Comments on EXI Profile, W3C Draft 31 July 2012
Rumen Kyusakov
2012-09-28

DISCLAIMER: The views and opinions expressed here are solely based on the author's experience and knowledge of the EXI specification and the EXI Profile. Many of the conclusions and comments might be biased by misunderstanding of these specifications or the author's lack of broader knowledge in the area.

1)
Subject: Evaluations of EXI in application areas with memory restrictions
Section: 1. Introduction
Paragraph: "Certain evaluations of EXI in the context of such areas..."

Comments:
It is good to have access to these evaluations. From my own perspective and use cases this profile will not bring much benefits. I am working with very small messages over low-bandwidth wireless links (up to 250 kb/s). Of course, memory is an issue but the compression is even more important. It would be good to have some statistics on what are the trade-offs (memory usage vs compression) by using the Profile in different use-cases (size and structure of the messages).

2)
Subject: Limiting the number of build-in grammars and Limiting the evolutions of build-in grammars
Section: 2. Grammar Capping
Paragraph: Whole section

Comments:
In the grammar capping section it is given a single mechanism to limit the number of build-in grammars by using the xsi:type attribute (possibly with xsd:anyType). There is no mentioning of the second very distinct mechanism described later on: limiting the evolutions of build-in grammars by restricting the number of dynamic productions inserted. Mixing up between these two mechanisms in the spec is the main source of confusion for me.

Example:
In the second paragraph:
"Note that the EXI profile can only limit grammar learning for schema-informed EXI streams but not for schema-less EXI streams. In the case where no schema is available, the "schemaId" element may be set to the empty value, so that all grammars derived from the built-in XML Schema types become available through xsi:type grammar switching, in particular the gramar corresponding to the xsd:anyType complex type."

The EXI Profile can also limit the "grammar learning" even in schema-less streams with the second mechanism (limiting dynamic productions). Without the information for the second mechanism the paragraph should read 'the "schemaId" element MUST be set to the empty value' and not 'may'.

(Note also the spelling error of "gramar" in the paragraph)

3)
Subject: Event code length for xsi:type attribute event
Section: 2.1 Grammar Learning Disabling Mechanism
Paragraph: "The xsi:type attribute event MUST always be represented by the AT(*) production whose event code length is 2."

Comments:
I am sure there is a good reason to require this but I cannot deduct it. Maybe a hint in the spec will make the job of the implementers easier. My guess is that this is connected to the separation of the "original" xsi:type attributes and the xsi:type attributes added because of the Profile.

(Note also: "For a given element E, the disabling of grammar learning E ..."
should be "For a given element E, the disabling of grammar learning for E ...";
"..., the following rule happens:" it is better "..., the following rules apply:")

4)
Subject: Limiting the evolutions of build-in grammars
Section: 2.1 Grammar Learning Disabling Mechanism
Paragraph: "If grammar learning is disabled in the case of a production insertion in a given grammar, named G, the following rule happens ..."

Comments:
This is the first time the second mechanism (Limiting the evolutions of build-in grammars) is mentioned in the spec. It is almost impossible to make the connection with the rest of the spec without reading it few/several times. In my opinion this mechanism should be clearly described and differentiated with the first one (using xsi:type).
A proper way of doing this is by mentioning it in the 2. Grammar Capping sections and then describing it as a separate subsection. Giving it a proper name would also help a lot as "grammar learning is disabled in the case of a production insertion" is definitely not a good one.

5)

Subject: Confusing description
Section: 2.2 Grammar Learning Disabling Parameters
Paragraph: "Grammar learning is disabled...
            ...
            Grammar learning is disabled in the case of a production ..."

Comments:
There should really be a better name for "Grammar learning is disabled in the case of a production ..." A suggestion: "Disabling the evolution of build-in grammars"?

6)
Subject: The Profile parameters
Section: 4. Parameters representation
Paragraph: "In such a case, the actual profile parameters (or fine-grained capping strategies) should be defined by an out-of-bound mechanism."

