

---

# Performance Comparison of the Java Implementation of ATerms and XML

Paul Klint  
Arnold Lankamp

## Table of Contents

Introduction .....	1
Considerations .....	2
ATerms versus XML .....	2
Potential influences on measurements .....	2
Experimental Setting .....	3
The test data generator .....	3
Software versions .....	3
Hardware configuration .....	3
Measurement procedure .....	3
Results .....	3
ATerm vs XML .....	4
ATerm ASCII vs SAF .....	5
Conclusions .....	8

## Introduction

ATerms are the ubiquitous data exchange format used in The Meta-Environment. ATerms were designed with a specific application domain in mind: exchanging parse tables, parse trees and other source code related data that emerges in the context of Interactive Development Environments and tools for source code analysis and transformation.

World-wide, XML is the defacto standard for data exchange in many application domains and an obvious question to ask is how ATerms and XML compare regarding

- speed (efficiency of reading and writing data), and
- size (the size of the intermediate data).

In this document we will compare the Java implementation of the ATerm library against the most commonly used Java XML Parser implementation (Xerces), by means of a series of experiments. In addition we will also take a look at the binary encoding of ATerms (SAF) and how it measures up to the ATerm ASCII format.

**Table 1. Summary of findings (no sharing)**

Format	Write	Read	Read+Write	Size
ATerm ASCII	+/-	--	-	+/-
ATerm SAF	++	++	++	++
XML	+/-	+/-	+/-	--
XML+ZIP	--	--	--	++

For the case that there is *no sharing*, our findings are shown in Table 1, “Summary of findings (no sharing)” (page 1) and can be summarized as follows:

- Writing ATerm ASCII and XML is about as fast.
- Parsing ATerm ASCII is significantly slower than parsing XML.
- The ATerm ASCII representation is more compact than XML.
- The binary ATerm SAF representation is the fastest to read and write and yields the smallest terms.
- Compressing XML documents using ZIP is reasonably expensive.
- When the amount of sharing in a document increases, the savings ATerm SAF (both in increased speed and smaller size) increase spectacularly.

In the absence of sharing, SAF is about two times as fast and yields results that are generally over two times as small compared to ATerm ASCII. However when we add sharing to the equation, the advantage ATerm SAF has over ATerm ASCII increases considerably. The benchmarks in the section called “Real life examples” (page 7) with 'real-life' ATerm examples confirm these observations.

In the section called “Considerations” (page 2) we sketch the general considerations to be taken into account when performing such a comparison and in the section called “Experimental Setting” (page 3) we describe the experimental setting. Results are presented in the section called “Results” (page 3) and our overall conclusions can be found in the section called “Conclusions” (page 8).

## Considerations

There are various considerations that influence a fair comparison of ATerms and XML.

### ATerms versus XML

There are various differences between ATerms and XML that may affect a comparison:

- ATerms may have annotations attached to each node; these annotations may contain arbitrary ATerms; XML elements may have attributes but these may only contain a single value.
- The implementation of ATerms supports the notion of maximal subterm sharing; XML has no counterpart for this.

### Potential influences on measurements

- Although we would prefer to compare the concepts underlying ATerms and XML, it is unavoidable that we can only measure specific implementations of these concepts. The implementation and optimization effort invested in XML implementations is uncomparable to the effort invested in the ATerm implementation.
- ATerms only support US-ASCII characters, while XML supports Unicode.
- The following properties of test data may influence the result:
  - The size of the data.
  - The amount of sharing.
  - The number of different function symbols/tags that is used.
  - The width and depth of the data.

- We will compare Java implementations of ATerms and XML and the unpredictable effects of the HotSpot compiler have to be taken into account.

## Experimental Setting

First we characterize our experimental setting.

### The test data generator

Due to the functional differences between ATerms and XML, we decided to create a synthetic benchmark that creates test data in both formats. The test data generator has the following parameters that are used to characterize the generated trees:

- Total number of nodes in the tree.
- Depth of the tree.
- Number of children per node.
- Percentual chance that any given node in the tree is shared.
- The amount of different 'node names' (element types / function symbols).

### Software versions

For the benchmarks the following software was used:

- Sun JDK 1.5.0\_12
- ATerm-Java, version 1.7pre.24750.52534.
- Xerces-J, version 2.6.2 (incorporated in JDK 1.5.0\_12).

### Hardware configuration

The benchmark was executed on a machine with the following specifications:

- Intel E6420 Core 2 Duo processor (2.13Ghz).
- 1GB DDR2-800 memory.

