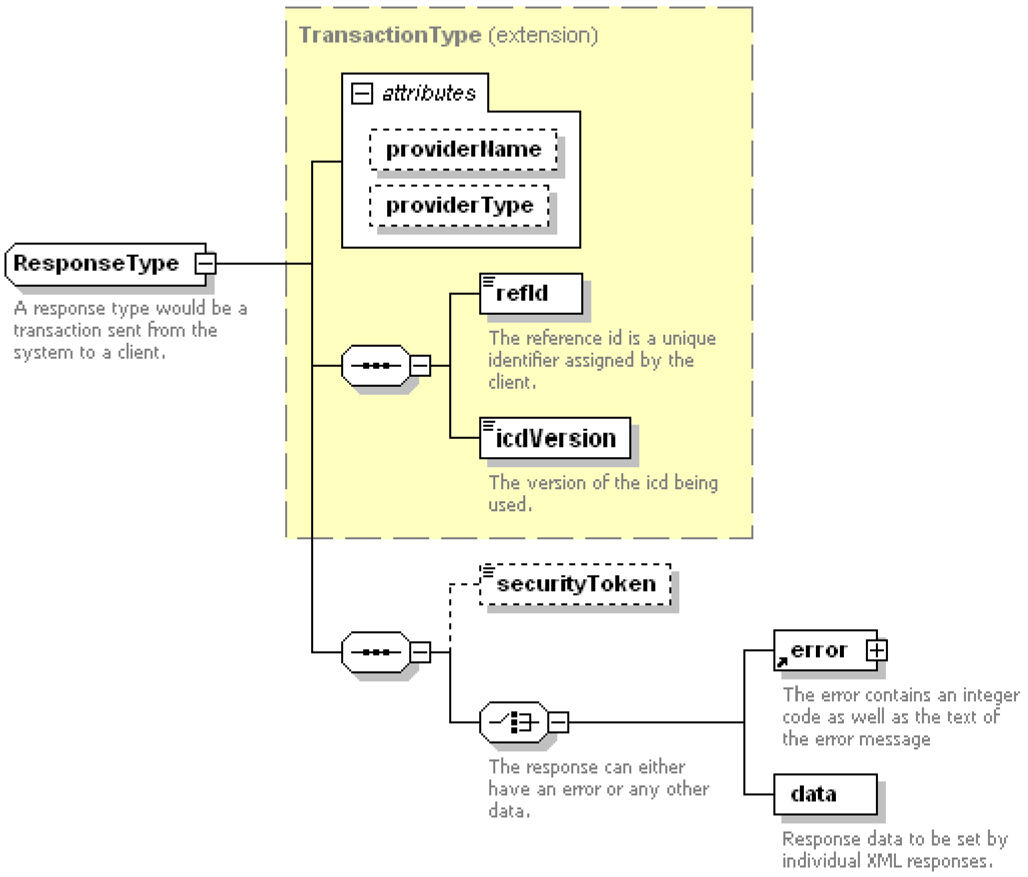
```
<xs:element name="securityToken" minOccurs="0">
  <xs:annotation>
    <xs:documentation>This token is provided to the client upon authorization.</xs:documentation>
  </xs:annotation>
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:minLength value="6"/>
      <xs:maxLength value="30"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Figure 2-7 – Required securityToken Element

2.2.2 Response

The response is a transaction that contains either an error or the response data. An error returned by the system will contain an error code, which is an integer value, and an error string containing the text of the error message. Other optional information may also be returned. If there is no error, the response data is returned. For responses where the client has added or modified data, the response will include the new data. The data is returned in the response to allow clients who have subscribed to the data changes to receive the same response and update their data. The optional security token is used by the Data Bus to route subsystem responses to the appropriate clients.

diagram


TransactionType (extension)
attributes
providerName
providerType
ResponseType
A response type would be a transaction sent from the system to a client.
refId
The reference id is a unique identifier assigned by the client.
icdVersion
The version of the icd being used.
securityToken
error
The error contains an integer code as well as the text of the error message
The response can either have an error or any other data.
data
Response data to be set by individual XML responses.

type extension of **TransactionType**

children **refId icdVersion securityToken error data**

attributes

Name	Type	Use	Default	Fixed	Annotation
providerName	identifier	optional			
providerType	identifier	optional			

annotation documentation A response type would be a transaction sent from the system to a client.

source

