FIX Antenna C++/.NET dictionaries format

Overview

FIX dictionaries are XML files containing metadata, that describe FIX messages, fields, and groups. FIX dictionaries are required by the FIX Engine for runtime message parsing, serialization and validation. Every FIX session requires a specific FIX dictionary to run. Dictionaries can be shared between sessions or used exclusively.

This metadata (dictionary) can be expressed in several formats, i.e.:

- EPAM products have their own format, that closely mimics the structure of official FIX specification, leveraging "block" elements and references, defining type system, and some more. Additionally, besides runtime metadata, it also contains human-readable textual details about message & tag meaning.
- The QuickFIX dictionary is natively supported by FIX Antenna C++ since FIX Antenna C++ 2.28.0 and no longer requires manual conversion to FIX Antenna format. The QuickFIX dictionary has a simplified format that does not use "blocks" and "references", and instead describes the same messages in a flatter format. It also skips textual details on content.
  QuickFIX Dictionary doesn't support conditional validation.

  > The QuickFIX dictionary is not yet natively supported by FIX Antenna JAVA or FIX Antenna .NET Core

A FIX dictionary can be:

- a self-sufficient dictionary that describes the FIX protocol;

> FIX Antenna JAVA & FIX Antenna .NET Core dictionaries have the same format as FIX Antenna C++/.NET dictionaries

- an additional dictionary that describes changes that are to be applied to another existing dictionary.

> FIX Antenna JAVA & FIX Antenna .NET Core do not support additional dictionaries

One or more FIX protocols are required for FIX session work.

FIX dictionaries are combined into parsers. Parsers are used to validate incoming messages, i.e. they check if messages satisfy defined protocols.

When FIX Antenna is started, all dictionaries from *DictionariesFilesList* are loaded. If dictionaries are standard, predefined standard parsers (like FIX40; FIX41; FIX42; FIX43; FIX44; FIXT11) are created basing on them. If any dictionary from *DictionariesFilesList* is custom, the parser is not automatically created and therefore such parsers should be listed manually in *AdditionalParsersList* parameter.

Overall there should be one session protocol (protocol which describes messages of session level) and one or more application protocols (protocols which describe messages of application level) defined in common parsers list (i.e. in standard parsers list and additional parsers list together).

Description for all standard FIX protocols can be found on fixopaedia.

Dictionary structure

Standalone dictionary structure

Below is the common structure of the standalone dictionary with used tags and attributes description.

Structure	Description
<fixdic>	Root tag of the dictionary.
@xmlns	xmlns="http://www.b2bits.com/FIXProtocol"
@xmlns:xsi	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

	@xsi:schemaLocation	xsi:schemaLocation="http://www.b2bits.com/FIXProtocol fixdic.xsd"
	@id	ID of the used dictionary. Valid values are: • FIX40 • FIX41 • FIX42 • FIX43 • FIX44 • FIX50 • FIX50SP1 • FIX50SP2 • FIX50SP2EP • FIXT11 • ID of the custom dictionary (refer to How to use custom dictionaries with FIXEdge for details)
	@fixversion	FIX version of the used dictionary. E.g.: 4.4, 4.2, T1.1, etc.
	@title	The name of the dictionary. E.g.: FIX 5.0 SP2 EP, FIX 4.4 (with errata 20030618)
	@version	Dictionary version.
	@date	Date of the last dictionary update.
	<typelist>	Section which describes types used in the dictionary.
	<typedef>	Definition of the type.
	@type	Type name. E.g.: Percentage, Amt, char
	@extends	Specified if the type extends one of existing types. E.g.: • <typedef type="Percentage" extends="float"> • <typedef type="Boolean" extends="char">
	@valuetype	Type of the value (specifies either type is multiple value type and type of the underlying value).
	@displayname	Name to be displayed (specifies either type is multiple value type and type of the underlying value).
	<descr>...</descr>	Description of the type.
	</typedef>	-
	</typelist>	-
	<fielddic>	Section which describes fields used in the dictionary.
	<valblockdef>	Definition of the block of element's values. Can be zero or more elements. E.g. for countries, currencies, products, securities, etc.
	@id	Unique identifier of the block.
	@name	Name of the block.
	<item>	Item of the block - name of the item.
	@val	Serial number of the item.
	@id	Unique identifier of the item. Id is an optional attribute and is used when description of an item value is longer than 4 words or has any special characters.
	<msgref>	Reference to the message where mentioned item is used - name of the message.
	@msgtype	Type of the reference message.
	</msgref>	-
	</item>	-
	<range>	Range of values to be supported - purpose of the range.
	@minval	Minimum value to be supported.
	@maxval	Maximum value to be supported.
	@type	Type of values.
	</range>	-