### Measurement procedure

The results that are summarized in the chapter below have been acquired in the following manner:

- For every benchmark a separate function was created which measured the absolute elapsed time that was required to run a certain operation one hundred times.
- Before starting a benchmark the code was 'warmed-up' first, so it was (fully) compiled before executing the actual test. This prevents compilation overhead from being included in the results.

## Results

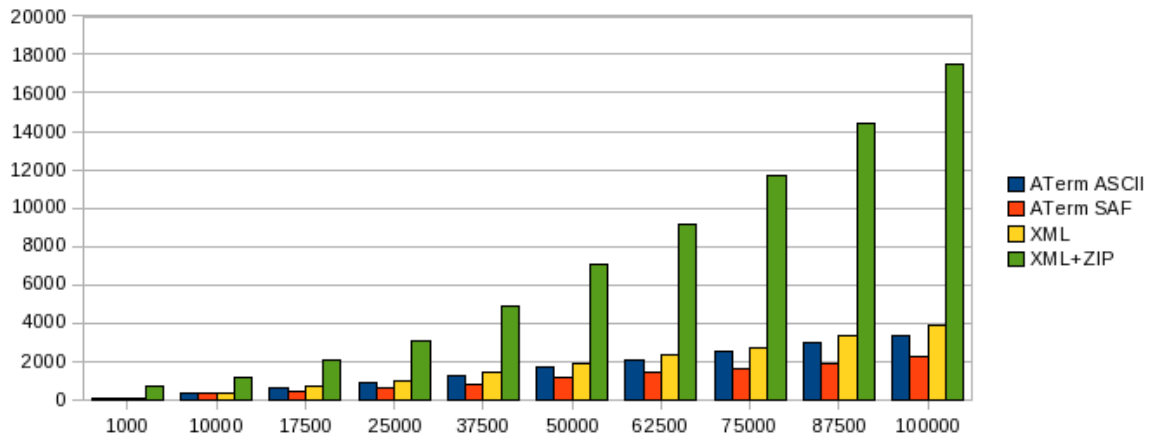
In this chapter we will take a look at the results that were produced by executing the different benchmarks.

## ATerm vs XML

In this section we will look at the transformation and parsing performance of the different formats and the size of the resulting documents. For completeness we have also included the results we obtained by applying ZIP compression to the serial representation of the produced XML documents, since this is the 'standard' way of reducing the size of large XML documents when sending them over the internet.

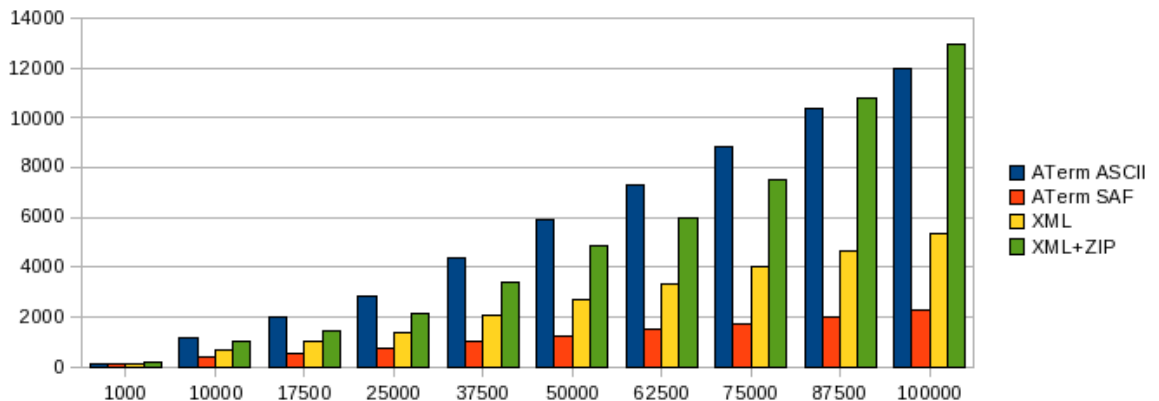
### Speed comparison

The graph below illustrates the way the ATerm formats and XML scale in terms of writing speed. Both ATerm ASCII and XML write their object model to a String in about the same amount of time; with the ATerm ASCII format being slightly on the upper hand. The binary ATerm SAF encoding is clearly the fastest. However, compressing an XML document using ZIP has a significant overhead.



