NISTIR 6581
2000

JAVA-Based XML Utility for the NIST Machine Tool Data Repository

Joe Falco
National Institute of Standards and Technology
100 Bureau Drive, Stop 823
Gaithersburg, MD 20899-8230
(301) 975-3455
falco@nist.gov

Abstract

The National Institute of Standards and Technology (NIST) is developing a specification for a standard format to represent machine tool performance test data. This data is used for machine acceptance, predictive maintenance, error compensation, and to evaluate the capability of a machine or contractor to manufacture parts to specified tolerances. This paper describes the implementation of a JAVA-based utility for converting data produced by several machine tool performance evaluation software packages into a unified Extensible Markup Language (XML) based standard data representation. The utility also provides XML based validation, viewing and editing capabilities, and a mechanism for client/server web based communications for database upload.

Introduction

A JAVA-based utility has been developed for the conversion, inspection, validation, modification and client/server upload of Extensible Markup Language (XML) based machine tool performance data. Machine tool performance data is used for machine acceptance, predictive maintenance, error compensation, and to evaluate the capability of a machine or contractor to manufacture parts to specified tolerances. Currently there exists a variety of software packages for the performance evaluation of machine tools. These packages employ different data models and use vendor specific data formats thus complicating data exchange, data storage in databases, and use of the data by third-party software. NIST has been working within a consortium, the NIST Machine Tool Performance Models and Machine Data Repository, to develop and implement a standardized computer interpretable representation [1] that allows for efficient archiving and exchange of performance test data throughout the life cycle of a machine tool. This work has led to the formation of a new standards technical committee, American Society of Mechanical Engineers (ASME) B5 TC56 Information Technology for Machine Tools. The primary goal in the development of the NIST Machine Tool XML Utility was to provide a platform independent application capable of converting performance data from a variety of vendor specific data files to the standard XML based format. An example of a raw vendor format and its associated XML file can be found in the appendix of this document. This implementation standard will be referred to here as the NIST Machine Tool Data Dictionary. Additional goals were to build, within this application, capabilities to view, validate, and edit XML files and to provide the mechanism to upload these files to an experimental database over the Internet.

XML, DTD & JAVA

XML is a standard framework for the description of data so that it can be easily exchanged and reinterpreted [2][3]. Like Hypertext Markup Language (HTML), XML is a markup language; however, XML was designed to describe data by defining tags, while HTML was designed to display data using a predefined set of tags. The ability to define a custom set of tags to structure and describe data is what makes XML extensible. XML was not designed to replace HTML, but rather, the two often compliment each other.

A Document Type Definition (DTD) defines the document structure with a list of legal elements for a particular XML data implementation. The purpose of a DTD is to define the legal building blocks of an XML document. A DTD can be defined within an XML document, or as a separate file that is identified within an XML document. Applications to check the validity of an XML document against a DTD are readily available.

JAVA is an object-oriented programming language capable of creating general-purpose applications and embedded programs "applets" in Web pages. JAVA is platform independent, meaning the application is compiled once and can run on any computer-platform provided the platform contains a JAVA Virtual Machine for running the compiled JAVA code. Likewise, browsers contain their own built-in JAVA Virtual Machine for running JAVA applets. JAVA programs consist of pieces called classes. Programmers can code their own classes and also take advantage of rich collections of existing classes in JAVA class libraries. The "hooks" for using these libraries are Application Programming Interfaces (APIs) which describe the structure and usage of the libraries.

JAVA and XML complement each other. XML provides platform-independent data (portable documents and data), and JAVA provides platform-independent processing (portable object oriented software solutions)[4][5]. The intrinsic hierarchical object oriented design of some of JAVA's APIs readily support the XML hierarchical representation of data, making the two extremely compatible for representing each other's structures. In addition, JAVA provides built in networking capabilities with its standard APIs for support of socket, Hypertext Transfer Protocol (HTTP), HTML, and server protocols. Although there are many XML tools and APIs based on other languages such as Perl and C, the majority of XML development is focused on JAVA, which is emerging as the industry's language of choice for processing XML.

