User-Designed Standardized File Loader
udStFiLr XML
Users Guide

Version 3.00(r)

J. J. Weimer

September 6, 2007

Distribution Details
Developed with Igor Pro Version: 6.0.2
Procedure Files: udStFiLrXML and udStFiLrXML.lcl
Demos: none (uses SimpleStandardLoaderPanelDemo)
Experiments: none XOPS: none Help Files: none
Requires Packages: udFLStandardFunctions and udFLStandardStructure
Also Requires: a working XSLT engine (xsltproc on MacOS or ?? on WinXX)

Abstract

The udStFiLrXML procedure is designed to read an XML file into Igor Pro according to rules in a user-generated XSL file. Values can be stored a parameters (variables or strings), keyword-list strings (in two variations), or waves. In addition, code can be included in the XSL file to run Igor Pro functions or operations.

This file loader procedure is compliant with proposed standards for user designed file loaders, and it requires the procedures developed for such file loaders. It can be used directly as a "plug-in" module" for the simple file loader panel or other standards compliant file loader user interfaces.

Applications for this file loader include Igor Pro procedures that need access to databases of numeric or string values that are provided in XML format. An example XLS that reads in values (atomic number, density, electron configuration, ...) for 112 elements from an XML periodic table (obtained directly from the Web) is provided. An additional XLS that reads header information and row-based intensity data from an x-ray diffraction system is also given.

This document was generated using LaTeX

Contents

1 Summary

The udStFiLrXML procedure is designed to read an XML file into Igor Pro according to rules in a user-generated XSL file. Values can be stored in a variety of ways, and Igor Pro commands can be issued from within the XSL.

Applications include Igor Pro procedures that need access to databases of values that are otherwise provided in XML format.

An example that reads in values (atomic number, density, electron configuration, ...) for 112 elements from an XML periodic table (obtained directly from the Web) is provided. An additional example reads a database into a wave, sets the wave scaling, and attaches a wave note.

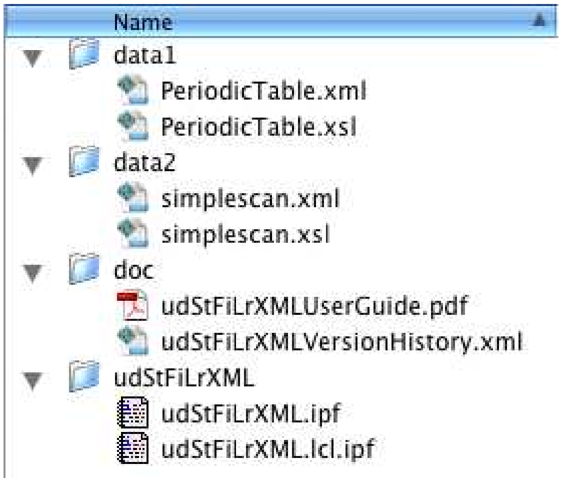
2 Setup

2.1 Requirements

This procedure has only been tested on Igor Pro 6.0 and is defined with a minimum requirement of Igor Pro 6.0. Please see the section Known Limitations in section 5 to learn how this procedure may be used with Igor Pro 5.0.

2.2 Package Contents

The procedure is provided in a ZIP archive. Unpacking the archive will reveal the primary (root) folder udStFiLrXMLN, where N is the version number. The directory structure inside this folder is shown below.



Name

- data1
  - PeriodicTable.xml
  - PeriodicTable.xsl
- data2
  - simplescan.xml
  - simplescan.xsl
- doc
  - udStFiLrXMLUserGuide.pdf
  - udStFiLrXMLVersionHistory.xml
- udStFiLrXML
  - udStFiLrXML.ipf
  - udStFiLrXML.lcl.ipf

---

data1
This directory contains XML data from a periodic table of elements and an XSL file to load it into waves.

data2
This directory contains XML data from an x-ray diffraction system and an XSL file to show how to load the row of intensity data, load header information, and set the wave scaling.

doc
This directory contains the documentation (this document) and an XML version history.

udStFiLrXML
This directory contains two files. The file udStFiLrXML.ipf contains the code to read, parse, and store the XML data into Igor Pro. The file udStFiLrXML.lcl.ipf contains dialog and alert strings in different languages.

