[MS-FSLRDS]:
Linguistic Resource Data Structure

Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation for protocols, file formats, languages, standards as well as overviews of the interaction among each of these technologies.
- **Copyrights.** This documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you may make copies of it in order to develop implementations of the technologies described in the Open Specifications and may distribute portions of it in your implementations using these technologies or your documentation as necessary to properly document the implementation. You may also distribute in your implementation, with or without modification, any schema, IDL's, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications.
- **No Trade Secrets.** Microsoft does not claim any trade secret rights in this documentation.
- **Patents.** Microsoft has patents that may cover your implementations of the technologies described in the Open Specifications. Neither this notice nor Microsoft's delivery of the documentation grants any licenses under those or any other Microsoft patents. However, a given Open Specification may be covered by Microsoft's Open Specification Promise (available here: http://www.microsoft.com/interop/osp) or the Community Promise (available here: http://www.microsoft.com/interop/cp/default.mspx). If you would prefer a written license, or if the technologies described in the Open Specifications are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplg@microsoft.com.
- **Trademarks.** The names of companies and products contained in this documentation may be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights.
- **Fictitious Names.** The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted in this documentation are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

Reservation of Rights. All other rights are reserved, and this notice does not grant any rights other than specifically described above, whether by implication, estoppel, or otherwise.

Tools. The Open Specifications do not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments you are free to take advantage of them. Certain Open Specifications are intended for use in conjunction with publicly available standard specifications and network programming art, and assumes that the reader either is familiar with the aforementioned material or has immediate access to it.

Revision Summary

Date	Revision History	Revision Class	Comments
11/06/2009	0.1	Major	Initial Availability
02/19/2010	1.0	Minor	Updated the technical content
03/31/2010	1.01	Editorial	Revised and edited the technical content
04/30/2010	1.02	Editorial	Revised and edited the technical content
06/07/2010	1.03	Editorial	Revised and edited the technical content
06/29/2010	1.04	Editorial	Changed language and formatting in the technical content.
07/23/2010	1.04	No change	No changes to the meaning, language, or formatting of the technical content.
09/27/2010	1.04	No change	No changes to the meaning, language, or formatting of the technical content.
11/15/2010	1.04	No change	No changes to the meaning, language, or formatting of the technical content.
12/17/2010	1.04	No change	No changes to the meaning, language, or formatting of the technical content.

Table of Contents

1 Introduction

This document specifies the Linguistic Resource Data Structures. These structures provide file formats that allow linguistic dictionaries to be represented as XML files. A typical scenario for using these structures is to store linguistic data that is employed by different applications.

1.1 Glossary

The following terms are defined in [MS-GLOS]:

XML

The following terms are defined in [MS-OFCGLOS]:

automaton
dictionary
property extraction

The following terms are specific to this document:

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as described in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information. Please check the archive site, http://msdn2.microsoft.com/en-us/library/E4BD6494-06AD-4aed-9823-445E921C9624, as an additional source.

[MS-FSRS] Microsoft Corporation, "Resource Store Protocol Specification", November 2009.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, http://www.ietf.org/rfc/rfc2119.txt

1.2.2 Informative References

[HopcroftUllman] Hopcroft, John and Ullman, J., "Introduction to Automata Theory, Languages and Computation (1st ed.)", Reading Mass: Addison-Wesley. ISBN 0-201-02988-X, 1979, Introduction to Automata Theory, Languages and Computation (1st ed.)

[MS-GLOS] Microsoft Corporation, "Windows Protocols Master Glossary", March 2007.

[MS-OFCGLOS] Microsoft Corporation, "Microsoft Office Master Glossary", June 2008.

[RFC1952] Deutsch, P., "GZIP file format specification version 4.3", May 1996, http://www.ietf.org/rfc/rfc1952.txt

1.3 Structure Overview (Synopsis)

This specification documents two file formats: An **automaton** file format, and a plain text **dictionary** file format. Both file formats use **XML** files to represent linguistic information.

The automaton file format uses XML files to represent a deterministic finite state machine that contains:

- A finite set of states.
- A finite set of transitions that are annotated with transition labels.
- A finite set of output values.

The dictionary file format uses XML files to contain keys, and optional values for these keys, in plain text format.

A finite state machine (FSM) is a mechanism that reads an input string and optionally writes an output string. Starting at a specially designated state called the **start state**, the FSM reads one character of its input string and, if there is a transition to another state that is labeled with that character, the FSM performs this transition. There is a special set of states called the **final states**, which are states without transitions to other states.

A deterministic finite state machine (DFSM) is an FSM that, for every state and every input character, has only one transition to the state that is labeled with that character. The automata that these structures represent are always deterministic.