Application Implementation

The NIST Machine Tool XML Utility is implemented as a client side application, meaning it runs on a users local computer platform. One reason for this was to keep a significant portion of the processing load on the client side leaving the server to handle database operations. Additionally, the software design is modular so the only data being passed between the client and server, as well as between the different modules within the client application, is an XML document. This makes the client processing independent of the server processing and simplifies future implementations of software tools. These tools may include analysis tools for XML based machine tool performance data both prior to data upload and after data retrieval from the database. Lastly, the application resides on the client side and the server network address and XML processing calls are configurable. This provides the capability for uploading machine tool performance data to different servers where server locality could range from local private Intranets holding a proprietary data repository to the Internet holding a public data repository.

The JAVA API for XML Parsing (JAXP) [6][7] is the Sun Microsystems JAVA[1] standard extension for XML which provides 100 % conformance to the XML 1.0 Specification, the Simple API for XML (SAX) 1.0, Document Object Model (DOM) Level 1 Core, and XML namespace recommendations from the Word Wide Web Consortium (W3C). This API provides the basic functionality for reading, manipulating, and generating XML documents through pure JAVA APIs and provides a standard interface for integrating any XML-compliant parser with a JAVA application. Depending on the needs of the application, developers have the flexibility to swap between several available JAVA-based XML parsers without making application code changes. XML-compliant parsers provide a platform and language neutral interface that allows programs to dynamically access and update content, structure and style of XML documents. The NIST Machine Tool XML Utility uses the JAXP default parser, Sun's JAVA Project X.

1. DISCLAIMER: Any mention of commercial products within this paper is for information only and does not imply recommendation or endorsement by NIST.

There are two methods of accessing XML documents. The first is event-driven, serial access using SAX. This is the fastest and least memory intensive mechanism for working with XML documents. This type of parsing has traversal and modification limitations. A document can be traversed from top to bottom, once, and the document cannot be rearranged. Document traversals such as this are fast and are ideal for tasks such as database uploads and DTD validation. The second method constructs a DOM to access an XML Document. The DOM API is ideal for interactive applications because the entire object model for the document is present in memory as a tree structure, where it can be accessed and manipulated by the user. Each node of the DOM tree structure contains one of the components from an XML structure. The most common types of nodes are element nodes and text nodes. The DOM API provides mechanisms to create nodes, remove nodes, change their contents, and traverse the node hierarchy. The client side NIST Machine Tool XML Utility utilizes both of these APIs depending on the task at hand.

