

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: March 9, 2019

J. Gould
VeriSign, Inc.
September 5, 2018

Data Set File Format
draft-gould-regext-dataset-02

Abstract

This document defines a Data Set File (DSF) format that can be used to define and pass bulk data between a client and a server. This format is extensible to pass any set of simple data types in a set of records contained in the body of the file. The file format also supports storing the result of processing the data set that MAY be generated by the server and returned to the client. The file format consists of an XML definition header and a Comma-Separated Values (CSV) data section delimited by "-----BEGIN DATA SET-----" and "-----END DATA SET-----" lines. The XML definition header defines the format of the CSV data section, contains additional meta-data, and optionally includes a digital signature to identify the source and ensure that the data has not been tampered with between the parties (source, client, and server). The interface (manual or automated) that is used to submit the file and receive the result file is not defined within this document.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on March 9, 2019.

Internet-Draft

Data Set File Format

September 2018

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.	Introduction	3
1.1.	Conventions Used in This Document	4
2.	General Conventions	5
2.1.	Date and Time	5
2.2.	Checksum	5
2.3.	Type and Subtype	5
2.4.	Fields	6
2.4.1.	Base Field Types	6
2.4.2.	Data Set Field Elements	8
2.4.2.1.	Field Element Extensibility	8
2.5.	Routing	10
3.	Data Set File (DSF) Format	11
3.1.	Header Format	12
3.1.1.	<dataSet:defData> element	12
3.1.2.	<dataSet:encodedSignedDefData> element	13
3.1.2.1.	Signed Definition Data	13
3.1.3.	<dataSet:resultData> element	15
3.2.	Body Format	18
4.	Object Description	18
4.1.	Domain Name Object	18
4.2.	Host Object	25
4.3.	Contact Object	31
4.4.	Verification Code Object	39
5.	Result Codes	41
6.	Example Data Set File (DSF) Result Files	44
6.1.	Example Data Set File (DSF) with Result Data	44

7.	Formal Syntax	47
7.1.	Data Set Schema	47
7.2.	Domain Name Schema	56
7.3.	Host Schema	60
7.4.	Contact Schema	62

7.5.	Verification Code Schema	67
7.6.	Routing Schema	69
8.	IANA Considerations	69
8.1.	XML Namespace	69
9.	Security Considerations	71
10.	Normative References	72
Appendix A.	Acknowledgements	73
Appendix B.	Change History	74
B.1.	Change from 00 to 01	74
B.2.	Change from 01 to 02	74
	Author's Address	74

[1.](#) Introduction

This document defines a Data Set File (DSF) format that can be used to define and pass bulk data between a client and a server. The client and server MAY leverage a provisioning protocol like EPP, as defined in [\[RFC5730\]](#), to provision individual objects, but the provisioning protocol MAY NOT handle all provisioning use cases. This document defines a format for bulk provisioning requests that can be generated by authorized third parties, can be generated by clients that choose not to leverage the provisioning protocol, and can be supported by servers to process bulk requests in a controlled manner that throttles the processing against the provisioning database. Bulk requests can include court orders, out-of-band client requests to fix data, the backfill of data like contact or verification code data, and a large set of ad-hoc needs.

This format is extensible to pass any set of simple data types in a set of records contained in the body of the file. The file format also supports storing the result of processing the data set that MAY be generated by the server and returned to the client. The file format consists of an XML definition header and a Comma-Separated Values (CSV) data section delimited by "-----BEGIN DATA SET-----" and "-----END DATA SET-----" lines. The XML definition header defines the format of the CSV data section, contains additional meta-data,

and optionally includes a digital signature to identify the source and ensure that the data has not been tampered with between the parties (source, client, and server) using XML Signature [[W3C.CR-xmlsig-core2-20120124](#)]. All XML Signature [[W3C.CR-xmlsig-core2-20120124](#)] data is "base64" encoded, in conformance with [[RFC2045](#)], by default to mitigate any validation errors due to whitespace formatting. The interface (manual or automated) that is used to submit the file and receive the result file is not defined within this document and must be documented independently. Please see [Section 9](#) for a description of the issues associated with transport security.

[1.1](#). Conventions Used in This Document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC 2119](#) [[RFC2119](#)].

XML is case sensitive. Unless stated otherwise, XML specifications and examples provided in this document MUST be interpreted in the character case presented in order to develop a conforming implementation.

The following XML abbreviations are used:

- "eppcom" XML namespace prefix refers to "urn:ietf:params:xml:ns:eppcom-1.0" XML namespace in [[RFC5730](#)].
- "domain" XML namespace prefix refers to "urn:ietf:params:xml:ns:domain-1.0" XML namespace in [[RFC5731](#)].
- "host" XML namespace prefix refers to "urn:ietf:params:xml:ns:host-1.0" XML namespace in [[RFC5732](#)].
- "contact" XML namespace prefix refers to "urn:ietf:params:xml:ns:contact-1.0" XML namespace in [[RFC5733](#)].
- "dataSet-1.0" is used as an abbreviation for "urn:ietf:params:xml:ns:dataSet-1.0".
- "dataSet" XML namespace prefix refers to "urn:ietf:params:xml:ns:dataSet-1.0" XML namespace.
- "dsfDomain-1.0" is used as an abbreviation for "urn:ietf:params:xml:ns:dsfDomain-1.0".
- "dsfDomain" XML namespace prefix refers to "urn:ietf:params:xml:ns:dsfDomain-1.0" XML namespace.

"dsfHost-1.0" is used as an abbreviation for
"urn:ietf:params:xml:ns:dsfHost-1.0".
"dsfHost" XML namespace prefix refers to
"urn:ietf:params:xml:ns:dsfHost-1.0" XML namespace.
"dsfContact-1.0" is used as an abbreviation for
"urn:ietf:params:xml:ns:dsfContact-1.0".
"dsfContact" XML namespace prefix refers to
"urn:ietf:params:xml:ns:dsfContact-1.0" XML namespace.
"dsfVerificationCode-1.0" is used as an abbreviation for
"urn:ietf:params:xml:ns:dsfVerificationCode-1.0".
"dsfVerificationCode" XML namespace prefix refers to
"urn:ietf:params:xml:ns:dsfVerificationCode-1.0" XML namespace.
"dsfRouting-1.0" is used as an abbreviation for
"urn:ietf:params:xml:ns:dsfRouting-1.0".

Implementations MUST NOT depend on the XML namespace prefix used in this document and instead MUST employ a proper namespace-aware XML parser and serializer to interpret and output the XML.

[2.](#) General Conventions

This document includes a general set of attributes and terms that are described here.

[2.1.](#) Date and Time

Numerous fields indicate "dates", such as the creation date of the DSF. These fields SHALL contain timestamps indicating the date and time in UTC as specified in [[RFC3339](#)], with no offset from the zero meridian.

[2.2.](#) Checksum

The checksum of the body of the file, as defined by the body rule of the Data Set File (DSF) ABNF ([Section 3](#)), MUST use CRC32, that is the algorithm used in the ISO 13239 standard [[iso13239](#)] and in [section 8.1.1.6.2](#) of ITU-T recommendation V.42 [[itu-t-V.42](#)].

[2.3.](#) Type and Subtype

There can be many different Data Set File (DSF) types supported based

on the objects, operations, and the specific scenarios. To make it easier to negotiate the client's intent of the DSF with the server's set of supported DSF files, the DSF supports the definition of a type and an optional sub-type. The supported list of types and sub-types is up to server policy. The server SHOULD provide the possible set of supported DSF type and sub-type values to the parties generating the DSF. The mechanism for providing the supported DSF type and sub-type values is not defined in this document but MAY require registration within an IANA registry.

The `<dataSet:type>` element formally defines the DSF type, which indicates the expected set of fields defined under the `<dataSet:fields>` element and the expected operation to be taken. An OPTIONAL "subType" attribute defines the sub-type that is used to further refine the purpose of the DSF.

It is recommended that the server follows a consistent naming scheme for the `<dataSet:type>` values. One option is to delimit the value with a '.' and to include the object ("domain", "host", "contact", "verificationCode", etc.), the operation ("create", "renew", "transfer", "delete", "update", etc.), and the scoped name of the DSF. For example, to support the bulk update of encoded signed verification codes, the `<dataSet:type>` value can be set to "verificationCode.update.encodedSignedCode". To further sub-divide the "verificationCode.update.encodedSignedCode" type by locality, the "subType" attribute can be set to the locality name. For example,

the "china" verification profile supports two verification codes of type "domain" and "real-name", so the "verificationCode.update.encodedSignedCode" DSF type can include "china" for the "subType" attribute, as in `<dataSet:type subType="china">verificationCode.update.encodedSignedCode</dataSet:type>`.

[2.4.](#) Fields

The `<dataSet:fields>` element, an element in the header ([Section 3.1](#)), contains an ordered list of CSV fields used in the body of the file delimited by the character defined by the OPTIONAL "sep" attribute, with a default value of ",". A field child element substitutes for the `<dataSet:field>` abstract element. Each element defines the format of the CSV field contained in the body. The `<dataSet:field>` elements support the "type" attribute that defines the XML schema

simple type of the field element.

The abstract `<dataSet:field>` element does not directly define any attributes, but there are a couple attributes that a data set processor SHOULD support including:

"isRequired": When the "isRequired" attribute is "true", it indicates that the field MUST be non-empty in the body and when set to "false", it indicates that the field MAY be empty in the body. The "isRequired" attribute MAY be specifically set for the field elements in the XML schema and MAY be overridden by specifying the attribute in the fields under the `<dataSet:fields>` element. If no "isRequired" attribute is defined, the default value is "false".

"isPrimaryKey": When the "isPrimaryKey" attribute is "true", it indicates that the field is part of the primary key of the records. The combination of the primary key fields MUST be unique across the set of records in the file. The "isPrimaryKey" attribute MAY be specifically set for the field elements in the XML schema and MAY be overridden by specifying the attribute in the fields under the `<dataSet:fields>` element. If no "isPrimaryKey" attribute is defined, the default value is "false".

[2.4.1.](#) Base Field Types

New field types can be defined that reside in the CSV records. To make the definition of new field types easier, a set of base field types are defined in the `dataSet-1.0` XML schema that include:

"dataSet:fieldOptionalType": The "dataSet:fieldOptionalType" type can be extended by a field that can be empty ("isRequired" =

"false") and is not a member of the primary key ("isPrimaryKey" = "false").

"dataSet:fieldRequiredType": The "dataSet:fieldRequiredType" type can be extended by a field that cannot be empty ("isRequired" = "true") and is not a member of the primary key ("isPrimaryKey" = "false").

"dataSet:fieldPrimaryKeyType": The "dataSet:fieldPrimaryKeyType" type can be extended by a field that cannot be empty

("isRequired" = "true") and is a member of the primary key ("isPrimaryKey" = "true").

"dataSet:fListItemType": The "dataSet:fListItemType" type is an extension of the "dataSet:fieldOptionalType" type that adds an OPTIONAL "op" attribute that reflects the operation to apply for the list item. The default "op" value is "replace". The list of possible "op" values is include below. A "dataSet:fListItemType" field with "op" set to "replace" MUST NOT be mixed with a matching "dataSet:fListItemType" field with "op" set to either "add" or "remove".

"replace": Specifies that the list item will replace what is currently set for the object. The list of items provided will replace the existing list of items set on the object. The concrete "dataSet:fListItemType" field along with the DSF type can determine what is the possible set of list item values.

"add": Specifies that the list item will be added to the object if it is not already set.

"rem": Specifies that the list item will be removed from the object and expects it to already be set.

Example definition of the <dataSet:fName> field element that extends the "dataSet:fieldPrimaryKeyType" base field type with the XML type dataSet:labelType and with the required "class" attribute:

```
<element name="fName" type="dataSet:fNameType"
  substitutionGroup="dataSet:field"/>

<complexType name="fNameType">
  <complexContent>
    <extension base="dataSet:fieldPrimaryKeyType">
      <sequence/>
      <attribute name="type" type="token"
        default="dataSet\:labelType"/>
      <attribute name="class" type="token"
        use="required"/>
    </extension>
  </complexContent>
</complexType>
```

The dataSet-1.0 XML schema predefines a set of field elements that can be used as child elements of the <dataSet:fields> element of the header to define the CSV record fields. The dataSet-1.0 field elements with the XML schema type value and the base type include:

<dataSet:fName>: Field that represents the name of an object. This field extends a "dataSet:fieldPrimaryKeyType", as defined in [Section 2.4.1](#), with the type="token". The required "class" attribute defines the class of the object, like the use of "domain" for a domain name. It is recommended to use an object specific name field if available, like <dsfDomain:fName>.

<dataSet:fAuthInfo>: Object authorization information with type="eppcom:pwAuthInfoType".

<dataSet:fResultCode>: Field that represents the result of processing an individual record using the codes defined in [Section 5](#). The field extends a "dataSet:fieldRequiredType", as defined in [Section 2.4.1](#), with the type="dataSet:resultCodeType".

<dataSet:fResultMsg>: Field containing the human-readable description of the result code. The field extends a "dataSet:fieldOptionalType", as defined in [Section 2.4.1](#), with the type="normalizedString". The field includes the OPTIONAL "lang" attribute with the default value of "en" (English).

<dataSet:fResultReason>: Field containing the human-readable message that describes the reason for the error processing the record. The field extends the "dataSet:fieldOptionalType", as defined in [Section 2.4.1](#), with the type="normalizedString". The language of the reason is identified the OPTIONAL "lang" attribute. If not specified, the default attribute value is "en" (English).

Example <dataSet:fields> definition for a result file generated by a server that includes a domain name with a result code, with an OPTIONAL result message, and an OPTIONAL result reason:

```
<dataSet:fields>
  <dataSet:fName class="domain"/>
  <dataSet:fResultCode/>
  <dataSet:fResultMsg/>
  <dataSet:fResultReason/>
</dataSet:fields>
```

[2.4.2.1](#). Field Element Extensibility

New field elements MAY be defined in separate XML schemas and referenced as child elements of the <dataSet:fields> element. The convention is to prefix all field elements with a lowercase "f", as in <dataSet:fName> for the object name or <dataSet:fResultCode> for

the result code. The following are the steps to define a new field element:

1. Define a new XML schema or extend an existing XML schema to include the definition of the field element. The XML schema will have its own XML namespace that will be referenced in the header of the Data Set File (DSF).
2. Define a new XML schema complexType that is a direct or indirect extension of the "dataSet:fieldType" complexType. The new complexType MUST directly, or indirectly define through extension, the following attributes:

"type": Defines the XML schema data type of the CSV field. The "default" value of the attribute is used to define the default CSV field type using an XML schema simple data type. Any XML namespaces MUST be escaped ("\" instead of ":") so the XML parser of the header can ignore the reference during XML validation. The "type" attribute is used during CSV validation leveraging the XML schema simple data type to define the format. The "type" attribute can be overridden when referencing the field element in the header. The "type" attribute for the "dataSet:fResultCodeType" complexType is defined as `<attribute name="type" default="dataSet\\:resultCodeType">`. When using the default XML namespace, no escaping is necessary. For example for the "dataSet:fResultMsg" complexType, the "type" attribute is defined as `<attribute name="type" default="normalizedString"/>`. A validator of the Data Set File (DSF) will apply the XML schema type to the CSV fields.