For a more comprehensive description of deterministic finite state machines and the related algorithms, see [HopcroftUllman].

Deterministic finite state machines can be employed in the encoding of linguistic data, for example, into dictionaries that are used for spell checking or **property extraction**.

1.4 Relationship to Protocols and Other Structures

The structures that are described in this document can be used as linguistic dictionaries that are uploaded to and downloaded from the Resource Store, as specified in [MS-FSRS].

The XML files that are described in this document can optionally be compressed with the "gzip" tool, as described in [RFC1952]. In that case, the files require the extension ".xml.gz"; otherwise, they require the ".xml" extension.

1.5 Applicability Statement

The structures described in this document are used to store linguistic data that is shared between applications. Typical examples are applications that create and use spell checking data, or applications that create and use property extraction data.

The dictionary files, which are ideal for smaller amounts of data, can be edited manually. The automaton files are not usually edited manually; instead, they are automatically created or updated from other dictionary sources by using the algorithms that are described in [HopcroftUllman].

1.6 Versioning and Localization

None.

1.7 Vendor-Extensible Fields

None.

2 Structures

2.1 XML Automaton Files

The terminology (such as "state", "transition") in this section is used in the same technical sense as in section 1.3.

2.1.1 Part Enumerations

None.

2.1.2 Extensions

None.

2.1.3 Global Elements

2.1.3.1 fsa Element

This element MUST be the root element for the file. In this sequence, the **fsa** element contains one **states** element, one **values** element, and one **meta** element, as follows:

- **states**: A **CT_states** element
- **values**: A **CT_values** element
- **meta**: A **CT_meta** element

The following shows the XML schema of the **fsa** element:

```
<xs:element name="fsa">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="1" maxOccurs="1" name="states" type="CT_states" />
      <xs:element minOccurs="1" maxOccurs="1" name="values" type="CT_values" />
      <xs:element minOccurs="1" maxOccurs="1" name="meta" type="CT_meta" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

2.1.4 Global Attributes

None.

2.1.5 Complex Types

2.1.5.1 CT_states

This complex type is referenced by the **fsa** element, has no attributes, and contains a list of states.

The child element of the **CT_states** element is as follows:

- **s**: A **CT_state** element

The following shows the XML schema of the **CT_states** complex type:

```
<xs:complexType name="CT_states">
  <xs:sequence>
    <xs:element minOccurs="1" maxOccurs="unbounded" name="s" type="CT_state" />
  </xs:sequence>
</xs:complexType
```

2.1.5.1.1 CT_state

This complex type referenced by the **CT_states** element represents a state of the automaton. The **CT_state** element can optionally contain transitions to other states represented by a sequence of **t** elements. If the **CT_state** element contains such transitions, the state is called a non-final state, otherwise the state is called a final state.

The child element of the **CT_state** element is as follows:

- **t**: A **CT_transition element**

The attributes of the **CT_state** element are represented in the following table.

Name	Description
i	The identifier of this state, an unsigned integer.
v	The value of this state, an unsigned integer. This is a reference to a **v** element in the same automaton, that is, the integer MUST correspond to the value of an **i** attribute of a **v** element in the same automaton.

The following shows the XML schema of the **CT_state** complex type:

```
<xs:complexType name="CT_state">
  <xs:sequence>
    <xs:element minOccurs="0" maxOccurs="unbounded" name="t" type="CT_transition" />
  </xs:sequence>
  <xs:attribute name="i" type="xs:unsignedInt" use="required" />
  <xs:attribute name="v" type="xs:unsignedInt" use="optional" />
</xs:complexType>
```

2.1.5.1.1.1 CT_transition

This complex type is referenced by the **CT_state** element, has no child elements, and represents a transition of the automaton from the state that contains this element, to the target state that is denoted by the attribute **t**. This process reads a single character, that is, the label of the transition, the attribute **l**.

The attributes of the **CT_transition** element are represented in the following table.

Name	Description
l	The label of the transition, consisting of an unsigned integer which represents the byte value of a single character. For example the character "e" is represented by its ASCII value "101". It is the responsibility of the application reading the automaton to interpret this integer value as a character in a given encoding, which can be different from the encoding of the XML file.

Name	Description
t	The target state of the transition, an unsigned integer. This is a reference to an **s** element in the same automaton, that is, the integer MUST correspond to the value of an **i** attribute of an **s** element in the same automaton.

The following shows the XML schema of the **CT_transition** complex type:

```
<xs:complexType name="CT_transition">
  <xs:attribute name="l" type="xs:unsignedByte" use="required"/>
  <xs:attribute name="t" type="xs:unsignedInt" use="required" />
</xs:complexType>
```

2.1.5.2 CT_values

This complex type is referenced by the **fsa** element, has no attributes, and contains a list of output values represented by a non-empty sequence of **v** elements.

The child element of the **CT_values** element is as follows:

- **v**: A **CT_value** element

The following shows the XML schema of the **CT_values** complex type:

```
<xs:complexType name="CT_values">
  <xs:sequence>
    <xs:element minOccurs="1" maxOccurs="unbounded" name="v" type="CT_value" />
  </xs:sequence>
</xs:complexType>
```

2.1.5.2.1 CT_value

This complex type is referenced by the **CT_values** element, has no child elements, and represents the output value of a state of the automaton, which is referenced by the **v** attribute of an **s** element.

The attributes of the **CT_value** element are represented in the following table.

Name	Description
i	The identifier of this value, an unsigned integer.
v	The output value, a string in the encoding denoted by the "encoding" attribute of the file.

The following shows the XML schema of the **CT_value** complex type:

```
<xs:complexType name="CT_value">
  <xs:attribute name="i" type="xs:unsignedInt" use="required" />
  <xs:attribute name="v" type="xs:string" use="required" />
</xs:complexType>
```

2.1.5.3 CT_meta

This complex type is referenced by the **fsa** element, has no child elements, and contains information about the start state of the automaton.

The attributes of the **CT_meta** element are represented in the following table.

Name	Description
startstate	This attribute represents the start state of the automaton as an unsigned integer identifier.

The following shows the XML schema of the **CT_meta** complex type:

```
<xs:complexType name="CT_meta">
  <xs:attribute name="startstate" type="xs:unsignedInt" use="required" />
</xs:complexType>
```

2.1.6 Simple Types

None.

2.2 XML Dictionary Files

2.2.1 Part Enumerations

None.

2.2.2 Extensions

None.

2.2.3 Global Elements

2.2.3.1 dictionary Element

This element, which has no attributes, MUST be the root element for the file. This element optionally contains a sequence of **entry** elements.

The **dictionary** element contains the following child element:

- **entry**: A **CT_entry** element

The following shows the XML schema of the **dictionary** element:

```
<xs:element name="dictionary">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" name="entry" type="CT_entry" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
```

2.2.4 Global Attributes

None.

2.2.5 Complex Types

2.2.5.1 CT_entry

This complex type is referenced by the **dictionary** element, has no child elements, and represents an entry in the dictionary containing a key and, optionally, a value.

The attributes of the **CT_entry** element are represented in the following table.

Name	Description
key	This attribute denotes the key of the dictionary entry, that is, the entry string itself. The value of the attribute is a string in the encoding of the XML file.
value	This optional attribute denotes a value that is associated with a dictionary entry. The value of the attribute is a string in the encoding of the XML file.

The following shows the XML schema of the **CT_entry** complex type:

```
<xs:complexType name="CT_entry">
  <xs:attribute name="key" type="xs:string" use="required" />
  <xs:attribute name="value" type="xs:string" use="optional" />
</xs:complexType>
```

2.2.6 Simple Types

None.

3 Structure Examples

3.1 Example XML Dictionary File

The following example file represents a dictionary in the key/value format. The key "test" has the integer value "25", the key "others" has the value "other", and the key "none" is not associated with a value.

```
<?xml version="1.0" encoding="UTF-8"?>
<dictionary>
  <entry key="test" value="25" />
  <entry key="none" />
  <entry key="others" value="other"/>
</dictionary>
```

3.2 Example XML Automaton File

This section describes an automaton based on the example shown in section 3.1 in the automaton format. The following figure represents the states of the automaton with circles; arrows represent the transitions between states.

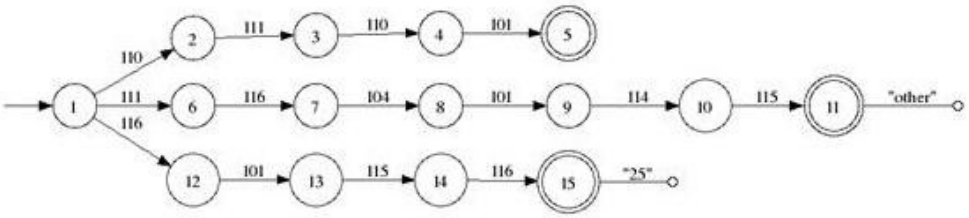

Figure 1: Example of a finite state machine represented by an automaton

The start state is the state numbered with 1. The final states, that is, the states that have optional output values are represented by double circles in the figure. The output values of the final states are shown in quotation marks. The transitions are identified by the byte values of their labels; for example, "e" is represented as "101".

This finite state machine is represented by the following example of an automaton XML file. The XML information in this example represents the same information that is described the example shown in section 3.1 in the automaton format.