```
<xs:complexType name="ResponseType" abstract="true">
 <xs:annotation>
  <xs:documentation>A response type would be a transaction sent from the system to a client.</xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="TransactionType">
   <xs:sequence>
    <xs:element name="securityToken" minOccurs="0"/>
    <xs:choice>
     <xs:annotation>
      <xs:documentation>The response can either have an error or any other data.</xs:documentation>
     </xs:annotation>
     <xs:element name="error">
      <xs:annotation>
       <xs:documentation>The error contains an integer code as well as the text of the error message</xs:documentation>
      </xs:annotation>
      <xs:complexType>
       <xs:sequence>
        <xs:element name="errorCode" type="xs:integer">
         <xs:annotation>
          <xs:documentation>The error code may be used by the client.</xs:documentation>
         </xs:annotation>
        </xs:element>
        <xs:element name="errorMap" type="xs:string" minOccurs="0">
         <xs:annotation>
```

```
            <xs:documentation>Name of table to lookup the error code.</xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="errorString" type="xs:string">
          <xs:annotation>
            <xs:documentation>This string contains the error text set by the originating process.</xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="errorData" type="xs:string" minOccurs="0">
          <xs:annotation>
            <xs:documentation>Additional error information.</xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="equipmentId" type="xs:string" minOccurs="0">
          <xs:annotation>
            <xs:documentation>The string contains the equipment identifier</xs:documentation>
          </xs:annotation>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="data" type="responseData">
    <xs:annotation>
      <xs:documentation>Response data to be set by individual XML responses.</xs:documentation>
    </xs:annotation>
  </xs:element>
 </xs:choice>
</xs:sequence>
</xs:extension>
</xs:complexContent>
</xs:complexType>
```

Figure 2-8 – ResponseType Abstract Type

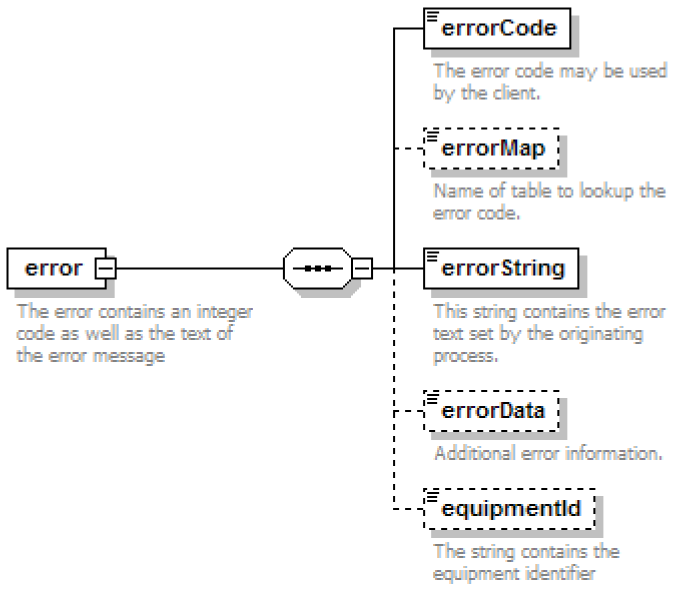
diagram

properties	isRef	0
	minOcc	0
	maxOcc	1

source `<xs:element name="securityToken" minOccurs="0"/>`

Figure 2-9 – Optional securityToken Element

diagram


error
The error contains an integer code as well as the text of the error message
errorCode
The error code may be used by the client.
errorMap
Name of table to lookup the error code.
errorString
This string contains the error text set by the originating process.
errorData
Additional error information.
equipmentId
The string contains the equipment identifier

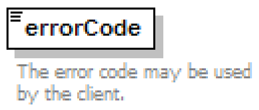
children **errorCode errorMap errorString errorData equipmentId**

annotation documentation The error contains an integer code as well as the text of the error message

source
```
<xs:element name="error">
 <xs:annotation>
  <xs:documentation>The error contains an integer code as well as the text of the error message</xs:documentation>
 </xs:annotation>
 <xs:complexType>
  <xs:sequence>
   <xs:element name="errorCode" type="xs:integer">
    <xs:annotation>
     <xs:documentation>The error code may be used by the client.</xs:documentation>
    </xs:annotation>
   </xs:element>
   <xs:element name="errorMap" type="xs:string" minOccurs="0">
    <xs:annotation>
     <xs:documentation>Name of table to lookup the error code.</xs:documentation>
    </xs:annotation>
   </xs:element>
   <xs:element name="errorString" type="xs:string">
    <xs:annotation>
     <xs:documentation>This string contains the error text set by the originating process.</xs:documentation>
    </xs:annotation>
   </xs:element>
   <xs:element name="errorData" type="xs:string" minOccurs="0">
    <xs:annotation>
     <xs:documentation>Additional error information.</xs:documentation>
    </xs:annotation>
   </xs:element>
   <xs:element name="equipmentId" type="xs:string" minOccurs="0">
    <xs:annotation>
     <xs:documentation>The string contains the equipment identifier</xs:documentation>
    </xs:annotation>
   </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Figure 2-10 – Required error Element