Comments:
The maximumNumberOfBuiltInElementGrammars and localValuePartitions parameters should not be mandatory to be set/communicated-out-of-bound in my opinion. For the number of build in grammar the encoder knows the number of the grammars so no need to be included in the header. On the decoder side if the implementation is "smart" it won't create a new build-in element grammar before examining the next event. If it is a xsi:type="SOME_EXISTING_TYPE", there is no need to create a new build-in element grammar.
For the localValuePartitions again - no issues for the encoder if the parameter is not included in the header. The decoder on the other hand can create local value tables only if it has enough memory to do so. If it cannot support local value tables, it will only expect string values represented literally in the stream. Receiving a string as a compact identifier will produce error during decoding in any way if the decoder does not have enough memory.

7)
Subject: Unclear
Section: C Prefix Workarounds (Non-Normative)
Paragraph: All

Comments:
All the paragraph could be summarize by saying that the application should use only ONE prefix per namespace. Then the two requirements are fulfilled.

8)
Subject: Striping of Profile xsi:type attributes
Section: E.2 Grammar Restriction Decoder Considerations
Paragraph: "An EXI profile decoder SHOULD strip any xsi:type attribute with the xsd:anyType value from the infoset that corresponds to the grammar learning disabling mechanism."

Comments:
It is strange to have "SHOULD" in a (Non-Normative) section. In any case a non-Profile EXI decoder will include the Profile xsi:type attributes.

8)
Subject: Summary
Section: All spec
Paragraph: All

Comments:
The conformance with the Profile does not guarantee in any way a bounded memory for processing (for example not setting valuePartitionCapacity to 0 is enough to make a Profile complying implementation consuming huge amount of memory). Moreover a non-conforming to the Profile implementation can still use the Limiting the number of build-in grammars mechanism without declaring that in the EXI header.
Without being an expert in the field my personal experience is that the EXI spec and even more the EXI Profile spec are very much requiring the users to be aware of some implementation issues in order to use the format in an optimal way.
For example, I was involved to some small extend in the Smart Energy Profile 2.0 standard/draft that considers using EXI. Main goal is compactness but also lean processing. The knowledge on what is the optimal way to set all the parameters (schemaId, preserve, selfContained, valueMaxLength, valuePartitionCapacity and now the Profile parameters) is simply not there.
The great flexibility provided by having so many things to tune is working against the wide spread adoption. The users should not be burdened by knowing what all these parameters mean especially the Profile ones that are purely connected to the implementation.
Again, without claiming to be an expert and have a wide range of use cases in mind, I think a more restrictive Profile that actually provides some guarantees and not so much freedom of setting implementation parameters will be much more useful. In any case the actual use of the EXI Profile will require some "Application-specific Profile" that defines the fixed values for all these parameters. So why not preset all these parameters and provide some guarantees with a good memory usage vs compression trade-off? I know one size does not fit all, but then why defining an EXI Profile in the first place?
In any case, if every single application sets these parameters in a different way then the implementations won't be able to interoperate due to some memory issues or size constrains. What I

see as a need is a Profile that provides some guarantees for resource constrained devices. For example, a conforming implementations should use only:

- maximumNumberOfBuiltInElementGrammars = 0
- valuePartitionCapacity = 0
- maximumNumberOfBuiltInProductions = 0
- localValuePartitions = 0
- selfContained = false
- preserve = all false
- fragment = false
- either strict schema mode OR "schemaId" set to the empty value schema-less mode

Note that by fixing the values of these parameters will not only make the RAM consumption predictive but also lower the programming memory which is also an issue in some cases. The implementations of the Profile will be much simpler as a codebase as compared to implementing the generic version of the Profiles as it is now. In fact, the Profile as it is now will just extend the code and make it more complex which I believe is the opposite of what is needed for embedded applications.