```
<?xml version="1.0" encoding="UTF-8"?>
<fsa>
  <states>
    <s i="5" v="1"/>
    <s i="4">
      <t l="101" t="5"/>
    </s>
    <s i="3">
      <t l="110" t="4"/>
    </s>
    <s i="2">
      <t l="111" t="3"/>
    </s>
    <s i="11" v="2"/>
```

```
    <s i="10">
      <t l="115" t="11"/>
    </s>
    <s i="9">
      <t l="114" t="10"/>
    </s>
    <s i="8">
      <t l="101" t="9"/>
    </s>
    <s i="7">
      <t l="104" t="8"/>
    </s>
    <s i="6">
      <t l="116" t="7"/>
    </s>
    <s i="15" v="3"/>
    <s i="14">
      <t l="116" t="15"/>
    </s>
    <s i="13">
      <t l="115" t="14"/>
    </s>
    <s i="12">
      <t l="101" t="13"/>
    </s>
    <s i="1">
      <t l="110" t="2"/>
      <t l="111" t="6"/>
      <t l="116" t="12"/>
    </s>
  </states>
  <values>
    <v i="1" v=""/>
    <v i="2" v="other"/>
    <v i="3" v="25"/>
  </values>
  <meta startstate="1"/>
</fsa>
```

3.3 Example XML Empty Automaton File

The following example file represents an empty automaton. It contains one state that is both an initial state and a final state, and which has an empty output value.

```
<?xml version="1.0" encoding="UTF-8"?>
<fsa>
  <states>
    <s i="1" v="1"/>
  </states>
  <values>
    <v i="1" v=""/>
  </values>
  <meta startstate="1"/>
</fsa>
```

4 Security Considerations

None.

5 Appendix A: XML Schemas

For ease of implementation, the following XML Schemas are provided.

5.1 Automata Files XML Schema

The XML automaton files adhere to the following XML schema.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <!-- **************** Main element structure ***************** -->
  <xs:element name="fsa">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="1" maxOccurs="1" name="states" type="CT_states" />
        <xs:element minOccurs="1" maxOccurs="1" name="values" type="CT_values" />
        <xs:element minOccurs="1" maxOccurs="1" name="meta" type="CT_meta" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <!-- **************** Complex types ***************** -->
  <xs:complexType name="CT_states">
    <xs:sequence>
      <xs:element minOccurs="1" maxOccurs="unbounded" name="s" type="CT_state" />
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="CT_values">
    <xs:sequence>
      <xs:element minOccurs="1" maxOccurs="unbounded" name="v" type="CT_value" />
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="CT_meta">
    <xs:attribute name="startstate" type="xs:unsignedInt" use="required" />
  </xs:complexType>

  <xs:complexType name="CT_state">
    <xs:sequence>
      <xs:element minOccurs="0" maxOccurs="unbounded" name="t" type="CT_transition" />
    </xs:sequence>
    <!-- state id -->
    <xs:attribute name="i" type="xs:unsignedInt" use="required" />
    <!-- state output: A value (integer reference to a value id) -->
    <xs:attribute name="v" type="xs:unsignedInt" use="optional" />
  </xs:complexType>

  <xs:complexType name="CT_value">
    <!-- value id -->
    <xs:attribute name="i" type="xs:unsignedInt" use="required" />
    <!-- -->
    <xs:attribute name="v" type="xs:string" use="required" />
  </xs:complexType>

  <xs:complexType name="CT_transition">
```

```
    <!-- transition label consisting of 1 character in the chosen encoding. -->
    <xs:attribute name="l" type="xs:unsignedByte" use="required"/>
    <!-- transition target: A state ID (integer) -->
    <xs:attribute name="t" type="xs:unsignedInt" use="required" />
  </xs:complexType>

</xs:schema>
```

5.2 Dictionary Files XML Schema

The XML dictionary files adhere to the following XML schema.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <!-- *************** Main element structure ***************** -->
  <xs:element name="dictionary">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="0" maxOccurs="unbounded" name="entry" type="CT_entry"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <!-- *************** Complex types ***************** -->
  <xs:complexType name="CT_entry">
    <xs:attribute name="key" type="xs:string" use="required" />
    <xs:attribute name="value" type="xs:string" use="optional" />
  </xs:complexType>
</xs:schema>
```

6 Appendix B: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include released service packs:

- Microsoft® FAST™ Search Server 2010

Exceptions, if any, are noted below. If a service pack or Quick Fix Engineering (QFE) number appears with the product version, behavior changed in that service pack or QFE. The new behavior also applies to subsequent service packs of the product unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms SHOULD or SHOULD NOT implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that the product does not follow the prescription.

7 Change Tracking

No table of changes is available. The document is either new or has had no changes since its last release.

8 Index