2.3 Installation

The procedure can be installed in one of two ways. In both cases, the entire directory `udStFiLrXML` is to be copied (or moved) to a specific location.

- To have the file loader available every time you start Igor Pro, move or copy the `udStFiLrXML` directory to the `Igor Procedures` directory of your local installation of Igor Pro.
- To have the file loader available only when you wish to include it, first move or copy the `udStFiLrXML` directory to the `User Procedures` directory of your local installation of Igor Pro. Then, when you wish to include the `udStFiLrXML` procedure in a specific experiment, open the Macros window while in Igor Pro and put the line

  ```
  #include "udStFiLrXML"
  ```

  somewhere directly after the `#pragma rtGlobals=1` directive that appears in this window.

This procedure can be used directly as a “plug-in” to the Simple Standard Loader Panel Demo, therefore no stand-alone experiment is provided as a user interface.

3 Use

The `udStFiLrXML` procedure can be used in one of two ways. It can be integrated into more complex routines as a file loader module. An example of this is where the procedure reads preference settings for a package that are stored externally in an XML database. In this case, the user is typically unaware of and has no direct interaction with the operation of the udStFiLrXML loader–it is just a functioning part of the larger package.

Alternatively, the `udStFiLrXML` procedure may be used as the primary method to load data in XML databases into Igor Pro for further processing. In this case, the user is typically prompted to select the data file(s) to load.

3.1 Processing Method

Previous versions of this procedure parsed the XML file entirely within Igor Pro. This version parses the XML using an XSLT engine. The XSLT engine uses the XSL file to parse the XML and to generate a result file. The result file is passed to the udStFiLrXML procedure for further processing (parsing).

The advantages of using an XSLT engine are in speed, flexibility, and durability. Because the XSLT engine is a compiled procedure, the translation runs faster than what can be provided by using Igor Pro coding (exclusive perhaps of XOPs). Because XSLT engines are well-developed, they provide far more options for you to decide how to input your data than I can generate in a reasonable time. Finally, because XSLT engines are well-maintained, they remove a significant portion of the burden from me to update this procedure.

Because this procedure is now intimately tied to an XSLT engine, having a basic familiarity with proper XML and XSL coding is now essential. In fact, I strongly recommend that you become conversant with the basic terms in XML and XSL as the FIRST step toward using this procedure. The success of any translation you want to make can be directly dependent on having such knowledge. The lessons needed to provide this knowledge are well beyond the scope of this document. A good starting point is the tutorial at http://www.w3schools.com/xsl/.

You will need to have an XSLT engine installed on your computer. On MacOS X, xsltproc is installed by default (at least with 10.4.x). You can check this by typing `xsltproc -V` (case sensitive) on the terminal. You can check for updates to xsltproc at http://www.xmlsoft.org/. On a WinXX system, you have to install an XSLT engine manually. The udStFiLrXML procedure works ONLY with AltovaXML found at http://www.altova.com/altovaxml.html. You must assure that AltovaXML is in your execution path - it must be executable at any directory level simply by typing AltovaXML.

I also recommend that you install a command-line or GUI type of processing checker to validate your XSL transformation before using it in this procedure. This is especially important to remove and control for spurious or incorrect carriage return/line feed sequences as files are transferred from Mac/Win/Unix systems. For the MacOS, I use the simple but effective tkxlstproc found here.

Finally, I cannot vouch completely for the robustness of the parsing algorithm to convert the HTML, especially with regard to how spaces, tabs, and carriage return/linefeed sequences are handled. Please see the section Known Limitations in section 5 for further comments about this.