"isRequired": See the definition of the "isRequired" attribute in [Section 2.4](#). A required field can extend the "dataSet:fieldRequiredType" complexType, as defined in [Section 2.4.1](#), or can explicitly specify the attribute. For example, the "isRequired" attribute is set in the "dataSet:fieldRequiredType" complexType as `<attribute name="isRequired" default="true">`.

"isPrimaryKey": See the definition of the "isPrimaryKey" attribute in [Section 2.4](#). A primary key field can extend the "dataSet:primaryKeyType" complexType, as defined in [Section 2.4.1](#), or can explicitly specify the attribute. For example, the "isPrimaryKey" attribute is set in the "dataSet:fieldPrimaryKeyType" complexType as `<attribute name="isPrimaryKey" default="true"/>`.

3. Define the field element that uses the new complexType defined above and that substitutes for the "dataSet:field" abstract element. An example of defining the "dataSet:fResultCode" field

element is `<element name="fResultCode" type="dataSet:fResultCodeType" substitutionGroup="dataSet:field"/>`.

An example of defining the "dataSet:fResultReason" field element is `<element name="fResultReason" type="dataSet:fResultMsgType" substitutionGroup="dataSet:field"/>`.

Example definition of the `<dataSet:fResultCode>` field element that is required to be non-empty and with the CSV field type "dataSet:resultCodeType":

```
<element name="fResultCode"
  type="dataSet:fResultCodeType"
  substitutionGroup="dataSet:field"/>

<complexType name="fResultCodeType">
  <complexContent>
    <extension base="dataSet:fieldRequiredType">
      <sequence/>
      <attribute name="type" type="token"
        default="dataSet\:resultCodeType"/>
    </extension>
  </complexContent>
</complexType>
```

Example definition of the `<dataSet:fResultMsg>` field element that may be empty (optional), with the CSV field type "normalizedString", and with the additional optional "lang" attribute:

```
<element name="fResultMsg"
  type="dataSet:fResultMsgType"
  substitutionGroup="dataSet:field"/>

<complexType name="fResultMsgType">
  <complexContent>
    <extension base="dataSet:fieldOptionalType">
      <sequence/>
      <attribute name="type" type="token"
        default="normalizedString"/>
      <attribute name="lang" type="language"
        default="en"/>
    </extension>
```

```
</complexContent>
</complexType>
```

[2.5.](#) Routing

A Data Set File (DSF) MAY include data that targets multiple independent backend services, but the object alone cannot be used to determine the appropriate backend service. An example is creating a Contact Object ([Section 4.3](#)) in the .EXAMPLE1 domain registry

Gould

Expires March 9, 2019

[Page 10]

Internet-Draft

Data Set File Format

September 2018

service and creating another Contact Object ([Section 4.3](#)) in the .EXAMPLE2 domain registry service within a single DSF, where .EXAMPLE1 and .EXAMPLE2 are independent domain registry services. The Data Set File (DSF) processor MAY coordinate with the backend services to process each of the DSF records, but it is dependent on the client to specify the target backend service with each record.

The Data Set File (DSF) field element used for routing a record to a specific backend service includes:

`<dsfRouting:fSubProduct>`: Contains the unique sub-product name of a backend service with type="token". It is up to server policy on the valid list of sub-product values. An example of a sub-product value is the Top Level Domain (TLD) of a domain registry service.

Routing is most applicable to objects like a Contact Object ([Section 4.3](#)) and a Host Object ([Section 4.2](#)), where the keys (`<dsfContact:fId>` for the Contact Object and `<dsfHost:fName>` for the Host Object) MAY NOT be enough to uniquely identify a target domain registry service. An example of using the `<dsfRouting:fSubProduct>` field for routing is provided with the "contact.create.routing" DSF in [Section 4.3](#).

[3.](#) Data Set File (DSF) Format

The Data Set File (DSF) format supports multiple types of requests and response files. All of these types of files support a general file format that includes a header that provides meta-data about the data including the what, who, when, and optionally the digital signature of the information, and the body containing the data. The Data Set File (DSF) syntax is specified using Augmented Backus-Naur

Form (ABNF) grammar [[RFC5234](#)] as follows:

Data Set File (DSF) ABNF

```
file           = header body
header         = dataSet-1-0 LF
dataSet-1-0    = 1*(1*VCHAR LF) ; compliant to dataSet-1.0
body          = start [data] end
start         = "-----BEGIN DATA SET-----" LF
data          = 1*VCHAR LF ; CSV compliant <dataSet:fields>
end           = "-----END DATA SET-----" *1LF
```

Gould

Expires March 9, 2019

[Page 11]

Internet-Draft

Data Set File Format

September 2018

[3.1.](#) Header Format

The header of the Data Set File (DSF) is an XML document that complies with the dataSet-1.0 XML schema. The purpose of the header is to provide the meta-data about the data contained in the body. The `<dataSet:definition>` element is the root XML element of the header and contains the following child element:

`<dataSet:defData>` or `<dataSet:encodedSignedDefData>` or `<dataSet:resultData>`:

- `<dataSet:defData>`: Provides meta-data about the body of the file that is not digitally signed, as described in [Section 3.1.1](#).
- `<dataSet:encodedSignedDefData>`: Contains the encoded form of the digitally signed `<dataSet:signedDefData>` element, as described in [Section 3.1.2](#).
- `<dataSet:resultData>`: Provides the high level result data with the processing of a request Data Set File (DSF) by the server, as described in [Section 3.1.3](#).

[3.1.1.](#) `<dataSet:defData>` element

The `<dataSet:defData>` element contains the meta-data of the body of the file that is not digitally signed. The `<dataSet:defData>` element contains the following child elements:

<dataSet:type>: The type value and OPTIONAL "subType" attribute value of the Data Set File (DSF), as defined in [Section 2.3](#).

<dataSet:fields>: Ordered list of CSV fields as defined in [Section 2.4](#).

<dataSet:dataSetId>: OPTIONAL unique identifier of the Data Set File (DSF). The client SHOULD assign a unique identifier when generating the Data Set File (DSF) and set it in the <dataSet:dataSetId> element.

<dataSet:crDate>: Contains the date and time of the Data Set File (DSF) creation.

Example <dataSet:defData> element for setting verification codes on a domain name:

```
<dataSet:defData>
  <dataSet:fields>
    <dataSet:type subType="locality">
      verificationCode.update.encodedSignedCode
    </dataSet:type>
    <dsfDomain:fName/>
    <dsfVerificationCode:fEncodedSignedCode type="domain"/>
    <dsfVerificationCode:fEncodedSignedCode type="real-name"/>
  </dataSet:fields>
  <dataSet:dataSetId>abc-123</dataSet:dataSetId>
  <dataSet:crDate>2016-04-03T22:00:00.0Z</dataSet:crDate>
</dataSet:defData>
```

[3.1.2](#). <dataSet:encodedSignedDefData> element

The `<dataSet:encodedSignedDefData>` element contains the encoded form of the digitally signed `<dataSet:signedDefData>` element, described in [Section 3.1.2.1](#), with the encoding defined by the "encoding" attribute with the default "encoding" value of "base64". The "base64" encoded text of the `<dataSet:signedDefData>` element MUST conform to [\[RFC2045\]](#).

Example `<dataSet:encodedSignedDefData>` element, where "..." represents continuation of data for brevity:

```
<dataSet:encodedSignedDefData encoding="base64">
  ICAgICAgPHZlcmlmaWNhdGlvbKNvZGU6c2lnbmVkJQ29kZQogICAgICAgIHhtbG5z
  ...
  b25Db2RlOnNpZ25lZENvZGU+Cg==
</dataSet:encodedSignedDefData>
```

[3.1.2.1](#). Signed Definition Data

The Signed Definition Data, represented by the `<dataSet:signedDefData>` element, is a signed extension of the `<dataSet:defData>` element, defined in [Section 3.1.1](#). The `<dataSet:signedDefData>` provides the meta-data about the body of the file, that is a fragment of XML that is digitally signed using XML Signature [\[W3C.CR-xmlsig-core2-20120124\]](#). The `<dataSet:signedDefData>` element includes a required "id" attribute of type XSD ID for use with the IDREF URI from the Signature element. Setting the "id" attribute value to "signedData" is recommended. The certificate of the issuer MUST be included with the Signature so it can be chained with the issuer's certificate by the validating client. A checksum, as defined by the [Section 2.2](#), of the body of

the file, as defined by the body rule of the Data Set File (DSF) ABNF, is included in the digitally signed data to ensure that it's covered by the Signature. The `<dataSet:signedDefData>` element contains the following child elements:

`<dataSet:type>`: The type value and OPTIONAL "subType" attribute value of the Data Set File (DSF), as defined in [Section 2.3](#).

`<dataSet:fields>`: Ordered list of CSV fields as defined in [Section 2.4](#).

`<dataSet:dataSetId>` OPTIONAL unique identifier of the data set. The source SHOULD assign a unique identifier when generating the data

set and set it in the <dataSet:dataSetId> element.

<dataSet:crDate>: Contains the date and time of the data set creation.

<dataSet:cksum>: Checksum of the body of the file, as defined by the body rule of the Data Set File (DSF) ABNF, using the mechanism defined by the [Section 2.2](#).

<Signature>: XML Signature [[W3C.CR-xmlsig-core2-20120124](#)] for the <dataSet:signedDefData>. Use of a namespace prefix, like "dsig", is recommended for the XML Signature [[W3C.CR-xmlsig-core2-20120124](#)] elements.

Example <dataSet:signedDefData> element, where "... " represents continuation of data for brevity:

```
<dataSet:signedDefData
  xmlns:dataSet=
    "urn:ietf:params:xml:ns:dataSet-1.0"
  xmlns:dsfDomain=
    "urn:ietf:params:xml:ns:dsfDomain-1.0"
  xmlns:dsfVerificationCode=
    "urn:ietf:params:xml:ns:dsfVerificationCode-1.0"
  id="signedData">
  <dataSet:type subType="locality">
    verificationCode.update.encodedSignedCode
  </dataSet:type>
  <dataSet:fields>
    <dsfDomain:fName/>
    <dsfVerificationCode:fCode type="domain"/>
    <dsfVerificationCode:fCode type="real-name"/>
  </dataSet:fields>
    <dataSet:dataSetId>abc-123</dataSet:dataSetId>
    <dataSet:crDate>2016-04-03T22:00:00.0Z</dataSet:crDate>
    <dataSet:cksum>6F2B988F</dataSet:cksum>
  <Signature xmlns="http://www.w3.org/2000/09/xmlsig#">
    <SignedInfo>
      <CanonicalizationMethod
        Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
```

```
    <SignatureMethod
      Algorithm="http://www.w3.org/2001/04/xmlsig-more#rsa-sha256"/>
    <Reference URI="#signedData">
      <Transforms>
```

```

    <Transform
Algorithm="http://www.w3.org/2000/09/xmlsig#enveloped-signature"/>
    </Transforms>
    <DigestMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
    <DigestValue>wgyW3nZPoEfptlhRILKn0QnbdU6ArM7ShrAfHgDFg=
    </DigestValue>
    </Reference>
    </SignedInfo>
    <SignatureValue>
jMu4PfyQGijBF0GWSEPFJCjmywCEqR2h4LD+ge6XQ+JnmKFFCuCZS/3SLKAx0L1w
...
ipJsXNa6osTUw1CzA7jfwA==
    </SignatureValue>
    <KeyInfo>
    <X509Data>
    <X509Certificate>
MIIESTCCAzGgAwIBAgIBAjANBgkqhkiG9w0BAQsFADBimQswCQYDVQQGEwJVUzEL
...
AdXitTWFipaIGea9lEGFM0L9+Bg7XzNn4nVLXokyEB3bgS4scG6QznX23FGk
    </X509Certificate>
    </X509Data>
    </KeyInfo>
    </Signature>
</dataSet:signedDefData>

```

[3.1.3.](#) <dataSet:resultData> element

The <dataSet:resultData> element contains the high level result data with the processing of a request Data Set File (DSF) by the server. the "code" attribute describes the success or failure of the processing using the code values defined in [Section 5](#). The <dataSet:resultData> element contains the following child elements:

- <dataSet:type>: OPTIONAL type value and OPTIONAL "subType" attribute value associated with the request Data Set File (DSF), as defined in [Section 2.3](#). The <dataSet:type> element MUST be returned if the request header is successfully parsed.
- <dataSet:fields>: OPTIONAL ordered list of CSV fields as defined in [Section 2.4](#). If the <dataSet:fields> element is not defined, then the data as defined by the data rule of the Data Set File (DSF) ABNF MUST be empty.
- <dataSet:dataSetId>: OPTIONAL identifier of the Data Set File (DSF) formed using the <dataSet:dataSetId> element associated with the

request if supplied by the client and the request header is successfully parsed.

<dataSet:msg>: Human-readable description of the result code. The language of the result is identified via an OPTIONAL "lang" attribute. If not specified, the default attribute value is "en" (English).

<dataSet:reason>: OPTIONAL human-readable message that describes the reason for the error. The language of the reason is identified via an OPTIONAL "lang" attribute. If not specified, the default attribute value is "en" (English).

<dataSet:records>: OPTIONAL summary record result data. The summary data is associated with the count of the records (lines in request data). The <dataSet:records> element contains the following child elements:

<dataSet:total>: Total count of the records (lines in request data) processed by the server.

<dataSet:success>: Total count of the records (lines in request data) successfully processed by the server.

<dataSet:failed>: Total count of the records (lines in request data) that failed processing by the server.