diagram



type **xs:integer**

annotation documentation The error code may be used by the client.

source
```
<xs:element name="errorCode" type="xs:integer">
  <xs:annotation>
    <xs:documentation>The error code may be used by the client.</xs:documentation>
  </xs:annotation>
</xs:element>
```

Figure 2-11 – Required errorCode Element

diagram

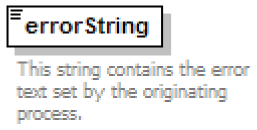
type **xs:string**

annotation documentation Name of table to lookup the error code.

source
```
<xs:element name="errorMap" type="xs:string" minOccurs="0">
  <xs:annotation>
    <xs:documentation>Name of table to lookup the error code.</xs:documentation>
  </xs:annotation>
</xs:element>
```

Figure 2-12 – Optional errorMap Element

diagram


errorString
This string contains the error text set by the originating process.

type **xs:string**

annotation documentation This string contains the error text set by the originating process.

source
```
<xs:element name="errorString" type="xs:string">
  <xs:annotation>
    <xs:documentation>This string contains the error text set by the originating process.</xs:documentation>
  </xs:annotation>
</xs:element>
```

Figure 2-13 – Required errorString Element

diagram

type **xs:string**

annotation documentation Additional error information.

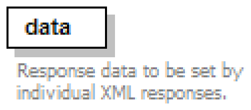
source
```
<xs:element name="errorData" type="xs:string" minOccurs="0">
  <xs:annotation>
    <xs:documentation>Additional error information.</xs:documentation>
  </xs:annotation>
</xs:element>
```

Figure 2-14 – Optional errorData Element

diagram		
type	**xs:string**	
annotation	documentation	The string contains the equipment identifier

source
```
<xs:element name="equipmentId" type="xs:string" minOccurs="0">
  <xs:annotation>
    <xs:documentation>The string contains the equipment identifier</xs:documentation>
  </xs:annotation>
</xs:element>
```

Figure 2-15 – Optional equipmentId Element

diagram		
type	**responseData**	
annotation	documentation	Response data to be set by individual XML responses.