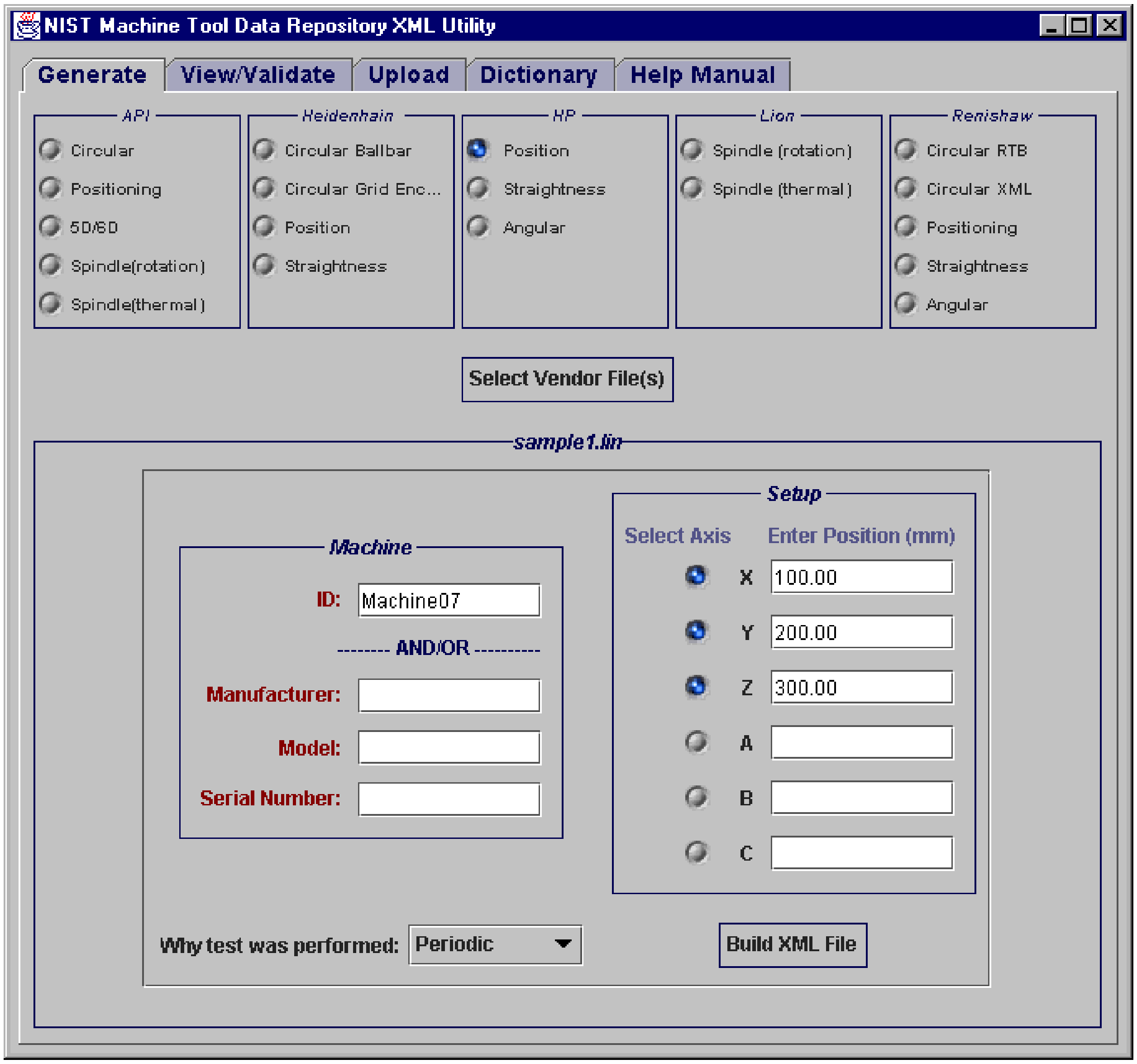
Application Overview

The application is comprised of 3 major components including a parsing utility, a view/edit utility, and an upload utility. The parsing utility handles the conversion of vendor formatted machine tool performance data into an XML based data representation in accordance to the NIST Machine Tool Data Dictionary. The view/edit utility provides capability to view and edit an XML document in the form of a graphical tree. This utility also provides the capability to check the validity of an XML document against NIST data structure and element content. The upload utility communicates XML documents to a server using a standard network protocol. All of these utilities operate independently, with an XML file being the input or output of the operation. In addition to the three utilities, both online help and access to the NIST Machine Tool Data Dictionary are provided as HTML utilities within the application.

The parsing utility implements several classes for parsing vendor performance data files. Currently, data files that are supported include but will not be limited to:

Circular	**Position**	**Angular**	**Straight**	**Spindle** (axis of rotation	**Spindle** (thermal)
Automated Precision Inc. (API)	Renishaw	Renishaw	Renishaw	API	API
Heidenhain ballbar	API 5D/6D	API 5D/6D	API 5D/6D	Lion	Lion
Heidenhain grid encoder	Heidenhain	Hewlett-Packard (HP)	Heidenhain		
Renishaw RTB	HP		HP		
Renishaw XML					

The result of a vendor parsing operation is a NIST Machine Tool Data Dictionary compliant XML file that requires additional user input for required information that is not included in the vendor data file. The input of this information is accomplished by parsing the XML file to create a DOM tree structure. Based on the specific vendor file, information is extracted from the tree and a form is produced for information input and modification as shown in Figure 1. Once the form is completed, the DOM is updated and the XML file is regenerated. This is all accomplished using the DOM API.



sample1.lin

HP: [✓ Position]

Machine

ID: Machine07

-------- AND/OR ---------

Manufacturer:

Model:

Serial Number:

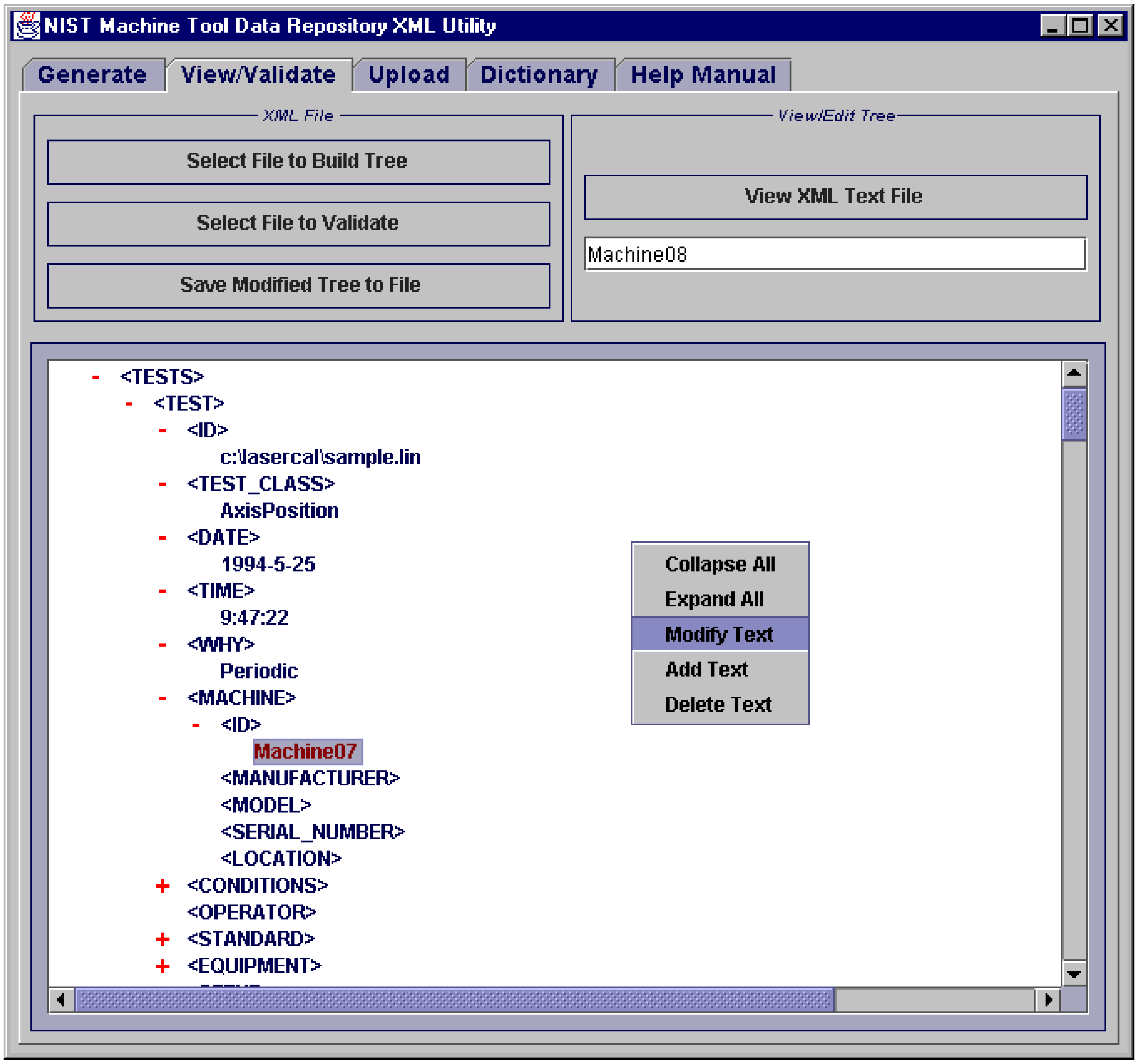
Setup

Select Axis		Enter Position (mm)
●	X	100.00
●	Y	200.00
●	Z	300.00
	A	
	B	
	C	

Why test was performed: Periodic

Figure 1. Parsing Utility

Viewing and editing operations are accomplished using a built-in JAVA API, called a J-Tree, in conjunction with the DOM API. A J-Tree builds a graphical representation of a user selected XML document. This "view" of the DOM can also be used as an interface for modifying the DOM. A snapshot of the view/edit utility containing a J-Tree representation of an XML document being edited is shown in Figure 2.