Example successful <dataSet:resultData> element:

```
<dataSet:resultData code="1000">
  <dataSet:type subType="locality">
    verificationCode.update.encodedSignedCode
  </dataSet:type>
  <dataSet:fields>
    <dsfDomain:fName/>
    <dataSet:fResultCode/>
    <dataSet:fResultMsg/>
    <dataSet:fResultReason/>
  </dataSet:fields>
  <dataSet:dataSetId>abc-123</dataSet:dataSetId>
  <dataSet:svTRID>54322-XYZ</dataSet:svTRID>
  <dataSet:msg>Code Set processed successfully.</dataSet:msg>
  <dataSet:records>
    <dataSet:total>2</dataSet:total>
    <dataSet:success>2</dataSet:success>
    <dataSet:failed>0</dataSet:failed>
  </dataSet:records>
</dataSet:resultData>
```

Example `<dataSet:resultData>` element with some failures:

```
<dataSet:resultData code="1001">
  <dataSet:type subType="locality">
    verificationCode.update.encodedSignedCode
  </dataSet:type>
  <dataSet:fields>
    <dsfDomain:fName/>
    <dataSet:fResultCode/>
    <dataSet:fResultMsg/>
    <dataSet:fResultReason/>
  </dataSet:fields>
  <dataSet:dataSetId>abc-123</dataSet:dataSetId>
  <dataSet:svTRID>54322-XYZ</dataSet:svTRID>
  <dataSet:msg>Success with failures</dataSet:msg>
  <dataSet:records>
    <dataSet:total>4</dataSet:total>
    <dataSet:success>1</dataSet:success>
    <dataSet:failed>3</dataSet:failed>
  </dataSet:records>
</dataSet:resultData>
```

Example failed `<dataSet:resultData>` element based on malformed request file:

```
<dataSet:resultData code="2000">
  <dataSet:svTRID>54322-XYZ</dataSet:svTRID>
  <dataSet:msg>File syntax error</dataSet:msg>
  <dataSet:reason lang="en">Malformed request file
</dataSet:reason>
</dataSet:resultData>
```

Example failed `<dataSet:resultData>` element based on header syntax error:

```
<dataSet:resultData code="2001">
  <dataSet:svTRID>54322-XYZ</dataSet:svTRID>
  <dataSet:msg>Header syntax error</dataSet:msg>
  <dataSet:reason lang="en">Header XML syntax error
</dataSet:reason>
```

</dataSet:resultData>

Example failed <dataSet:resultData> element based on body syntax error:

```
<dataSet:resultData code="2002">
  <dataSet:type subType="locality">
    verificationCode.update.encodedSignedCode
  </dataSet:type>
  <dataSet:dataSetId>abc-123</dataSet:dataSetId>
  <dataSet:svTRID>54322-XYZ</dataSet:svTRID>
  <dataSet:msg>Body syntax error</dataSet:msg>
  <dataSet:reason lang="en">Invalid with fields definition
</dataSet:reason>
</dataSet:resultData>
```

[3.2.](#) Body Format

The body of the Data Set File (DSF) is a set of Comma-Separated Values (CSV) data that includes a leading "-----BEGIN CODE SET-----" line and a trailing "-----END CODE SET-----" line. The format of the CSV data is defined by the <dataSet:fields> element in the header ([Section 3.1](#)), that includes the delimiter and the ordered set of fields ([Section 2.4](#)) with their XML simple type and descriptor attributes like "isRequired", "isPrimaryKey", "class", and "codeType".

Example body for a result file generated by a server that includes a success and a set of failed records with a mix of messages and reasons.:

```
-----BEGIN DATA SET-----
domain1.example,1000,Success,
domain2.example,2303,Object does not exist,
domain3.example,2201,Authorization error,
```

domain4.example,2302,Object exists,domain code already
-----END DATA SET-----

4. Object Description

This section describes the base objects supported by this specification:

4.1. Domain Name Object

The domain name object is based on the EPP domain name mapping specified in [[RFC5731](#)].

The Data Set File (DSF) field elements specific to the domain name object include:

Gould Expires March 9, 2019 [Page 18]

Internet-Draft Data Set File Format September 2018

<dsfDomain:fName>: Domain name field with type="eppcom:labelType" and isRequired="true".
<dsfDomain:fPeriod>: Domain name initial or extension period of the expiration date with type="domain:pLimitType".
<dsfDomain:fPeriodUnit>: Domain name initial or extension period unit of the expiration date with type="domain:pUnitType".
<dsfDomain:fNs>: Domain name server is an extension of the "dataSet:fListItemType" field, described in [Section 2.4.1](#), with type="eppcom:labelType".
<dsfDomain:fContact>: Domain contact identifier with type="eppcom:clIDType" and with required "role" attribute. The possible values of the "role" attribute include "registrant", "admin", "tech", and "billing".
<dsfDomain:fStatus>: The status of the domain name is an extension of the "dataSet:fListItemType" field, described in [Section 2.4.1](#), with type="domain:statusValueType".
<dsfDomain:fKeyTag>: Contains the DS key tag value per [[RFC5910](#)] with type="unsignedShort".
<dsfDomain:fDsAlg>: Contains the DS algorithm value per [[RFC5910](#)] with type="unsignedByte".
<dsfDomain:fDigestType>: Contains the DS digest type value per [[RFC5910](#)] with type="unsignedByte".
<dsfDomain:fDigest>: Contains the DS digest value per [[RFC5910](#)] with type="hexBinary".
<dsfDomain:fFlags>: Contains the flags field value per [[RFC5910](#)] with type="unsignedShort".

<dsfDomain:fProtocol>: Contains the Key protocol value per [[RFC5910](#)] with type="unsignedByte".
<dsfDomain:fKeyAlg>: Contains the Key algorithm value per [[RFC5910](#)] with type="unsignedByte".
<dsfDomain:fPubKey>: Contains the public key value per [[RFC5910](#)] with type="secDNS:keyType".
<dsfDomain:fMaxSigLife>: Indicates a child's preference for the number of seconds after signature generation when the parent's signature on the DS information provided by the child will expire with type="secDNS:maxSigLifeType" defined in [[RFC5910](#)].

The following are example DSF files using the domain object fields. The different forms of domain object DSF files is up to server policy.

Example of a "domain.update.replaceClientStatuses" DSF that will set the client statuses of a domain name to those specified in the DSF record. The client statuses set will be replaced by the set of client statuses specified. Empty statuses will result in the removal of those statuses if previously set. Non-empty statuses will be added if not previously set. Non-empty statuses will result in no change if previously set. All five client statuses defined in [[RFC5731](#)] are defined in the DSF in fixed order.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<dataSet:definition
  xmlns:dataSet=
    "urn:ietf:params:xml:ns:dataSet-1.0"
  xmlns:dsfDomain=
    "urn:ietf:params:xml:ns:dsfDomain-1.0">
  <dataSet:defData>
    <dataSet:type>
      domain.update.replaceClientStatuses
    </dataSet:type>
```

```

<dataSet:fields>
  <dsfDomain:fName/>
  <!-- clientDeleteProhibited -->
  <dsfDomain:fStatus/>
  <!-- clientHold -->
  <dsfDomain:fStatus/>
  <!-- clientRenewProhibited -->
  <dsfDomain:fStatus/>
  <!-- clientTransferProhibited -->
  <dsfDomain:fStatus/>
  <!-- clientUpdateProhibited -->
  <dsfDomain:fStatus/>
</dataSet:fields>
<dataSet:dataSetId>abc-123</dataSet:dataSetId>
<dataSet:crDate>2016-04-03T22:00:00.0Z</dataSet:crDate>
</dataSet:defData>
</dataSet:definition>
-----BEGIN DATA SET-----
domain1.example,clientDeleteProhibited,,,clientUpdateProhibited
domain2.example,,,clientHold,,,
domain3.example,,,clientRenewProhibited,clientTransferProhibited,
domain4.example,,,,,
-----END DATA SET-----

```

Example of a "domain.update.addRemoveStatus" DSF that will add a status and remove a status for each domain name. Any status can be empty to indicate no action for that domain name.

```

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<dataSet:definition
  xmlns:dataSet=
    "urn:ietf:params:xml:ns:dataSet-1.0"
  xmlns:dsfDomain=
    "urn:ietf:params:xml:ns:dsfDomain-1.0">
  <dataSet:defData>
    <dataSet:type>

```

```

        domain.update.addRemoveNs
    </dataSet:type>
    <dataSet:fields>
        <dsfDomain:fName/>
        <dsfDomain:fNs op="add"/>
        <dsfDomain:fNs op="remove"/>
    </dataSet:fields>
    <dataSet:dataSetId>abc-123</dataSet:dataSetId>
    <dataSet:crDate>2016-04-03T22:00:00.0Z</dataSet:crDate>
</dataSet:defData>
</dataSet:definition>
-----BEGIN DATA SET-----
domain1.example,ns1.domain1.example,n2.domain1.example
domain2.example,ns1.domain1.example,
domain3.example,,ns2.domain1.example
-----END DATA SET-----

```

Example of a "domain.update.replaceNs" DSF that will set the name servers of a domain name to those specified in the DSF record. The server supports setting up to 13 name servers to a domain name, so there are 13 <dsfDomain:fNs> defined. All existing name servers set will be replaced with the name servers specified.

```

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<dataSet:definition
  xmlns:dataSet=
    "urn:ietf:params:xml:ns:dataSet-1.0"
  xmlns:dsfDomain=
    "urn:ietf:params:xml:ns:dsfDomain-1.0">
  <dataSet:defData>
    <dataSet:type>
      domain.update.replaceNs
    </dataSet:type>
    <dataSet:fields>
      <dsfDomain:fName/>
      <dsfDomain:fNs/>
      <dsfDomain:fNs/>
      <dsfDomain:fNs/>
      <dsfDomain:fNs/>
      <dsfDomain:fNs/>
      <dsfDomain:fNs/>
      <dsfDomain:fNs/>
      <dsfDomain:fNs/>
      <dsfDomain:fNs/>
      <dsfDomain:fNs/>
      <dsfDomain:fNs/>
      <dsfDomain:fNs/>
      <dsfDomain:fNs/>
    </dataSet:fields>
    <dataSet:dataSetId>abc-123</dataSet:dataSetId>
    <dataSet:crDate>2016-04-03T22:00:00.0Z</dataSet:crDate>
  </dataSet:defData>
</dataSet:definition>
-----BEGIN DATA SET-----
domain1.example,ns1.domain1.example,n2.domain1.example,,,,,,,,,
domain2.example,ns1.domain1.example,,,,,,,,,
domain3.example,,,,,,,,,
-----END DATA SET-----

```

Example of a "domain.update.addRemoveNs" DSF that will add a name server and remove a name server for each domain name. Any name server can be empty to indicate no action for that domain name.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<dataSet:definition
  xmlns:dataSet=
    "urn:ietf:params:xml:ns:dataSet-1.0"
  xmlns:dsfDomain=
    "urn:ietf:params:xml:ns:dsfDomain-1.0">
  <dataSet:defData>
    <dataSet:type>
      domain.update.addRemoveNs
    </dataSet:type>
    <dataSet:fields>
      <dsfDomain:fName/>
      <dsfDomain:fNs op="add"/>
      <dsfDomain:fNs op="remove"/>
    </dataSet:fields>
    <dataSet:dataSetId>abc-123</dataSet:dataSetId>
    <dataSet:crDate>2016-04-03T22:00:00.0Z</dataSet:crDate>
  </dataSet:defData>
</dataSet:definition>
-----BEGIN DATA SET-----
domain1.example,ns1.domain1.example,n2.domain1.example
domain2.example,ns1.domain1.example,
domain3.example,,ns2.domain1.example
-----END DATA SET-----
```

Example of a "domain.update.contacts" DSF that supports updating the contacts (registrant, admin, tech, and billing) for each domain name. All four contacts fields are set as required using the "isRequired" attribute and any existing contacts set for each domain name are replaced.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<dataSet:definition
  xmlns:dataSet=
    "urn:ietf:params:xml:ns:dataSet-1.0"
  xmlns:dsfDomain=
    "urn:ietf:params:xml:ns:dsfDomain-1.0">
  <dataSet:defData>
    <dataSet:type>
      domain.update.contacts
    </dataSet:type>
    <dataSet:fields>
      <dsfDomain:fName/>
      <dsfDomain:fContact role="registrant" isRequired="true"/>
      <dsfDomain:fContact role="admin" isRequired="true"/>
      <dsfDomain:fContact role="tech" isRequired="true"/>
      <dsfDomain:fContact role="billing" isRequired="false"/>
    </dataSet:fields>
    <dataSet:dataSetId>abc-123</dataSet:dataSetId>
    <dataSet:crDate>2016-04-03T22:00:00.0Z</dataSet:crDate>
  </dataSet:defData>
</dataSet:definition>
-----BEGIN DATA SET-----
domain1.example,jd1234,sh813,sh813,sh813
domain2.example,jd1234,sh813,sh813,
-----END DATA SET-----
```

Example of a "domain.create.standard" DSF that supports creating each domain name record with a standard set of fields / attributes for a thick domain name.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<dataSet:definition
  xmlns:dataSet=
    "urn:ietf:params:xml:ns:dataSet-1.0"
  xmlns:dsfDomain=
    "urn:ietf:params:xml:ns:dsfDomain-1.0">
  <dataSet:defData>
    <dataSet:type>
      domain.create.standard
    </dataSet:type>
    <dataSet:fields>
      <dsfDomain:fName/>
      <dsfDomain:fPeriod/>
      <dsfDomain:fNs/>
      <dsfDomain:fNs/>
      <dsfDomain:fContact role="registrant"/>
      <dsfDomain:fContact role="admin"/>
      <dsfDomain:fContact role="tech"/>
      <dsfDomain:fContact role="billing"/>
      <dataSet:fAuthInfo/>
    </dataSet:fields>
    <dataSet:dataSetId>abc-123</dataSet:dataSetId>
    <dataSet:crDate>2016-04-03T22:00:00.0Z</dataSet:crDate>
  </dataSet:defData>
</dataSet:definition>
-----BEGIN DATA SET-----
domain1.example,1,ns1.dom.example,,jd1234,sh813,sh813,sh813,2fooBAR
domain2.example,1,,,jd1234,sh813,sh813,sh813,2fooBAR
-----END DATA SET-----
```

[4.2.](#) Host Object

The host object is based on the EPP host mapping specified in [\[RFC5732\]](#).

The Data Set File (DSF) field elements specific to the host object include:

<dsfHost:fName>: Host name field with type="eppcom:labelType" and
isRequired="true".
<dsfHost:fNewName>: Host new name field used to rename the host with
type="eppcom:labelType".
<dsfHost:fAddrVersion>: The address version ("v4" or "v6") is an
extension of the "dataSet:fListItemType" field, described in

[Section 2.4.1](#), with type="eppcom:addrStringType". The
"dsfHost.fAddrVersion" field usually corresponds with a following
"dsfHost:fAddr" field. The "dsfHost:fAddrVersion" and
"dsfHost.fAddr" fields are included in pairs.
<dsfHost:fAddr>: The address is an extension of the
"dataSet:fListItemType" field, described in [Section 2.4.1](#), with
type="eppcom:addrStringType". The "dsfHost.fAddr" field usually
corresponds with a previous "dsfHost:fAddrVersion" field that
defines the type of address. The "dsfHost:fAddrVersion" and
"dsfHost.fAddr" fields are included in pairs.
<dsfHost:fStatus>: The status of the host is an extension of the
"dataSet:fListItemType" field, described in [Section 2.4.1](#), with
type="host:statusValueType".