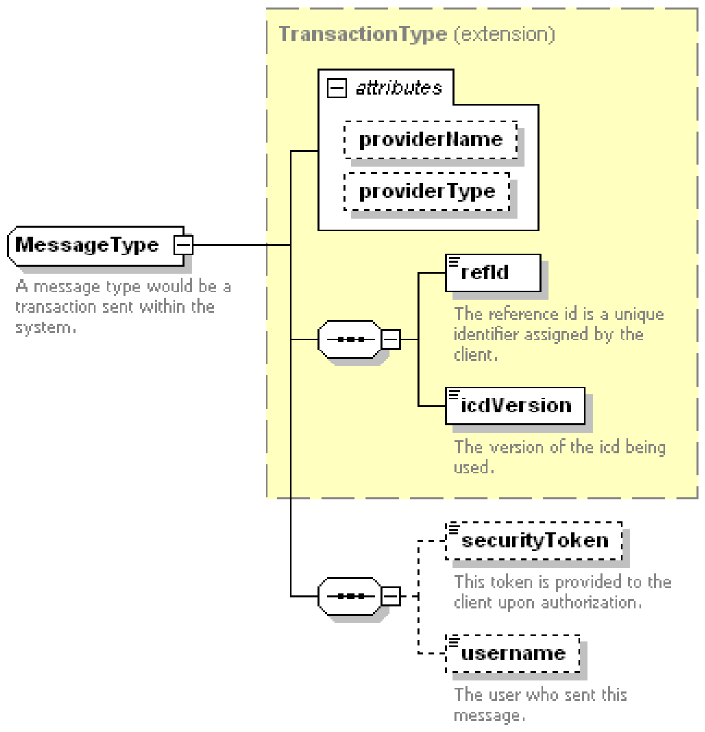
source
```
<xs:element name="data" type="responseData">
  <xs:annotation>
    <xs:documentation>Response data to be set by individual XML responses.</xs:documentation>
  </xs:annotation>
</xs:element>
```

Figure 2-16 – Required data Element

2.2.3 Message

A message (Figure 2-17) is a transaction that contains an optional *username* and *securityToken*. If sent from a client, both of these must be set for a subsystem to respond. When sent to a client, the *securityToken* must be sent for the Data Bus to route the message to the appropriate client. Neither are required for messages sent between a subsystem and a driver. Messages are sent asynchronously when a response is not required.

diagram


TransactionType (extension)
attributes
providerName
providerType
MessageType
A message type would be a transaction sent within the system.
refId
The reference id is a unique identifier assigned by the client.
icdVersion
The version of the icd being used.
securityToken
This token is provided to the client upon authorization.
username
The user who sent this message.

type extension of TransactionType

children refId icdVersion securityToken username

attributes

Name	Type	Use	Default	Fixed	Annotation
providerName	identifier	optional			
providerType	identifier	optional			

annotation documentation A message type would be a transaction sent within the system.

source

```
<xs:complexType name="MessageType" abstract="true">
 <xs:annotation>
  <xs:documentation>A message type would be a transaction sent within the system.</xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="TransactionType">
   <xs:sequence>
    <xs:element name="securityToken" minOccurs="0">
     <xs:annotation>
      <xs:documentation>This token is provided to the client upon authorization.</xs:documentation>
     </xs:annotation>
     <xs:simpleType>
      <xs:restriction base="xs:string">
       <xs:minLength value="6"/>
       <xs:maxLength value="30"/>
      </xs:restriction>
     </xs:simpleType>
    </xs:element>
    <xs:element name="username" type="identifier" minOccurs="0">
     <xs:annotation>
      <xs:documentation>The user who sent this message.</xs:documentation>
     </xs:annotation>
    </xs:element>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
```

Figure 2-17 - MessageType Abstract Type

2.3 Initial Client-Subsystem Communication

The following communications are valid for all regular subsystems (i.e., DMS, CCTV, TSS). The three commands listed would be the general sequence that should occur when a client connects to a subsystem.

2.3.1 Authenticate

Before any other commands can be sent, a client must send an authenticate request to register with the regular subsystem. The authenticate request is a transaction type which contains two additional fields: *username* and *password*. The *username* is the name of the application or user that is connecting to the subsystem. The application/user must have an associated password that the subsystem can retrieve from the database. The *password* sent as part of this request will be encrypted using Message Digest 5 (MD5) hashing.

The system uses the *username* and *password* to verify the client's privileges. If the authentication is successful, a *securityToken* will be returned to the client. If not successful, an error message will be returned. The *securityToken* returned by the subsystem to the client must be sent with each additional request and will be used to validate the client's ability to perform the request.

2.3.2 Subscribe

Another request that a client might send to the system is a subscribe request. The client can specify what types of data updates should be sent. Then, if that data changes at a later time, the client will receive unsolicited responses with updated data. The subscribe response will return true values for data to which the client has successfully subscribed The subscribe may fail if the client does not have permission to retrieve the data types requested. If a client subscribes more than once, the latest subscription will override previously stored subscription data in the system.