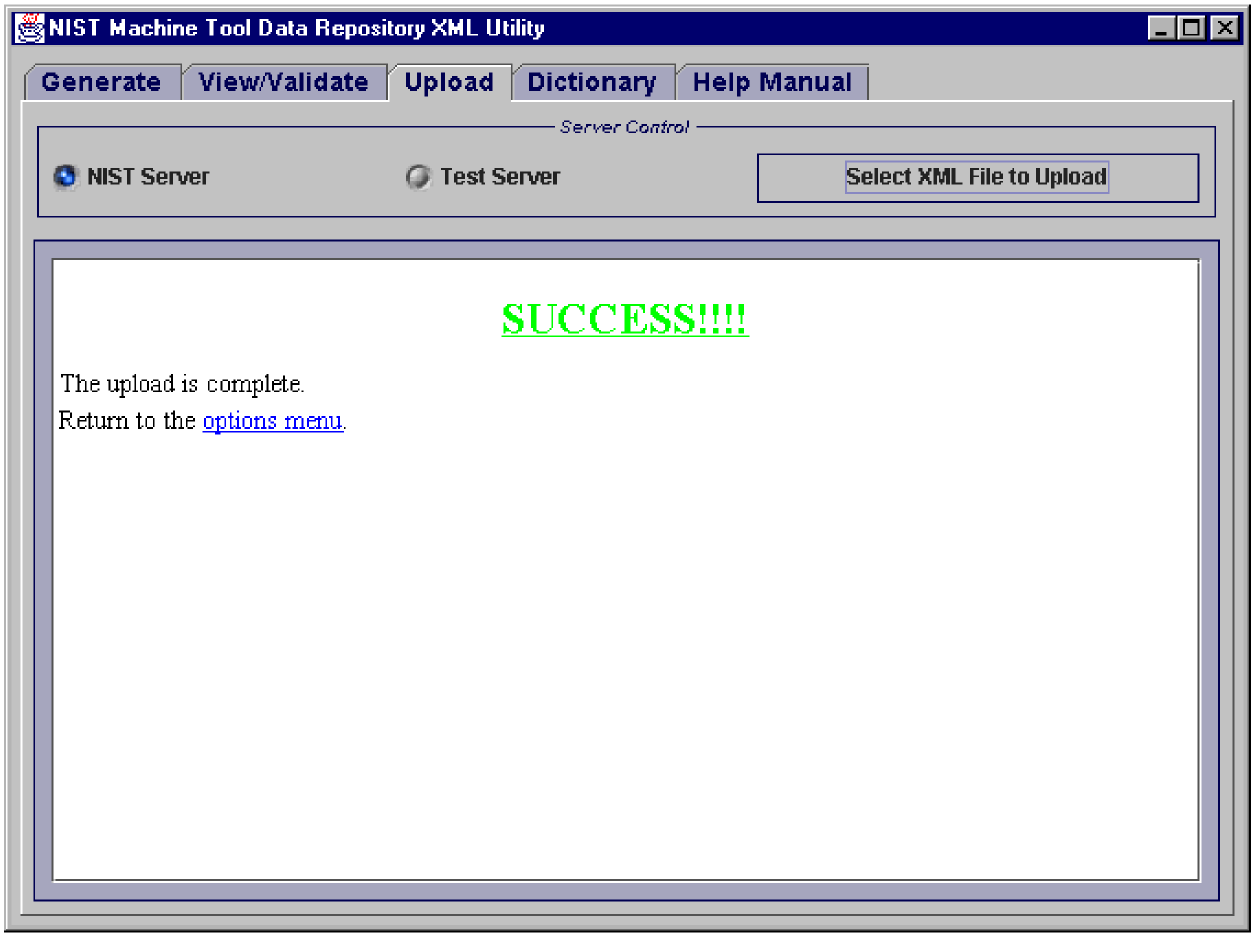
Machine08

```
- <TESTS>
   - <TEST>
      - <ID>
           c:\lasercal\sample.lin
      - <TEST_CLASS>
           AxisPosition
      - <DATE>
           1994-5-25
      - <TIME>
           9:47:22
      - <WHY>
           Periodic
      - <MACHINE>
         - <ID>
              Machine07
           <MANUFACTURER>
           <MODEL>
           <SERIAL_NUMBER>
           <LOCATION>
      + <CONDITIONS>
        <OPERATOR>
      + <STANDARD>
      + <EQUIPMENT>
```