The following are example DSF files using the host object fields.
The different forms of host object DSF files is up to server policy.

Example of a "host.update.replaceClientStatuses" DSF that will set the client statuses of a host to those specified in the DSF record. The client statuses set will be replaced by the set of client statuses specified. Empty statuses will result in the removal of those statuses if previously set. Non-empty statuses will be added if not previously set. Non-empty statuses will result in no change if previously set. All two client statuses defined in [\[RFC5732\]](#) are defined in the DSF in fixed order.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<dataSet:definition
  xmlns:dataSet=
    "urn:ietf:params:xml:ns:dataSet-1.0"
  xmlns:dsfHost=
    "urn:ietf:params:xml:ns:dsfHost-1.0">
  <dataSet:defData>
    <dataSet:type>
      host.update.replaceClientStatuses
    </dataSet:type>
    <dataSet:fields>
      <dsfHost:fName/>
```

```

        <!-- clientDeleteProhibited -->
        <dsfHost:fStatus/>
        <!-- clientUpdateProhibited -->
        <dsfHost:fStatus/>
    </dataSet:fields>
    <dataSet:dataSetId>abc-123</dataSet:dataSetId>
    <dataSet:crDate>2016-04-03T22:00:00.0Z</dataSet:crDate>
</dataSet:defData>
</dataSet:definition>
-----BEGIN DATA SET-----
ns1.domain.example,clientDeleteProhibited,clientUpdateProhibited
ns2.domain.example,,clientUpdateProhibited
ns3.domain.example,clientDeleteProhibited,
ns4.domain.example,,
-----END DATA SET-----

```

Example of a "host.update.addRemoveStatus" DSF that will add a status and remove a status for each host. Any status can be empty to indicate no action for that host.

```

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<dataSet:definition
  xmlns:dataSet=
    "urn:ietf:params:xml:ns:dataSet-1.0"
  xmlns:dsfHost=
    "urn:ietf:params:xml:ns:dsfHost-1.0">
  <dataSet:defData>
    <dataSet:type>
      host.update.addRemoveClientStatuses
    </dataSet:type>
  </dataSet:defData>
</dataSet:definition>

```

```

<dataSet:fields>
  <dsfHost:fName/>
  <dsfHost:fStatus op="add"/>
  <dsfHost:fStatus op="remove"/>
</dataSet:fields>
<dataSet:dataSetId>abc-123</dataSet:dataSetId>
<dataSet:crDate>2016-04-03T22:00:00.0Z</dataSet:crDate>
</dataSet:defData>
</dataSet:definition>
-----BEGIN DATA SET-----
ns1.domain.example,clientDeleteProhibited,clientUpdateProhibited
ns2.domain.example,,clientUpdateProhibited
ns3.domain.example,clientUpdateProhibited,
-----END DATA SET-----

```

Example of a "host.update.replaceAddr" DSF that will set the addresses of a host to those specified in the DSF record. The server supports setting up to 6 addresses to a host, so there are 6 <dsfHost:fAddrVersion> and <dsfHost:fAddr> field pairs defined. All existing addresses set will be replaced with the addresses specified.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
```

```

<dataSet:definition
  xmlns:dataSet=
    "urn:ietf:params:xml:ns:dataSet-1.0"
  xmlns:dsfHost=
    "urn:ietf:params:xml:ns:dsfHost-1.0">
  <dataSet:defData>
    <dataSet:type>
      host.update.replaceAddr
    </dataSet:type>
    <dataSet:fields>
      <dsfHost:fName/>
      <dsfHost:fAddrVersion/>
      <dsfHost:fAddr/>
      <dsfHost:fAddrVersion/>
      <dsfHost:fAddr/>
      <dsfHost:fAddrVersion/>
      <dsfHost:fAddr/>
      <dsfHost:fAddrVersion/>
      <dsfHost:fAddr/>
      <dsfHost:fAddrVersion/>
      <dsfHost:fAddr/>
      <dsfHost:fAddrVersion/>
      <dsfHost:fAddr/>
    </dataSet:fields>
    <dataSet:dataSetId>abc-123</dataSet:dataSetId>
    <dataSet:crDate>2016-04-03T22:00:00.0Z</dataSet:crDate>
  </dataSet:defData>
</dataSet:definition>
-----BEGIN DATA SET-----
ns1.domain.example,v4,192.0.2.2,v4,192.0.2.29,,,,,,,,,
ns2.domain.example,v6,1080:0:0:0:8:800:200C:417A,,,,,,,,,
ns3.domain.example,,,,,,,,,
-----END DATA SET-----

```

Example of a "host.update.addRemoveAddr" DSF that will add an address and remove an address for each host. Any address can be empty to indicate no action for that host. The <dsfHost:fAddrVersion> and <dsfHost:fAddr> field pairs are provided with matching list operations.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<dataSet:definition
  xmlns:dataSet=
    "urn:ietf:params:xml:ns:dataSet-1.0"
  xmlns:dsfHost=
    "urn:ietf:params:xml:ns:dsfHost-1.0">
  <dataSet:defData>
    <dataSet:type>
      host.update.addRemoveAddr
    </dataSet:type>
    <dataSet:fields>
      <dsfHost:fName/>
      <dsfHost:fAddrVersion op="add"/>
      <dsfHost:fAddr op="add"/>
      <dsfHost:fAddrVersion op="remove"/>
      <dsfHost:fAddr op="remove"/>
    </dataSet:fields>
    <dataSet:dataSetId>abc-123</dataSet:dataSetId>
    <dataSet:crDate>2016-04-03T22:00:00.0Z</dataSet:crDate>
  </dataSet:defData>
</dataSet:definition>
-----BEGIN DATA SET-----
ns1.domain.example,v4,192.0.2.2,v4,192.0.2.29
ns2.domain.example,v6,1080:0:0:0:8:800:200C:417A,,
ns3.domain.example,,v4,192.0.2.29
-----END DATA SET-----
```

Example of a "host.create.standard" DSF that supports creating each host record with a standard set of fields / attributes for a host. There is an example of creating an internal host (ns1.domain.example and ns2.domain.example) with up to two addresses and an external host (ns1.domain.external) without addresses.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<dataSet:definition
  xmlns:dataSet=
    "urn:ietf:params:xml:ns:dataSet-1.0"
  xmlns:dsfHost=
    "urn:ietf:params:xml:ns:dsfHost-1.0">
  <dataSet:defData>
    <dataSet:type>
      host.create.standard
    </dataSet:type>
    <dataSet:fields>
      <dsfHost:fName/>
      <dsfHost:fAddrVersion/>
      <dsfHost:fAddr/>
      <dsfHost:fAddrVersion/>
      <dsfHost:fAddr/>
    </dataSet:fields>
    <dataSet:dataSetId>abc-123</dataSet:dataSetId>
    <dataSet:crDate>2016-04-03T22:00:00.0Z</dataSet:crDate>
  </dataSet:defData>
</dataSet:definition>
-----BEGIN DATA SET-----
ns1.domain.example,v4,192.0.2.2,v4,192.0.2.29
ns2.domain.example,v6,1080:0:0:0:8:800:200C:417A,
ns1.domain.external,,,,
-----END DATA SET-----
```

[4.3.](#) Contact Object

The contact object is based on the EPP host mapping specified in [\[RFC5733\]](#).

Some elements MAY be provided in either internationalized form ("int") or provided in localized form ("loc"). Those elements use a field value or "isLoc" attribute to specify the form used. If an "isLoc" attribute is used, a value of "true" indicates the use of the localized form and a value of "false" indicates the use of the

internationalized form. This MAY override the form specified for a parent element. A value of "int" is used to indicate the internationalized form and a value of "loc" is used to indicate the localized form. When the internationalized form ("int") is provided, the field value MUST be represented in a subset of UTF-8 that can be

represented in the 7-bit US-ASCII character set. When the localized form ("loc") is provided, the field value MAY be represented in unrestricted UTF-8. Some of the contact field elements specify the internationalized form with the isLoc="false" attribute.

The Data Set File (DSF) field elements specific to the contact object include:

<dsfContact:fId>: Contains the server-unique contact identifier with type="eppcom:clIDType" and isRequired="true".

<dsfContact:fEmail>: Contains the contact's email address with type="eppcom:minTokenType" and isRequired="true".

<dsfContact:fVoice>: Contains the contact's voice telephone number with type="contact:e164StringType".

<dsfContact:fVoiceExt>: Contains the contact's voice telephone number extension with type="token".

<dsfContact:fFax>: Contains the contact's facsimile telephone number with type="contact:e164StringType".

<dsfContact:fFaxExt>: Contains the contact's facsimile telephone number extension with type="token".

<dsfContact:fName>: Contains the contact's name of the individual or role represented by the contact with type="contact:postalLineType" and isRequired="true". An OPTIONAL "isLoc" attribute to used to indicate the localized or internationalized form.

<dsfContact:fStreet>: Contains the contact's contact's street address line with type="contact:fPostalLineType". An index attribute is required to indicate which street address line the field represents with index "0" for the first line and index "2" for the last line. An OPTIONAL "isLoc" attribute to used to indicate the localized or internationalized form.

<dsfContact:fCity>: Contains the contact's city with type="contact:postalLineType" and isRequired="true". An OPTIONAL "isLoc" attribute to used to indicate the localized or internationalized form.

<dsfContact:fCc>: Contains the contact's country code with

type="contact:ccType" and isRequired="true". An OPTIONAL "isLoc" attribute to used to indicate the localized or internationalized form.

<dsfContact:fPostalType>: Contains the form of the postal-address information with type="contact:postalLineType". This field specifies the form ("int" or "loc") of the <dsfContact:fName>, <dsfContact:fOrg>, <dsfContact:fStreet>, <dsfContact:fCity>, <dsfContact:fSp>, <dsfContact:fPc>, <dsfContact:fCc> fields.

<dsfContact:fOrg>: Contains the name of the organization with which the contact is affiliated with type="contact:optPostalLineType". An OPTIONAL "isLoc" attribute to used to indicate the localized or internationalized form.

<dsfContact:fSp>: Contains the contact's state or province with type="contact:optPostalLineType". An OPTIONAL "isLoc" attribute to used to indicate the localized or internationalized form.

<dsfContact:fPc>: Contains the contact's postal code with type="contact:pcType". An OPTIONAL "isLoc" attribute to used to indicate the localized or internationalized form.

<dsfContact:fDiscloseFlag>: Contains flag with a value of "true" or "1" (one) notes the preference to allow disclosure of the specified elements as an exception to the stated data-collection policy. A value of "false" or "0" (zero) notes a client preference to not allow disclosure of the specified elements as an exception to the stated data-collection policy with type="boolean". The additional fields define specific exceptional disclosure preferences based on the <dsfContact:fDiscloseFlag> field.

<dsfContact:fDiscloseNameLoc>: Exceptional disclosure preference flag for the localized form of the contact name with type="boolean".

<dsfContact:fDiscloseNameInt>: Exceptional disclosure preference flag for the internationalized form of the contact name with type="boolean".

<dsfContact:fDiscloseOrgLoc>: Exceptional disclosure preference flag for the localized form of the contact organization with type="boolean". with type="boolean".

<dsfContact:fDiscloseOrgInt>: Exceptional disclosure preference flag for the internationalized form of the contact organization with type="boolean". with type="boolean".

<dsfContact:fDiscloseAddrLoc>: Exceptional disclosure preference flag for the localized form of the contact address with

type="boolean".

<dsfContact:fDiscloseAddrInt>: Exceptional disclosure preference flag for the internationalized form of the contact address with type="boolean".

<dsfContact:fDiscloseVoice>: Exceptional disclosure preference flag of the contact voice telephone number with type="boolean".

<dsfContact:fDiscloseFax>: Exceptional disclosure preference flag of the contact facsimile telephone number with type="boolean".

<dsfContact:fDiscloseEmail>: Exceptional disclosure preference flag of the contact email address with type="boolean".

<dsfContact:fDiscloseAll>: Exceptional disclosure preference flag of all of the supported disclose fields with type="boolean". This single field represents the <dsfContact:fDiscloseFlag> field value for all of the disclose fields, that include <dsfContact:fDiscloseNameLoc>, <dsfContact:fDiscloseNameInt>, <dsfContact:fDiscloseOrgLoc>, <dsfContact:fDiscloseOrgInt>, <dsfContact:fDiscloseAddrLoc>, <dsfContact:fDiscloseAddrInt>, <dsfContact:fDiscloseVoice>, <dsfContact:fDiscloseFax>, and <dsfContact:fDiscloseEmail>.

The following are example DSF files using the contact object fields. The different forms of contact object DSF files is up to server policy.

Example of a "contact.update.replaceClientStatuses" DSF that will set the client statuses of a contact to those specified in the DSF record. The client statuses set will be replaced by the set of client statuses specified. Empty statuses will result in the removal of those statuses if previously set. Non-empty statuses will be added if not previously set. Non-empty statuses will result in no change if previously set. All three client statuses defined in [\[RFC5733\]](#) are defined in the DSF in fixed order.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<dataSet:definition
  xmlns:dataSet=
    "urn:ietf:params:xml:ns:dataSet-1.0"
  xmlns:dsfContact=
    "urn:ietf:params:xml:ns:dsfContact-1.0">
  <dataSet:defData>
    <dataSet:type>
      contact.update.replaceClientStatuses
```

```

</dataSet:type>
<dataSet:fields>
  <dsfContact:fId/>
  <!-- clientDeleteProhibited -->
  <dsfContact:fStatus/>
  <!-- clientTransferProhibited -->
  <dsfContact:fStatus/>
  <!-- clientUpdateProhibited -->
  <dsfContact:fStatus/>
</dataSet:fields>
<dataSet:dataSetId>abc-123</dataSet:dataSetId>
<dataSet:crDate>2016-04-03T22:00:00.0Z</dataSet:crDate>
</dataSet:defData>
</dataSet:definition>
-----BEGIN DATA SET-----
sh1111,clientDeleteProhibited,,clientUpdateProhibited
sh2222,,clientUpdateProhibited
sh3333,,clientTransferProhibited,
sh4444,,,
-----END DATA SET-----

```

Example of a "contact.update.addRemoveStatus" DSF that will add a status and remove a status for each contact. Any status can be empty to indicate no action for that contact.