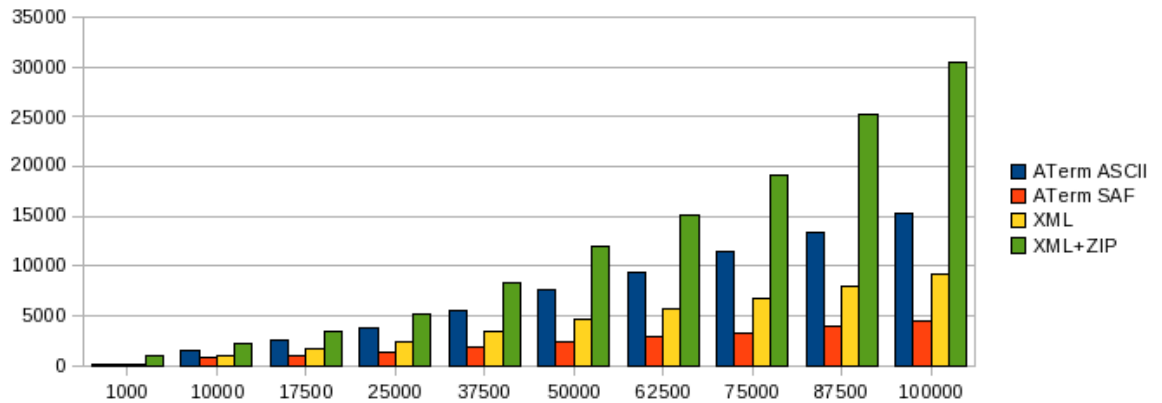
The time it takes (in ms) to serialize (write) a document with a certain number of nodes.

When it comes to parsing the tables are turned. XML is significantly faster than ATerm ASCII; this is mainly caused by the extra work needed to calculate the sharing of the nodes. ATerm SAF is the clear winner yet again. Additionally, as expected, decompression of Zipped XML documents has a noticeable performance overhead.



The time it takes (in ms) to parse (read) a document with a certain number of nodes.

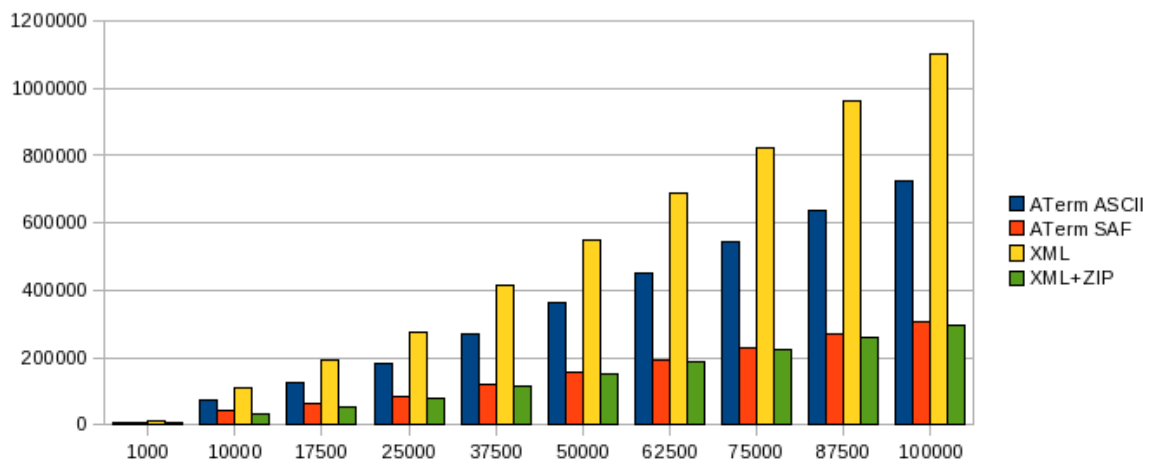
When we look at the combined serialization and parsing times, XML is the fastest of the two text representations. However even though the data does not contain any shared nodes, ATerm SAF is the clear winner. As for XML+ZIP, it's reasonably expensive.



The combined time (in ms) of serializing and parsing a document with a certain number of nodes.

## Size comparison

In terms of document size the ATerm ASCII representation is smaller than XML, mainly because it uses brackets as close tags, instead of the entire name of the element. The binary ATerm SAF encoding seems to be on par with the ZIP compression of XML.



The size (in bytes) of the serialized representation of a document with a certain number of nodes.