Figure 2. View/Edit Utility

Editing the XML file through this interface is limited to the confines of the NIST Machine Tool Data Dictionary and to the type and content of the particular XML component being modified. Only data values "text components" can be edited, added, or deleted and limitations are placed on what text nodes can be edited. For example, using this interface, a user cannot edit raw data values. Once an edit operation is performed, the user can then save the DOM to an XML document. The capability for DTD based validation of XML documents that were not generated using this application is also provided within the view/edit utility. This ensures that an XML document is compliant with the NIST Machine Tool Data Dictionary prior to database upload.

Submit operations send XML documents to the server using HTTP, Multipurpose Internet Mail Extensions (MIME) format. HTTP is a protocol for client server communication. MIME is a specification for enhancing the capabilities of HTTP. A user simply selects an XML document to upload to the database. Based on error feedback from an unsuccessful upload process (ie. improper machine identification number), a user may edit and save the document using the view/edit utility and resubmit it to the server. A user can also define the server and upload script to use. The upload utility is shown in Figure 3.



SUCCESS!!!!

The upload is complete.

Return to the options menu.

Figure 3. Upload Utility

Conclusions

A JAVA-based utility has been developed for the conversion, inspection, validation, modification and client/server upload of XML based machine tool performance data. This utility is being used by NIST and other consortium members to generate and upload XML based machine tool performance data to the NIST Machine Tool Data Repository. It currently provides support for 19 vendor-specific data formats.

The modular software design provides the potential to add capabilities such as parsers for additional vendor data formats and the retrieval and analysis of data from a machine tool data repository. Currently, retrieval is accomplished using HTML form documents and analysis is performed upon retrieval of data using Matlab, a commercially available mathematical analysis software package which is running on the database server. HTML documents constructed by the server using plot images and data from Matlab are returned to the client. This method presents a burden to the server and could be implemented on the client side as an additional capability. The only data that would be downloaded to the client would be the XML file returned from the database query.

References

[1] Soons, Hans, “NIST Machine Tool Data Dictionary,” working draft. http://mtrws.nist.gov/dictionary/dictionary.htm

[2] XML101, http://www.xml101.com/, tutorial.

[3] XML.COM, http://www.xml.com, information repository

[4] Web Developers Virtual Library, “XML & JAVA : The Perfect Pair,” http://www.wdvl.com/Authoring/Languages/XML/Java/index.html

[5] Sun Microsystems Inc., “Portable Data / Portable Code: XML & JAVA Technologies,” http://java.sun.com/xml/ncfocus.html

[6] Sun Microsystems Inc., “The JAVA API for XML Parsing (JAXP) Tutorial,” http://java.sun.com/xml/docs/tutorial/index.html

[7] Sun Microsystems Inc., “The JAVA API for XML Parsing – Version 1.0,” http://java.sun.com/xml/docs/api