```

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<dataSet:definition
  xmlns:dataSet=
    "urn:ietf:params:xml:ns:dataSet-1.0"
  xmlns:dsfContact=
    "urn:ietf:params:xml:ns:dsfContact-1.0">
  <dataSet:defData>
    <dataSet:type>
      contact.update.addRemoveClientStatuses
    </dataSet:type>
    <dataSet:fields>

```

```

    <dsfContact:fId/>
    <dsfContact:fStatus op="add"/>
    <dsfContact:fStatus op="remove"/>
  </dataSet:fields>
  <dataSet:dataSetId>abc-123</dataSet:dataSetId>
  <dataSet:crDate>2016-04-03T22:00:00.0Z</dataSet:crDate>
</dataSet:defData>
</dataSet:definition>
-----BEGIN DATA SET-----
sh1111,clientDeleteProhibited,clientUpdateProhibited
sh2222,,clientUpdateProhibited
sh3333,clientUpdateProhibited,
sh4444,clientTransferProhibited,
-----END DATA SET-----

```

Example of a "contact.create.standard" DSF that supports creating each contact record with a standard set of fields / attributes for a contact. A "|" character is used as a separator to minimize conflicting with the contact data. This is an example of creating two contacts, where indented data set lines are a continuation from the prior line.

```

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<dataSet:definition
  xmlns:dataSet=
    "urn:ietf:params:xml:ns:dataSet-1.0"
  xmlns:dsfContact=
    "urn:ietf:params:xml:ns:dsfContact-1.0">
  <dataSet:defData>
    <dataSet:type>

```

```

    contact.create.standard
</dataSet:type>
<dataSet:fields sep="|">
  <dsfContact:fId/>
  <dsfContact:fPostalType/>
  <dsfContact:fName/>
  <dsfContact:fOrg/>
  <dsfContact:fStreet index="0"/>
  <dsfContact:fStreet index="1"/>
  <dsfContact:fStreet index="2"/>
  <dsfContact:fCity/>
  <dsfContact:fSp/>
  <dsfContact:fPc/>
  <dsfContact:fCc/>
  <dsfContact:fVoice/>
  <dsfContact:fVoiceExt/>
  <dsfContact:fFax/>
  <dsfContact:fFaxExt/>
  <dsfContact:fEmail/>
  <dataSet:fAuthInfo/>
</dataSet:fields>
<dataSet:dataSetId>abc-123</dataSet:dataSetId>
<dataSet:crDate>2016-04-03T22:00:00.0Z</dataSet:crDate>
</dataSet:defData>
</dataSet:definition>
-----BEGIN DATA SET-----
sh1111|int|John Doe|Example Inc.|
  123 Example Dr.||||Dulles|VA|20166-6503|US|
  +1.7035555555|1234|+1.7035555556||
  johndoe@example.com|2fooBAR
sh2222|int|Jane Doe|Example Inc.|
  123 Example Dr.|Suite 2222||Dulles|VA|20166-6503|US|
  +1.7035555555||||
  janedoe@example.com|2fooBAR
-----END DATA SET-----

```

Example of a "contact.create.routing" DSF that extends the "contact.create.standard" DSF with the <dsfContact:fSubProduct> field ([Section 2.5](#)) to route the create to the appropriate Top Level Domain (TLD) or group of TLDs. A "|" character is used as a separator to

minimize conflicting with the contact data. This is an example of

creating two contacts, where indented data set lines are a continuation from the prior line.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<dataSet:definition
  xmlns:dataSet=
    "urn:ietf:params:xml:ns:dataSet-1.0"
  xmlns:dsfContact=
    "urn:ietf:params:xml:ns:dsfContact-1.0"
  xmlns:dsfRouting=
    "urn:ietf:params:xml:ns:dsfRouting-1.0">
  <dataSet:defData>
    <dataSet:type>
      contact.create.routing
    </dataSet:type>
    <dataSet:fields sep="|">
      <dsfRouting:fSubProduct/>
      <dsfContact:fId/>
      <dsfContact:fPostalType/>
      <dsfContact:fName/>
      <dsfContact:fOrg/>
      <dsfContact:fStreet index="0"/>
      <dsfContact:fStreet index="1"/>
      <dsfContact:fStreet index="2"/>
      <dsfContact:fCity/>
      <dsfContact:fSp/>
      <dsfContact:fPc/>
      <dsfContact:fCc/>
      <dsfContact:fVoice/>
      <dsfContact:fVoiceExt/>
      <dsfContact:fFax/>
      <dsfContact:fFaxExt/>
      <dsfContact:fEmail/>
      <dataSet:fAuthInfo/>
    </dataSet:fields>
    <dataSet:dataSetId>abc-123</dataSet:dataSetId>
    <dataSet:crDate>2016-04-03T22:00:00.0Z</dataSet:crDate>
  </dataSet:defData>
</dataSet:definition>
-----BEGIN DATA SET-----
EXAMPLE|sh1111|int|John Doe|Example Inc.|
  123 Example Dr.|||Dulles|VA|20166-6503|US|
  +1.7035555555|1234|+1.7035555556||
  johndoe@example.com|2fooBAR
EXAMPLE|sh2222|int|Jane Doe|Example Inc.|
  123 Example Dr.|Suite 2222||Dulles|VA|20166-6503|US|
  +1.7035555555| |||
  janedoe@example.com|2fooBAR
-----END DATA SET-----
```

[4.4.](#) Verification Code Object

The Verification Code Object is defined in the Verification Code Extension for the Extensible Provisioning Protocol [[I-D.gould-eppext-verificationcode](#)]. This section defines the Verification Code specific Data Set File (DSF) field elements and sample DSF files used to set verification codes with a bulk update.

The Verification Service Provider (VSP) is a certified party to verify that data is in compliance with the policies of a locality. A locality MAY require the client to have data verified in accordance with local regulations or laws utilizing data sources not available to the server. The VSP has access to the local data sources and is authorized to verify the data. Examples include verifying that the domain name is not prohibited and verifying that the domain name registrant is a valid individual, organization, or business in the locality. The data verified, and the objects and operations that require the verification code to be passed to the server is up to the policies of the locality. The verification code represents a marker that the verification was completed. The data verified by the VSP MUST be stored by the VSP along with the generated verification codes in order to address any compliance issues. The signer certificate and the digital signature of the verification code MUST be verified by the server.

The Data Set File (DSF) field elements specific to the Verification Code Object include:

<dsfVerificationCode:fCode>: Field that represents a Verification Code Token value. The field extends a "dataSet:fieldOptionalType", as defined in [Section 2.4.1](#), with the type="dataSet:verificationCodeValueType". The field includes the required "codeType" attribute that represents the type of verification code, as defined by the "type" attribute of the <verificationCode:signedCode> element in [[I-D.gould-eppext-verificationcode](#)]. The verification code field can be empty since it is a "dataSet:fieldOptionalType".

<dsfVerificationCode:fEncodedSignedCode>: Field that represents a Encoded Signed Code. The field extends a

"dataSet:fieldOptionalType", as defined in [Section 2.4.1](#), with the type="token". The field includes the OPTIONAL "encoding" attribute with the default value of "base64". The line breaks, defined in [[RFC2045](#)], MUST NOT be used with the "base64" Encoded Signed Code.

The Data Set File (DSF) used for setting of the verification codes MAY include fields from other objects and the fields specific to the Verification Code Object. There are two scenarios in passing the

verification codes based on who generates the DSF file, which includes:

VSP Generated DSF: The VSP generates the Verification Code Token values, represented with the <dsfVerificationCode:fCode> field, and digitally signs the DSF file by referencing the <dataSet:encodedSignedDefData> element in the DSF header.

Client Generated DSF: The client generates the DSF file with a set of encoded signed codes, represented with the <dsfVerificationCode:fEncodedSignedCode> field, collected from the VSP to bulk submit them to the server. The <dataSet:defData> element, described in [Section 3.1.1](#), is referenced in the DSF header.

The following examples reference verification codes supported by the China Locality, which includes the Domain Name Verification Code (DNVC) and the Real Name Verification Code (RNVC). Both China Locality verification codes are set on a domain name with the verification code type of "domain" used for the DNVC and the verification code type of "real-name" used for the RNVC. Other localities MAY include a different set of verification codes.

Example VSP Generated DSF that includes a domain name primary key field, a "domain" code token value field, and a "real-name" token value field. The "base64" value includes "..." representing continuation of data for brevity:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<dataSet:definition
  xmlns:dataSet=
    "urn:ietf:params:xml:ns:dataSet-1.0">
  <dataSet:encodedSignedDefData encoding="base64">
```

```

ICAgICAgPHZlcmlmaWNhdGlvbKNvZGU6c2lnbmVkJQ29kZQogICAgICAgIHhtbG5z
...
b25Db2RlOnNpZ25lZENvZGU+Cg==
</dataSet:encodedSignedDefData>
</dataSet:definition>
-----BEGIN DATA SET-----
domain1.example,0-abc111,0-abc222
domain2.example,0-abc333,0-abc444
domain3.example,0-abc555,
domain3.example,,0-abc666
-----END DATA SET-----

```

Example Client Generated Data Set File (DSF) that includes a domain name primary key field, a "domain" encoded signed code field, and a "real-name" encoded signed code field. The "base64" values include no line breaks and include "..." representing continuation of data for brevity:

```

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<dataSet:definition
  xmlns:dataSet=
    "urn:ietf:params:xml:ns:dataSet-1.0"
  xmlns:dsfDomain=
    "urn:ietf:params:xml:ns:dsfDomain-1.0"
  xmlns:dsfVerificationCode=
    "urn:ietf:params:xml:ns:dsfVerificationCode-1.0"
>
<dataSet:defData>
  <dataSet:type subType="china">
    verificationCode.update.encodedSignedCode
  </dataSet:type>
  <dataSet:fields>
    <dsfDomain:fName/>
    <dsfVerificationCode:fEncodedSignedCode type="domain"/>
    <dsfVerificationCode:fEncodedSignedCode type="real-name"/>
  </dataSet:fields>
  <dataSet:dataSetId>abc-123</dataSet:dataSetId>

```

```

    <dataSet:crDate>2016-04-03T22:00:00.0Z</dataSet:crDate>
  </dataSet:defData>
</dataSet:definition>
-----BEGIN DATA SET-----
domain1.example,ICAg...GlvbkN,CAGICA...u0mlldG
domain2.example,YW1zOb2R...GlvbkNvZGU6c2ln,
domain3.example,,CAGICAnZ...htbDpZmljYXMCIKI
-----END DATA SET-----

```

5. Result Codes

The result codes are based on and not equal to the reply codes defined in the EPP result codes of [\[RFC5730\]](#). Four decimal digits are used to describe the success or failure of the processing of the Data Set File (DSF) or the processing of an individual record. Each digit of the result code has specific significance.

The first digit denotes success or failure. The second digit denotes the result category, such as request syntax or security. The third and fourth digits provide explicit response detail within each result category.

There are two values for the first digit of the result code:

Gould	Expires March 9, 2019	[Page 41]
-------	-----------------------	-----------

Internet-Draft	Data Set File Format	September 2018
----------------	----------------------	----------------

1yzz: Positive result. The file or record was processed by the system successfully. Partial successful results MAY be included.

2yzz: Negative result. The file or record was not processed successfully. Partial successful results MAY NOT be included.

The second digit groups responses into one of five specific categories:

- x0zz: File or record syntax
- x1zz: Implementation-specific rules
- x2zz: Security
- x3zz: Data Management
- x4zz: Server System

Successful result codes:

1000 "Success":
 This is the result for a successfully processed file when all

records are processed successfully or the result for an individual record that is successfully processed.

1001 "Success with failures":

This is the result for a successfully processed file when some records are processed successfully and some records failed.

1002 "Success with all failures":

This is the result for a successfully processed file when all of the records failed.

Error result codes:

2000 "File syntax error":

This result code occurs when the overall file is syntactically incorrect.

2001 "Header syntax error":

This result code occurs when the header is syntactically incorrect.

2002 "Body syntax error":

This result code occurs when the body is syntactically incorrect.

2003 "Required parameter missing":

This result code occurs when a server receives a request for which a required parameter has not been provided.

2004 "Parameter value range error":

This result code occurs when a server receives a request whose value is outside the range of values specified in the protocol.

2005 "Parameter value syntax error":

This result code occurs when a server receives a request whose value is improperly formed.

2100 "Unimplemented protocol version":

This result code occurs when a server receives a request version, as defined by the XML namespace of the header, that is

not implemented by the server.

2102 "Unimplemented option":

This result code occurs when a server receives a request that contains a protocol option that is not implemented by the server.

2103 "Unimplemented extension":

This result code occurs when a server receives a request that contains an extension, as defined by the XML namespace used by a field element extension, that is not implemented by the server.

2104 "Billing failure":

This result code occurs when a server receives a request that cannot be completed due to a client-billing failure.

2201 "Authorization error":

This result code occurs when a server receives a request cannot be execute due to a client-authorization error.

2202 "Invalid authorization information":

This result code occurs when a server receives invalid authorization information to execute a request.

2302 "Object exists":

This result code occurs when a server receives a request to create an object that already exists in the repository.

2303 "Object does not exist":

This result code occurs when a server receives a request to transform an object that does not exist in the repository.

2304 "Object status prohibits operation":

This result code occurs when a server receives a request to transform an object that cannot be completed due to server policy or business practices.

2305 "Object association prohibits operation":

This result code occurs when a server receives a request to transform an object that cannot be completed due to dependencies to other objects that are associated with the

target object.

- 2306 "Parameter value policy error":
This result code occurs when a server receives a request that contains a parameter value that is syntactically valid but semantically invalid due to local policy.
- 2307 "Unimplemented object service":
This result code occurs when a server receives a request to operate on an object that is not supported by the server.
- 2308 "Data management policy violation":
This result code occurs when a server receives a request whose execution results in a violation of server data management policies.
- 2400 "Request failed":
This result code occurs when a server receives a request that cannot be executed due to an internal server error that is not related to the protocol.

The error result codes include some overlap that will require the server to decide which one to return based on server policy. For example, a "Header syntax error", represented by the error result code 2001, may have occurred for multiple reasons including a 2003 "Required parameter missing", a 2004 "Parameter value range error", or a 2005 "Parameter value syntax error". The overlapping reasons also apply at the record level, where an object may be syntactically incorrect with a 2005 "Parameter value syntax error", and also be against server policy with a 2306 "Parameter value policy error". The server SHOULD attempt to provide the most accurate error result code to match the error. For example, a syntax error defined by the protocol SHOULD take precedence over a server policy error.

[6.](#) Example Data Set File (DSF) Result Files

This section includes a set of Data Set File (DSF) result examples.

[6.1.](#) Example Data Set File (DSF) with Result Data

This section includes example Data Set File (DSF) files that reference the <dataSet:resultData> element, described in [Section 3.1.3](#), for providing the meta-data about the result file,

along with a set of fields that reference the <dataSet:fResultCode> field element, described in [Section 2.4.2](#), for including the result code in processing the domain name record in the request.

Example Data Set File (DSF) for a succesful result by the server:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<dataSet:definition
  xmlns:dataSet=
    "urn:ietf:params:xml:ns:dataSet-1.0"
  xmlns:dsfDomain=
    "urn:ietf:params:xml:ns:dsfDomain-1.0">
  <dataSet:resultData code="1000">
    <dataSet:fields>
      <dsfDomain:fName/>
      <dataSet:fResultCode/>
      <dataSet:fResultMsg/>
      <dataSet:fResultReason/>
    </dataSet:fields>
    <dataSet:dataSetId>abc-123</dataSet:dataSetId>
    <dataSet:svTRID>54322-XYZ</dataSet:svTRID>
    <dataSet:msg>Code Set processed successfully.</dataSet:msg>
    <dataSet:records>
      <dataSet:total>2</dataSet:total>
      <dataSet:success>2</dataSet:success>
      <dataSet:failed>0</dataSet:failed>
    </dataSet:records>
  </dataSet:resultData>
</dataSet:definition>
-----BEGIN DATA SET-----
domain1.example,1000,,
domain2.example,1000,,
-----END DATA SET-----
```