2.3.3 Retrieve Data

A client may also choose to retrieve available data from the subsystem. The client may specify what types of data should be retrieved. This request, used in conjunction with the subscribe request, is used to ensure the client has valid data objects for the subsystem.

2.4 Initial Subsystem-Driver Communication

Unlike client-subsystem connections, communications between a subsystem and a driver do not require authentication. Drivers do not communicate with the database, so when a subsystem connects to a driver, the subsystem must send all appropriate data to the driver. For example, when CCTV subsystem connects to the CCTV driver, a list of all cameras that driver should control is sent to the driver.

2.5 Common XML Commands

Each SunGuide subsystem is responsible for implementing the following commands. (Refer to sections Subsystem Schemas and Driver Schemas for further information regarding the commands described).

2.5.1 setSystemPropertiesReq

The setSystemPropertiesReq has settings for log mode and validation. Each subsystem is responsible for ensuring that the appropriate actions are taken when this request is received.

2.5.2 updateSystemDataMsg

The updateSystemDataMsg is request sent when user information has been modified outside of the subsystem. Each subsystem is responsible for ensuring that the appropriate action is taken when this request is received.

2.5.3 clientDisconnectMsg

The clientDisconnectMsg is request sent when a client has disconnected from the Data Bus. This allows each subsystem to ensure connection information for that user is cleared. Each subsystem is responsible for ensuring that the appropriate action is taken when this request is received.

2.6 Network Connection

A TCP/IP connection will be utilized to exchange data. A session begins when the client connects to the server and sends the authenticate request. The session ends when the connection closes. If the connection between the client and server is severed for any reason, the client must start anew by opening a TCP connection and sending the authenticate request. If the client wishes to resubscribe to data updates, a subscription request must also be resent each time the client reconnects.

2.7 Schema

The schemas for these transactions may be located in the Schemas directory. The objects directory contains common data schemas that are used by the various request/messages/responses. Schemas are organized in the following tree structure:

common

- messages
    - clientDisconnectMsg.xsd
    - deviceStatusUpdateMsg.xsd
    - startPollingMsg.xsd
    - updateConnectionsMsg.xsd
    - updateSystemDataMsg.xsd
    - userNotificationMsg.xsd
    - userUpdatedMsg.xsd
- objects
    - address.xsd
    - baud.xsd
    - common.xsd
    - contact.xsd
    - databaseId.xsd
    - direction.xsd
    - driver.xsd
    - equipmentGroup.xsd
    - equipmentLocation.xsd

  - equipmentStatus.xsd
  - font.xsd
  - id.xsd
  - idName.xsd
  - lName.xsd
  - location.xsd
  - Point.xsd
  - transaction.xsd
- requests
  - addEquipmentGroupReq.xsd
  - authenticateReq.xsd
  - connectionReq.xsd
  - copyEquipmentGroupReq.xsd
  - deleteEquipmentGroupReq.xsd
  - modifyEquipmentGroupReq.xsd
  - renameEquipmentGroupReq.xsd
  - setSystemPropertiesReq.xsd
  - synchronizeClockReq.xsd
- responses
  - addEquipmentGroupResp.xsd
  - authenticateResp.xsd
  - connectionResp.xsd
  - copyEquipmentGroupResp.xsd
  - deleteEquipmentGroupResp.xsd
  - errorResp.xsd
  - modifyEquipmentGroupResp.xsd
  - renameEquipmentGroupResp.xsd
  - setSystemPropertiesResp.xsd
  - synchronizeClockReq.xsd

Requests may be sent from a client to a subsystem or from a subsystem to a driver. Responses may be sent from a driver to a subsystem or a subsystem to a client. A message can be sent from any process to another process.

2.8 Examples

For example, if a client wishes to change the log level of a subsystem logging to the Status Logger, the client sends a setSystemPropertiesReq to the subsystem. The subsystem changes the log level as appropriate and sends a setSystemPropertiesResp to the calling client.