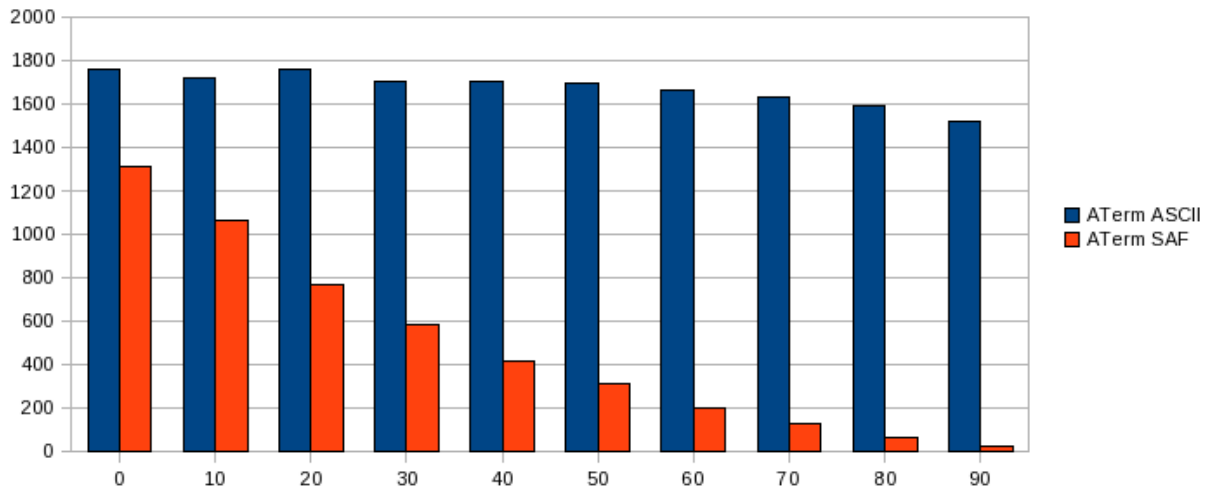
## ATerm ASCII vs SAF

For smaller documents the text (ASCII) representation of ATerms is sufficient. However when it comes to reading and writing large documents it is clearly unsuitable. Here we will take a look at how ATerm ASCII and SAF compare when sharing of noded is taken into consideration.

## Speed comparison

Writing ATerms in the SAF format ranges from moderately better to astonishingly better, when compared to ATerm ASCII, completely dependant on the amount of sharing in the document. The more sharing the less needs to be done when processing SAF.

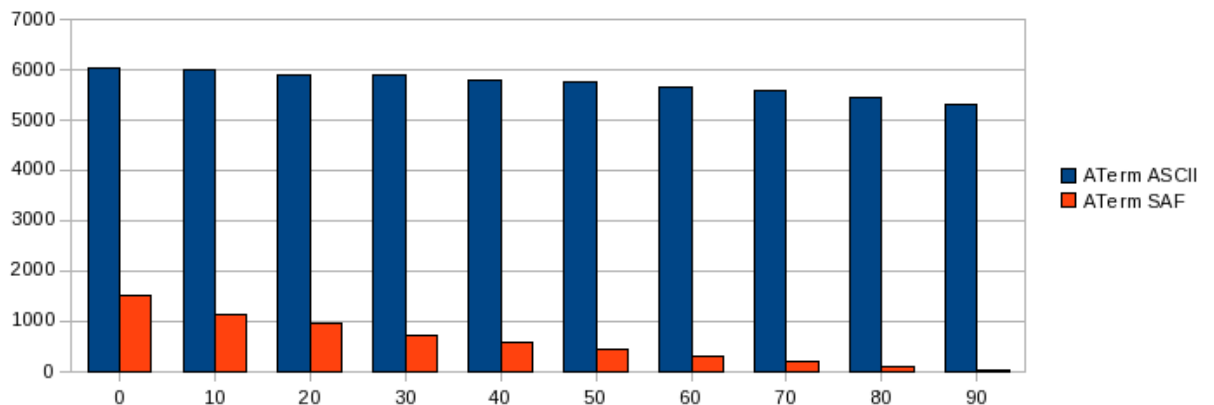
## Performance Comparison of the Java Implementation of ATerms and XML



The time (in ms) it takes to serialize (write) a document with 50000 nodes, given a certain percentage of sharing.

In terms for parsing performance, SAF is in a league of it's own; even in the complete absence of shared nodes it is still significantly faster. The more sharing the smaller the document and the less characters need to be parsed. Additionally, since the sharing is preserved in the SAF representation it does not need to be recalculated when parsing, which is one of the main bottlenecks.