Example Data Set File (DSF) for a partially successful result by the server:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<dataSet:definition
  xmlns:dataSet=
    "urn:ietf:params:xml:ns:dataSet-1.0"
  xmlns:dsfDomain=
    "urn:ietf:params:xml:ns:dsfDomain-1.0">
  <dataSet:resultData code="1001">
    <dataSet:fields>
      <dsfDomain:fName/>
      <dataSet:fResultCode/>
      <dataSet:fResultMsg/>
      <dataSet:fResultReason/>
    </dataSet:fields>
    <dataSet:dataSetId>abc-123</dataSet:dataSetId>
    <dataSet:svTRID>54322-XYZ</dataSet:svTRID>
    <dataSet:msg>Success with failures</dataSet:msg>
    <dataSet:records>
      <dataSet:total>4</dataSet:total>
      <dataSet:success>1</dataSet:success>
      <dataSet:failed>3</dataSet:failed>
    </dataSet:records>
  </dataSet:resultData>
</dataSet:definition>
-----BEGIN DATA SET-----
domain1.example,1000,Success,
domain2.example,2303,Object does not exist,
domain3.example,2201,Authorization error,
domain4.example,2302,Object exists,domain code already set
-----END DATA SET-----
```

Example Data Set File (DSF) for a failure result by the server:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<dataSet:definition
  xmlns:dataSet=
    "urn:ietf:params:xml:ns:dataSet-1.0">
  <dataSet:resultData code="2000">
    <dataSet:dataSetId>abc-123</dataSet:dataSetId>
    <dataSet:svTRID>54322-XYZ</dataSet:svTRID>
    <dataSet:msg>File syntax error</dataSet:msg>
    <dataSet:reason lang="en">Malformed request file
  </dataSet:reason>
  </dataSet:resultData>
</dataSet:definition>
-----BEGIN DATA SET-----
-----END DATA SET-----
```

[7.](#) Formal Syntax

Each schema is included in a sub-section, starting with the Data Set schema and followed by each object schema.

The formal syntax presented here is a complete schema representation of the object mapping suitable for automated validation of EPP XML instances. The BEGIN and END tags are not part of the schema; they are used to note the beginning and ending of the schema for URI registration purposes.

[7.1.](#) Data Set Schema

Data Set schema that represents the base XML schema for the Data Set File (DSF).

```

BEGIN
<?xml version="1.0" encoding="UTF-8"?>
<schema
  targetNamespace=
    "urn:ietf:params:xml:ns:dataSet-1.0"
  xmlns:dataSet=
    "urn:ietf:params:xml:ns:dataSet-1.0"
  xmlns:dsig="http://www.w3.org/2000/09/xmldsig#"
  xmlns="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">

  <annotation>
    <documentation>
      Data Set XML Schema
    </documentation>

```

Gould

Expires March 9, 2019

[Page 47]

Internet-Draft

Data Set File Format

September 2018

```

</annotation>

<import namespace="http://www.w3.org/2000/09/xmldsig#"
  schemaLocation="xmldsig-core-schema.xsd"/>

<element name="definition"
  type="dataSet:definitionType"/>

<complexType name="definitionType">
  <sequence>
    <choice>
      <element name="encodedSignedDefData"
        type="dataSet:encodedSignedDefDataType"/>
      <element name="defData"
        type="dataSet:defDataType"/>
      <element name="resultData"
        type="dataSet:resultDataType"/>
    </choice>
  </sequence>
</complexType>

<!-- field separator must be a single character -->
<simpleType name="sepType">
  <restriction base="string">
    <minLength value="1"/>
    <maxLength value="1"/>

```

```

    </restriction>
</simpleType>

<!-- Ordered list of field definitions for the csv -->
<complexType name="fieldsType">
  <sequence maxOccurs="unbounded">
    <element ref="dataSet:field"/>
  </sequence>
  <attribute name="sep" type="dataSet:sepType" default=","/>
</complexType>

<!-- defDataType -->
<complexType name="defDataType">
  <sequence>
    <element name="type"
      type="dataSet:typeType"/>
    <element name="fields"
      type="dataSet:fieldsType"/>
    <element name="dataSetId"
      type="dataSet:IdType"/>
    <element name="crDate"
      type="dateTime"/>
  </sequence>
</complexType>

```

```

  </sequence>
</complexType>

<!-- typeType -->
<complexType name="typeType">
  <simpleContent>
    <extension base="token">
      <attribute name="subType" type="token"/>
    </extension>
  </simpleContent>
</complexType>

<!-- signedData -->
<element name="signedDefData"
  type="dataSet:signedDefDataType"/>

<complexType name="signedDefDataType">
  <complexContent>
    <extension base="dataSet:defDataType">

```

```

        <sequence>
            <element name="cksum"
                type="token"/>
            <element ref="dsig:Signature"/>
        </sequence>
        <attribute name="id" type="ID" use="required"/>
    </extension>
</complexContent>
</complexType>

<!-- resultDataType -->
<complexType name="resultDataType">
    <sequence>
        <element name="type"
            type="dataSet:typeType"
            minOccurs="0"/>
        <element name="fields"
            type="dataSet:fieldsType"
            minOccurs="0"/>
        <element name="dataSetId"
            type="dataSet:IdType"
            minOccurs="0"/>
        <element name="svTRID"
            type="dataSet:IdType"/>
        <element name="msg"
            type="dataSet:msgType"/>
        <element name="reason"
            type="dataSet:msgType"
            minOccurs="0"/>
    </sequence>
</complexType>

```

```

        <element name="records"
            type="dataSet:recordsType"
            minOccurs="0"/>
    </sequence>
    <attribute name="code" type="dataSet:resultCodeType"
        use="required"/>
</complexType>

<complexType name="recordsType">
    <sequence>
        <element name="total" type="unsignedInt"/>
        <element name="success" type="unsignedInt"/>
    </sequence>
</complexType>

```

```

    <element name="failed" type="unsignedInt"/>
  </sequence>
</complexType>

<simpleType name="resultCodeType">
  <restriction base="unsignedShort">
    <!-- Success -->
    <enumeration value="1000"/>
    <!-- Success with failures -->
    <enumeration value="1001"/>
    <!-- Success with all failures -->
    <enumeration value="1002"/>
    <!-- File syntax error -->
    <enumeration value="2000"/>
    <!-- Invalid header -->
    <enumeration value="2001"/>
    <!-- Invalid body -->
    <enumeration value="2002"/>
    <!-- Required parameter missing -->
    <enumeration value="2003"/>
    <!-- Parameter value range error -->
    <enumeration value="2004"/>
    <!-- Parameter value syntax error -->
    <enumeration value="2005"/>
    <!-- Unimplemented protocol version -->
    <enumeration value="2100"/>
    <!-- Unimplemented option -->
    <enumeration value="2102"/>
    <!-- Unimplemented extension -->
    <enumeration value="2103"/>
    <!-- Billing failure -->
    <enumeration value="2104"/>
    <!-- Authorization error -->
    <enumeration value="2201"/>
    <!-- Invalid authorization information -->
    <enumeration value="2202"/>
  </restriction>
</simpleType>

```

```

  <!-- Object exists -->
  <enumeration value="2302"/>
  <!-- Object does not exist -->
  <enumeration value="2303"/>
  <!-- Object status prohibits operation -->

```

```

    <enumeration value="2304"/>
    <!-- Object association prohibits operation -->
    <enumeration value="2305"/>
    <!-- Parameter value policy error -->
    <enumeration value="2306"/>
    <!-- Unimplemented object service -->
    <enumeration value="2307"/>
    <!-- Data management policy violation -->
    <enumeration value="2308"/>
    <!-- Request failed -->
    <enumeration value="2400"/>
  </restriction>
</simpleType>

<simpleType name="IdType">
  <restriction base="token">
    <minLength value="3"/>
    <maxLength value="64"/>
  </restriction>
</simpleType>

<complexType name="msgType">
  <simpleContent>
    <extension base="normalizedString">
      <attribute name="lang" type="language"
        default="en"/>
    </extension>
  </simpleContent>
</complexType>

<!-- encodedSignedData -->
<element name="encodedSignedDefData"
  type="dataSet:encodedSignedDefDataType"/>

<complexType name="encodedSignedDefDataType">
  <simpleContent>
    <extension base="token">
      <attribute name="encoding"
        type="token" default="base64"/>
    </extension>
  </simpleContent>
</complexType>

```

```

<!-- Abstract field type -->
<element name="field" type="dataSet:fieldType"
  abstract="true"/>

<complexType name="fieldType">
  <sequence/>
</complexType>

<!-- fieldType with optional value (isRequired=false) -->
<complexType name="fieldOptionalType">
  <complexContent>
    <extension base="dataSet:fieldType">
      <sequence/>
      <attribute name="isRequired" type="boolean"
        default="false"/>
      <attribute name="isPrimaryKey" type="boolean"
        default="false"/>
    </extension>
  </complexContent>
</complexType>

<!-- fieldType with required value (isRequired=false) -->
<complexType name="fieldRequiredType">
  <complexContent>
    <extension base="dataSet:fieldType">
      <sequence/>
      <attribute name="isRequired" type="boolean"
        default="true"/>
      <attribute name="isPrimaryKey" type="boolean"
        default="false"/>
    </extension>
  </complexContent>
</complexType>

<!-- fieldType as primary key field (isPrimaryKey=true) -->
<complexType name="fieldPrimaryKeyType">
  <complexContent>
    <extension base="dataSet:fieldType">
      <sequence/>
      <attribute name="isRequired" type="boolean"
        default="true"/>
      <attribute name="isPrimaryKey" type="boolean"
        default="true"/>
    </extension>
  </complexContent>
</complexType>

<!-- fieldType as list item -->

```

```
<complexType name="fListItemType">
  <complexContent>
    <extension base="dataSet:fieldOptionalType">
      <sequence/>
      <attribute name="op"
        type="dataSet:listItemOpType"
        default="replace"/>
    </extension>
  </complexContent>
</complexType>

<!-- Enumerated list item "op" attribute values -->
<simpleType name="listItemOpType">
  <restriction base="token">
    <enumeration value="replace"/>
    <enumeration value="add"/>
    <enumeration value="remove"/>
  </restriction>
</simpleType>

<complexType name="fNameRequiredType">
  <complexContent>
    <extension base="dataSet:fieldRequiredType">
      <sequence/>
      <attribute name="type" type="token"
        default="eppcom\:labelType"/>
    </extension>
  </complexContent>
</complexType>

<element name="fName" type="dataSet:fNameType"
  substitutionGroup="dataSet:field"/>

<complexType name="fNameType">
  <complexContent>
    <extension base="dataSet:fieldPrimaryKeyType">
      <sequence/>
      <attribute name="type" type="token"
        default="dataSet\:labelType"/>
      <attribute name="class" type="token"
        use="required"/>
    </extension>
```

```

    </complexContent>
</complexType>

<!-- Authorization Information field -->
<element name="fAuthInfo" type="dataSet:fAuthInfoType"
    substitutionGroup="dataSet:field"/>

```

```

<complexType name="fAuthInfoType">
  <complexContent>
    <extension base="dataSet:fieldOptionalType">
      <sequence/>
      <attribute name="type" type="token"
        default="eppcom\:pwAuthInfoType"/>
    </extension>
  </complexContent>
</complexType>

```

```

<!-- General token type -->
<complexType name="fTokenType">
  <complexContent>
    <extension base="dataSet:fieldOptionalType">
      <sequence/>
      <attribute name="type" type="token"
        default="token"/>
    </extension>
  </complexContent>
</complexType>

```

```

<!-- boolean type -->
<complexType name="fBooleanType">
  <complexContent>
    <extension base="dataSet:fieldOptionalType">
      <sequence/>
      <attribute name="type" type="token"
        default="boolean"/>
    </extension>
  </complexContent>
</complexType>

```

```

<!-- fResultCode field -->
<element name="fResultCode"

```

```

    type="dataSet:fResultCodeType"
    substitutionGroup="dataSet:field"/>
<complexType name="fResultCodeType">
  <complexContent>
    <extension base="dataSet:fieldRequiredType">
      <sequence/>
      <attribute name="type" type="token"
        default="dataSet\:resultCodeType"/>
    </extension>
  </complexContent>
</complexType>

<!-- fResultMsg field -->

```

```

<element name="fResultMsg"
  type="dataSet:fResultMsgType"
  substitutionGroup="dataSet:field"/>
<complexType name="fResultMsgType">
  <complexContent>
    <extension base="dataSet:fieldOptionalType">
      <sequence/>
      <attribute name="type" type="token"
        default="normalizedString"/>
      <attribute name="lang" type="language"
        default="en"/>
    </extension>
  </complexContent>
</complexType>

<!-- fResultReason field -->
<element name="fResultReason"
  type="dataSet:fResultReasonType"
  substitutionGroup="dataSet:field"/>
<complexType name="fResultReasonType">
  <complexContent>
    <extension base="dataSet:fieldOptionalType">
      <sequence/>
      <attribute name="type" type="token"
        default="normalizedString"/>
      <attribute name="lang" type="language"
        default="en"/>
    </extension>
  </complexContent>
</complexType>

```

```

    </complexContent>
  </complexType>

  <!-- unsignedByte type -->
  <complexType name="fUnsignedByteType">
    <complexContent>
      <extension base="dataSet:fieldOptionalType">
        <sequence/>
        <attribute name="type" type="token"
          default="unsignedByte"/>
      </extension>
    </complexContent>
  </complexType>

  <!-- unsignedShort type -->
  <complexType name="fUnsignedShortType">
    <complexContent>
      <extension base="dataSet:fieldOptionalType">
        <sequence/>
        <attribute name="type" type="token"

```

```

      default="unsignedShort"/>
    </extension>
  </complexContent>
</complexType>

<!-- hexBinary type -->
<complexType name="fHexBinaryType">
  <complexContent>
    <extension base="dataSet:fieldOptionalType">
      <sequence/>
      <attribute name="type" type="token"
        default="hexBinary"/>
    </extension>
  </complexContent>
</complexType>

</schema>
END

```

Domain Name Object XML schema that defines the fields specific to the Domain Name Object.