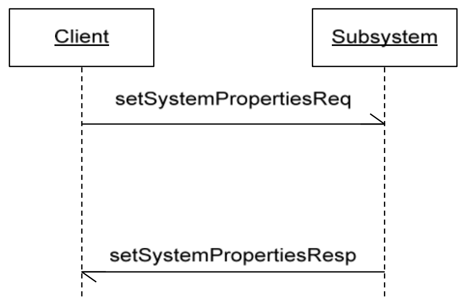


Figure 2-18 - Sample Transaction

The tables below show which requests can be sent from client to subsystem and subsystem to driver. The responses sent from driver to subsystem and subsystem to client are also specified. Messages are sent when a response is not required.

2.9 Subsystem Schemas

FC (From client), TC (To client), TD (To driver), FD (From driver)

Usage Description	Requests	FC	TD	Responses	FD	TC	Messages	TD	FD	TC
Used to add a new device group to a subsystem.	addEquipmentGroup Req	X		addEquipmentGroup Resp		X				
Used to register a client to with a subsystem.	authenticateReq	X		authenticateResp		X				
Used to inform a subsystem that a client has disconnected from the Data Bus.							clientDisconnectMsg	X		
Used to connect a source to a viewer or disconnect a source from a viewer.	connectionReq	X	X	connectionResp	X	X				
Used to copy and existing equipment group to a new group name.	copyEquipmentGrou pReq	X		copyEquipmentGrou pResp		X				
Used to delete an equipment group from the subsystem.	deleteEquipmentGro upReq	X		deleteEquipmentGrou pResp		X				

Usage Description	Requests	FC	TD	Responses	FD	TC	Messages	TD	FD	TC
Used to notify client when the status of a device changes from online to offline or offline to online.							deviceStatusUpdateMsg			X
Sent when no corresponding request can be used.				errorResp		X				
Used to modify the devices contained in an equipment group.	modifyEquipmentGroupReq	X		modifyEquipmentGroupResp		X				
Used to rename an existing equipment group.	renameEquipmentGroupReq	X		renameEquipmentGroupResp		X				
Used to turn schema validation off and/or set the log mode of the process.	setSystemPropertiesReq	X	X	setSystemPropertiesResp	X	X				
Used to initiate device polling for the specified device.							startPollingMsg	X		
Used to set the clock in a device to the time on the local server	synchronizeClockReq	X	X	synchronizeClockResp	X	X				
Used to update a camera connection.							updateConnectionsMsg		X	

Usage Description	Requests	FC	TD	Responses	FD	TC	Messages	TD	FD	TC
Used to request that the process update its cached data. This message will be sent if user information is modified outside of the system.							updateSystemDataMsg	X		
Used to notify a user. A string containing the text of a message is included.							UserNotificationMsg			X
Used to indicate that a user has logged in or out.							UserUpdatedMsg			X

2.10 Driver Schemas

TD (To driver), FD (From driver)

Usage Description	Requests	TD	Responses	FD	Messages	TD	FD
Used to connect a source to a viewer or disconnect a source from a viewer.	connectionReq	X	connectionResp	X			
Used to turn schema validation off and/or set the log mode of the process.	setSystemPropertiesReq	X	setSystemPropertiesResp	X			
Used to update a camera connection.					updateConnectionsMsg		X

Usage Description	Requests	TD	Responses	FD	Messages	TD	FD
Used to request that the process update its cached data. This message will be sent if user information is modified outside of the system.					updateSystemDataMsg	X	

3. Notes

Information about XML and schemas can be found at the World Wide Web Consortium (W3) website at http://www.w3.org.

Attachment 1

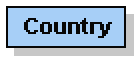
Documentation Key

Documentation Key

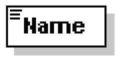
The documentation for the XML schemas uses notation that is unique to this type of document.

The following icons and terminology are used in the schema design documentation. This documentation is generated by XML Spy (www.altova.com).