Note that ATerm ASCII also becomes slightly faster as a result of sharing. This is an implementation artifact. The factory that keeps track of the shared ATerm objects uses weak-references to refer to them. Since a weak-reference needs to be checked by the VM at every garbage collection invocation (both major and minor) it incurs a certain performance overhead. If there is more sharing there are less objects in the factory and thus less overhead.

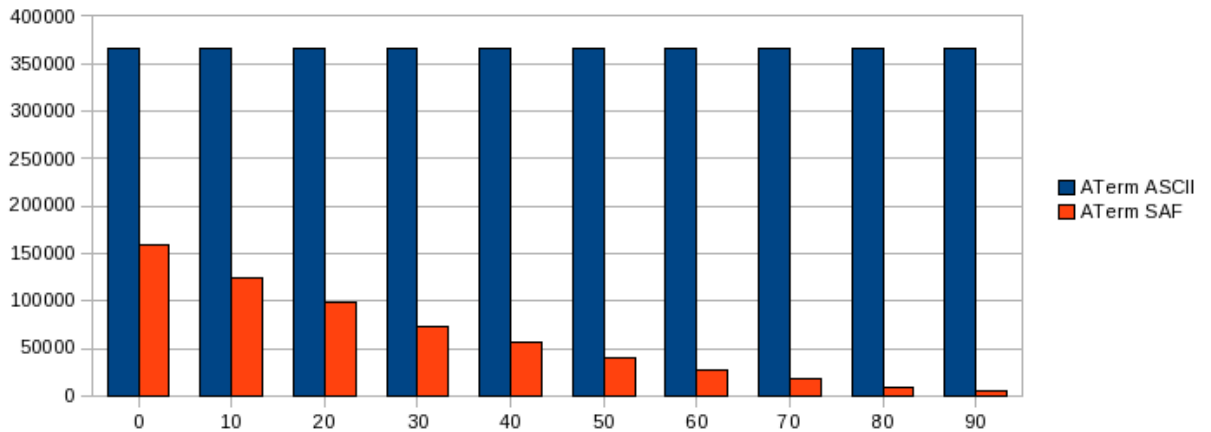


The time (in ms) it takes to parse (read) a document with 50000 nodes, given a certain percentage of sharing.

## Size comparison

In terms of document size, there are no real surprises. The more sharing, the smaller the SAF encoded document.

Performance Comparison of the Java Implementation of ATerms and XML

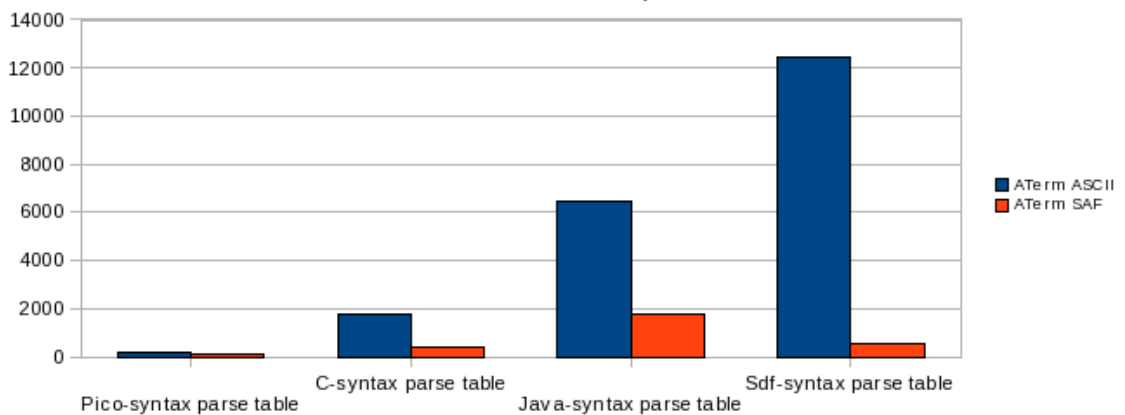


The size (in bytes) of the serialized representation of a document with 50000 nodes, given a certain percentage of sharing.