3.2 File Formats

With an XSLT engine, three files are part of the processing sequence (instead of two as in previous versions of this procedure). The XML file contains the data that is to be read into Igor Pro. The XSL file defines the translation instructions for the XSLT engine. Finally, the XSLT engine generates a result file that is passed to and defines the processing (parsing) instructions for Igor Pro.

The format of the result file from the XSLT engine can typically be set by a directive in the XSL as one three options: text, html, or xml. To process the result file properly within Igor Pro, the html option is required.

3.2.1 XML File

The XML file should have the basic format shown below. Please see a tutorial on XML for full details to the summary information given below.

```
<?xml version="1.0"?>
<DATABASE>
  <ELEMENT>
    <VALUE1>value1</VALUE1>
    <VALUE2>value2</VALUE2>
    <VALUE3>value3</VALUE3>
    ...
    <VALUEN>valueN</VALUEN>
  </ELEMENT>
  <ELEMENT>
    ...
    </ELEMENT>
    ...
  </DATABASE>
```

DATABASE
The name of the XML database. For example, the Periodic Table XML file has a DATABASE name of PeriodicTable.

ELEMENT
The name of the elements within the database. For example, all ELEMENTS within the Periodic Table XML file have the name ATOM.

VALUEj
The name of the j^{th} property of the given ELEMENT within the DATABASE. Within the Periodic Table XML file for example, ATOMS have VALUEs such as SYMBOL, NAME, ATOMIC_WEIGHT, DENSITY, and ELECTRONIC_CONFIGURATION.

valuej
The *value* (numeric or string) given to the j^{th} VALUE of the given ELEMENT. For example, for one ELEMENT in the Periodic Table XML file, a SYMBOL value Au is associated with the NAME value Gold. The $valuej$ parameter is never enclosed in quotes, regardless of whether it is a string value or a numeric value.
Multilevel nesting to indicate properties of VALUEs is supported in Ver. 3.00.

Variations of the above XML file format are permitted. The VALUEj names can include ATTRIBUTES in the following manner:

```
    <VALUEj ATTRIBUTE="attribute">valuej</VALUEj>
```

ATTRIBUTE
The designation indicating the VALUEj has a particular attribute.

attribute
The attribute associated with $valuej$. The attribute must be enclosed in quotations!

An alternative form of data storage in an XML file is shown by the section below.

```
<?xml version="1.0"?>
<DATABASE>
  <ELEMENT>
    <NAME>name1</NAME>
    <VALUES>value1 value2 value3 ... valueN</VALUES>
  </ELEMENT>
  <ELEMENT>
    <NAME>name2</NAME>
    <VALUES>value1 value2 value3 ... valueN</VALUES>
    ...
  </ELEMENT>
  ...
</DATABASE>
```

In the above format, the data is stored as a row within one element of the XML database. The data values may be numeric or text and must be separated by a single space only!

3.2.2 XSL File

The basic XSL file should have the format shown below. Again, you are instructed to find tutorials on XSL to clarify points made in the summary discussions below. Lines marked as REQUIRED are necessary to make the translation routine work properly, they are otherwise optional in well-formed XSL files.

```
<?xml version="1.0" encoding="UTF-8"?>
<?igorpro version=VERSION?> (REQUIRED!)

<xsl:stylesheet xmlns:xsl="..." version="1.0">
  <xsl:strip-space elements="*"/> (REQUIRED!)
  <xsl:output method="html" indent="yes"/> (REQUIRED!)

<xsl:template match="//ELEMENT">
...
</xsl:template>

</xsl:stylesheet>
```

VERSION

A string version number. As of 3.00, this value should be `"1.04"` (see the known limitations in Section 5).

Strip-Space and Output Designations

The strip-space designation assures that translation removes spurious space characters before the result file is read into Igor Pro for further parsing.

The required output method of the translation is HTML, for reasons illustrated in the examples later in this document. The indent designation assures that lines are properly formatted before the result file is read into Igor Pro for further processing.