Appendix

Example HP Position raw data file:

```
HP 5529A:       Linear Error Plot - X-Axis
Filename:       c:\lasercal\sample.lin
Acquisition date:       5/25/94 9:47:22 AM
Current date:   7/29/80 1:46:03 PM
Measurement Type:       LINEAR / Algebraic Sign Convention
Axis:   X
Travel Mode:    Bidirectional
Number of Target Positions:      11
Number of total data pairs:   110
Number of total data runs:       10
Expansion Coefficient:   11.7 PPM/°C
Position Value Units:   millimeters
Error Value Units -     micrometers

environmental data -    MIN     MAX     AVG
Air Temp (C)    020.00  020.00  020.00
Air Prs  (mm)   760.73  760.73  760.73
Air Hmd  (%)    050.00  050.00  050.00

Sorted by       run number
Run #   Pos #   Target Value    Error Value     Out of spec.
 1       1      0.E+00  -2.7E+00
 1       2      1.50041E+02     1.E-01
 1       3      3.00082E+02     -5.0999999E+00
 1       4      4.50123E+02     5.E-01
 1       5      6.00164E+02     -8.5E+00
 1       6      7.50205E+02     -5.E+00
 1       7      9.00246E+02     -6.3999996E+00
 1       8      1.050287E+03    -5.5999999E+00
 1       9      1.200328E+03    -8.3000002E+00
 1       10      1.350369E+03    -7.5E+00
 1       11      1.50041E+03     -7.8999996E+00
 .
 .
 .
 10      1      0.E+00  1.4000001E+00
 10      2      1.50041E+02     5.8000002E+00
 10      3      3.00082E+02     -8.9999998E-01
 10      4      4.50123E+02     2.8E+00
 10      5      6.00164E+02     -7.8999996E+00
 10      6      7.50205E+02     -3.E+00
 10      7      9.00246E+02     -3.4000001E+00
 10      8      1.050287E+03    -2.1000001E+00
 10      9      1.200328E+03    -2.1000001E+00
 10      10      1.350369E+03    -1.4999999E+00
 10      11      1.50041E+03     -2.8000002E+00
```