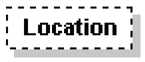
Element Symbols



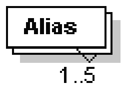
Mandatory single element. This element is required in the XML for the XML to be valid with respect to the schema. Details: MinOcc=1, MaxOcc=1



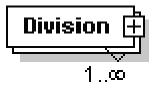
Mandatory single element, containing Parsed Character Data (#PC-Data). This element is required in the XML for the XML to be valid with respect to the schema.
The content may be simple content or mixed complex content.
Details: MinOcc=1, MaxOcc=1, type=xsd:string, content=simple.



Single optional element. This element is not required in the XML. Details: MinOcc=0, MaxOcc=1

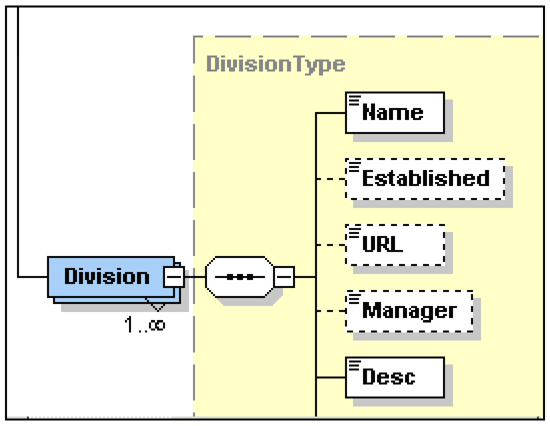


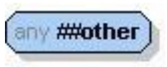
Mandatory multiple element. This element is required in the XML for the XML to be valid with respect to the schema. No more than five of these elements may exist. Details MinOcc=1, MaxOcc=5.



Mandatory multiple element containing child elements.
Details: MinOcc=1, MaxOcc=unbounded, type=DivisionType, content=complex.

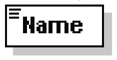
The element expanded to show the child elements.





"Any" can be a placeholder for any element from a certain namespace.

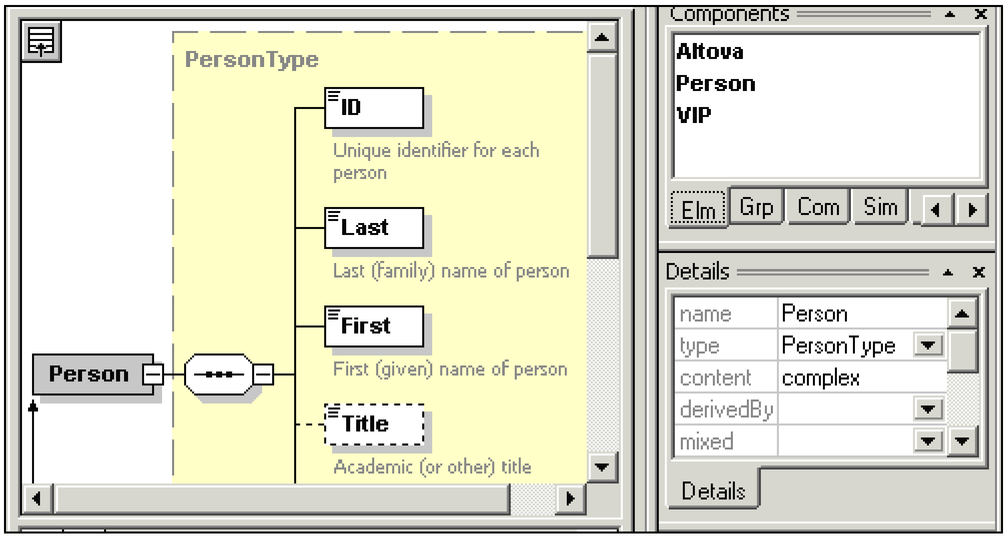
Simple types:
A "simple type" element is defined as a datatype that only contains values and no element or attributes.



Details: name=Name, type=xsd:string, content=simple.

Complex types:
"Complex type" is a datatype which may contain attributes, elements and text.

References to "external" complex types are displayed with a yellow background. In the example below, the Person element references the PersonType complex element.