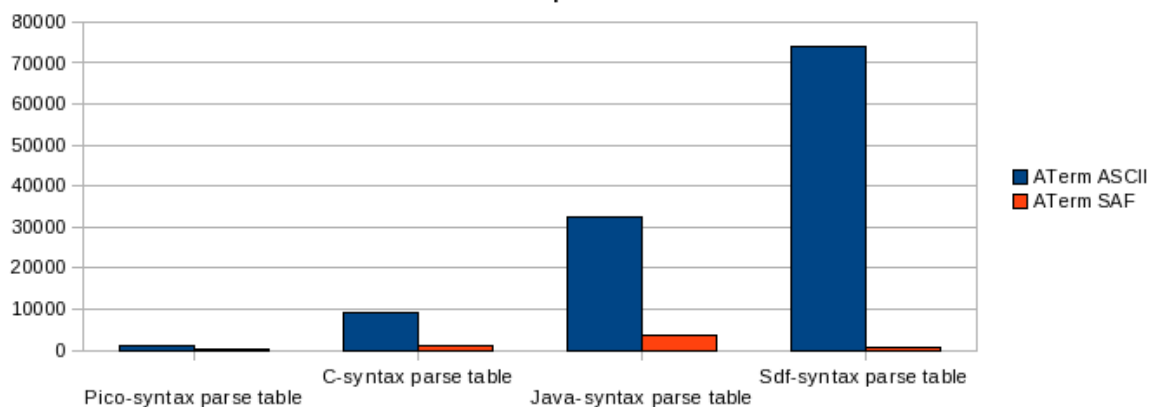
### Real life examples

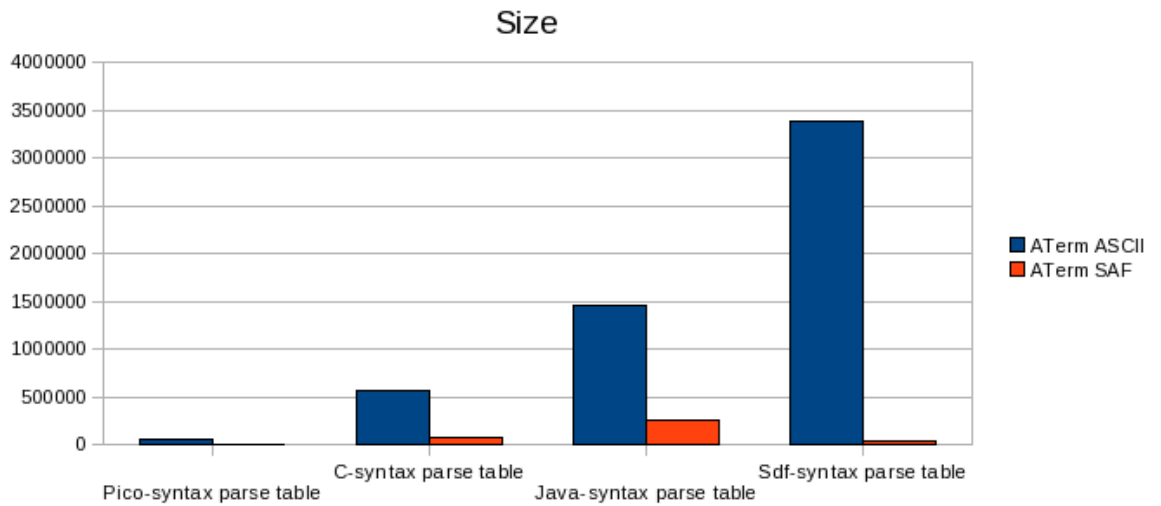
To provide some insight into the 'real-life' performance of both ATerm formats we executed a number of tests with documents from the Meta-Environment. Parse tables that were generated from a (ASF-)SDF specification to be precise. Generally these parse tables contain lots of shared nodes, which the ATerm SAF encoding can take advantage of. As can be observed, using ATerm SAF instead of ATerm ASCII has a noticeable benefit, especially in the case of the SDF-syntax parse table example; here it reduced parsing speed by a factor 100 and achieved a compression rate of over 99.6%. Admittedly this is an extreme case though.

Transformer Speed



Parser Speed





## Conclusions

We can conclude from the benchmarks that the XML format has a definite performance advantage in terms of parsing speed over ATerm ASCII. The main reason for this is the global sharing of ATerm objects; while this reduces overall memory usage it also introduces a noticeable overhead. However, documents using the ATerm ASCII representation are significantly smaller.

ATerm SAF is the clear winner in all the tests. It dominates in both transformation and parsing speed as in small document size. When sharing is also taken into consideration, SAF becomes even better.

In contrast, applying ZIP compression to XML documents, while reducing their size, does introduce a significant overhead. So unless bandwidth or diskpace is an issue, this doesn't seem to be a good idea.