Example HP Position XML data file:

```
<?xml version="1.0" encoding="UTF-8"?>

<TESTS>
  <TEST>
    <ID>c:\lasercal\sample.lin</ID>
    <TEST_CLASS>AxisPosition</TEST_CLASS>
    <DATE>1994-5-25</DATE>
    <TIME>9:47:22</TIME>
    <WHY>Periodic</WHY>
    <MACHINE>
      <ID>Machine08</ID>
      <MANUFACTURER />
      <MODEL />
      <SERIAL_NUMBER />
      <LOCATION />
    </MACHINE>
    <CONDITIONS>
      <CLAMPED_AXES />
      <COMPENSATION />
      <COMPENSATION_ID />
      <COOLANT />
      <DRIVE_STATUS />
      <TEMP_ENVIRONMENT>020.00</TEMP_ENVIRONMENT>
    </CONDITIONS>
    <OPERATOR />
    <STANDARD>
      <ORGANIZATION />
      <NUMBER />
      <NAME />
      <YEAR />
    </STANDARD>
    <EQUIPMENT>
      <ID />
      <COMPONENT>
        <ID />
         <DESCRIPTION>Laserhead</DESCRIPTION>
         <MANUFACTURER>Hewlett Packard/Agilent</MANUFACTURER>
         <MODEL>5519A</MODEL>
         <SERIAL_NUMBER />
         <CALIBRATION_DATE />
         <CALIBRATION_EXP_DATE />
         <CERTIFICATE_NUMBER />
         <CALIBRATION_ORGANIZATION />
      </COMPONENT>
      <COMPONENT>
        <ID />
        <DESCRIPTION>AirSensor</DESCRIPTION>
        <MANUFACTURER>Hewlett Packard/Agilent</MANUFACTURER>
        <MODEL>10751C</MODEL>
        <SERIAL_NUMBER />
        <CALIBRATION_DATE />
        <CALIBRATION_EXP_DATE />
        <CERTIFICATE_NUMBER />
    <CALIBRATION_ORGANIZATION />
  </COMPONENT>
  <SOFTWARE>
    <ID />
     <MANUFACTURER>Hewlett Packard/Agilent</MANUFACTURER>
     <NAME>HP5529A Dynamic Calibrator</NAME>
     <VERSION_NUMBER>2.1</VERSION_NUMBER>
  </SOFTWARE>
  <RESOLUTION>0.00001</RESOLUTION>
  <SAMPLE_RATE_RAW>33000</SAMPLE_RATE_RAW>
  <EQUIPMENT_CLASS>LaserInterferometer</EQUIPMENT_CLASS>
  <FILTER_OFFSET>No</FILTER_OFFSET>
  <TEMP_REFERENCE_COMP>Yes</TEMP_REFERENCE_COMP>
  <TEMP_REFERENCE_SENSOR />
  <AIR_HUMIDITY_COMP>Yes</AIR_HUMIDITY_COMP>
  <AIR_PRESSURE_COMP>Yes</AIR_PRESSURE_COMP>
  <DEADPATH_COMP>Yes</DEADPATH_COMP>
  <VOL_COMP_METHOD>Sensor</VOL_COMP_METHOD>
</EQUIPMENT>
<SETUP>
  <ID />
  <MEAS_MODE>Static</MEAS_MODE>
  <MEASURAND>X</MEASURAND>
  <MEAS_METHOD>Direct</MEAS_METHOD>
  <AXIS>X</AXIS>
  <START_POINT>
     <AXIS_POSITION>
       <AXIS>X</AXIS>
       <POSITION>100.00</POSITION>
     </AXIS_POSITION>
     <AXIS_POSITION>
       <AXIS>Y</AXIS>
       <POSITION>200.00</POSITION>
    </AXIS_POSITION>
    <AXIS_POSITION>
      <AXIS>Z</AXIS>
      <POSITION>200.00</POSITION>
    </AXIS_POSITION>
  </START_POINT>
  <TOOL_VECTOR>
    <X />
    <Y />
    <Z />
  </TOOL_VECTOR>
  <SPINDLE_NUMBER />
  <TURRET_NUMBER />
  <FEEDRATE />
  <DEADPATH />
  <OVERSHOOT />
  <TEMP_MATERIAL_COMP>No</TEMP_MATERIAL_COMP>
  <TEMP_MATERIAL_SENSOR />
  <TEMP_ADDITIONAL_SENSOR />
  <SAMPLES_AVERAGED />
  <SENSOR_OFFSET />
  <REVERSAL />
  <TARGETS>0.0 150.041 300.082 450.123 600.164 750.205 900.246 1050.287 1200.328 1350.369 1500.41
```

```
      </TARGETS>
      <REPETITIONS>1</REPETITIONS>
      <TRIGGER_MODE>Target</TRIGGER_MODE>
      <TRIGGER_DWELL />
      <TRIGGER_WIDTH />
      <TRIGGER_STABILITY />
    </SETUP>
    <RUN_DATA>
      <APPROACH_DIRECTION>Positive</APPROACH_DIRECTION>
      <LS_OFFSET_ERROR />
      <TEMP_REFERENCE>
        <NAME />
        <DATA>020.00</DATA>
      </TEMP_REFERENCE>
      <TEMP_MATERIAL />
      <TEMP_ADDITIONAL />
      <AIR_HUMIDITY>050.00</AIR_HUMIDITY>
      <AIR_PRESSURE>0.101422</AIR_PRESSURE>
      <ELAPSED_TIME />
      <POINTS>-0.002700 0.000100 -0.005100 0.000500 -0.008500 -0.005000 -0.006400 -0.005600 -0.008300 -
0.007500 -0.007900 </POINTS>
    </RUN_DATA>
      •
      •
      •
    <RUN_DATA>
      <APPROACH_DIRECTION>Negative</APPROACH_DIRECTION>
      <LS_OFFSET_ERROR />
      <TEMP_REFERENCE>
        <NAME />
        <DATA>020.00</DATA>
      </TEMP_REFERENCE>
      <TEMP_MATERIAL />
      <TEMP_ADDITIONAL />
      <AIR_HUMIDITY>050.00</AIR_HUMIDITY>
      <AIR_PRESSURE>0.101422</AIR_PRESSURE>
      <ELAPSED_TIME />
      <POINTS>-0.002800 -0.001500 -0.002100 -0.002100 -0.003400 -0.003000 -0.007900 0.002800 -0.000900
0.005800 0.001400 </POINTS>
    </RUN_DATA>
    <COMMENT />
  </TEST>
</TESTS>
```