BEGIN

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

<schema targetNamespace="urn:ietf:params:xml:ns:dsfDomain-1.0"
  xmlns:dsfDomain="urn:ietf:params:xml:ns:dsfDomain-1.0"
  xmlns:dataSet="urn:ietf:params:xml:ns:dataSet-1.0"
  xmlns:secDNS="urn:ietf:params:xml:ns:secDNS-1.1"
  xmlns:domain="urn:ietf:params:xml:ns:domain-1.0"
  xmlns:eppcom="urn:ietf:params:xml:ns:eppcom-1.0"
  xmlns="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">

  <!--
  Import common element types
  -->
  <import namespace="urn:ietf:params:xml:ns:eppcom-1.0"
    schemaLocation="eppcom-1.0.xsd"/>
  <import namespace="urn:ietf:params:xml:ns:domain-1.0"
    schemaLocation="domain-1.0.xsd"/>
  <import namespace="urn:ietf:params:xml:ns:secDNS-1.1"
    schemaLocation="secDNS-1.1.xsd"/>
  <import namespace="urn:ietf:params:xml:ns:dataSet-1.0"
    schemaLocation="dataSet-1.0.xsd"/>
```

Gould

Expires March 9, 2019

[Page 56]

Internet-Draft

Data Set File Format

September 2018

```
<annotation>
  <documentation>
    Domain Name Data Set File (DSF) Object
  </documentation>
</annotation>

<!-- Domain name field -->
<element name="fName" type="dataSet:fNameRequiredType"
  substitutionGroup="dataSet:field"/>

<!-- Registration Period -->
<element name="fPeriod" type="dsfDomain:fPeriodType"
  substitutionGroup="dataSet:field"/>
```

```

<complexType name="fPeriodType">
  <complexContent>
    <extension base="dataSet:fieldOptionalType">
      <sequence/>
      <attribute name="type" type="token"
        default="domain\:pLimitType"/>
    </extension>
  </complexContent>
</complexType>

<!-- Registration Period Unit -->
<element name="fPeriodUnit" type="dsfDomain:fPeriodUnitType"
  substitutionGroup="dataSet:field"/>

<complexType name="fPeriodUnitType">
  <complexContent>
    <extension base="dataSet:fieldOptionalType">
      <sequence/>
      <attribute name="type" type="token"
        default="domain\:pUnitType"/>
    </extension>
  </complexContent>
</complexType>

<!-- Contact field -->
<element name="fContact" type="dsfDomain:fContactsType"
  substitutionGroup="dataSet:field"/>

<complexType name="fContactsType">
  <complexContent>
    <extension base="dataSet:fieldOptionalType">
      <sequence/>
      <attribute name="type" type="token"
        default="dataSet\:contactRoleType"/>
    </extension>
  </complexContent>
</complexType>

```

```

      <attribute name="role" type="dsfDomain:contactRoleType"
        use="required"/>
    </extension>
  </complexContent>
</complexType>

<simpleType name="contactRoleType">

```

```

    <restriction base="token">
      <enumeration value="registrant"/>
      <enumeration value="admin"/>
      <enumeration value="tech"/>
      <enumeration value="billing"/>
    </restriction>
  </simpleType>

  <!-- Domain name server -->
  <element name="fNs" type="dsfDomain:fNsType"
    substitutionGroup="dataSet:field"/>

  <complexType name="fNsType">
    <complexContent>
      <extension base="dataSet:fListItemType">
        <sequence/>
        <attribute name="type" type="token"
          default="domain\:pLimitType"/>
      </extension>
    </complexContent>
  </complexType>

  <!-- DNSSEC field types -->

  <!-- Maximum signature lifetime field -->
  <element name="fMaxSigLife" type="dsfDomain:fMaxSigLifeType"
    substitutionGroup="dataSet:field"/>
  <complexType name="fMaxSigLifeType">
    <complexContent>
      <extension base="dataSet:fieldOptionalType">
        <sequence/>
        <attribute name="type" type="token"
          default="secDNS\:maxSigLifeType"/>
      </extension>
    </complexContent>
  </complexType>

  <!-- Key tag field -->
  <element name="fKeyTag" type="dataSet:fUnsignedShortType"
    substitutionGroup="dataSet:field"/>

```

```

<!-- DS Algorithm field -->
<element name="fDsAlg" type="dataSet:fUnsignedByteType"
  substitutionGroup="dataSet:field"/>

<!-- Digest type field -->
<element name="fDigestType" type="dataSet:fUnsignedByteType"
  substitutionGroup="dataSet:field"/>

<!-- Digest field -->
<element name="fDigest" type="dataSet:fHexBinaryType"
  substitutionGroup="dataSet:field"/>

<!-- Flags field -->
<element name="fFlags" type="dataSet:fUnsignedShortType"
  substitutionGroup="dataSet:field"/>

<!-- Protocol field -->
<element name="fProtocol" type="dataSet:fUnsignedByteType"
  substitutionGroup="dataSet:field"/>

<!-- Key Algorithm field -->
<element name="fKeyAlg" type="dataSet:fUnsignedByteType"
  substitutionGroup="dataSet:field"/>

<!-- Public Key field -->
<element name="fPubKey" type="dsfDomain:fPubKeyType"
  substitutionGroup="dataSet:field"/>
<complexType name="fPubKeyType">
  <complexContent>
    <extension base="dataSet:fieldOptionalType">
      <sequence/>
      <attribute name="type" type="token"
        default="secDNS\:keyType"/>
    </extension>
  </complexContent>
</complexType>

<!-- Domain status field -->
<element name="fStatus" type="dsfDomain:fStatusType"
  substitutionGroup="dataSet:field"/>

<!-- Domain status based on domain-1.0.xsd -->
<complexType name="fStatusType">
  <complexContent>
    <extension base="dataSet:fListItemType">
      <sequence/>
      <attribute name="type" type="token"
        default="domain\:statusValueType"/>
    </extension>
  </complexContent>
</complexType>

```

Internet-Draft

Data Set File Format

September 2018

```
        </extension>
    </complexContent>
</complexType>

<!--
End of schema.
-->
</schema>
END
```

[7.3.](#) Host Schema

Host Object XML schema that defines the fields specific to the Host Object.

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

<schema targetNamespace="urn:ietf:params:xml:ns:dsfHost-1.0"
  xmlns:dsfHost="urn:ietf:params:xml:ns:dsfHost-1.0"
  xmlns:dataSet="urn:ietf:params:xml:ns:dataSet-1.0"
  xmlns:host="urn:ietf:params:xml:ns:host-1.0"
  xmlns:eppcom="urn:ietf:params:xml:ns:eppcom-1.0"
  xmlns="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">

  <!--
  Import common element types
  -->
  <import namespace="urn:ietf:params:xml:ns:eppcom-1.0"
    schemaLocation="eppcom-1.0.xsd"/>
  <import namespace="urn:ietf:params:xml:ns:host-1.0"
    schemaLocation="host-1.0.xsd"/>
  <import namespace="urn:ietf:params:xml:ns:dataSet-1.0"
    schemaLocation="dataSet-1.0.xsd"/>

  <annotation>
    <documentation>
      Host Name Data Set File (DSF) Object
    </documentation>
  </annotation>

  <!-- Host name field -->
```

```

<element name="fName" type="dataSet:fNameRequiredType"
  substitutionGroup="dataSet:field"/>

<!-- Host new name field -->
<element name="fNewName" type="dsfHost:fNewNameType"

```

```

  substitutionGroup="dataSet:field"/>

<complexType name="fNewNameType">
  <complexContent>
    <extension base="dataSet:fieldOptionalType">
      <sequence/>
      <attribute name="type" type="token"
        default="eppcom\:labelType"/>
    </extension>
  </complexContent>
</complexType>

<!-- IP address field -->
<element name="fAddr" type="dsfHost:fAddrType"
  substitutionGroup="dataSet:field"/>

<complexType name="fAddrType">
  <complexContent>
    <extension base="dataSet:fListItemType">
      <sequence/>
      <attribute name="type" type="token"
        default="host\:addrStringType"/>
    </extension>
  </complexContent>
</complexType>

<!-- IP address version field linked to the fAddr field -->
<element name="fAddrVersion" type="dsfHost:fAddrVersionType"
  substitutionGroup="dataSet:field"/>
<complexType name="fAddrVersionType">
  <complexContent>
    <extension base="dataSet:fListItemType">
      <sequence/>
      <attribute name="type" type="token"
        default="host\:ipType"/>
    </extension>
  </complexContent>
</complexType>

```

```

    </complexContent>
  </complexType>

  <!-- Host status field -->
  <element name="fStatus" type="dsfHost:fStatusType"
    substitutionGroup="dataSet:field"/>

  <!-- Host status based on host-1.0.xsd -->
  <complexType name="fStatusType">
    <complexContent>
      <extension base="dataSet:fListItemType">
        <sequence/>
      </extension>
    </complexContent>
  </complexType>

```

Gould

Expires March 9, 2019

[Page 61]

Internet-Draft

Data Set File Format

September 2018

```

    <attribute name="type" type="token"
      default="host\:statusValueType"/>
  </extension>
</complexContent>
</complexType>

<!--
End of schema.
-->
</schema>
END

```

[7.4.](#) Contact Schema

Contact Object XML schema that defines the fields specific to the Contact Object.

BEGIN

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

```

<schema targetNamespace="urn:ietf:params:xml:ns:dsfContact-1.0"
  xmlns:dsfContact="urn:ietf:params:xml:ns:dsfContact-1.0"
  xmlns:dataSet="urn:ietf:params:xml:ns:dataSet-1.0"
  xmlns:contact="urn:ietf:params:xml:ns:contact-1.0"
  xmlns:eppcom="urn:ietf:params:xml:ns:eppcom-1.0"
  xmlns="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">

```

```
<!--
```

Import common element types.

```
-->
<import namespace="urn:ietf:params:xml:ns:eppcom-1.0"
  schemaLocation="eppcom-1.0.xsd"/>
<import namespace="urn:ietf:params:xml:ns:contact-1.0"
  schemaLocation="contact-1.0.xsd"/>
<import namespace="urn:ietf:params:xml:ns:dataSet-1.0"
  schemaLocation="dataSet-1.0.xsd"/>

<annotation>
  <documentation>
    Contact Data Set File (DSF) Object
  </documentation>
</annotation>

<!-- Server-unique contact identifier field -->
<element name="fId" type="dsfContact:fIdType"
  substitutionGroup="dataSet:field"/>
```

Gould

Expires March 9, 2019

[Page 62]

Internet-Draft

Data Set File Format

September 2018

```
<complexType name="fIdType">
  <complexContent>
    <extension base="dataSet:fieldRequiredType">
      <sequence/>
      <attribute name="type" type="token"
        default="eppcom\:clIDType"/>
    </extension>
  </complexContent>
</complexType>

<!-- Is Registrar Contact field -->
<element name="fIsRegistrarContact"
  type="dataSet:fBooleanType"
  substitutionGroup="dataSet:field"/>

<!-- voice and fax telephone number fields -->
<element name="fVoice" type="dsfContact:fE164StringType"
  substitutionGroup="dataSet:field"/>
<element name="fFax" type="dsfContact:fE164StringType"
  substitutionGroup="dataSet:field"/>
```

```

<complexType name="fE164StringType">
  <complexContent>
    <extension base="dataSet:fieldOptionalType">
      <sequence/>
      <attribute name="type" type="token"
        default="contact\:e164StringType"/>
    </extension>
  </complexContent>
</complexType>

<!-- voice and fax telephone extension fields -->
<element name="fVoiceExt" type="dataSet:fTokenType"
  substitutionGroup="dataSet:field"/>
<element name="fFaxExt" type="dataSet:fTokenType"
  substitutionGroup="dataSet:field"/>

<!-- contact email address field -->
<element name="fEmail" type="dsfContact:fEmailType"
  substitutionGroup="dataSet:field"/>
<complexType name="fEmailType">
  <complexContent>
    <extension base="dataSet:fieldRequiredType">
      <sequence/>
      <attribute name="type" type="token"
        default="eppcom\:minTokenType"/>
    </extension>
  </complexContent>

```

```

  </complexContent>
</complexType>

<!--
  Postal type field
  ("loc" = localized, "int" = internationalized)
-->
<element name="fPostalType" type="dsfContact:fPostalTypeType"
  substitutionGroup="dataSet:field"/>
<complexType name="fPostalTypeType">
  <complexContent>
    <extension base="dataSet:fieldOptionalType">
      <sequence/>

```

```

        <attribute name="type" type="token"
        default="contact\:postalInfoEnumType"/>
    </extension>
</complexContent>
</complexType>

<!-- Standard postal line field -->
<complexType name="fPostalLineType">
    <complexContent>
        <extension base="dataSet:fieldRequiredType">
            <sequence/>
            <attribute name="type" type="token"
            default="contact\:postalLineType"/>
            <attribute name="isLoc" type="boolean"/>
        </extension>
    </complexContent>
</complexType>

<!-- Standard optional postal line field -->
<complexType name="fOptPostalLineType">
    <complexContent>
        <extension base="dataSet:fieldOptionalType">
            <sequence/>
            <attribute name="type" type="token"
            default="contact\:optPostalLineType"/>
            <attribute name="isLoc" type="boolean"/>
        </extension>
    </complexContent>
</complexType>

<!-- Name of the individual or role field -->
<element name="fName" type="dsfContact:fPostalLineType"
    substitutionGroup="dataSet:field"/>

```

```

<!-- Name organization field -->
<element name="fOrg" type="dsfContact:fOptPostalLineType"
    substitutionGroup="dataSet:field"/>

<!-- Street address line field with required index attribute -->
<!-- starting with index 0. -->
<element name="fStreet" type="dsfContact:fStreetType"

```

```

    substitutionGroup="dataSet:field"/>
<complexType name="fStreetType">
  <complexContent>
    <extension base="dsfContact:fOptPostalLineType">
      <sequence/>
      <attribute name="index" type="int"
        use="required"/>
    </extension>
  </complexContent>
</complexType>

<!-- Contact's city field -->
<element name="fCity" type="dsfContact:fPostalLineType"
  substitutionGroup="dataSet:field"/>

<!-- Contact's state or province field -->
<element name="fSp" type="dsfContact:fOptPostalLineType"
  substitutionGroup="dataSet:field"/>

<!-- Contact's postal code field -->
<element name="fPc" type="dsfContact:fPcType"
  substitutionGroup="dataSet:field"/>
<complexType name="fPcType">
  <complexContent>
    <extension base="dataSet:fieldOptionalType">
      <sequence/>
      <attribute name="type" type="token"
        default="contact\:pcType"/>
      <attribute name="isLoc" type="boolean"/>
    </extension>
  </complexContent>
</complexType>

<!-- Contact's country code field -->
<element name="fCc" type="dsfContact:fCcType"
  substitutionGroup="dataSet:field"/>
<complexType name="fCcType">
  <complexContent>
    <extension base="dataSet:fieldRequiredType">
      <sequence/>

