EARTH OBSERVATION FILE HANDLING Reference Manual

Generated by Doxygen 1.2.14

Thu Oct 26 10:45:15 2017

Contents

1 EARTH OBSERVATION FILE HANDLING Main Page

Software User Manual

EE-MA-DMS-GS-008 Issue 4.14

16/11/2017

Introduction

The EO File Handling Library provides a simple programming interface for creating, modifying, writing and reading XML files tailored to the XML usage needs of the Earth Observation Missions Ground Segment files.

It is built on top of the GNOME libxml2 library but it hides most of the details associated to file parsing, in-memory representation and file writing.

How to compile and link user applications

The EO_FILE_HANDLING software library has the following dependencies:

- Third party libraries:
  - POSIX thread library: libpthread.so (Note: this library is normally pre-installed in Linux and MacOS platforms. For Windows platforms, pthread.lib is included in the distribution package, with license LGPL);
  - LIBXML2 library (this library is included in the distribution package. Their usage terms and conditions are available in the file TERMS_AND_CONDITIONS.TXT which is part of the distribution package).

The following is required to compile and link a Software application that uses the EO_FILE_HANDLING software library functions (it is assumed that the required EOCFI and third-part libraries are located in directory *cfi_lib_dir* and the required header files are located in *cfi_include_dir*, see [GEN_SUM] for installation procedures):

1. include the following header files in the source code:
 explorer_file_handling.h(for a C application)

2. use the following compile and link options:

   - LINUX and MacOS platforms:
 −I*cfi_include_dir* -L*cfi_lib_dir* -lexplorer_file_handling -lxml2 -lm -lc -lpthread
   - WINDOWS platforms:
 −/I *cfi_include_dir* /libpath: *cfi_lib_dir* libexplorer_file_-handling.lib libxml2.lib pthread.lib Ws2_32.lib

All functions in this library have a name starting with the prefix xf_. All constants (macros and enumerations) in this library have a name starting with XF_.

To avoid problems in linking user applications with the EO_FILE_HANDLING software due to the existence of duplicated names, the user application should not name any global software item beginning with either xf_ or XF_.

Library Usage Guide

The interface to an XML file represented as a tree

Any well-formed XML file may be viewed as a hierarchy of elements, attributes and values that can be represented as a tree. In fact this is the in-memory representation used by libxml when an XML file is read into memory.

Although the EO_FILE_HANDLING library hides all these details to the programmer, a few basic concepts need to be presented here to understand how to use the library functions in order to achieve a given goal. We assume that the user has some basic knowledge of XML and understands the concepts of XML elements, attributes and text nodes.

- An element B which is contained directly within an element A is referred to as *child* of A. Unsurprisingly, A is said to be the *parent* of B.
- Two elements, B and C, that are children of a common parent A are said to be *sibling* elements.

The EO_FILE_HANDLING library uses iterators for traversing the in-memory representation of an XML file as built by libxml. An iterator is an abstraction of a pointer to a specific element within a collection. The EO_FILE_HANDLING library uses an iterator to XML elements since elements are the basic building blocks of XML files.

Opening a file for reading and/or modification

The first operation a user must perform in order to use an existing XML file for reading and/or modifying it is loading the file content into memory. This operation is performed by the function xf_tree_init_parser. This function takes as an input the file name and returns an integer value. The EO_FILE_HANDLING library may handle in parallel up to XF_MAX_FILES_NUMBER. When a file is read, xf_tree_init_parser assigns a unique number to that file and returns it to the user. This unique number must be used thereon in order to access the in-memory representation of that file.

Reading a file sequentially

The in-memory representation of an XML file used by libxml allows an optimum traversal using a first-child-next-sibling algorithm. This algorithm ensures that all elements within the tree are visited only once using recursively the following rules:

- If an element has children elements that have not been visited, the next element to visit is its first child.
- If all element children have been visited or the element has no children, the next element to visit is its next sibling.
- If an element has no next sibling, then go to the parent and apply the second rule.

Starting with an iterator pointing to the root element, all elements are visited once and only once until the iterator returns to the root element.

A file may be read sequentially using the family of functions `xf_tree_read_*_element_value`. These functions will return the value of the first element found whose name matches exactly the element name provided as input typed as requested in the function name.

The EO_FILE_HANDLING library uses a global iterator for each file (do not mistake the concept of global iterator with that of a global variable). When a file is read using the `xf_tree_init_parser` function, this global iterator points automatically to the root element. Each function uses a local iterator for searching the requested element.

The global iterator behaves as follows:

- If the requested element is found, the global iterator is set to point to the found element.
- When an `xf_tree_read_*_element_value` function is called several times, the search will start just after the point where the global iterator was left in the preceding call.
- In case the requested element name is not found, the global iterator will not be moved
- If the element is found and an error occurs, i.e., because of a type conversion error or an empty element, the iterator will point to the found element.
- The `xf_tree_rewind` function allows to reset the iterator to the root element in order to start over from the beginning.

Sometimes, it is desirable to have a function that reads all elements with a given name and creates an array of values. This functionality is achieved by the `xf_tree_read_*_array_value` family of functions. A call to any of these functions will first set the iterator pointing to the root element, then will traverse the file looking for the requested elements using the `xf_tree_read_*_element_value` family of functions and will return the array of values leaving the iterator pointing to the last element read.

Attributes can be read with the family of functions `xf_tree_read_*_attribute_value` specifying the attribute name as well as the name of the element that contains it. If the global iterator is pointing to an element name different from the specified one, this element is first looked for, the iterator is moved to it and then the requested attribute is looked for within that element. If the iterator is already pointing to an element whose name matches the specified one, the attribute is directly searched for within that element.

Reading a file using path expressions

Traversing a file sequentially may not always be the optimum way of accessing information if we just need to read a few values. Random access to the in-memory representation of an XML file is achieved using XPath expressions. XPath expressions allow addressing XML elements and attributes using a syntax that resembles the one used for addressing files and directories in a hierarchical file system and is based on the following rules:

- Element names are separated by the / character.
- XPath expressions may be absolute (i.e., starting with the / character and listing all elements from the root to the requested one), or relative to the current position of the iterator (i.e., without the leading /).
- . is used for the current element, and .. for the parent element as it is used for directories.
- If an element has a collection of children with the same name, individual elements may be accessed using their element name followed by an integer index between square brackets, being 1 the index to the first element (i.e., A[1], for the first occurrence of element A, A[2] for the second and so on)
- Attribute names are preceded by the @ character.

The `xf_tree_path_read_*` family of functions implement random access file reading using XPath expressions. The global iterator has the same behaviour as in the `xf_tree_read_*` familliy of functions.

Searching for elements with a given value

The EO_FILE_HANDLING library provides two functions for searching the XPath expression of an element with a given value.

- The `xf_tree_find_string_value_element` function uses an element name and a string value and returns the first element found whose value matches the value provided as input.
- The `xf_tree_find_string_value_path` function uses an XPath expression and a string value. The XPath expression must contain the `[*]` character sequence in order to indicate the elements in a sequence (list) that are iterated.

The search functions set the file iterator pointing to the found element and return its absolute XPath. If the search operation returns an error, the iterator is not moved.

Creating new elements or removing existing elements

A file can be created in memory from scratch using the `xf_tree_create` function. Like the `xf_tree_-init_parser` that reads an existing XML file from disk, it returns a unique integer value that must be used thereon to access the in-memory representation of an XML file just created.

When a file is created in memory, the first operation that must be performed is the creation of the root element, which is done by the `xf_tree_create_root` function.

From that point, elements may be added using as input a reference element to which they are attached and the created element name. The element to which the new element is attached is referenced using an XPath expression. The following possibilities are available.

- Add a child to the reference element using `xf_tree_add_child`. If the reference element already has children, the new element will be appended at the end of the children list.
- Add a sibling after the reference element using `xf_tree_add_next_sibling`. This will insert the new element just after the reference one.
- Add a sibling before the reference element using `xf_tree_add_previous_sibling`. This will insert the new element just before the reference one.
- Attributes are added with the `xf_tree_add_attribute` function using the name of the element that contains them as the reference element.

After a new element has been added to the in-memory representation of the XML file, the global iterator is pointing to the last added element. Therefore, relative XPath expressions used just after an element insertion are relative to the inserted element.

Existing elements are removed using the xf_tree_remove function with an XPath expression addressing them. If the requested element has children, all of them will be removed recursively.

Setting and/or modifying element values

When elements and attributes are created, they have no value. Element and attribute values may be set and/or modified with the `xf_tree_set_*_node_value` family of functions using an XPath expression for addressing the requested element or attribute. Format information as in the C standard library `printf` family of functions must be provided when setting an element or attribute value.

Writing an XML file to disk

An in-memory representation of an XML file can be written to a file on disk with the `xf_tree_write` function. Note that if the in-memory representation is loaded from a file, it may be written under a different file name.

Releasing memory

The in-memory representation of an XML file normally takes a significant amount of memory space, typically between 7 and 10 times the size of the file on disk. For this reason it is important to provide the means for releasing the memory allocated after being used. This can be done using the `xf_tree_cleanup_*` family of functions.

<**font size=+1**>**Earth Observation Header Functions**</**font**>

The EO_FILE_HANDLING library includes a set of functions for the creation of the file headers and filenames for the Earth Observation Missions according to the File Format Standards Document (CS-TN-ESA-GS-0154). These provide the following functionalities:

- Create a fixed header with empty values and an empty variable header
- Set the value of a fixed header element given the element name
- Get the value of a fixed header element given the element name
- Get the values of all fixed header elements
- Create a standard filename from its component items
- Decompose a standard filename in its individual components

The EO_FILE_HANDLING library has been designed as a general purpose simple and reusable library for reading and writing XML files. The Earth Observation Header functions are only relevant to those users developing applications for the ESA Earth Observation Missions. For this reason and in order to provide an easy way to disable these functions for users not interested in them, their inclusion is controlled by a conditional compilation preprocessor directive.

- Users linking their applications with other higher level EARTH OBSERVATION CFI libraries do not need to do anything in order to include the Earth Observation Header functions in the compilation and linking process.
- Users linking their applications only with the EO_FILE_HANDLING library must define the `XF_-EARTH_EXPLORER_HEADER` macro in order to add the header functions to the compilation and linking process. This may be done in two ways:
  - By adding this macro definition in the code before the preprocessor directive that includes the `explorer_file_handling.h` header file
  - By adding this macro definition in the compiler command using the -D flag

Runtime Performances

The library performance has been measured by dedicated test procedures run in 5 different platforms under the below specified machines:

OS ID	Processor	OS	RAM
LINUX64	Intel(R) Xeon(R) CPU E5-2470 0 @ 2.30GHz (16 cores)	GNU LINUX 2.6.35-22-generic (Ubuntu 10.10)	16 GB
LINUX32 LEGACY	Intel(R) Core(TM)2 Quad CPU Q8400 @ 2.66GHz	GNU LINUX 2.6.24-16-generic (Ubuntu 8.04)	4 GB
LINUX64 LEGACY	Intel(R) Core(TM)2 Quad CPU Q8400 @ 2.66GHz	GNU LINUX 2.6.24-16-generic (Ubuntu 8.04)	4 GB
MACIN64	Intel Core i7 4 cores @2,6 GHz	MAC OSX V10.10	16 GB
WINDOWS32	Intel(R) Xeon(R) CPU ES-2630 @ 2.40 GHz 2.40 GHz	Microsoft Windows 7 Professional	16 GB
WINDOWS64	Intel(R) Xeon(R) CPU ES-2630 @ 2.40 GHz 2.40 GHz	Microsoft Windows 7 Professional	16 GB

The table below shows the time (in miliseconds - ms) each function takes to be run under each platform:

Function ID	WIN-DOWS32	WIN-DOWS64	LINUX64	LINUX64-LEGACY	LINUX32-LEGACY	MACIN64
xf_tree_-init_parser	0.300000	0.200000	0.000000	0.000000	0.000000	0.000000
xf_tree_-create_root	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
xf_tree_-add_child	0.001625	0.001250	0.000000	0.001250	0.002500	0.001250
xf_tree_-add_next_-sibling ∗ 1000 elements	0.002000	0.002000	0.000000	0.010000	0.000000	0.000000
xf_tree_-add_-previous_-sibling	0.002000	0.002000	0.000000	0.000000	0.000000	0.010000

xf_tree_- set_- integer_- node_value	0.115300	0.123900	0.087000	0.314000	0.114000	0.060000
xf_tree_- read_- integer_- element_- value	0.000325	0.000282	0.000210	0.000240	0.000410	0.000250
xf_tree_- read_- integer_- element_- array ∗ 10000 elements	2.070000	1.820000	1.300000	2.400000	2.400000	1.900000
xf_tree_- path_read_- integer_- node_value	0.226000	0.244000	0.170000	0.840000	0.270000	0.110000
xf_tree_- path_read_- integer_- node_array ∗ 10000 elements	1.730000	1.520000	0.900000	2.400000	2.000000	1.800000
xf_tree_- set_real_- node_value	0.116800	0.125200	0.086000	0.304000	0.115000	0.060000

xf_tree_-read_real_-element_-value *with xf_tree_-rewind	0.000500	0.000500	0.001000	0.000000	0.001000	0.001000
xf_tree_-read_real_-element_-array * 10000 elements	4.010000	3.330000	1.900000	3.300000	3.200000	2.000000
xf_tree_-path_read_-real_node_-value	0.225000	0.244000	0.170000	0.840000	0.260000	0.120000
xf_tree_-path_read_-real_node_-array * 10000 elements	3.670000	3.030000	1.500000	3.000000	2.800000	1.900000
xf_tree_-set_string_-node_value	0.115900	0.124400	0.087000	0.295000	0.113000	0.059000
xf_tree_-read_-string_-element_-value * with xf_-tree_rewind	0.000300	0.000300	0.000000	0.000000	0.001000	0.001000

xf_tree_-read_-string_-element_-array * 10000 elements	0.010000	0.010000	0.000000	0.000000	0.000000	0.000000
xf_tree_-path_read_-string_-node_value	0.226000	0.243000	0.170000	0.850000	0.250000	0.110000
xf_tree_-path_read_-string_-node_array * 10000 elements	0.020000	0.020000	0.000000	0.000000	0.100000	0.000000
xf_tree_-add_-attribute	0.115800	0.124900	0.087000	0.296000	0.112000	0.061000
xf_tree_-read_-integer_-attribute	0.000321	0.000249	0.000230	0.000260	0.000340	0.000370
xf_tree_-read_real_-attribute	0.000530	0.000432	0.000310	0.000370	0.000460	0.000400
xf_tree_-read_-string_-attribute	0.000305	0.000233	0.000210	0.000290	0.000350	0.000350
xf_tree_go_-to_path_-node	0.225100	0.242800	0.168000	0.876000	0.258000	0.116000
xf_tree_go_-to_-element_-node * with xf_tree_-rewind	0.000200	0.000200	0.000000	0.000000	0.000000	0.000000
xf_tree_go_-to_next_-element_-node	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
xf_tree_-create_-header	0.002900	0.002400	0.002000	0.002000	0.003000	0.002000
xf_tree_-get_fixed_-header_-item	0.000382	0.000360	0.000350	0.000490	0.000570	0.000280

xf_tree_-get_fixed_-header_-items	0.003600	0.003000	0.003000	0.005000	0.006000	0.003000
xf_tree_-set_fixed_-header_-items	0.020700	0.018000	0.018000	0.021000	0.026000	0.019000
xf_read_-filename_-items	0.000447	0.000451	0.000340	0.000480	0.000570	0.000280
xf_tree_-find_-string_-value_-element * timing includes xf_tree_-rewind	2.838000	2.523000	2.010000	3.680000	3.820000	2.990000
xf_tree_-find_-string_-value_path	1.598000	1.352000	1.360000	2.800000	2.160000	1.900000
xf_tree_-get_-current_-element_-name	0.000089	0.000071	0.000050	0.000060	0.000070	0.000060
xf_tree_-get_path	0.145000	0.165000	0.130000	0.820000	0.230000	0.110000
xf_tree_-rewind	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
xf_tree_-write	9.800000	9.800000	16.000000	24.000000	37.000000	16.000000
xf_copy_-node * Copied node with 10000 elements	41.000000	39.000000	40.000000	70.000000	60.000000	60.000000
xf_tree_-remove_-node * /Earth_-Explorer_-File/Data_-Block2	5.000000	6.000000	10.000000	10.000000	10.000000	10.000000
xf_set_-schema	36.650002	35.830002	51.500000	74.699997	81.400002	57.099998
xf_tree_-get_-namespace	0.000769	0.000605	0.000410	0.000650	0.000760	0.000650

Note that when the value 0.000000 is defined for a function in a certain platform, it means that its running time is lower than 1 nano-second and so it can be considered as 0.

2   EARTH OBSERVATION FILE HANDLING File Index

2.1   EARTH OBSERVATION FILE HANDLING File List

Here is a list of all documented files with brief descriptions:

explorer_file_handling.h (Public header file for the explorer_file_handling library)   11

3   EARTH OBSERVATION FILE HANDLING File Documentation

3.1   explorer_file_handling.h File Reference

Public header file for the explorer_file_handling library.

Defines

- #define XF_MAX_FILES_NUMBER 10
- #define XF_MAX_ERROR_MSG_LENGTH 512
- #define XF_MAX_XML_NODE_NAME_LENGTH 64
- #define XF_MAX_VALUE_LENGTH 512
- #define XF_MAX_ARRAY_SIZE 10000
- #define XF_MAX_ATTR_ARRAY_SIZE 10
- #define XF_MAX_PATH_LENGTH 256
- #define XF_MAX_FILENAME_LENGTH 64

Enumerations

- enum XF_CFI_General_err_enum { XF_CFI_FIRST_ELEMENT = -40, XF_CFI_ARRAY_-EXCEEDED, XF_CFI_LOCK_ERROR, XF_CFI_UNLOCK_ERROR, XF_CFI_VALUE_NOT_-FOUND, XF_CFI_FILENAME_TOO_LONG, XF_CFI_WRONG_HEADER_ELEMENT, XF_CFI_-WRONG_HEADER_FILE_EXTENSION, XF_CFI_WRONG_HEADER_FILE_FORMAT, XF_-CFI_EMPTY_ELEMENT, XF_CFI_NO_ARRAY_PATH, XF_CFI_SAVING_DOC_ERROR, XF_-CFI_INVALID_FILE_FORMAT, XF_CFI_INVALID_FORMAT, XF_CFI_REMOVING_NODE_-ERROR, XF_CFI_ADDING_SIBLING_ERROR, XF_CFI_CREATING_NODE_ERROR, XF_-CFI_ROOT_ALREADY_EXISTS, XF_CFI_CREATING_DOC_ERROR, XF_CFI_WRONG_FILE_-DESCRIPTOR, XF_CFI_CURRENT_NODE_IS_NULL, XF_CFI_MAX_FILES_REACHED, XF_-CFI_NODE_NAME_TOO_LONG, XF_CFI_PATH_TOO_LONG, XF_CFI_NOT_A_TERMINAL_-ELEMENT, XF_CFI_NO_ATTRIBUTES, XF_CFI_VALUE_OUT_OF_RANGE, XF_CFI_NOT_A_-DOUBLE, XF_CFI_NOT_A_LONG, XF_CFI_NO_ELEMENT_FOUND, XF_CFI_ROOT_NODE_-IS_NULL, XF_CFI_NOT_AN_ELEMENT_NODE, XF_CFI_NO_ELEMENT_REQUESTED, XF_CFI_DOC_NOT_PARSED_OR_CREATED, XF_CFI_MEMORY_ERROR, XF_CFI_GETTING_-ROOT_ERROR, XF_CFI_PARSING_FILE_ERROR, XF_CFI_BAD_INPUT_ARGUMENT, XF_-CFI_GENERIC_ERROR, XF_CFI_UNKNOWN_ERROR_CODE, XF_CFI_OK = 0, XF_CFI_-ROOT_ALREADY_REACHED, XF_CFI_TOO_MANY_ELEMENTS, XF_CFI_MAX_LENGTH_-EXCEEDED, XF_CFI_GENERIC_WARNING, XF_CFI_HEADER_ALREADY_EXISTS, XF_-CFI_LAST_ELEMENT }

List of error and warning codes.

- enum XF_File_extension_type_enum { XF_HEADER_FORMAT_EEF, XF_HEADER_FORMAT_-
HDR, XF_HEADER_FORMAT_DBL, XF_HEADER_FORMAT_NONE, XF_HEADER_-
FORMAT_EOF }

    List of XML Header files allowed.

- enum XF_Set_schema_enum { XF_CHANGE_SCHEMA_AND_VERSION, XF_CHANGE_-
SCHEMA, XF_DELETE_ALL, XF_DELETE_SCHEMA }

    List of actions to do when calling to xf_set_schema.

- enum XF_Sat_id_enum { XF_SAT_DEFAULT = 0 , XF_SAT_ERS1 = 11, XF_SAT_ERS2 = 12,
XF_SAT_ENVISAT = 21, XF_SAT_METOP1 = 31, XF_SAT_METOP2 = 32, XF_SAT_METOP3
= 33, XF_SAT_CRYOSAT = 41, XF_SAT_ADM = 51, XF_SAT_GOCE = 61, XF_SAT_SMOS = 71,
XF_SAT_TERRASAR = 81, XF_SAT_EARTHCARE = 91, XF_SAT_SWARM_A = 101, XF_SAT_-
SWARM_B = 102, XF_SAT_SWARM_C = 103 }

    List of Satellite Ids allowed.

Functions

- long xf_tree_init_parser (char ∗file, long ∗error)

    Loads an XML file into memory.

- void xf_tree_cleanup_parser (long ∗fd, long ∗error)

    Releases parser resources.

- void xf_tree_cleanup_all_parser (void)

    Releases all parser resources.

- void xf_tree_get_current_element_name (long ∗fd, char ∗∗name, long ∗error)

    Gets the name of the current element.

- void xf_tree_read_integer_element_value (long ∗fd, char ∗element, long ∗value, long ∗error)

    Reads an integer number value.

- void xf_tree_read_integer_element_array (long ∗fd, char ∗element, long ∗array, long ∗length, long
∗error)

    Reads an array of integer number values.

- void xf_tree_read_real_element_value (long ∗fd, char ∗element, double ∗value, long ∗error)

    Reads a real number value.

- void xf_tree_read_real_element_array (long ∗fd, char ∗element, double ∗array, long ∗length, long
∗error)

    Reads an array of real number values.

- void xf_tree_read_string_element_value (long ∗fd, char ∗element, char ∗∗value, long ∗error)

    Reads a string value.

- void xf_tree_read_string_element_array (long *fd, char *element, char ***array, long *length, long *error)

  Reads an array of string values.

- void xf_tree_read_string_attribute (long *fd, char *element, char *attribute_name, char **attribute_-value, long *error)

  Reads an attribute as string.

- void xf_tree_read_integer_attribute (long *fd, char *element, char *attribute_name, long *attribute_-value, long *error)

  Reads an attribute as a long.

- void xf_tree_read_real_attribute (long *fd, char *element, char *attribute_name, double *attribute_-value, long *error)

  Reads an attribute as a double.

- void xf_tree_path_read_string_node_value (long *fd, char *path, char **value, long *error)

  Reads a string value given the path to the node.

- void xf_tree_path_read_integer_node_value (long *fd, char *path, long *value, long *error)

  Reads a integer value given the path to the node.

- void xf_tree_path_read_real_node_value (long *fd, char *path, double *value, long *error)

  Reads a double value given the path to the node.

- void xf_tree_path_read_string_node_array (long *fd, char *path, char ***array, long *length, long *error)

  Reads an array of string characters given the Xpath to the node.

- void xf_tree_path_read_integer_node_array (long *fd, char *path, long *array, long *length, long *error)

  Reads an array of long values given the Xpath to the node.

- void xf_tree_path_read_real_node_array (long *fd, char *path, double *array, long *length, long *error)

  Reads an array of double values given the Xpath to the node.

- void xf_tree_get_namespace (long *fd, char *node_name, long *num_ns, char *prefix[XF_MAX_-ARRAY_SIZE], char *url[XF_MAX_ARRAY_SIZE], long *error)

  Reads all the namespace that applies to a given node.

- void xf_tree_rewind (long *fd, long *error)

  Sets the read pointer to the beginning of the file.

- void xf_tree_go_to_path_node (long *fd, char *path, long *error)

  Goes to the node specified in the input XPath.

- void xf_tree_go_to_element_node (long *fd, char *element, long *error)

  Goes to the next element in the tree.

- void xf_tree_go_to_next_element_node (long ∗fd, long ∗error)

    Goes to the next element in the tree.

- void xf_tree_get_path (long ∗fd, char ∗path, long ∗error)

    Gets the XPath of the current tree pointer.

- long xf_tree_create (long ∗error)

    Create a memory representation of an empty XML document.

- void xf_tree_create_root (long ∗id, char ∗name, long ∗error)

    Create the root element of an XML document.

- void xf_tree_add_child (long ∗id, char ∗parent, char ∗name, long ∗error)

    Add a new element to an XML document as a child of parent.

- void xf_tree_add_next_sibling (long ∗id, char ∗current, char ∗name, long ∗error)

    Add a new element to an XML document after current.

- void xf_tree_add_previous_sibling (long ∗id, char ∗current, char ∗name, long ∗error)

    Add a new element to an XML document before current.

- void xf_tree_add_attribute (long ∗id, char ∗current, char ∗name, long ∗error)

    Add a new attribute carried by an element.

- void xf_tree_remove_node (long ∗id, char ∗name, long ∗error)

    Remove a node.

- void xf_tree_set_integer_node_value (long ∗id, char ∗name, long ∗value, char ∗format, long ∗error)

    Set an integer node value.

- void xf_tree_set_real_node_value (long ∗id, char ∗name, double ∗value, char ∗format, long ∗error)

    Set a real number node value.

- void xf_tree_set_string_node_value (long ∗id, char ∗name, char ∗value, char ∗format, long ∗error)

    Set a string node value.

- void xf_tree_write (long ∗id, char ∗name, long ∗error)

    Write the data to a file on disk.

- void xf_set_schema (char ∗filename, char ∗schema, long ∗action, long ∗error)

    It sets the schema in the root element of the filename.

- void xf_tree_find_string_value_element (long ∗fd, char ∗element, char ∗value, char ∗found_path, long ∗error)

    Find the XPath name of an element given an element name and value.

- void xf_tree_find_string_value_path (long ∗fd, char ∗path, char ∗value, char ∗found_path, long ∗error)

    Find the XPath name of an element given an XPath expression and value.

- void xf_basic_error_msg (long error_code, char ∗error_message)

  Gets default message corresponding to the input error code.

- void xf_verbose ()

  Set verbosity flag.

- void xf_silent ()

  Unset verbosity flag.

- void xf_tree_create_header (long ∗fd, long ∗file_extension_type, long ∗error)

  Generates a new blank Header including both Fixed and Variable headers.

- void xf_tree_set_fixed_header_item (long ∗fd, char ∗item_id, char ∗item_value, long ∗error)

  Sets the value of any element of a Header.

- void xf_tree_set_fixed_header_items (long ∗fd, char ∗file_name, char ∗file_description, char ∗notes, char ∗mission, char ∗file_class, char ∗file_type, char ∗validity_start, char ∗validity_stop, long ∗file_version, char ∗system, char ∗creator, char ∗creator_version, char ∗creation_date, long ∗error)

  Sets the value of all Fixed Header elements.

- void xf_tree_get_fixed_header_item (long ∗fd, char ∗item_id, char ∗∗item_value, long ∗error)

  Gets the value of any element of a Header.

- void xf_tree_get_fixed_header_items (long ∗fd, char ∗∗file_name, char ∗∗file_description, char ∗∗notes, char ∗∗mission, char ∗∗file_class, char ∗∗file_type, char ∗∗validity_start, char ∗∗validity_stop, long ∗file_version, char ∗∗system, char ∗∗creator, char ∗∗creator_version, char ∗∗creation_date, long ∗error)

  Gets the value of all Fixed Header elements.

- void xf_create_filename (long ∗satellite_id, char ∗file_class, char ∗file_type, char ∗instance_id, long ∗file_extension_type, char ∗filename, long ∗error)

  Generates a complete Cryosat File Name.

- void xf_read_filename_items (char ∗filename, long ∗satellite_id, char ∗file_class, char ∗file_type, char ∗instance_id, long ∗error)

  Reads a filename and returns its parameters.

- void xf_copy_node (long ∗fd, char ∗dest_node, char ∗src_file, char ∗source_node, long ∗error)

  Copy a node element from a file in other file.

- long xf_check_library_version ()

  Get the library version.

3.1.1   Detailed Description

Public header file for the explorer_file_handling library.

Version:

    4.14

Date:
16/11/2017

Copyright DEIMOS SPACE S.L.U

Project: EOCFI

This product includes software developed by the GNOME Project `http://xmlsoft.org/`

3.1.2   Define Documentation

3.1.2.1   #define XF MAX ARRAY SIZE 10000

Maximum number of elements in an array of values

3.1.2.2   #define XF MAX ATTR ARRAY SIZE 10

Maximum number of attributes

3.1.2.3   #define XF MAX ERROR MSG LENGTH 512

Maximum number of characters in error message

3.1.2.4   #define XF MAX FILENAME LENGTH 64

Maximum number of characters in a filename

3.1.2.5   #define XF MAX FILES NUMBER 10

Maximum number of open files allowed

3.1.2.6   #define XF MAX PATH LENGTH 256

Maximum number of characters in a path

3.1.2.7   #define XF MAX VALUE LENGTH 512

Maximum number of characters in content value of XML nodes

3.1.2.8   #define XF MAX XML NODE NAME LENGTH 64

Maximum number of characters in element or attribute name

3.1.3   Enumeration Type Documentation

3.1.3.1   enum XF CFI General err enum

List of error and warning codes.

- OK : code = 0
- Warning : code > 0
- Error : code < 0

Enumeration values:

XF_CFI_FIRST_ELEMENT First element

XF_CFI_ARRAY_EXCEEDED Maximum number of elements in array exceeded

XF_CFI_LOCK_ERROR Could not lock other running threads

XF_CFI_UNLOCK_ERROR Could not unlock other blocked threads

XF_CFI_VALUE_NOT_FOUND The requested value has not been found

XF_CFI_FILENAME_TOO_LONG The filename is too long

XF_CFI_WRONG_HEADER_ELEMENT Element not allowed to be included in a header

XF_CFI_WRONG_HEADER_FILE_EXTENSION File can?t contain a header

XF_CFI_WRONG_HEADER_FILE_FORMAT The header file contains unwanted tags

XF_CFI_EMPTY_ELEMENT Element is empty

XF_CFI_NO_ARRAY_PATH The given path is not for an array

XF_CFI_SAVING_DOC_ERROR Unable to save the XML document into disk

XF_CFI_INVALID_FILE_FORMAT Unable to read an item. Invalid file format

XF_CFI_INVALID_FORMAT Printing format provided is not valid

XF_CFI_REMOVING_NODE_ERROR Unable to remove an element node

XF_CFI_ADDING_SIBLING_ERROR Unable to add a sibling node

XF_CFI_CREATING_NODE_ERROR Unable to create an element node

XF_CFI_ROOT_ALREADY_EXISTS Root element already exists

XF_CFI_CREATING_DOC_ERROR Error when creating a new XML document

XF_CFI_WRONG_FILE_DESCRIPTOR File descriptor out of range

XF_CFI_CURRENT_NODE_IS_NULL Current node is NULL. Rewind is needed

XF_CFI_MAX_FILES_REACHED The max. number of open files has been reached. No more files are allowed

XF_CFI_NODE_NAME_TOO_LONG Name of node exceeds the maximum allowed

XF_CFI_PATH_TOO_LONG Path length exceeds the maximum allowed

XF_CFI_NOT_A_TERMINAL_ELEMENT Current node has other xml elements as value

XF_CFI_NO_ATTRIBUTES Current node has no attributes

XF_CFI_VALUE_OUT_OF_RANGE Element value (integer or real) is out of range

XF_CFI_NOT_A_DOUBLE Value to be converted is not a double

XF_CFI_NOT_A_LONG Value to be converted is not an integer

XF_CFI_NO_ELEMENT_FOUND No element was found in the search

XF_CFI_ROOT_NODE_IS_NULL Root node is NULL. Initialization is needed

XF_CFI_NOT_AN_ELEMENT_NODE Current node is not an xml element node

XF_CFI_NO_ELEMENT_REQUESTED There is no element name set to be searched

XF_CFI_DOC_NOT_PARSED_OR_CREATED No XML document has been parsed or created

XF_CFI_MEMORY_ERROR Unable to ask for memory

XF_CFI_GETTING_ROOT_ERROR Unable to get the root element during initialisation

XF_CFI_PARSING_FILE_ERROR Error during initialisation

XF_CFI_BAD_INPUT_ARGUMENT Bad input argument

XF_CFI_GENERIC_ERROR Generic error

XF_CFI_UNKNOWN_ERROR_CODE Error code for unknown error codes

XF_CFI_OK OK

XF_CFI_ROOT_ALREADY_REACHED Warning used when the path goes up too much

XF_CFI_TOO_MANY_ELEMENTS The number of elements in the file exceeds the size of the storage variable

XF_CFI_MAX_LENGTH_EXCEEDED Length of element name or value exceeds the size of the storage variable

XF_CFI_GENERIC_WARNING Generic warning

XF_CFI_HEADER_ALREADY_EXISTS Header already exists

XF_CFI_LAST_ELEMENT Last element

3.1.3.2 enum XF_File_extension_type_enum

List of XML Header files allowed.

Enumeration values:

XF_HEADER_FORMAT_EEF XML file with Header and Datablock (Earth Explorer mission)

XF_HEADER_FORMAT_HDR XML file with Header

XF_HEADER_FORMAT_DBL XML file with Datablock

XF_HEADER_FORMAT_NONE No extension when generating filename

XF_HEADER_FORMAT_EOF XML file with Header and Datablock (Earth Observation mission)

3.1.3.3 enum XF_Sat_id_enum

List of Satellite Ids allowed.

Enumeration values:

XF_SAT_DEFAULT Default satellite Id

XF_SAT_ERS1 Satellite Id for ERS1

XF_SAT_ERS2 Satellite Id for ERS2

XF_SAT_ENVISAT Satellite Id for Envisat

XF_SAT_METOP1 Satellite Id for MetOp1

XF_SAT_METOP2 Satellite Id for MetOP2

XF_SAT_METOP3 Satellite Id for MetOp3

XF_SAT_CRYOSAT Satellite Id for Cryosat

XF_SAT_ADM Satellite Id for Aeolus

XF_SAT_GOCE Satellite Id for GOCE

XF_SAT_SMOS Satellite Id for SMOS

XF_SAT_TERRASAR Satellite Id for TERRASAR

XF_SAT_EARTHCARE Satellite Id for EARTHCARE

XF_SAT_SWARM_A Satellite Id for SWARM A

XF_SAT_SWARM_B Satellite Id for SWARM B

XF_SAT_SWARM_C Satellite Id for SWARM C

3.1.3.4 enum XF_Set_schema_enum

List of actions to do when calling to xf_set_schema.

Enumeration values:

    XF_CHANGE_SCHEMA_AND_VERSION Change schema and version

    XF_CHANGE_SCHEMA Change schema

    XF_DELETE_ALL Delete schema and version attributes

    XF_DELETE_SCHEMA Delete schema attribute

3.1.4 Function Documentation

3.1.4.1 void xf_basic_error_msg (long *error_code*, char ∗ *error_message*)

Gets default message corresponding to the input error code.

Parameters:

    error_code (IN): Code of the error or warning to be read.

    error_message (OUT): Error message associated to the input code.

This function returns the error message associated with the input error code. If the error code is unknown, an adequate error message is returned.

- Assumptions :
    - It is assumed that the size of the *error_message* variable is equal or greater than *XF_MAX_ERROR_MSG_LENGTH*
- Errors : N/A
- Warnings : N/A
- Reference : N/A

3.1.4.2 xf_check_library_version ()

Get the library version.

This function prints in the standard output the library version

- Assumptions : None
- Errors : N/A
- Warnings : N/A
- Reference : None

3.1.4.3 xf_copy_node (long ∗ *fd*, char ∗ *dest_node*, char ∗ *src_file*, char ∗ *source_node*, long ∗ *error*)

Copy a node element from a file in other file.

Parameters:

    fd (IN): Input file descriptor

    dest_node (IN): Destination node name

src_file (IN): Source filename

source_node (IN): Node to copy

errr (OUT): Error code

This function copies a node from the src_file into the dest_node within the file whose descriptor is fd

- Assumptions :
- Errors :

    – XF_CFI_NO_ELEMENT_FOUND
    – XF_CFI_PARSING_FILE_ERROR

- Warnings : N/A
- Reference : None

**3.1.4.4   void xf_create_filename (long ∗ *mission_id*, char ∗ *file_class*, char ∗ *file_type*, char ∗ *instance_-*
id, long ∗ *file_extension_type*, char ∗ *filename*, long ∗ *error*)**

Generates a complete Cryosat File Name.

Parameters:

mission_id (IN): Satellite ID to add the following to the filename. See *XF_Sat_id_enum*

file_class (IN): Type of activity for which the file is used

file_type (IN): File Type

instance_id (IN): Makes the file unique

file_extension_type (IN): Type of XML file to add Header. See *XF_File_extension_type_enum*

filename (OUT): Filename generated

error (OUT): Error code

This function generates a complete Cryosat File Name

- Assumptions :

    – XF_EARTH_EXPLORER_HEADER macro is defined

- Errors :

    – XF_CFI_MAX_LENGTH_EXCEEDED
    – XF_CFI_BAD_INPUT_ARGUMENT

- Warnings : N/A
- Reference : None

**3.1.4.5   void xf_read_filename_items (char ∗ *filename*, long ∗ *satellite_id*, char ∗ *file_class*, char ∗ *file_-*
type, char ∗ *instance_id*, long ∗ *error*)**

Reads a filename and returns its parameters.

Parameters:

filename (IN): Input filename

satellite_id (OUT): Satellite ID corresponding to the file. See *XF_Sat_id_enum*

file_class (OUT): Type of activity for which the file is used

file_type (OUT): File Type

instance_id (OUT): Makes the file unique

error (OUT): Error code

This function generates a complete Cryosat File Name

- Assumptions :
    - XF_EARTH_EXPLORER_HEADER macro is defined
- Errors :
    - XF_CFI_MAX_LENGTH_EXCEEDED
    - XF_CFI_BAD_INPUT_ARGUMENT
- Warnings : N/A
- Reference : None

3.1.4.6 void xf_set_schema (char ∗ *filename*, char ∗ *schema*, long ∗ *action*, long ∗ *error*)

It sets the schema in the root element of the filename.

filename (IN):
   filename

schema (IN):
   schema path.

action (IN):
   flag to indicate what to do:

- XF_CHANGE_SCHEMA_AND_VERSION: Change schema and version
- XF_CHANGE_SCHEMA: Change schema
- XF_DELETE_ALL: Delete schema and version attributes
- XF_DELETE_SCHEMA: Delete schema attribute

error(OUT):
   Error code.

This function sets the schema in the root element of the filename

- Assumptions : N/A
- Errors :
- XF_CFI_PARSING_FILE_ERROR
- XF_CFI_SAVING_DOC_ERROR
- Warnings : N/A
- Reference : See the *libxml* documentation

3.1.4.7 void xf_silent ()

Unset verbosity flag.

This function unset the verbosity flag. When the flag is unset the errors raised by explorer_file_handling functions as well as the ones raised by libxml are NOT displayed to stdedd.

3.1.4.8 void xf_tree_add_attribute (long ∗ *id*, char ∗ *current*, char ∗ *name*, long ∗ *error*)

Add a new attribute carried by an element.

Parameters:

  id (IN): Descriptor of the XML document memory representation

  current (IN): XPath name of the element

  name (IN): New attribute name

  error (OUT): Error code

This function adds a new attribute to the *current* element

- Assumptions : N/A
- Errors :

    – XF_CFI_BAD_INPUT_ARGUMENT
    – XF_CFI_WRONG_FILE_DESCRIPTOR
    – XF_CFI_DOC_NOT_PARSED_OR_CREATED
    – XF_CFI_ROOT_NODE_IS_NULL
    – XF_CFI_PATH_TOO_LONG
    – XF_CFI_NODE_NAME_TOO_LONG
    – XF_CFI_NO_ELEMENT_FOUND

- Warnings :

    – XF_CFI_ROOT_ALREADY_REACHED

- Reference : See the *libxml* documentation

3.1.4.9 void xf_tree_add_child (long ∗ *id*, char ∗ *parent*, char ∗ *name*, long ∗ *error*)

Add a new element to an XML document as a child of parent.

Parameters:

  id (IN): Descriptor of the XML document memory representation

  parent (IN): Parent XPath name

  name (IN): New element name

  error (OUT): Error code

This function adds a new element to an XML document at the end of the *parent* child list

- Assumptions : N/A
- Errors :

    – XF_CFI_BAD_INPUT_ARGUMENT
    – XF_CFI_WRONG_FILE_DESCRIPTOR
    – XF_CFI_DOC_NOT_PARSED_OR_CREATED
    – XF_CFI_ROOT_NODE_IS_NULL
    – XF_CFI_PATH_TOO_LONG
    – XF_CFI_NODE_NAME_TOO_LONG
    – XF_CFI_NO_ELEMENT_FOUND
    – XF_CFI_CREATING_NODE_ERROR
    – XF_CFI_ADDING_SIBLING_ERROR

- Warnings :

    – XF_CFI_ROOT_ALREADY_REACHED

- Reference : See the *libxml* documentation

3.1.4.10   void xf_tree_add_next_sibling (long ∗ *id*, char ∗ *current*, char ∗ *name*, long ∗ *error*)

Add a new element to an XML document after current.

Parameters:

   id (IN): Descriptor of the XML document memory representation

   current (IN): XPath name of the element used as reference for the new element insertion

   name (IN): New element name

   error (OUT): Error code

This function adds a new element to an XML document after the *current* element

- Assumptions : N/A
- Errors :

    – XF_CFI_BAD_INPUT_ARGUMENT
    – XF_CFI_WRONG_FILE_DESCRIPTOR
    – XF_CFI_DOC_NOT_PARSED_OR_CREATED
    – XF_CFI_ROOT_NODE_IS_NULL
    – XF_CFI_PATH_TOO_LONG
    – XF_CFI_NODE_NAME_TOO_LONG
    – XF_CFI_NO_ELEMENT_FOUND
    – XF_CFI_CREATING_NODE_ERROR
    – XF_CFI_ADDING_SIBLING_ERROR

- Warnings :

    – XF_CFI_ROOT_ALREADY_REACHED

- Reference : See the *libxml* documentation

3.1.4.11   void xf_tree_add_previous_sibling (long ∗ *id*, char ∗ *current*, char ∗ *name*, long ∗ *error*)

Add a new element to an XML document before current.

Parameters:

   id (IN): Descriptor of the XML document memory representation

   current (IN): XPath name of the element used as reference for the new element insertion

   name (IN): New element name

   error (OUT): Error code

This function adds a new element to an XML document before the *current* element

- Assumptions : N/A
- Errors :

    – XF_CFI_BAD_INPUT_ARGUMENT
    – XF_CFI_WRONG_FILE_DESCRIPTOR
    – XF_CFI_DOC_NOT_PARSED_OR_CREATED
    – XF_CFI_ROOT_NODE_IS_NULL
    – XF_CFI_PATH_TOO_LONG
    – XF_CFI_NODE_NAME_TOO_LONG
    – XF_CFI_NO_ELEMENT_FOUND
    – XF_CFI_CREATING_NODE_ERROR
    – XF_CFI_ADDING_SIBLING_ERROR

- Warnings :

    – XF_CFI_ROOT_ALREADY_REACHED

- Reference : See the *libxml* documentation

3.1.4.12   void xf_tree_cleanup_all_parser (void)

Releases all parser resources.

This function releases the memory resources taken by the XML parser for all the open files.

This function is not thread safe and it should not be called from a multi-thread program.

- Assumptions : N/A
- Errors : N/A
- Warnings : N/A
- Reference : See the *libxml* documentation

3.1.4.13   void xf_tree_cleanup_parser (long ∗ *fd*, long ∗ *error*)

Releases parser resources.

Parameters:
  fd (IN): File descriptor
  error (OUT): Error code

This function releases the memory resources taken by the XML parser for the corresponding file descriptor given in input.

- Assumptions : N/A
- Errors :

    – XF_CFI_WRONG_FILE_DESCRIPTOR

- Warnings : N/A
- Reference : See the *libxml* documentation

3.1.4.14   long xf_tree_create (long ∗ *error*)

Create a memory representation of an empty XML document.

Parameters:
  error (OUT): Error code

Returns:
  A descriptor to be used in all functions for access, insertion and removal of element and attributes from the memory representation of the XML document, and for writing its content to a file.

This function must be called first by the user application in order to create an XML file from scratch

- Assumptions : N/A
- Errors :

    – XF_CFI_MAX_FILES_REACHED
    – XF_CFI_MEMORY_ERROR
    – XF_CFI_CREATING_DOC_ERROR

- Warnings : N/A
- Reference : See the *libxml* documentation

3.1.4.15 void xf_tree_create_header (long ∗ fd, long ∗ file_extension_type, long ∗ error)

Generates a new blank Header including both Fixed and Variable headers.

Parameters:

fd (IN): File descriptor

file_extension_type (IN): Type of XML file to add Header to. See *XF_File_extension_type_enum*

error (OUT): Error code

This function writes the complete Cryosat Header (full Fixed Header + Variable Header). If the XML file already has a Header this function shall not rewrite it again.

- Assumptions :

    - XF_EARTH_EXPLORER_HEADER macro is defined

- Errors :

    - XF_CFI_WRONG_FILE_DESCRIPTOR
    - XF_CFI_DOC_NOT_PARSED_OR_CREATED
    - XF_CFI_CREATING_NODE_ERROR
    - XF_CFI_NO_ELEMENT_FOUND
    - XF_CFI_WRONG_HEADER_FILE_EXTENSION
    - XF_CFI_HEADER_ALREADY_EXISTS

- Warnings : N/A
- Reference : See the *libxml* documentation

3.1.4.16 void xf_tree_create_root (long ∗ id, char ∗ name, long ∗ error)

Create the root element of an XML document.

Parameters:

id (IN): Descriptor of the XML document memory representation

name (IN): Root element name

error (OUT): Error code

This function must be called first by the user application in order to create an XML file from scratch

- Assumptions : N/A
- Errors :

    - XF_CFI_WRONG_FILE_DESCRIPTOR
    - XF_CFI_DOC_NOT_PARSED_OR_CREATED
    - XF_CFI_BAD_INPUT_ARGUMENT
    - XF_CFI_ROOT_ALREADY_EXISTS
    - XF_CFI_CREATING_NODE_ERROR

- Warnings : N/A
- Reference : See the *libxml* documentation

3.1.4.17 void xf_tree_find_string_value_element (long ∗ _fd_, char ∗ _element_, char ∗ _value_, char ∗ _found_path_, long ∗ _error_)

Find the XPath name of an element given an element name and value.

Parameters:

    _fd_ (IN): File descriptor

    element (IN): Element name used for the search operation

    value (IN): Value

    _found_path_ (OUT): XPath expression of the found element

    error (OUT): Error code

This function finds the XPath name of the next element whose name and value match the _element_ and _value_ provided as input

This function iterates through all nodes in the file, starting from the current location

The search operation is based on the exact lexicographical match of both the element name and the value

- Assumptions :

    - It is assumed that ∗_found_path_ has no memory reserved and points to NULL

- Errors :

    - XF_CFI_WRONG_FILE_DESCRIPTOR
    - XF_CFI_DOC_NOT_PARSED_OR_CREATED
    - XF_CFI_ROOT_NODE_IS_NULL
    - XF_CFI_BAD_INPUT_ARGUMENT
    - XF_CFI_NO_ELEMENT_REQUESTED
    - XF_CFI_NO_ELEMENT_FOUND
    - XF_CFI_NOT_AN_ELEMENT_NODE
    - XF_CFI_NOT_A_TERMINAL_ELEMENT
    - XF_CFI_PATH_TOO_LONG
    - XF_CFI_VALUE_NOT_FOUND

- Warnings :

    - XF_CFI_MAX_LENGTH_EXCEEDED

- Reference : See the _libxml_ documentation
- Memory allocation Note: Memory is dynamically allocated internally for ∗_found_path_. The user must free this memory after using it.

3.1.4.18 void xf_tree_find_string_value_path (long ∗ _fd_, char ∗ _path_, char ∗ _value_, char ∗ _found_path_, long ∗ _error_)

Find the XPath name of an element given an XPath expression and value.

Parameters:

    _fd_ (IN): File descriptor

    path (IN): XPath expression used for the search operation

    value (IN): Value

    _found_path_ (OUT): XPath expression of the found element

    error (OUT): Error code

This function finds the XPath name of the first element whose XPath name matches the *path* expression and whose value matches the *value* provided as input.

The XPath expression in the *path* input parameter must reference a list of elements with the characters '[∗]', which must appear once and only once in the path.

This function is intended for searching the XPath expression of an element within a list whose value is unique. For a more general search, use the *xf_tree_find_string_value_element* function instead.

The search operation is based on the exact lexicographical match of both the element path name and the value.

- Assumptions :

  - It is assumed that ∗*found_path* has no memory reserved and points to NULL

- Errors :

  - XF_CFI_WRONG_FILE_DESCRIPTOR
  - XF_CFI_DOC_NOT_PARSED_OR_CREATED
  - XF_CFI_ROOT_NODE_IS_NULL
  - XF_CFI_CURRENT_NODE_IS_NULL
  - XF_CFI_NOT_AN_ELEMENT_NODE
  - XF_CFI_BAD_INPUT_ARGUMENT
  - XF_CFI_NO_ELEMENT_REQUESTED
  - XF_CFI_PATH_TOO_LONG
  - XF_CFI_NODE_NAME_TOO_LONG
  - XF_CFI_NOT_A_TERMINAL_ELEMENT
  - XF_CFI_NO_ELEMENT_FOUND

- Warnings :

  - XF_CFI_ROOT_ALREADY_REACHED
  - XF_CFI_MAX_LENGTH_EXCEEDED
  - XF_CFI_TOO_MANY_ELEMENTS

- Reference : See the *libxml* documentation
- Memory allocation Note: Memory is dynamically allocated internally for ∗*found_path*. The user must free this memory after using it.

3.1.4.19 void xf_tree_get_current_element_name (long ∗ *fd*, char ∗∗ *name*, long ∗ *error*)

Gets the name of the current element.

Parameters:

   fd (IN): File descriptor

   name (OUT): Name of the current element

   error (OUT): Error code

This function gets the name of the current element node.

- Assumptions :

  - It is assumed that ∗*name* has no memory reserved and points to NULL

- Errors :

  - XF_CFI_WRONG_FILE_DESCRIPTOR

– XF_CFI_DOC_NOT_PARSED_OR_CREATED
– XF_CFI_ROOT_NODE_IS_NULL
– XF_CFI_BAD_INPUT_ARGUMENT
– XF_CFI_NOT_AN_ELEMENT_NODE

- Warnings :

    – XF_CFI_MAX_LENGTH_EXCEEDED

- Reference : See the *libxml* documentation
- Memory allocation Note: Memory is dynamically allocated internally for *∗name*. The user must free this memory after using it.

3.1.4.20   void xf_tree_get_fixed_header_item (long ∗ *fd*, char ∗ *item_id*, char ∗∗ *item_value*, long ∗ *error*)

Gets the value of any element of a Header.

Parameters:

   fd (IN): File descriptor

   item_id (IN): Name of the element which value we want to retrieve

   item_value (OUT): Value of the requested element

   error (OUT): Error code

This function gets the value of a Fixed Header node.

- Assumptions :

    – XF_EARTH_EXPLORER_HEADER macro is defined
    – It is assumed that *∗item_value* has no memory reserved and points to NULL

- Errors :

    – XF_CFI_WRONG_FILE_DESCRIPTOR
    – XF_CFI_DOC_NOT_PARSED_OR_CREATED
    – XF_CFI_WRONG_HEADER_ELEMENT
    – XF_CFI_NO_ELEMENT_FOUND
    – XF_CFI_INVALID_FILE_FORMAT
    – XF_CFI_EMPTY_ELEMENT

- Warnings : N/A
- Reference : See the *libxml* documentation
- Memory allocation Note: Memory is dynamically allocated internally for *∗item_value*. The user must free this memory after using it.

3.1.4.21   void xf_tree_get_fixed_header_items (long ∗ *fd*, char ∗∗ *file_name*, char ∗∗ *file_description*, char ∗∗ *notes*, char ∗∗ *mission*, char ∗∗ *file_class*, char ∗∗ *file_type*, char ∗∗ *validity_start*, char ∗∗ *validity_stop*, long ∗ *file_version*, char ∗∗ *system*, char ∗∗ *creator*, char ∗∗ *creator_version*, char ∗∗ *creation_date*, long ∗ *error*)

Gets the value of all Fixed Header elements.

Parameters:

   fd (IN): File descriptor

file_name (OUT): Value of the element file_name

file_description (OUT): Value of the element file_description

notes (OUT): Value of the element notes

mission (OUT): Value of the element mission

file_class (OUT): Value of the element file_class

file_type (OUT): Value of the element file_type

validity_start (OUT): Value of the element validity_start

validity_stop (OUT): Value of the element validity_stop

file_version (OUT): Value of the element file_version

system (OUT): Value of the element system

creator (OUT): Value of the element creator

creator_version (OUT): Value of the element creator_version

creation_date (OUT): Value of the element creation_date

error (OUT): Error code

This function sets the value of a Fixed Header node.

- Assumptions :

  - XF_EARTH_EXPLORER_HEADER macro is defined
  - All (char∗) arguments have no memory reserved and point to NULL

- Errors :

  - XF_CFI_WRONG_FILE_DESCRIPTOR
  - XF_CFI_DOC_NOT_PARSED_OR_CREATED
  - XF_CFI_WRONG_HEADER_ELEMENT
  - XF_CFI_NO_ELEMENT_FOUND
  - XF_CFI_INVALID_FORMAT

- Warnings : N/A
- Reference : See the *libxml* documentation
- Memory allocation Note: Memory is dynamically allocated internally for each char∗ argument. The user must free this memory after using it.

3.1.4.22 void xf_tree_get_namespace (long ∗ *fd*, char ∗ *node_name*, long ∗ *num_ns*, char ∗ *prefix*[XF_-MAX_ARRAY_SIZE], char ∗ *url*[XF_MAX_ARRAY_SIZE], long ∗ *error*)

Reads all the namespace that applies to a given node.

Parameters:

  fd (IN): File descriptor

  path (IN): Path/node name to the element. It can be the whole path name of the node or just the following node with the given name. If an empty string is provided, then the namespace is searched in the root element

  num_ns (OUT): number of namespaces found

  prefix (OUT): Array of output prefix.

  url (OUT): array with the URL

  error (OUT): Error code

Reads all the namespace that applies to a given node

- Assumptions :

    – There are no more than 50 namespaces in that node

- Errors :

    – XF_CFI_VALUE_NOT_FOUND
    – XF_CFI_ARRAY_EXCEEDED

- Warnings :

    –

- Reference : See the *libxml* documentation

3.1.4.23 void xf_tree_get_path (long ∗ *fd*, char ∗ *path*, long ∗ *error*)

Gets the XPath of the current tree pointer.

Parameters:

   fd (IN): File descriptor

   path (OUT): Path of the current element node.

   error (OUT): Error code

Gets the Xpath of the current tree pointer of the file referred by *fd*.

- Assumptions : N/A
- Errors :

    – XF_CFI_WRONG_FILE_DESCRIPTOR
    – XF_CFI_DOC_NOT_PARSED_OR_CREATED
    – XF_CFI_ROOT_NODE_IS_NULL
    – XF_CFI_CURRENT_NODE_IS_NULL
    – XF_CFI_NOT_AN_ELEMENT_NODE
    – XF_CFI_PATH_TOO_LONG

- Warnings : N/A
- Reference : See the *libxml* documentation

3.1.4.24 void xf_tree_go_to_element_node (long ∗ *fd*, char ∗ *element*, long ∗ *error*)

Goes to the next element in the tree.

Parameters:

   fd (IN): File descriptor

   element (IN): Element name. If it is NULL, means it is on the current element name

   error (OUT): Error code

Sets the current tree pointer of the file referred by *fd* to the *element* node.

- Assumptions : N/A
- Errors :

- – XF_CFI_WRONG_FILE_DESCRIPTOR
- – XF_CFI_DOC_NOT_PARSED_OR_CREATED
- – XF_CFI_ROOT_NODE_IS_NULL
- – XF_CFI_NO_ELEMENT_REQUESTED
- – XF_CFI_NO_ELEMENT_FOUND

- • Warnings : N/A
- • Reference : See the *libxml* documentation

3.1.4.25 void xf_tree_go_to_next_element_node (long ∗ *fd*, long ∗ *error*)

Goes to the next element in the tree.

Parameters:

   fd (IN): File descriptor

   error (OUT): Error code

Sets the current tree pointer of the file referred by *fd* to the next XML element node in the tree, if not NULL.

- • Assumptions : N/A
- • Errors :

   - – XF_CFI_WRONG_FILE_DESCRIPTOR
   - – XF_CFI_DOC_NOT_PARSED_OR_CREATED
   - – XF_CFI_ROOT_NODE_IS_NULL
   - – XF_CFI_NO_ELEMENT_FOUND

- • Warnings : N/A
- • Reference : See the *libxml* documentation

3.1.4.26 void xf_tree_go_to_path_node (long ∗ *fd*, char ∗ *path*, long ∗ *error*)

Goes to the node specified in the input XPath.

Parameters:

   fd (IN): File descriptor

   path (IN): Path to the element.

   error (OUT): Error code

Sets the current tree pointer of the file referred by *fd* to the node specified in *path*.

- • Assumptions :

   - – If the last node of the path is an attribute, it is not taken into account, so that the tree pointer is set to the previous element node.

- • Errors :

   - – XF_CFI_WRONG_FILE_DESCRIPTOR
   - – XF_CFI_DOC_NOT_PARSED_OR_CREATED
   - – XF_CFI_ROOT_NODE_IS_NULL
   - – XF_CFI_BAD_INPUT_ARGUMENT
   - – XF_CFI_PATH_TOO_LONG

– XF CFI NODE NAME TOO LONG
– XF CFI NO ELEMENT FOUND

- Warnings :

  – XF CFI ROOT ALREADY REACHED

- Reference : See the *libxml* documentation

3.1.4.27 long xf tree init parser (char ∗ *file*, long ∗ *error*)

Loads an XML file into memory.

Parameters:
   file (IN): File name

   error (OUT): Error code

Returns:
   A file descriptor of long type is returned

This function reads an XML into memory and builds a tree representation of the file data ready for being accessed by the functions in the EO FILE HANDLING library. It returns a file descriptor to be used for referencing the file in the EO file handling routines. If the maximum number of open files has already been reached, an error is returned.

- Assumptions : N/A
- Errors :

  – XF CFI MAX FILES REACHED
  – XF CFI BAD INPUT ARGUMENT
  – XF CFI MEMORY ERROR
  – XF CFI PARSING FILE ERROR
  – XF CFI GETTING ROOT ERROR

- Warnings : N/A
- Reference : See the *libxml* documentation

3.1.4.28 void xf tree path read integer node array (long ∗ *fd*, char ∗ *path*, long ∗ *array*, long ∗ *length*, long ∗ *error*)

Reads an array of long values given the Xpath to the node.

Parameters:
   fd (IN): File descriptor

   path (IN): Path to the element. It must reference a list of elements with the characters '[∗]', which must appear once and only once in the path

   array (OUT): Array of values

   length (OUT): Actual array size

   error (OUT): Error code

This function reads the values of all elements in the file referred by the *fd* whose Xpath matches *path* and returns them as an array of longs. Please noticed that in case there are several conversion errors in the array elements, only the last one is published.

- Assumptions :

  - It is assumed that the size of the array variable which shall contain the elements is equal or greater than *XF_MAX_ARRAY_SIZE*

- Errors :

  - XF_CFI_WRONG_FILE_DESCRIPTOR
  - XF_CFI_DOC_NOT_PARSED_OR_CREATED
  - XF_CFI_ROOT_NODE_IS_NULL
  - XF_CFI_BAD_INPUT_ARGUMENT
  - XF_CFI_NO_ELEMENT_REQUESTED
  - XF_CFI_PATH_TOO_LONG
  - XF_CFI_NODE_NAME_TOO_LONG
  - XF_CFI_NOT_A_TERMINAL_ELEMENT
  - XF_CFI_NO_ELEMENT_FOUND
  - XF_CFI_NOT_A_LONG
  - XF_CFI_VALUE_OUT_OF_RANGE

- Warnings :

  - XF_CFI_ROOT_ALREADY_REACHED
  - XF_CFI_MAX_LENGTH_EXCEEDED
  - XF_CFI_TOO_MANY_ELEMENTS

- Reference : See the *libxml* documentation

3.1.4.29 void xf_tree_path_read_integer_node_value (long ∗ *fd*, char ∗ *path*, long ∗ *value*, long ∗ *error*)

Reads a integer value given the path to the node.

Parameters:

    fd (IN): File descriptor

    path (IN): Path to the element.

    value (OUT): Element value.

    error (OUT): Error code

This function gets the value of a node (element or attribute) specified with an XPath, as long, from the file referred by the *fd*.

- Assumptions : N/A
- Errors :

  - XF_CFI_WRONG_FILE_DESCRIPTOR
  - XF_CFI_DOC_NOT_PARSED_OR_CREATED
  - XF_CFI_ROOT_NODE_IS_NULL
  - XF_CFI_CURRENT_NODE_IS_NULL
  - XF_CFI_BAD_INPUT_ARGUMENT
  - XF_CFI_NO_ELEMENT_REQUESTED
  - XF_CFI_PATH_TOO_LONG
  - XF_CFI_NODE_NAME_TOO_LONG
  - XF_CFI_NOT_A_TERMINAL_ELEMENT
  - XF_CFI_NO_ELEMENT_FOUND
  - XF_CFI_NOT_A_LONG
  - XF_CFI_VALUE_OUT_OF_RANGE

- Warnings :

  - XF_CFI_ROOT_ALREADY_REACHED
  - XF_CFI_MAX_LENGTH_EXCEEDED

- Reference : See the *libxml* documentation

3.1.4.30 void xf_tree_path_read_real_node_array (long ∗ *fd*, char ∗ *path*, double ∗ *array*, long ∗ *length*, long ∗ *error*)

Reads an array of double values given the Xpath to the node.

Parameters:

    fd (IN): File descriptor

    path (IN): Path to the element. It must reference a list of elements with the characters '[∗]', which must appear once and only once in the path

    array (OUT): Array of values

    length (OUT): Actual array size

    error (OUT): Error code

This function reads the values of all elements in the file referred by the *fd* whose Xpath matches *path* and returns them as an array of doubles. Please noticed that in case there are several conversion errors in the array elements, only the last one is published.

- Assumptions :

    – It is assumed that the size of the array variable which shall contain the elements is equal or greater than *XF_MAX_ARRAY_SIZE*

- Errors :

    – XF_CFI_WRONG_FILE_DESCRIPTOR
    – XF_CFI_DOC_NOT_PARSED_OR_CREATED
    – XF_CFI_ROOT_NODE_IS_NULL
    – XF_CFI_BAD_INPUT_ARGUMENT
    – XF_CFI_NO_ELEMENT_REQUESTED
    – XF_CFI_PATH_TOO_LONG
    – XF_CFI_NODE_NAME_TOO_LONG
    – XF_CFI_NOT_A_TERMINAL_ELEMENT
    – XF_CFI_NO_ELEMENT_FOUND
    – XF_CFI_NOT_A_DOUBLE
    – XF_CFI_VALUE_OUT_OF_RANGE

- Warnings :

    – XF_CFI_ROOT_ALREADY_REACHED
    – XF_CFI_MAX_LENGTH_EXCEEDED
    – XF_CFI_TOO_MANY_ELEMENTS

- Reference : See the *libxml* documentation

3.1.4.31 void xf_tree_path_read_real_node_value (long ∗ *fd*, char ∗ *path*, double ∗ *value*, long ∗ *error*)

Reads a double value given the path to the node.

Parameters:

    fd (IN): File descriptor

    path (IN): Path to the element.

    value (OUT): Element value.

    error (OUT): Error code

This function gets the value of a node (element or attribute) specified with an XPath, as double, from the file referred by the *fd*.

- Assumptions : N/A
- Errors :

  - XF CFI WRONG FILE DESCRIPTOR
  - XF CFI DOC NOT PARSED OR CREATED
  - XF CFI ROOT NODE IS NULL
  - XF CFI CURRENT NODE IS NULL
  - XF CFI BAD INPUT ARGUMENT
  - XF CFI NO ELEMENT REQUESTED
  - XF CFI PATH TOO LONG
  - XF CFI NODE NAME TOO LONG
  - XF CFI NOT A TERMINAL ELEMENT
  - XF CFI NO ELEMENT FOUND
  - XF CFI NOT A DOUBLE
  - XF CFI VALUE OUT OF RANGE

- Warnings :

  - XF CFI ROOT ALREADY REACHED
  - XF CFI MAX LENGTH EXCEEDED

- Reference : See the *libxml* documentation

3.1.4.32 void xf tree path read string node array (long ∗ *fd*, char ∗ *path*, char ∗∗∗ *array*, long ∗ *length*, long ∗ *error*)

Reads an array of string characters given the Xpath to the node.

Parameters:

  fd (IN): File descriptor

  path (IN): Path to the element. It must reference a list of elements with the characters '[∗]', which must appear once and only once in the path

  array (OUT): Array of values

  length (OUT): Actual array size

  error (OUT): Error code

This function reads the values of all elements in the file referred by the *fd* whose Xpath matches *path* and returns them as an array of strings (character arrays). Please noticed that in case there are several conversion errors in the array elements, only the last one is published.

- Assumptions :

  - It is assumed that the size of the array variable which shall contain the elements is equal or greater than *XF MAX ARRAY SIZE*
  - It is assumed that ∗*array* is an array of (char∗) already reserved by the user: i.e: array = calloc(XF MAX ARRAY SIZE, sizeof(char ∗));

- Errors :

  - XF CFI WRONG FILE DESCRIPTOR
  - XF CFI DOC NOT PARSED OR CREATED
  - XF CFI ROOT NODE IS NULL

- – XF_CFI_BAD_INPUT_ARGUMENT
- – XF_CFI_NO_ELEMENT_REQUESTED
- – XF_CFI_PATH_TOO_LONG
- – XF_CFI_NODE_NAME_TOO_LONG
- – XF_CFI_NOT_A_TERMINAL_ELEMENT
- – XF_CFI_NO_ELEMENT_FOUND

- Warnings :

  - – XF_CFI_ROOT_ALREADY_REACHED
  - – XF_CFI_MAX_LENGTH_EXCEEDED
  - – XF_CFI_TOO_MANY_ELEMENTS

- Reference : See the *libxml* documentation
- Memory allocation Note: Memory is allocated internally for each *∗array* element . The user must free this memory after using it.

3.1.4.33   void xf_tree_path_read_string_node_value (long ∗ *fd*, char ∗ *path*, char ∗∗ *value*, long ∗ *error*)

Reads a string value given the path to the node.

Parameters:

    fd (IN): File descriptor

    path (IN): Path to the element.

    value (OUT): Element value.

    error (OUT): Error code

This function gets the value of a node (element or attribute) specified with an XPath, as string, from the file referred by the *fd*.

- Assumptions :

  - – It is assumed that *∗value* has no memory reserved and points to NULL

- Errors :

  - – XF_CFI_WRONG_FILE_DESCRIPTOR
  - – XF_CFI_DOC_NOT_PARSED_OR_CREATED
  - – XF_CFI_ROOT_NODE_IS_NULL
  - – XF_CFI_CURRENT_NODE_IS_NULL
  - – XF_CFI_BAD_INPUT_ARGUMENT
  - – XF_CFI_NO_ELEMENT_REQUESTED
  - – XF_CFI_PATH_TOO_LONG
  - – XF_CFI_NODE_NAME_TOO_LONG
  - – XF_CFI_NOT_A_TERMINAL_ELEMENT
  - – XF_CFI_NO_ELEMENT_FOUND

- Warnings :

  - – XF_CFI_ROOT_ALREADY_REACHED
  - – XF_CFI_MAX_LENGTH_EXCEEDED

- Reference : See the *libxml* documentation
- Memory allocation Note: Memory is dynamically allocated internally for *∗value*. The user must free this memory after using it.

3.1.4.34   void xf_tree_read_integer_attribute (long ∗ fd, char ∗ element, char ∗ attribute_name, long ∗ attribute_value, long ∗ error)

Reads an attribute as a long.

Parameters:

    fd (IN): File descriptor

    element (IN): Element name. If it is NULL, means it is on the current element name

    attribute_name (IN): Attribute name

    attribute_value (OUT): Attribute value

    error (OUT): Error code

This function reads the value of the attribute with name *attribute_name*, from the element specified in *element* (if NULL, it is assumed it has already been set) in the file referred by the *fd*, and returns it as a characters long. If the node name is not equal to the one set for searching, the tree pointer is moved forward to the next element requested.

- Assumptions : N/A
- Errors :

    - XF_CFI_WRONG_FILE_DESCRIPTOR
    - XF_CFI_DOC_NOT_PARSED_OR_CREATED
    - XF_CFI_ROOT_NODE_IS_NULL
    - XF_CFI_CURRENT_NODE_IS_NULL
    - XF_CFI_BAD_INPUT_ARGUMENT
    - XF_CFI_NO_ELEMENT_REQUESTED
    - XF_CFI_NO_ELEMENT_FOUND
    - XF_CFI_NO_ATTRIBUTES
    - XF_CFI_NOT_A_LONG
    - XF_CFI_VALUE_OUT_OF_RANGE

- Warnings :

    - XF_CFI_MAX_LENGTH_EXCEEDED

- Reference : See the *libxml* documentation

3.1.4.35   void xf_tree_read_integer_element_array (long ∗ fd, char ∗ element, long ∗ array, long ∗ length, long ∗ error)

Reads an array of integer number values.

Parameters:

    fd (IN): File descriptor

    element (IN): Element name

    array (OUT): Array of values

    length (OUT): Actual array size

    error (OUT): Error code

This function reads the values of all elements in the file referred by the *fd* whose name match *element* and returns them as an array of integers. Please noticed that in case there are several conversion errors in the array elements, only the last one is published.

- Assumptions :

    – It is assumed that the size of the array variable which shall contain the elements is equal or greater than *XF_MAX_ARRAY_SIZE*

- Errors :

    – XF_CFI_WRONG_FILE_DESCRIPTOR
    – XF_CFI_DOC_NOT_PARSED_OR_CREATED
    – XF_CFI_ROOT_NODE_IS_NULL
    – XF_CFI_BAD_INPUT_ARGUMENT
    – XF_CFI_NO_ELEMENT_REQUESTED
    – XF_CFI_ROOT_NODE_IS_NULL
    – XF_CFI_NOT_AN_ELEMENT_NODE
    – XF_CFI_NOT_A_TERMINAL_ELEMENT
    – XF_CFI_NO_ELEMENT_FOUND
    – XF_CFI_NOT_A_LONG
    – XF_CFI_VALUE_OUT_OF_RANGE

- Warnings :

    – XF_CFI_MAX_LENGTH_EXCEEDED
    – XF_CFI_TOO_MANY_ELEMENTS

- Reference : See the *libxml* documentation

3.1.4.36 void xf_tree_read_integer_element_value (long ∗ *fd*, char ∗ *element*, long ∗ *value*, long ∗ *error*)

Reads an integer number value.

Parameters:

   fd (IN): File descriptor

   element (IN): Element name. A NULL value means an iteration.

   value (OUT): Element value

   error (OUT): Error code

This function reads the value of the next element in the file referred by the *fd* whose name matches *element* and returns it as an integer.

- Assumptions : N/A
- Errors :

    – XF_CFI_WRONG_FILE_DESCRIPTOR
    – XF_CFI_DOC_NOT_PARSED_OR_CREATED
    – XF_CFI_ROOT_NODE_IS_NULL
    – XF_CFI_BAD_INPUT_ARGUMENT
    – XF_CFI_NO_ELEMENT_REQUESTED
    – XF_CFI_NO_ELEMENT_FOUND
    – XF_CFI_NOT_AN_ELEMENT_NODE
    – XF_CFI_NOT_A_TERMINAL_ELEMENT
    – XF_CFI_NOT_A_LONG
    – XF_CFI_VALUE_OUT_OF_RANGE

- Warnings :

    – XF_CFI_MAX_LENGTH_EXCEEDED

- Reference : See the *libxml* documentation

3.1.4.37 void xf_tree_read_real_attribute (long ∗ *fd*, char ∗ *element*, char ∗ *attribute_name*, double ∗ *attribute_value*, long ∗ *error*)

Reads an attribute as a double.

Parameters:

fd (IN): File descriptor

element (IN): Element name. If it is NULL, means it is on the current element name

attribute_name (IN): Attribute name

attribute_value (OUT): Attribute value

error (OUT): Error code

This function reads the value of the attribute with name *attribute_name*, from the element specified in *element* (if NULL, it is assumed it has already been set) in the file referred by the *fd*, and returns it as a characters double. If the node name is not equal to the one set for searching, the tree pointer is moved forward to the next element requested.

- Assumptions : N/A
- Errors :
  - XF_CFI_WRONG_FILE_DESCRIPTOR
  - XF_CFI_DOC_NOT_PARSED_OR_CREATED
  - XF_CFI_ROOT_NODE_IS_NULL
  - XF_CFI_CURRENT_NODE_IS_NULL
  - XF_CFI_BAD_INPUT_ARGUMENT
  - XF_CFI_NO_ELEMENT_REQUESTED
  - XF_CFI_NO_ELEMENT_FOUND
  - XF_CFI_NO_ATTRIBUTES
  - XF_CFI_NOT_A_DOUBLE
  - XF_CFI_VALUE_OUT_OF_RANGE
- Warnings :
  - XF_CFI_MAX_LENGTH_EXCEEDED
- Reference : See the *libxml* documentation

3.1.4.38 void xf_tree_read_real_element_array (long ∗ *fd*, char ∗ *element*, double ∗ *array*, long ∗ *length*, long ∗ *error*)

Reads an array of real number values.

Parameters:

fd (IN): File descriptor

element (IN): Element name

array (OUT): Array of values

length (OUT): Actual array size

error (OUT): Error code

This function reads the values of all elements in the file referred by the *fd* whose name match *element* and returns them as an array of doubles. Please noticed that in case there are several conversion errors in the array elements, only the last one is published.

- Assumptions :

  – It is assumed that the size of the array variable which shall contain the elements is equal or greater than *XF_MAX_ARRAY_SIZE*

- Errors :

  – XF_CFI_WRONG_FILE_DESCRIPTOR
  – XF_CFI_DOC_NOT_PARSED_OR_CREATED
  – XF_CFI_ROOT_NODE_IS_NULL
  – XF_CFI_BAD_INPUT_ARGUMENT
  – XF_CFI_NO_ELEMENT_REQUESTED
  – XF_CFI_ROOT_NODE_IS_NULL
  – XF_CFI_NOT_AN_ELEMENT_NODE
  – XF_CFI_NOT_A_TERMINAL_ELEMENT
  – XF_CFI_NO_ELEMENT_FOUND
  – XF_CFI_NOT_A_DOUBLE
  – XF_CFI_VALUE_OUT_OF_RANGE

- Warnings :

  – XF_CFI_MAX_LENGTH_EXCEEDED
  – XF_CFI_TOO_MANY_ELEMENTS

- Reference : See the *libxml* documentation

3.1.4.39   void xf_tree_read_real_element_value (long ∗ *fd*, char ∗ *element*, double ∗ *value*, long ∗ *error*)

Reads a real number value.

Parameters:

  fd (IN): File descriptor

  element (IN): Element name. A NULL value means an iteration.

  value (OUT): Element value

  error (OUT): Error code

This function reads the value of the next element in the file referred by the *fd* whose name matches *element* and returns it as a double.

- Assumptions : N/A
- Errors :

  – XF_CFI_WRONG_FILE_DESCRIPTOR
  – XF_CFI_DOC_NOT_PARSED_OR_CREATED
  – XF_CFI_ROOT_NODE_IS_NULL
  – XF_CFI_BAD_INPUT_ARGUMENT
  – XF_CFI_NO_ELEMENT_REQUESTED
  – XF_CFI_NO_ELEMENT_FOUND
  – XF_CFI_NOT_AN_ELEMENT_NODE
  – XF_CFI_NOT_A_TERMINAL_ELEMENT
  – XF_CFI_NOT_A_DOUBLE
  – XF_CFI_VALUE_OUT_OF_RANGE

- Warnings :

  – XF_CFI_MAX_LENGTH_EXCEEDED

- Reference : See the *libxml* documentation

3.1.4.40 void xf_tree_read_string_attribute (long ∗ *fd*, char ∗ *element*, char ∗ *attribute_name*, char ∗∗ *attribute_value*, long ∗ *error*)

Reads an attribute as string.

Parameters:

> *fd* (IN): File descriptor
>
> *element* (IN): Element name. A value of NULL means that the operation is performed on the current element name
>
> *attribute_name* (IN): Attribute name
>
> *attribute_value* (OUT): Attribute value
>
> *error* (OUT): Error code

This function reads the value of the attribute with name *attribute_name*, from the element specified in *element* (if NULL, it is assumed it has already been set) in the file referred by the *fd*, and returns it as a characters string. If the node name is not equal to the one set for searching, the tree pointer is moved forward to the next element requested.

- Assumptions :

  - It is assumed that ∗*attribute_value* has no memory reserved and points to NULL

- Errors :

  - XF_CFI_WRONG_FILE_DESCRIPTOR
  - XF_CFI_DOC_NOT_PARSED_OR_CREATED
  - XF_CFI_ROOT_NODE_IS_NULL
  - XF_CFI_CURRENT_NODE_IS_NULL
  - XF_CFI_BAD_INPUT_ARGUMENT
  - XF_CFI_NO_ELEMENT_REQUESTED
  - XF_CFI_NO_ELEMENT_FOUND
  - XF_CFI_NO_ATTRIBUTES

- Warnings :

  - XF_CFI_MAX_LENGTH_EXCEEDED

- Reference : See the *libxml* documentation
- Memory allocation Note: Memory is dynamically allocated internally for ∗*attribute_value*. The user must free this memory after using it.

3.1.4.41 void xf_tree_read_string_element_array (long ∗ *fd*, char ∗ *element*, char ∗∗∗ *array*, long ∗ *length*, long ∗ *error*)

Reads an array of string values.

Parameters:

> *fd* (IN): File descriptor
>
> *element* (IN): Element name
>
> *array* (OUT): Array of values
>
> *length* (OUT): Actual array size
>
> *error* (OUT): Error code

This function reads the values of all elements in the file referred by the *fd* whose name match *element* and returns them as an array of characters strings.

- Assumptions :

  – It is assumed that the size of the array variable which shall contain the elements is equal or greater than *XF_MAX_ARRAY_SIZE*
  – It is assumed that *∗array* is an array of (char∗) already reserved by the user: i.e: array = calloc(XF_MAX_ARRAY_SIZE, sizeof(char ∗));

- Errors :

  – XF_CFI_WRONG_FILE_DESCRIPTOR
  – XF_CFI_DOC_NOT_PARSED_OR_CREATED
  – XF_CFI_ROOT_NODE_IS_NULL
  – XF_CFI_BAD_INPUT_ARGUMENT
  – XF_CFI_NO_ELEMENT_REQUESTED
  – XF_CFI_ROOT_NODE_IS_NULL
  – XF_CFI_NOT_AN_ELEMENT_NODE
  – XF_CFI_NOT_A_TERMINAL_ELEMENT
  – XF_CFI_NO_ELEMENT_FOUND

- Warnings :

  – XF_CFI_MAX_LENGTH_EXCEEDED
  – XF_CFI_TOO_MANY_ELEMENTS

- Reference : See the *libxml* documentation
- Memory allocation Note: Memory is allocated internally for each *∗array* element . The user must free this memory after using it.

3.1.4.42 void xf_tree_read_string_element_value (long ∗ *fd*, char ∗ *element*, char ∗∗ *value*, long ∗ *error*)

Reads a string value.

Parameters:

 fd (IN): File descriptor

 element (IN): Element name. If it is NULL, means it is an iteration.

 value (OUT): Element value

 error (OUT): Error code

This function reads the value of the next element in the file referred by the *fd* whose name matches *element* and returns it as a characters string.

- Assumptions :

  – It is assumed that *∗value* has no memory reserved and points to NULL

- Errors :

  – XF_CFI_WRONG_FILE_DESCRIPTOR
  – XF_CFI_DOC_NOT_PARSED_OR_CREATED
  – XF_CFI_ROOT_NODE_IS_NULL
  – XF_CFI_BAD_INPUT_ARGUMENT
  – XF_CFI_NO_ELEMENT_REQUESTED
  – XF_CFI_NO_ELEMENT_FOUND

- XF CFI NOT AN ELEMENT NODE
- XF CFI NOT A TERMINAL ELEMENT

- Warnings :

    - XF CFI MAX LENGTH EXCEEDED

- Reference : See the *libxml* documentation
- Memory allocation Note: Memory is dynamically allocated internally for ∗*value*. The user must free this memory after using it.

3.1.4.43   void xf tree remove node (long ∗ *id*, char ∗ *name*, long ∗ *error*)

Remove a node.

Parameters:
    id   (IN): Descriptor of the XML document memory representation

    name   (IN): XPath node name. It may be either an element or an attribute

    error   (OUT): Error code

This function unlinks the *name* node and all its descendants from the memory representation of the XML document.

- Assumptions : N/A
- Errors :

    - XF CFI BAD INPUT ARGUMENT
    - XF CFI WRONG FILE DESCRIPTOR
    - XF CFI DOC NOT PARSED OR CREATED
    - XF CFI ROOT NODE IS NULL
    - XF CFI PATH TOO LONG
    - XF CFI NODE NAME TOO LONG
    - XF CFI NO ELEMENT FOUND
    - XF CFI REMOVING NODE ERROR

- Warnings :

    - XF CFI ROOT ALREADY REACHED

- Reference : See the *libxml* documentation

3.1.4.44   void xf tree rewind (long ∗ *fd*, long ∗ *error*)

Sets the read pointer to the beginning of the file.

Parameters:
    fd   (IN): File descriptor

    error   (OUT): Error code

Sets the parser pointer to the root element of the document being parsed, which is referred by the *fd*.

- Assumptions :

    - If the root node is NULL this function does not care

- Errors :

  – XF_CFI_WRONG_FILE_DESCRIPTOR
  – XF_CFI_DOC_NOT_PARSED_OR_CREATED
  – XF_CFI_ROOT_NODE_IS_NULL

- Warnings : N/A
- Reference : See the *libxml* documentation

3.1.4.45   void xf_tree_set_fixed_header_item (long ∗ *fd*, char ∗ *item_id*, char ∗ *item_value*, long ∗ *error*)

Sets the value of any element of a Header.

Parameters:

    fd (IN): File descriptor

    item_id (IN): Name of the element which value we want to modify

    item_value (IN): New value assigned for the node

    error (OUT): Error code

This function sets the value of a Fixed Header node.

- Assumptions :

  – XF_EARTH_EXPLORER_HEADER macro is defined

- Errors :

  – XF_CFI_WRONG_FILE_DESCRIPTOR
  – XF_CFI_DOC_NOT_PARSED_OR_CREATED
  – XF_CFI_WRONG_HEADER_ELEMENT
  – XF_CFI_NO_ELEMENT_FOUND
  – XF_CFI_INVALID_FORMAT

- Warnings : N/A
- Reference : See the *libxml* documentation

3.1.4.46   void xf_tree_set_fixed_header_items (long ∗ *fd*, char ∗ *file_name*, char ∗ *file_description*, char ∗ *notes*, char ∗ *mission*, char ∗ *file_class*, char ∗ *file_type*, char ∗ *validity_start*, char ∗ *validity_stop*, long ∗ *file_version*, char ∗ *system*, char ∗ *creator*, char ∗ *creator_version*, char ∗ *creation_date*, long ∗ *error*)

Sets the value of all Fixed Header elements.

Parameters:

    fd (IN): File descriptor

    file_name (IN): Value of the element file_name

    file_description (IN): Value of the element file_description

    notes (IN): Value of the element notes

    mission (IN): Value of the element mission

    file_class (IN): Value of the element file_class

    file_type (IN): Value of the element file_type

validity_start (IN): Value of the element validity_start

validity_stop (IN): Value of the element validity_stop

file_version (IN): Value of the element file_version

system (IN): Value of the element system

creator (IN): Value of the element creator

creator_version (IN): Value of the element creator_version

creation_date (IN): Value of the element creation_date

error (OUT): Error code

This function sets the value of a Fixed Header node.

- Assumptions :
  - XF_EARTH_EXPLORER_HEADER macro is defined
- Errors :
  - XF_CFI_WRONG_FILE_DESCRIPTOR
  - XF_CFI_DOC_NOT_PARSED_OR_CREATED
  - XF_CFI_INVALID_FORMAT
- Warnings : N/A
- Reference : See the *libxml* documentation

3.1.4.47 void xf_tree_set_integer_node_value (long ∗ *id*, char ∗ *name*, long ∗ *value*, char ∗ *format*, long ∗ *error*)

Set an integer node value.

Parameters:

id (IN): Descriptor of the XML document memory representation

name (IN): XPath node name. It may be either an element or an attribute

value (IN): Integer value

format (IN): Integer representation format

error (OUT): Error code

Replace the value of the *name* node by the integer *value*. *format* may be any valid format specification accepted by the printf family of C functions, i.e. "%+010d"

- Assumptions : N/A
- Errors :
  - XF_CFI_BAD_INPUT_ARGUMENT
  - XF_CFI_WRONG_FILE_DESCRIPTOR
  - XF_CFI_DOC_NOT_PARSED_OR_CREATED
  - XF_CFI_ROOT_NODE_IS_NULL
  - XF_CFI_PATH_TOO_LONG
  - XF_CFI_NODE_NAME_TOO_LONG
  - XF_CFI_NO_ELEMENT_FOUND
  - XF_CFI_INVALID_FORMAT
- Warnings :
  - XF_CFI_ROOT_ALREADY_REACHED
  - XF_CFI_MAX_LENGTH_EXCEEDED
- Reference : See the *libxml* documentation

3.1.4.48 void xf_tree_set_real_node_value (long ∗ id, char ∗ name, double ∗ value, char ∗ format, long ∗ error)

Set a real number node value.

Parameters:

    id (IN): Descriptor of the XML document memory representation

    name (IN): XPath node name. It may be either an element or an attribute

    value (IN): Real number value

    format (IN): Real number representation format

    error (OUT): Error code

Replace the value of the *name* node by the double *value*. *format* may be any valid format specification accepted by the printf family of C functions, i.e. "%+012.6lf"

- Assumptions : N/A
- Errors :

    - XF_CFI_BAD_INPUT_ARGUMENT
    - XF_CFI_WRONG_FILE_DESCRIPTOR
    - XF_CFI_DOC_NOT_PARSED_OR_CREATED
    - XF_CFI_ROOT_NODE_IS_NULL
    - XF_CFI_PATH_TOO_LONG
    - XF_CFI_NODE_NAME_TOO_LONG
    - XF_CFI_NO_ELEMENT_FOUND
    - XF_CFI_INVALID_FORMAT

- Warnings :

    - XF_CFI_ROOT_ALREADY_REACHED
    - XF_CFI_MAX_LENGTH_EXCEEDED

- Reference : See the *libxml* documentation

3.1.4.49 void xf_tree_set_string_node_value (long ∗ id, char ∗ name, char ∗ value, char ∗ format, long ∗ error)

Set a string node value.

Parameters:

    id (IN): Descriptor of the XML document memory representation

    name (IN): XPath node name. It may be either an element or an attribute

    value (IN): String value

    format (IN): String representation format

    error (OUT): Error code

Replace the value of the *name* node by the string *value*. *format* may be any valid format specification accepted by the printf family of C functions, i.e. "%10s"

- Assumptions : N/A
- Errors :

    - XF_CFI_BAD_INPUT_ARGUMENT

- – XF_CFI_WRONG_FILE_DESCRIPTOR
- – XF_CFI_DOC_NOT_PARSED_OR_CREATED
- – XF_CFI_ROOT_NODE_IS_NULL
- – XF_CFI_PATH_TOO_LONG
- – XF_CFI_NODE_NAME_TOO_LONG
- – XF_CFI_NO_ELEMENT_FOUND
- – XF_CFI_INVALID_FORMAT

- Warnings :

  - – XF_CFI_ROOT_ALREADY_REACHED
  - – XF_CFI_MAX_LENGTH_EXCEEDED

- Reference : See the *libxml* documentation

3.1.4.50   void xf_tree_write (long ∗ *id*, char ∗ *name*, long ∗ *error*)

Write the data to a file on disk.

Parameters:

    id (IN): Descriptor of the XML document memory representation

    name (IN): File name. If it is "-", then the standard output is used

    error (OUT): Error code

Write an XML document previously opened or created to a file on disk.

The empty tags in a XML file can be written in two ways:

- short format:
- long format: The used format depends on the libxml's global variable "xmlSaveNo-EmptyTags" (defined in libxml/globals.h). This variable is set to 0 by default (so that the short format is written). To use the long format, the variable has to be set to 1.

- Assumptions : N/A
- Errors :

  - – XF_CFI_WRONG_FILE_DESCRIPTOR
  - – XF_CFI_DOC_NOT_PARSED_OR_CREATED
  - – XF_CFI_ROOT_NODE_IS_NULL
  - – XF_CFI_BAD_INPUT_ARGUMENT
  - – XF_CFI_CREATING_NODE_ERROR
  - – XF_CFI_ADDING_SIBLING_ERROR
  - – XF_CFI_SAVING_DOC_ERROR
- Warnings : N/A
- Reference : See the *libxml* documentation

3.1.4.51   void xf_verbose ()

Set verbosity flag.

This function set the verbosity flag. When the flag is set the errors raised by explorer_file_handling functions as well as the ones raised by libxml are displayed to stdedd.

Index