ELEMENT

A name of a node as it appears in the XML file. The capitalization of ELEMENT names in the XSL should follow exactly with those given in the XML.

The translation of the XML that is done by the XSLT engine relies on what is provided for a template. The XSL file may have more than one template. Each template is applied at its particular node.

Four templates will likely be within your XSL.

- `match = "/"`

  This is the root node template. It matches at the topmost level of the XML tree. The template defined in this match will include the primary layout of the document that is sent to Igor Pro.

- `match = "*"`

  This is the match for every node within the XML tree. Use this template to define processing that should occur for every `ELEMENT` throughout the XML.

- `match = "text()"`

  This template matches all text nodes within the XML tree.

- `match = "ELEMENT"`

  This template only matches when the node has the designation `ELEMENT`.

A detailed discussion of XSL commands within the bounds of a template is beyond the scope of this document. Specific examples below can serve as guides.

3.2.3 HTML Result File

The file that is passed from the XSLT engine to Igor Pro must be in html format. The parsing within Igor Pro recognizes specific html coding to handle data storage. They are based on html tags. In addition, the result file can direct Igor Pro to execute commands. Here is an overview of the html tags recognized with Igor Pro.

`<DIV>...</DIV>`
The `DIV` tag is a directive to assign contents to parameters, either as variables or strings.

`<ol>...</ol>` *or* `<ul>...</ul>`
The `ol` or `ul` tag is a directive to assign contents to keyword=value string lists.

`<table>...</table>`
The `table` tag is a directive to assign contents to waves or matrices.

`[code]...[/code]`
The `code` tag is a directive to execute the contents as functions or operations.

Note that UPPER or lower case designation of the tags is irrelevant for the input of the html tags to Igor Pro.

3.3 Storage Options

In general, all storage options use an html tag and an additional directive, the `id = "..."` directive. The tag defines how to store the contents and the `id` directive defines how and where to store them. Some tags require the `id` directives, others do not. In any case, the `id` directive should be placed directly within the opening tag designation. General examples of proper and improper tag + directive formats are shown below.

`<DIV id = "string"> ... </DIV>` - PROPER FORMAT

`<table id = "waves"> ... </table>` - PROPER FORMAT

`<DIV> ... </DIV>` - IMPROPER FORMAT

`<table> ... </table>` - IMPROPER FORMAT

Further clarification of the storage options is given on a case-by-case basis.

3.3.1 Parameters

The contents within `DIV` tags are stored as parameters. The proper format of the html coding to store a `value` as a string or variable is

```
<DIV id = "string/variable">
<p id = "name">value</p>
...
</DIV>
```

The `DIV` tag directs Igor Pro to create parameters. The `id` directive defines whether the parameters are to be strings or variables. Each `<p> ... </p>` line within the contents of a `DIV` block defines an individual parameters. The `id = "name"` directive establishes the name of the parameter, and the `value` within the `p` tags gives its value.

Example 3.3.1.A

The following code stores three string parameters, header, startTime, and sample. The value of header is set by text. The other two parameters contain both text and selections from the XML file.

```
<DIV id="string">
<p id="header">FILE HEADER\r</p>
<p id="startTime">Start: <xsl:value-of select="startTime">\r</p>
<p id="sample">Sample: <xsl:value-of select="sampleType">\r</p>
</DIV>
```

Note that text content is NOT quoted and the `\r` is used to designate where a line break (carriage return) is to be placed.

Example 3.3.1.B

The following code stores two variable parameters, Sscan and Escan.

```
<DIV id="variable">
<p id="Sscan"><xsl:value-of select="startScan"></p>
<p id="Escan"><xsl:value-of select="endScan"></p>
</DIV>
```

Mixing string and variable storage within one `DIV` is illegal.

3.3.2 Keyword-Value Lists

The contents within `ol` or `ul` tags are stored as keyword-value lists. Order lists use the format `keyword = value` and unordered lists use the format `keyword:value`. The proper format to store keyword-value lists is

---

```
<ol id = "listname">
<li id = "name">value</li>
...
</ol>
```

The `ol` (or `ul`) tag directs Igor Pro to create keyword-value lists. The `id` directive at this level defines whether the name of the list string. Each `<li> ... </li>` line within the contents of a `ol` block defines an individual keyword-value. The `id = "name"` directive within the `li` establishes the name of the keyword, and the `value` within the `li` tags gives its value.

Example 3.3.2.A

The following code stores a keyword=value list string named graphprefs. The string contains information about the graph title, scalings, and labels on the x-axis and y-axis.

```
<ol id="graphprefs">
<li id="title"><xsl:value-of select="title"></li>
<li id="xlabel"><xsl:value-of select="axis/x/label"></li>
<li id="xscale"><xsl:value-of select="axis/x/scaling"></li>
<li id="ylabel"><xsl:value-of select="axis/y/label"></li>
<li id="yscale"><xsl:value-of select="axis/y/scaling"></li>
</ol>
```

3.3.3 Waves

The contents within `table` tags are stored as waves. Because two variations of wave storage exist, the `id` tag in the table code is required. The proper format of the html coding to store values as waves is

```
<table id = "waves">
<tr id = "name">
  <td>value</td>
  ...
</tr>
</table>
```

The `table` tag directs Igor Pro to create waves. The `id` directive defines the application as wave storage. Each `<tr> ... </tr>` segment within the contents of a `table` block defines an individual wave. The `id = "name"` directive establishes the name of the wave. The `value` within the `td` tags gives the values of each element of the wave.

To be clear - every ROW `<tr>...</tr>` within a table is a NEW WAVE. Every DATA `<td>...</td>` element within a row is a value that is stored in the given wave.

The parsing procedure in Igor Pro will create a string or a numeric wave appropriately based on the first value that is stored in the wave.

Example 3.3.3.A

The following code stores a set of values into waves called ToC and ts.

```
<table id="waves">
<tr id="ToC">
  <xsl:for-each select="dataPoints">
    <td><xsl:value-of select="Temp"></td>
  </xsl:for-each>
</tr>
<tr id="ts">
  <xsl:for-each select="dataPoints">
    <td><xsl:value-of select="Time"></td>
  </xsl:for-each>
</tr>
</table>
```

3.3.4 Row-Vector Waves

This is an alternative option to store values as waves when they appear in the XML as a single row vector. The proper format of the html coding to store row-vector values as waves is

---

```
<table id = "row-vector">
<tr id = "name">
  <td>value</td>
</tr>
</table>
```

The `table` tag directs Igor Pro to create waves. The `id` directive defines this application as wave storage from a row-vector. Each `<tr> ... </tr>` segment within the contents of a `table` block defines an individual wave. The `id = "name"` directive establishes the name of the wave. The `value` within the `td` tags gives the values of each element of the wave.

The parsing procedure in Igor Pro will create a string or a numeric wave appropriately based on the first value that is stored in the wave.

Note, this option could possibly also be handled as `waves` storage by means of clever coding (using `<xls:for-each ...>` statements) within the XSL itself. Examples of such are solicited from users.

Example 3.3.4.A

The following code stores a set of row-vector values into a wave called intensities.

```
<table id="row-vector">
<tr id="intensities">
  <td><xsl:value-of select="intensities"></td>
</tr>
</table>
```

3.3.5 Matrices (not yet fully implemented!)

A method to input matrices is under consideration. Please note your interest in such a feature on the Igor Exchange Web site.

3.4 Coding

The XSL file can contain directives to Igor Pro as executable statements. The proper format of the html coding to direct Igor Pro via functions or operations is

```
[code]
code statement
...
[/code]
```

All codes are executed in the data folder where values are stored. They can reference global parameters, waves, or matrices created during the storage operations as long as they proceed the storage operation itself. The best place to put coding directives in the XSL is within the root template AFTER a statement to apply all other templates.

Example 3.4.0.A

The following XSL stores string parameters and manipulates them afterward.

```
<xsl:template match="/">
  <body>
  <xsl:apply-templates/>

  [code]
  header += startTime + user
  killstrings/Z startTime, user
  [/code]

  </body>
</template>

<xsl:template match="Comments">
<DIV id="string">
<p id="header">FILE HEADER\r---\r</p>
<p id="startTime">Start: <xsl:value-of select="start"> with </p>
<p id="user">User: <xsl:value-of select="userID">\r</p>
</DIV>
</template>
```

The result of this XSL translation will be a single string parameter named header with the contents such as:

FILE HEADER
—
Start: December 12, 2008 with User: apsmith

3.5 Further Examples

Please see the two example XSL/XML sets provided in the directories `data1` and `data2` for further examples.

3.6 Standard User Interface Modes

As a standardized file loader, `udStFiLrXML` has three modes of operation: Auto, Auto-XML, and Manual. Further details of these modes are presented again in the section For Programmers.

Auto Mode

In Auto mode, the procedure must be provided with all information needed to automatically load and parse the files. In particular, this includes the file names for the XSL and XML files. This mode is typically used by programmers who wish to integrate the routine into a larger package.

Auto-XML Mode

In Auto-XML mode, the procedure will present a dialog box requesting the user to select an XSL file. The procedure then searches in the same directory for an XML file with the same name.

As an example, the following two files can be processed in this mode:

```
mydata.xsl
mydata.xml
```

At the first dialog prompt, the user should select the `mydata.xsl` file. The routine will automatically find the `mydata.xml` file and process it.

Both files must be in the same directory!

Manual Mode
In Manual mode, the procedure will present a dialog box requesting the user to select an XSL file followed by a dialog box requesting the user to select an XML file. Both files can be in separate directories.

You can test all three of these modes in the Simple Standard Loader Panel Demo using the data provided in the directories `data1` and `data2`.

3.7 Localization

You can change the dialog and alert texts to a different language. This is done by modifying the `udStFiLrXML.lcl.ipf` file.

The file starts with a compiler directive that is a definition of the current language. Below this are the language specific alert and dialog strings. A subset of the file is shown below.

```
#define ENGLISH

#ifdef ENGLISH
StrConstant alert00 = "..."
StrConstant alert01 = "..."
StrConstant alert01a = "..."
StrConstant alert02 = "..."
...
#endif
```

To change the language, first confirm that a command block `#ifdef ... #endif` exists for the language you want to use. Then, change the language after the `#define` directive to that language. Please use ALL UPPER CASE LETTERS when doing this!

The line given below changes the language from English to German, assuming the block of code `#ifdef DEUTSCH ... #endif` exists to define the strings in German.

```
#define DEUTSCH
```

To create a new language localization, copy the entire existing block of code found between `#ifdef ENGLISH ... #endif`. Paste this at the bottom of the localization file. Change the `#ifdef ENGLISH` to reflect the language you wish to define. Then, change all of the string text (between quotation marks) to reflect the proper translation of the alert and dialog strings. Do not change the names of the alert or dialog string constants when doing this.

I encourage anyone who develops a new language localization for this procedure to forward it to me so that I can post it for others to use.

3.8 For Programmers

Programmers who wish to use the procedure directly in their own Igor Pro routines are encouraged to review the guidelines below. Only one point of top-level access exists.

Function udStFiLrXML(udFL)

This function is accessed according to the rules defined in the proposed standard document. The relevant parameters are outlined below.

Descriptors

```
udFL.mimetype = "TEXT"
udFL.extensions = ".xsl;.xml;"
udFL.procModes = "Auto;Auto XML;Manual;"
udFL.itemsType = (2^0 + 2^1 + 2^2)
udFL.itemsDim[0]=0
udFL.setNames=1
```

When queried with `udFL.itemsType` set, the procedure also returns the following:

```
2
  udFL.Dim[0] = 3
4
  udFL.Dim[0] = 0
```

Inputs

The inputs will control the loader in the following manner:

```
udFL.userCtrl ... sets processing mode
udFL.userData ... used during programming (see below)
udFL.reportCtrl ... used only sparingly
```

The "Silent" mode of control and the "Normal" mode of control are the same. The "Verbose" mode of control keeps the parsing notebook resident after it has been used and reports progress of the loading, parsing, and saving operations throughout.

Localizers

The inputs will control the loader in the following manner:

```
udFL.pathStr ... according to standards
udFL.fileList ... according to standards
udFL.returnCtrl ... currently unused
```

In "Auto" mode, the values of `udFL.pathStr` and `udFL.fileList` can be passed in one of two ways. In the first case, `udFL.pathStr` should contain the full path to two files in the same directory. Correspondingly, `udFL.fileList` should contain just the names of the two files, with the XSL file first and the XML file second in the list. In the second method, the value of `udFL.pathStr` should be passed empty (as `""`), and `udFL.fileList` should contain the full path to the XSL file followed by the full path to the XML file.

Returns

The file loader will return -1 on a fatal error. It may have a mixture of its own internal alerts and standardized alerts via `udFL.errCode` and `udFL.errMsg`.

4 Package

This file loader no longer requires creation of a Package data folder. Any remaining references to such will be removed in future releases.

5 Known Limitations

- The procedure parses the XSL and XML using an XSLT engine. You are advised to learn how to access such engines using command-line or GUI tools on you system. Testing your XSL translation using such tools BEFORE applying this routine to read an XML file is strongly recommended!
- The algorithm used to parse the HTML in Igor Pro may have problems deciphering spaces, carriage return/linefeed sequences, or tabs when they appear before HTML tags or within HTML elements. If you create what you believe is a valid

XML/XSL set to generate a valid HTML (as proven by external testing), and it cannot be parsed within the udStFiLrXML procedure, please prepare a ZIP archive of the experiment and the XML + XSL for me to check.

- On WinXX systems, the AltovaXML parsing process creates a xxxTMP.txt file for every XML job. The TMP file is supposed to be deleted but often is not. This file can be deleted manually.
- The procedures may possibly be used with Igor Pro 5.0 by making the modifications below. I do not assure this will work - please report if it does.
  - First, put the udStFiLrXML folder in the `User Procedures` folder of Igor Pro. This modification will NOT work when the udStFiLrXML folder is in the `Igor Procedures` folder!
  - Open the udStFiLrXML.ipf file within Igor Pro.
  - Change the `#pragma IgorVersion=6.0` to `#pragma IgorVersion=5.0`
  - Change the line that states `#include ":udStFiLrXML.lcl"` to read exactly as `#include "udStFiLrXML.lcl"` (remove the colon before the file name).
  - Save the procedure file.
- To use the stand-alone procedure when the `udStFiLrXML` folder is placed in the `Igor Procedures` directory, comment out the line `#include "udStFiLrXML"` in the procedure window of the stand-alone experiment.
- Please pay attention to the `igorpro` version number you give in your XSL. This will assure that your data is handled by the correct version of the `udStFiLrXML` procedure file.
- Algebraic manipulations are handled within `code` segments.
- Do NOT quote string values when providing them as input values to parameters or lists. Follow carefully the formats used in the examples.

6 Acknowledgments

Many thanks go to Shaun Roe who was instrumental in establishing the proper format for the XSL file in the transition from 1.12 to 2.0.

Thanks to Luc Ortega for sending the XRD data as a motivation to create the row-wave method of input and to code the routines using XSLT engines.

Thanks to Holger Taschenberger for the clarification and hints about how to handle ExecuteScriptText on WinXX systems as the final piece of the puzzle to creating an XSLT engine compatible version for WinXX systems.

7 Contact

Suggestions, bug reports, and feature requests should be posted on the project page at the Igor Exchange Web site. I will be using it exclusively to track this project.

8 Legalize

This software is free to use but not free to modify and subsequently market further as per the standard terms of public domain software.

Enjoy!

9 Version History

An XML file outlining the version history is provided in the docs folder.

3.00r (2007.09.06)
- now uses XSLT engine for both systems: MacOS uses xsltproc, WinXX uses AltovaXML
- updated information in this document

3.00b1 (2007.09.03)
- now uses XSLT engine: MacOS uses xsltproc, WinXX uses ???
- can read parameters (strings or variables), keyword-lists (using = sign or :), and waves (as standard waves or row-vectors)
- can embed Igor Pro commands within XSL
- no longer requires internal globals (therefore, it has no Package contents)

2.12r (2007.08.25)
- changed name from xml2igorpro to udStFiLr XML to indicate standard compliance (version 2.04r is technically the last version of xml2igorpro)
- included udFL.setNames=1 parameter in initialization
- fixed an incorrect test of `V_flag` instead of `S_filenames` after Open dialog box

---

2.11r (2007.08.22)
- verbose mode prints status at key points during processing
- file loader returns the items list and number of items
- re-distributed coding in the initialize and query segments
- fixed a missing alert code problem that prevented compiling
- open dialogs now check only for file extensions exclusive of mimetype
- itemsType now conform to BIT-wise values

2.10r (2007.08.20)
- rewrote the code to conform to the working proposal on standardizing file loaders
- checks for proper values of the optional KEY
- added row-wave as storage option
- require <?igorpro version =1.03 (to read as row-wave)

2.04r (2007.06.05)
- changed <?xmligorpro version to <?igorpro version (in preparation for use of xslt engines)
-added localization file to have dialogs and alerts in different languages
- put file and localization file in a directory (director structure changed accordingly)
- added text-matrix as storage option
- require <?igorpro version =1.02 (to read as text-matrix)
- request (not require) that all lines in XSL end in ";" or in text (units string) (in preparation for use of xslt engines)

2.03r (2007.05.07)
- added optional `filepath` parameters to `inputXSLFile(...)` and `inputXMLFile(...)`
- added procedure information to the Users Guide
- updated Known Limitations information in the Users Guide

2.02b0 (2007.05.03)
- significant code changes to improve speed and reliability of parsing XML
- moved coding for how=1 in parseXML to its proper location
- missing <KEY></KEY> lines in XML are now stored as "keyword=" or as blank values in wave (you must use version `"1.01"` in the `<?xmligorpro ...>` header!)
- return 0 cases in parseXML now kill notebook and reset datafolder
- added Package Folder information to the Users Guide
- added Known Limitations information to the Users Guide

2.01b1 (2007.05.02)
- fixed ordering of lines in XSL and clarified format requirements in the Users Guide

---

2.01b0 (2007.04.30) (not released)
- used an `<?xmligorpro version="..." input-as="..."?>` directive to control versioning of the xml parsing and input directives to the parser
- removed the `<xsl:parameter ...>` directive as a way to define the input method
- procedure now checks for compatible XML version information in the XSL

2.00b0 (2007.04.29) (not released)

- major change in format of XSL file (therefore the major version number change)
- cleaned up hidden notebook and other junk after abort of XSL or XML input
- can now input XML data to waves
- only one keyword=value input type supported
- can now define pre-processing of numeric values using algebraic formula in XSL
- unit designations stored as <xsl:value-of ...>; “UNITS”
- bug fixes and code revisions

1.12b0 (2007.04.24)

- changed versioning to include letter (a, b, r) and minor number
- made pragma rtGlobals=1
- put Static Function states on internal functions
- can use KEY as name for storage wave (default is “name”)
- has how=0, how=1, and how=2 storage states
- reads UNITS in XLM file
- code changes and revisions

1.11

- changed names of files and folder from ParseXML to XML2IgorPro
- first public posting

1.10

- added "how" switch to inputXMLFile() function for future use (currently unused option) (not posted)

1.00 - first development (not posted)