```

```

<attribute name="type" type="token"

```

```

        default="contact\:ccType"/>
        <attribute name="isLoc" type="boolean"/>
    </extension>
</complexContent>
</complexType>

<!-- Disclosure element fields -->
<!-- Flag of "1" to allow disclosure
      and "0" to disallow disclosure -->
<element name="fDiscloseFlag" type="dsfContact:fBoolean"
  substitutionGroup="dataSet:field"/>
<!-- Disclosure of localized name
      based on fDiscloseFlag? -->
<element name="fDiscloseNameLoc" type="dsfContact:fBoolean"
  substitutionGroup="dataSet:field"/>
<!-- Disclosure of internationalized name
      based on fDiscloseFlag? -->
<element name="fDiscloseNameInt" type="dsfContact:fBoolean"
  substitutionGroup="dataSet:field"/>
<!-- Disclosure of localized org
      based on fDiscloseFlag? -->
<element name="fDiscloseOrgLoc" type="dsfContact:fBoolean"
  substitutionGroup="dataSet:field"/>
<!-- Disclosure of internationalized org
      based on fDiscloseFlag? -->
<element name="fDiscloseOrgInt" type="dsfContact:fBoolean"
  substitutionGroup="dataSet:field"/>
<!-- Disclosure of localized address
      based on fDiscloseFlag? -->
<element name="fDiscloseAddrLoc" type="dsfContact:fBoolean"
  substitutionGroup="dataSet:field"/>
<!-- Disclosure of internationalized address
      based on fDiscloseFlag? -->
<element name="fDiscloseAddrInt" type="dsfContact:fBoolean"
  substitutionGroup="dataSet:field"/>
<!-- Disclosure voice telephone number
      based on fDiscloseFlag? -->
<element name="fDiscloseVoice" type="dsfContact:fBoolean"
  substitutionGroup="dataSet:field"/>
<!-- Disclosure facsimile telephone number
      based on fDiscloseFlag? -->
<element name="fDiscloseFax" type="dsfContact:fBoolean"
  substitutionGroup="dataSet:field"/>
<!-- Disclosure email address
      based on fDiscloseFlag? -->
<element name="fDiscloseEmail" type="dsfContact:fBoolean"
  substitutionGroup="dataSet:field"/>

```

```
<complexType name="fBoolean">
  <complexContent>
    <extension base="dataSet:fieldOptionalType">
      <sequence/>
      <attribute name="type" type="token"
        default="boolean"/>
    </extension>
  </complexContent>
</complexType>

<!-- Contact status field -->
<element name="fStatus" type="dsfContact:fStatusType"
  substitutionGroup="dataSet:field"/>

<!-- Host status based on contact-1.0.xsd -->
<complexType name="fStatusType">
  <complexContent>
    <extension base="dataSet:fListItemType">
      <sequence/>
      <attribute name="type" type="token"
        default="contact\:statusValueType"/>
    </extension>
  </complexContent>
</complexType>

<!--
End of schema.
-->
</schema>
END
```

[7.5.](#) Verification Code Schema

Verification Code Object XML schema that defines the fields specific to the Verification Code Object.

```
BEGIN
<?xml version="1.0" encoding="UTF-8"?>
<schema
  targetNamespace=
    "urn:ietf:params:xml:ns:dsfVerificationCode-1.0"
  xmlns:dsfVerificationCode=
    "urn:ietf:params:xml:ns:dsfVerificationCode-1.0"
  xmlns:dataSet="urn:ietf:params:xml:ns:dataSet-1.0"
  xmlns="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
```

<annotation>

```
<documentation>
  Verification Code Data Set File (DSF) Object
</documentation>
</annotation>

<import namespace="urn:ietf:params:xml:ns:dataSet-1.0"
  schemaLocation="dataSet-1.0.xsd"/>

<!-- fCode field -->
<element name="fCode" type="dsfVerificationCode:fCodeType"
  substitutionGroup="dataSet:field"/>
<complexType name="fCodeType">
  <complexContent>
    <extension base="dataSet:fieldOptionalType">
      <sequence/>
      <attribute name="type" type="token"
        default="dsfVerificationCode\:verificationCodeValueType"/>
      <attribute name="codeType" type="token"/>
    </extension>
  </complexContent>
</complexType>

<!-- Format of verification code value -->
<simpleType name="verificationCodeValueType">
  <restriction base="token">
    <pattern value="\d+-[A-Za-z0-9]+"/>
  </restriction>
</simpleType>

<!-- fEncodedSignedCode field -->
<element name="fEncodedSignedCode"
  type="dsfVerificationCode:fEncodedSignedCodeType"
  substitutionGroup="dataSet:field"/>
<complexType name="fEncodedSignedCodeType">
  <complexContent>
    <extension base="dataSet:fieldOptionalType">
      <sequence/>
      <attribute name="type" type="token"
        default="token"/>
    </extension>
  </complexContent>
</complexType>
```

```

        <attribute name="encoding" type="token"
            default="base64"/>
    </extension>
</complexContent>
</complexType>

</schema>
END

```

[7.6.](#) Routing Schema

Routing XML schema that defines the fields specific to routing.

```

BEGIN
<?xml version="1.0" encoding="UTF-8"?>
<schema
  targetNamespace=
    "urn:ietf:params:xml:ns:dsfRouting-1.0"
  xmlns:dsfRouting=
    "urn:ietf:params:xml:ns:dsfRouting-1.0"
  xmlns:dataSet="urn:ietf:params:xml:ns:dataSet-1.0"
  xmlns="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">

  <annotation>
    <documentation>
      Routing Data Set File (DSF) Fields
    </documentation>
  </annotation>

  <import namespace="urn:ietf:params:xml:ns:dataSet-1.0"
    schemaLocation="dataSet-1.0.xsd"/>

  <!-- fSubProduct field -->
  <element name="fSubProduct" type="dsfRouting:fSubProductType"
    substitutionGroup="dataSet:field"/>
  <complexType name="fSubProductType">
    <complexContent>
      <extension base="dataSet:fieldOptionalType">
        <sequence/>
        <attribute name="type" type="token"

```

```
        default="token"/>
    </extension>
</complexContent>
</complexType>

</schema>
END
```

[8.](#) IANA Considerations

[8.1.](#) XML Namespace

This document uses URNs to describe XML namespaces and XML schemas conforming to a registry mechanism described in [[RFC3688](#)].

Registration request for the dataSet namespace:

Gould	Expires March 9, 2019	[Page 69]
-------	-----------------------	-----------

Internet-Draft	Data Set File Format	September 2018
----------------	----------------------	----------------

URI: ietf:params:xml:ns:dataSet-1.0
Registrant Contact: See the "Author's Address" section of this document.
XML: None. Namespace URIs do not represent an XML specification.

Registration request for the dataSet XML schema:

URI: ietf:params:xml:schema:dataSet-1.0
Registrant Contact: See the "Author's Address" section of this document.
XML: See the "Formal Syntax" section of this document.

Registration request for the dsfDomain namespace:

URI: ietf:params:xml:ns:dsfDomain-1.0
Registrant Contact: See the "Author's Address" section of this document.
XML: None. Namespace URIs do not represent an XML specification.

Registration request for the dsfDomain XML schema:

URI: ietf:params:xml:schema:dsfDomain-1.0
Registrant Contact: See the "Author's Address" section of this document.
XML: See the "Formal Syntax" section of this document.

Registration request for the dsfHost namespace:

URI: ietf:params:xml:ns:dsfHost-1.0

Registrant Contact: See the "Author's Address" section of this document.

XML: None. Namespace URIs do not represent an XML specification.

Registration request for the dsfHost XML schema:

URI: ietf:params:xml:schema:dsfHost-1.0

Registrant Contact: See the "Author's Address" section of this document.

XML: See the "Formal Syntax" section of this document.

Registration request for the dsfContact namespace:

URI: ietf:params:xml:ns:dsfContact-1.0

Registrant Contact: See the "Author's Address" section of this document.

XML: None. Namespace URIs do not represent an XML specification.

Registration request for the dsfContact XML schema:

URI: ietf:params:xml:schema:dsfContact-1.0

Registrant Contact: See the "Author's Address" section of this document.

XML: See the "Formal Syntax" section of this document.

Registration request for the dsfVerificationCode namespace:

URI: ietf:params:xml:ns:dsfVerificationCode-1.0

Registrant Contact: See the "Author's Address" section of this document.

XML: None. Namespace URIs do not represent an XML specification.

Registration request for the dsfVerificationCode XML schema:

URI: ietf:params:xml:schema:dsfVerificationCode-1.0

Registrant Contact: See the "Author's Address" section of this document.

XML: See the "Formal Syntax" section of this document.

Registration request for the dsfRouting namespace:

URI: ietf:params:xml:ns:dsfRouting-1.0

Registrant Contact: See the "Author's Address" section of this document.

XML: None. Namespace URIs do not represent an XML specification.

Registration request for the dsfRouting XML schema:

URI: ietf:params:xml:schema:dsfRouting-1.0

Registrant Contact: See the "Author's Address" section of this document.

XML: See the "Formal Syntax" section of this document.

9. Security Considerations

The data supported by the Data Set File (DSF) format MAY include information that needs to be protected with the use of a secure transport. For example, the <dataSet:fAuthInfo> field provides object authorization information, and as described in [[RFC5731](#)], both the client and the server MUST ensure that it is stored and exchanged with high-grade encryption mechanisms. The transport leveraged to exchange the DSF containing sensitive or secure data MUST protect it from inadvertent disclosure.

XML Signature [[W3C.CR-xmlsig-core2-20120124](#)] is used in this document to verify that the Data Set originated from a trusted source and that it wasn't tampered with in transit from the source to the client to the server. To support multiple source keys, the source

certificate chain MUST be included in the <X509Certificate> elements of the Signed Def Data ([Section 3.1.2.1](#)) and MUST chain up and be verified by the server against a set of trusted certificates.

It is RECOMMENDED that signed definition data does not include white-spaces between the XML elements in order to mitigate risks of invalidating the digital signature when transferring of signed codes between applications takes place.

Use of XML canonicalization SHOULD be used when generating the signed code. SHA256/RSA-SHA256 SHOULD be used for digesting and signing.

The size of the RSA key SHOULD be at least 2048 bits.

10. Normative References

[I-D.gould-eppext-verificationcode]

Gould, J., "Verification Code Extension for the Extensible Provisioning Protocol (EPP)", [draft-gould-eppext-verificationcode-04](#) (work in progress), September 2016.

[iso13239]

the International Organization for Standardization, "ISO 13239", December 2007,
<http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=37010>.

[itu-t-V.42]

International Telecommunication Union, "ITU-T V.42 Series V: Data Communication Over the Telephone Network", March 2002, <<https://www.itu.int/rec/T-REC-V.42-200203-I>>.

[RFC2045]

Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", [RFC 2045](#), DOI 10.17487/RFC2045, November 1996, <<https://www.rfc-editor.org/info/rfc2045>>.

[RFC2119]

Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.

[RFC3339]

Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", [RFC 3339](#), DOI 10.17487/RFC3339, July 2002, <<https://www.rfc-editor.org/info/rfc3339>>.

[RFC3688]

Mealling, M., "The IETF XML Registry", [BCP 81](#), [RFC 3688](#), DOI 10.17487/RFC3688, January 2004, <<https://www.rfc-editor.org/info/rfc3688>>.

[RFC5234]

Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, [RFC 5234](#), DOI 10.17487/RFC5234, January 2008, <<https://www.rfc-editor.org/info/rfc5234>>.

- [RFC5730] Hollenbeck, S., "Extensible Provisioning Protocol (EPP)", STD 69, [RFC 5730](#), DOI 10.17487/RFC5730, August 2009, <<https://www.rfc-editor.org/info/rfc5730>>.
- [RFC5731] Hollenbeck, S., "Extensible Provisioning Protocol (EPP) Domain Name Mapping", STD 69, [RFC 5731](#), DOI 10.17487/RFC5731, August 2009, <<https://www.rfc-editor.org/info/rfc5731>>.
- [RFC5732] Hollenbeck, S., "Extensible Provisioning Protocol (EPP) Host Mapping", STD 69, [RFC 5732](#), DOI 10.17487/RFC5732, August 2009, <<https://www.rfc-editor.org/info/rfc5732>>.
- [RFC5733] Hollenbeck, S., "Extensible Provisioning Protocol (EPP) Contact Mapping", STD 69, [RFC 5733](#), DOI 10.17487/RFC5733, August 2009, <<https://www.rfc-editor.org/info/rfc5733>>.
- [RFC5910] Gould, J. and S. Hollenbeck, "Domain Name System (DNS) Security Extensions Mapping for the Extensible Provisioning Protocol (EPP)", [RFC 5910](#), DOI 10.17487/RFC5910, May 2010, <<https://www.rfc-editor.org/info/rfc5910>>.
- [W3C.CR-xmlsig-core2-20120124]
Cantor, S., Roessler, T., Eastlake, D., Yiu, K., Reagle, J., Solo, D., Datta, P., and F. Hirsch, "XML Signature Syntax and Processing Version 2.0", World Wide Web Consortium CR CR-xmlsig-core2-20120124, January 2012, <<http://www.w3.org/TR/2012/CR-xmlsig-core2-20120124>>.

[Appendix A](#). Acknowledgements

The author wishes to acknowledge the work done by Scott Hollenbeck in the EPP [RFC 5730](#) for providing some of the concepts and text used in this document.

Special suggestions that have been incorporated into this document were provided by John Colosi, Deepak Deshpande, Scott Hollenbeck, Suzy Strier, and Srikanth Veeramachaneni.

[Appendix B](#). Change History

[B.1](#). Change from 00 to 01

1. Fixed the ABNF.
2. Added support for the 1002 "Success with all failures" result code to indicate that the file was successfully processed but that all of the records failed.
3. Updated the description for the 1000 "Success" and the 1001 "Success with failures" result codes.

[B.2](#). Change from 01 to 02

1. Made small editorial changes based upon a review.

Author's Address

James Gould
VeriSign, Inc.
12061 Bluemont Way
Reston, VA 20190
US

Email: jgould@verisign.com
URI: <http://www.verisign.com>