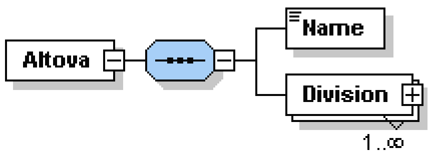

PersonType
ID
Unique identifier for each person
Last
Last (family) name of person
First
First (given) name of person
Title
Academic (or other) title
Person
Components
Altova
Person
VIP
Elm
Grp
Com
Sim
Details

name	Person
type	PersonType
content	complex
derivedBy	
mixed	

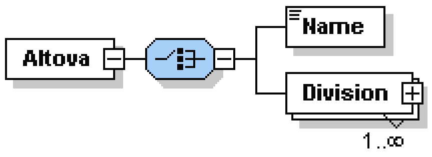
Compositors

A "Compositor" defines an ordered sequence of sub elements (child elements).

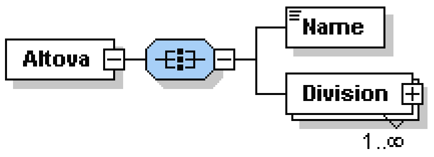
Sequence:



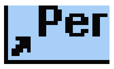
Choice:



All:



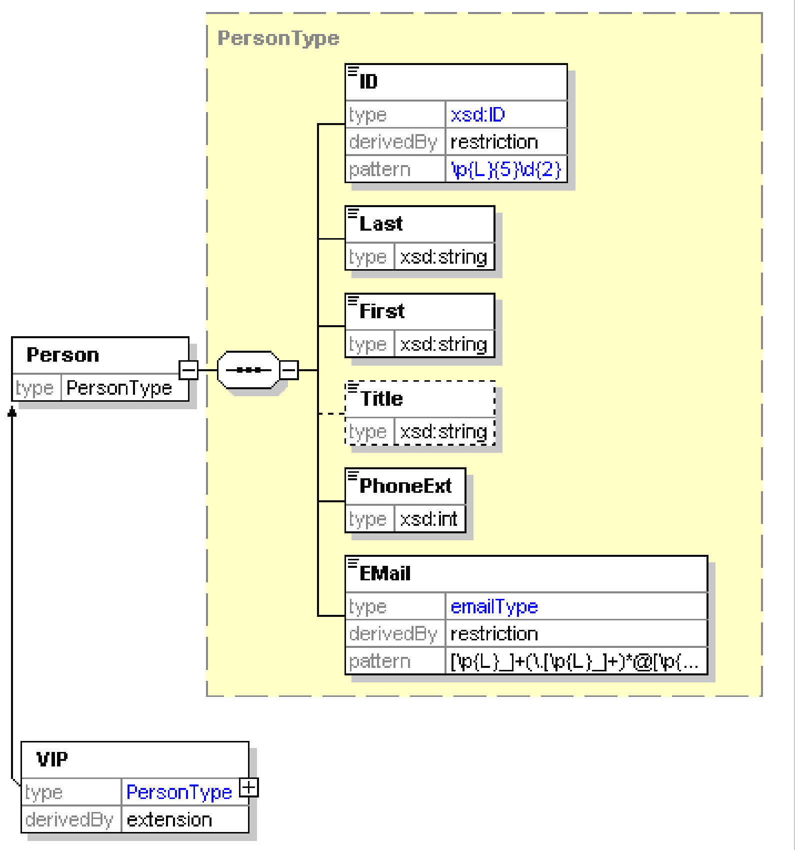
References to global components
If a schema references a component in another schema, a link icon appears at the bottom left of the element symbol.



Document example

The following element “Person” shows an example of the generated documentation. The Person is a PersonType which is displayed within the dotted lines. All of the elements of PersonType are depicted in this documentation. All elements are required except for Title, which is optional as designated by the dotted lines around the element box. The type of each element is shown as well as the restrictions if any exist. Attribute information, including type and whether required, is depicted in the table.

element **Person**

diagram	
namespace	http://www.xmlspy.com/schemas/Altova/orgchart
type	**PersonType**
children	**ID Last First Title PhoneExt EMail**
used by	complexType **DivisionType**
attributes	Name Type Use Value Mgr xsd:boolean required Prg xsd:boolean required Des xsd:boolean optional
source	<xsd:element name="Person" type="PersonType"/>