Invocations of streamable processing

This document describes the various ways streamable processing, and its counterpart, non-streamable processing, can be instigated by either the API or by using a declared streamable construct.

It raised a few questions:

1) What happens if a non-streamable construct is given a streamed node (i.e., on invocation). I couldn't find this written out in the spec.
   A: I think the consensus is that no error is raised and the streamed node is buffered in full (notwithstanding the processor's freedom to stream it anyway if it can)

2) What happens if a streamable invocation construct is given a non-streamed node?
   A: I think what should happen is that it is treated as a grounded node, i.e., the node is not processed using streaming, but we don't specify this (we could consider this an API matter). See also Bug 29690, which shows that we currently mandate that you *must* use streaming.

3) Can the IMS be a sequence of streamed, or mixed nodes?
   A: I think it can, this is just API dependent. We have no Note or suggestion in either direction though.

4) Can the IMS be something else than a streamed document node?
   A: Yes, it can be of any node type, though I don't think we say this very explicitly.

5) Can the IMS be a streamed string or continuous sequence of items (this was a question during XML London 2016)?
   A: No, as a sequence has a beginning and end in XDM. Again, I don't think we are explicit about this, but it follows from the XDM definition of sequences and atomics.

6) Can fn:unparsed-text-lines be streamed?
   A: No, because the function is considered deterministic, multiple invocations *must* return the same string.

Key

(use mouseover on a term in the blue table to see a short description popup, doesn't work in all PDF readers):

NORMAL	normal operation, non streamed	Means: input tree is not a streamed node, process without streaming, similar to XSLT 2.0 invocation
MUST	must stream	Current rules mandate that streamable processor *must* process the input tree using streaming
ATTEMPT	attempt processing	For non-streaming processors, if the input tree is a streamed node, it *must* attempt to process it by buffering the input tree as a whole, or fail doing so.
ATT-CHOOSE	attempt / E3430 / non-strm	Processor must provide options for each of the following (19.10 Rule 2): a) Attempt streaming if it can b) Or raise XTSE3430 c) Process non-streaming
ERR-CHOOSE	error? / normal / E3430	When input tree is not streamed, construct is declared streamable but not guaranteed streamable, then the consensus seems to be, at user option: a) Raise error, because it cannot attempt streaming (?) b) Process non-streaming, do not raise error c) Raise XTSE3430
OPT-ERR	opt / error? /non-strm	If IMS is a streamed node, initial construct is not streamable, then behavior is processor dependent, consensus seems to be: a) Optionally attempt streaming b) Raise an error (?) c) Process non-streaming
ERR-NORM	error? / normal	When input tree is not a streamed node, construct tree is streamable and is invoked as initial construct. This scenario is not clear from the spec, I think.

Key to reading the table:

Heading	Description
input	shows a method with which you can give the processor a streamed document or node
invocation method	one of the three ways of invoking a transformation
declared streamable	whether or not the *invocation construct* (term is mine) is declared streamable
guaranteed streamable	whether static analysis shows that any construct is guaranteed streamable
streaming processor	how a streaming processor is supposed to behave
non-streaming processor	how a non-streaming processor is supposed to behave

Table of invocation variants

Input	invocation method	declared streamable	Guaranteed streamable	Streaming processor	Non-streaming processor
IMS is streamed	apply-templates	no	no[1]	opt-error?	attempt
		yes	yes	must	
		yes	no	attempt-choose	
IMS is non-streamed	apply-templates	no	no[1]	normal	normal
		yes	yes	error-or-normal	
		yes	no	error-choose	
IMS a not a node	apply-templates	no	no[1]	normal	normal
		yes	yes	error-or-normal	
		yes	no	error-choose	
context-item or global context item is streamed node	apply-templates or call-template	–	–	API matter, will likely attempt processing non-streaming	process CI non-streaming
doc / document / collection return a streamed node	n/a	n/a	n/a	F&O allows non-deterministic behavior upon request, so I think there's room for this, but the spec has no rules here	n/a
xsl:stream tree is streamed	n/a	no[2]	no[1]	must not stream	attempt
		yes	yes	must	
		yes	no	attempt-choose	
xsl:stream tree is non-streamed	n/a	no	no[1]	normal	normal
		yes	yes	error-or-normal	
		yes	no	error-choose	
xsl:merge-source tree is streamed	n/a	no	no[1]	opt-error?	attempt
		yes	yes	must	
		yes	no	attempt-choose	
xsl:merge-source tree is non-streamed	n/a	no	no[1]	normal	normal
		yes	yes	error-or-normal	
		yes	no	error-choose	
xsl:function arg is streamed	initial-function	no	no[1]	opt-error?	attempt
		yes	yes	must	
		yes	no	attempt-choose	
xsl:function arg is non-streamed	initial-function	no	no[1]	normal	normal
		yes	yes	error-or-normal	
		yes	no	error-choose	

xsl:param initiated to a streamed node	n/a	no	no[1]	Spec has no rules here, API may allow it, but behavior will be implementation-dependent	process param non-streaming
		yes	yes		
		yes	no		
extension functions or instructions providing a streamed node	n/a	no	no[1]	implementation-defined[3]	implementation-defined
		yes	yes		
		yes	no		

[1] A non-streamable construct is automatically not guaranteed streamable, even if the invocation mandates that no streamability rule checks should take place

[2] This feature is currently under debate in issue 29472, the idea is to allow xsl:stream to behave non-streaming, similar to the fn:doc function, in which case streamability guarantees are not checked.

[3] The effect of an extension function or instruction invoking streamability is implementation defined. However, we disallow implementation defined behavior to behave differently than the spec, do we allow, say, an extension attribute to change an instruction into streamable that otherwise would not? (use-case, say you allow <xsl:variable ext:streamable="true">, which allows throughput processing of the stream in xsl:variable, and this would enforce certain streamability guarantee rules, is that OK?)