<multi>	List of items. The field can contain multiple items separated by space.
<item>...</item>	The structure is the same as described for ***fixdic/fielddic/valblockdef/item.***
</multi>	-
<descr>...</descr>	Description.
</valblockdef>	-
<fielddef>	Definition of the field.
@tag	Tag of the field.
@name	Name of the field.
@type	Type of the field.
@transport	Transport.
@lenfield	Only for fielddef with type="data" or type="XMLData"
<alias>...</alias>	Alternative field name.
<multi>	List of items. The field can contain multiple items separated by space.
<item>...</item>	The structure is the same as described for ***fixdic/fielddic/valblockdef/item.***
</multi>	-
<item>...</item>	Can be zero or more item elements in the field definition. Only one item can be specified for the field. The structure is the same as described for ***fixdic/fielddic/valblockdef/item.***
<range>...</range>	The structure is the same as described for ***fixdic/fielddic/valblockdef/range.***
<valblock idref="..."/>	Reference to the id of the *<valblockdef>* element.
<descr>...</descr>	Description of the field.
</fielddef>	-
</fielddic>	-
<msgdic>	Section which describes messages used in the dictionary.
<blockdef>	Definition of the block. Can be zero or more definitions.
@id	Identifier of the block definition.
@name	Name of the block definition.
@transport	Transport.
<field>	Definition of the field.
@tag	Field tag.
@name	Field name.
@req	Specified if the field is mandatory or not.
@condreq	Specified if there is a condition when the field is mandatory.
<comment>...</comment>	Comment.
</field>	-
<block>	Block description.
@idref	Reference to the id of the block.
@req	Specified if the block is mandatory or not.
@condreq	Specified if there is a condition when the field is mandatory.
<comment>...</comment>	Comment.
</block>	-
<group>	Group description.
@nofield	Tag of the group.
@startfield	Start field.
<field>...</field>	The structure is the same as described for ***fixdic/msgdic/blockdef/field.***
<block>...</block>	The structure is the same as described for ***fixdic/msgdic/blockdef/block.***

<group>...</group>	The structure is the same as described for ***fixdic/msgdic/blockdef/group.***
</group>	-
<descr>...</descr>	Description.
</blockdef>	-
<msgdef>	Definition of the message. Can be zero or more definitions.
@msgtype	Message type.
@name	Message name.
@admin	Defines if the message is session level or application level message.
<alias>...</alias>	Alternative message name. Can be zero or more elements.
<field>...</field>	The structure is the same as described for ***fixdic/msgdic/blockdef/field.***
<block>...</block>	The structure is the same as described for ***fixdic/msgdic/blockdef/block.***
<group>...</group>	The structure is the same as described for ***fixdic/msgdic/blockdef/group.***
<descr>...</descr>	Description.
</msgdef>	-
</msgdic>	-
<descr>...</descr>	Description.
</fixdic>	-

Additional dictionary specifics

Below is the commom structure of the additional dictionary with specific tags and attributes description.

Structure	Description
<fixdics>	Root tag of the additional dictionary.
<update>	Dictionary update operation. Can be zero or one *<update>* element.
<fixdic>	Dictionary to be updated. Can be zero or more *<fixdic>* elements.
@id	ID of the additional dictionary.
@parent_id	ID of the original dictionary additional dictionary based on. Valid values are: • FIX40 • FIX41 • FIX42 • FIX43 • FIX44 • FIX50 • FIX50SP1 • FIX50SP2 • FIX50SP2EP • FIXT11
@fixversion	FIX version additional dictionary based on.
@title	The name of the additional dictionary.
@version	Additional dictionary version.
@date	Date of the last additional dictionary update.
...	*<typedef>/<valblockdef>/<fielddef>/<blockdef>/<msgdef>* to be added/updated. The structure of listed elements is the same as described in standalone dictionary structure.
</fixdic>	-
</update>	-
</fixdic>	-

Please see the article How to use custom dictionaries for the cases of additional dictionaries usage.

> There is no way to remove elements from the base FIX dictionary

How to manage a list of dictionaries in FIXAntenna based applications

List of dictionaries to be used by FIXAntenna based applications can be managed using the ***DictionariesFilesList*** property in `engine.properties` configuration file.

This parameter contains a list of names of XML files with definitions and extensions of the FIX protocols delimited by semicolon.

> QuickFIX formatted dictionaries use a capitalized file name without an extension at the end as a dictionary ID.
>
> For example, for a dictionary with the filename FIX44.xml, the FIX protocol version/parser id is FIX44

It may contain both standard and custom dictionaries:

```
DictionariesFilesList = ../../data/fixdic40.xml;../../data/fixdic41.xml;../../data/fixdic42.xml;../../data
/fixdic43.xml;../../data/fixdic44.xml;../../data/fixdic50.xml;../../data/fixdic50sp1.xml;../../data/fixdic50sp2.
xml;../../data/fixdict11.xml;../../data/additional_dict.xml
```

Dictionary customization

As mentioned above, dictionaries can be customized for the needs of a particular exchange.

Refer to How to use custom dictionaries with FIXEdge article for details.

How to check that dictionary doesn't have errors

Simple Client can be used in order to check that the dictionary doesn't have errors.

Below is the step-by-step instruction how to check the dictionary with the help of Simple Client:

1. Open the folder with SimpleClient;
2. Specify prepared dictionary in *DictionariesFilesList* property in `engine.properties` configuration file Simple Client refers on;
3. Run the Simple Client;
4. Check the results of the launch:
    a. If the specified dictionary doesn't have errors, Simple Client will be launched successfully;
    b. If the specified dictionary has any error, Simple Client won't be started and the message with the reason will appear: