
Juphoon Protocol Framework

XML

Published: June 2007

For more information on Juphoon Protocol Framework, see <http://www.juphoon.com>

Juphoon XML Function Definition

Juphoon System Software Corporation.

<http://www.juphoon.com>

Tel: +86-574-87304379 +86-574-87287820

Fax: +86-574-87304379

Text Part Number: 108-101-02-01

Copyright © 2007, Juphoon System Software Corporation.

All rights reserved.

Contents

1. INTRODUCTION.....	9
1.1 PURPOSE.....	9
1.2 AUDIENCE	9
1.3 SCOPE.....	9
1.4 DEFINITIONS, ACRONYMS, AND ABBREVIATIONS	9
2. SYSTEM ENVIRONMENT.....	10
2.1 BASIC DATA TYPES	10
2.2 PLATFORM TYPES.....	10
3. USER INTERFACES	11
3.1 BASIC CONCEPTS.....	11
3.1.1 DOM.....	11
3.1.2 SAX.....	11
3.2 DOM INTERFACES.....	11
3.2.1 <i>Interface Structures</i>	12
3.2.1.1 ST_DOM_NODE.....	12
3.2.1.2 EN_DOM_NODE_TYPE.....	13
3.2.1.3 ST_DOM_STR.....	13
3.2.1.4 ST_DOM_DOC.....	14
3.2.1.5 ST_DOM_NNODE_MAP.....	14
3.2.1.6 ST_DOM_CDATA	14
3.2.1.7 ST_DOM_ATTR.....	14
3.2.1.8 ST_DOM_ELEM	15
3.2.1.9 ST_DOM_DOC.....	15
3.2.1.10 ST_DOM_DFRAG.....	15
3.2.1.11 ST_DOM_TXT	15
3.2.1.12 ST_DOM_COMMT	15
3.2.1.13 ST_DOM_CDATA_SEC	15
3.2.1.14 ST_DOM_PI.....	16
3.2.1.15 ST_ZOS_DBUF	16
3.2.2 <i>Dom_NodeGetType</i>	16
3.2.3 <i>Dom_NodeGetName</i>	17
3.2.4 <i>Dom_NodeSetName</i>	18
3.2.5 <i>Dom_NodeGetVal</i>	18
3.2.6 <i>Dom_NodeSetVal</i>	19
3.2.7 <i>Dom_NodeGetParent</i>	20
3.2.8 <i>Dom_NodeHasChilds</i>	20
3.2.9 <i>Dom_NodeGetChilds</i>	21
3.2.10 <i>Dom_NodeGetFristChild</i>	22
3.2.11 <i>Dom_NodeGetLastChild</i>	22
3.2.12 <i>Dom_NodeGetPrevSibling</i>	23

3.2.13	<i>Dom_NodeGetNextSibling</i>	24
3.2.14	<i>Dom_NodeHasAttrs</i>	24
3.2.15	<i>Dom_NodeGetOwnerDoc</i>	25
3.2.16	<i>Dom_NodeInsertBefore</i>	26
3.2.17	<i>Dom_NodeReplacChild</i>	26
3.2.18	<i>Dom_NodeRemChild</i>	27
3.2.19	<i>Dom_NodeAppendChild</i>	28
3.2.20	<i>Dom_NodeClone</i>	29
3.2.21	<i>Dom_NodeLstGetItem</i>	29
3.2.22	<i>Dom_NodeLstGetLen</i>	30
3.2.23	<i>Dom_NNodeMapGetNItem</i>	31
3.2.24	<i>Dom_NNodeMapSetNItem</i>	32
3.2.25	<i>Dom_NnodeMapRemNItem</i>	32
3.2.26	<i>Dom_NNodeMapGetItem</i>	33
3.2.27	<i>Dom_NNodeMapGetLen</i>	34
3.2.28	<i>Dom_CDataGetSubData</i>	35
3.2.29	<i>Dom_CDataAppendData</i>	35
3.2.30	<i>Dom_CDataInsertData</i>	36
3.2.31	<i>Dom_CDataDelData</i>	37
3.2.32	<i>Dom_CDataReplace</i>	38
3.2.33	<i>Dom_CDataSetData</i>	38
3.2.34	<i>Dom_CDataSplit</i>	39
3.2.35	<i>Dom_AttrGetName</i>	40
3.2.36	<i>Dom_AttrGetVal</i>	41
3.2.37	<i>Dom_AttrSetVal</i>	41
3.2.38	<i>Dom_AttrGetDQuote</i>	42
3.2.39	<i>Dom_AttrSetDQuote</i>	43
3.2.40	<i>Dom_ElemGetTagName</i>	43
3.2.41	<i>Dom_ElemGetAttr</i>	44
3.2.42	<i>Dom_ElemSetAttr</i>	45
3.2.43	<i>Dom_ElemRemAttr</i>	46
3.2.44	<i>Dom_ElemGetAttrNode</i>	46
3.2.45	<i>Dom_ElemSetAttrNode</i>	47
3.2.46	<i>Dom_ElemRemAttrNode</i>	48
3.2.47	<i>Dom_ElemGetElemsByTagName</i>	49
3.2.48	<i>Dom_ElemGetSingleElem</i>	50
3.2.49	<i>Dom_ElemHasAttr</i>	50
3.2.50	<i>Dom_DocCreateElem</i>	51
3.2.51	<i>Dom_DocCreateDocFrag</i>	52
3.2.52	<i>Dom_DocCreateTxt</i>	52
3.2.53	<i>Dom_DocCreateCommt</i>	53
3.2.54	<i>Dom_DocCreateCDataSec</i>	54
3.2.55	<i>Dom_DocCreatePi</i>	55
3.2.56	<i>Dom_DocCreateAttr</i>	56

3.2.57	<i>Dom_DocGetElemsByTagName</i>	57
3.2.58	<i>Dom_DocGetElem</i>	57
3.2.59	<i>Dom_DocLoad</i>	58
3.2.60	<i>Dom_DocLoadXml</i>	59
3.2.61	<i>Dom_DocSave</i>	59
3.2.62	<i>Dom_DocSaveXml</i>	60
3.2.63	<i>Dom_DocCreate</i>	61
3.2.64	<i>Dom_DocDelete</i>	61
3.2.65	<i>Dom_DocImportNode</i>	62
3.3	DOM UTILITY INTERFACES.....	63
3.3.1	<i>Dom_SecGetKey</i>	63
3.3.2	<i>Dom_SecGetKeyX</i>	64
3.3.3	<i>Dom_SecGetKeyS</i>	64
3.3.4	<i>Dom_SecGetVal</i>	65
3.3.5	<i>Dom_SecGetValX</i>	66
3.3.6	<i>Dom_SecGetStr</i>	67
3.3.7	<i>Dom_SecGetUl</i>	68
3.3.8	<i>Dom_SecGetUc</i>	69
3.3.9	<i>Dom_SecGetUs</i>	69
3.3.10	<i>Dom_SecGetBool</i>	70
3.3.11	<i>Dom_KeyGetVal</i>	71
3.3.12	<i>Dom_KeyGetStr</i>	71
3.3.13	<i>Dom_KeyGetUl</i>	72
3.3.14	<i>Dom_KeyGetUs</i>	73
3.3.15	<i>Dom_KeyGetUc</i>	74
3.3.16	<i>Dom_KeyGetBool</i>	74
3.3.17	<i>Dom_KeyGetAttrStr</i>	75
3.3.18	<i>Dom_KeyGetAttrUl</i>	76
3.3.19	<i>Dom_KeyGetAttrUs</i>	77
3.3.20	<i>Dom_KeyGetAttrUc</i>	78
3.3.21	<i>Dom_DocPutRoot</i>	78
3.3.22	<i>Dom_SecPutKey</i>	79
3.3.23	<i>Dom_SecSetStr</i>	80
3.3.24	<i>Dom_SecSetUl</i>	81
3.3.25	<i>Dom_SecSetUs</i>	81
3.3.26	<i>Dom_SecSetUc</i>	82
3.3.27	<i>Dom_SecSetBool</i>	83
3.3.28	<i>Dom_KeyPutVal</i>	84
3.3.29	<i>Dom_KeySetUl</i>	84
3.3.30	<i>Dom_KeySetUs</i>	85
3.3.31	<i>Dom_KeySetUc</i>	86
3.3.32	<i>Dom_KeySetBool</i>	86
3.3.33	<i>Dom_KeySetAttrStr</i>	87
3.3.34	<i>Dom_KeySetAttrUl</i>	88

3.3.35	<i>Dom_KeySetAttrUs</i>	89
3.3.36	<i>Dom_KeySetAttrUc</i>	89
3.3.37	<i>Dom_KeySetAttrBool</i>	90
3.4	SAX INTERFACES	91
3.4.1	<i>Interface Structures</i>	91
3.4.1.1	ST_SAX1_ACTION.....	91
3.4.1.2	ST_SAX1_DTD_ACTION	91
3.4.1.3	ST_ZOS_USTR.....	91
3.4.2	<i>Sax1_SetDftAction</i>	92
3.4.3	<i>Sax1_SetDtdAction</i>	92
3.4.4	<i>Sax1_SetDocAction</i>	93
3.4.5	<i>Sax1_SetResolveAction</i>	93
3.4.6	<i>Sax1_SetErrAction</i>	94
3.4.7	<i>Sax1_ParseData</i>	95
3.4.8	<i>Sax1_ParseFile</i>	95
3.5	CODEC INTERFACES.....	96
3.5.1	<i>Interface Structures</i>	96
3.5.1.1	ST_XML_ENCODE_MSG.....	96
3.5.1.2	ST_XML_ERR_INFO.....	97
3.5.1.3	ST_XML_MSG.....	97
3.5.1.4	ST_ZOS_EBUF.....	97
3.5.1.5	ST_XML_DECODE_MSG	98
3.5.2	<i>Xml_ErrInit</i>	98
3.5.3	<i>Xml_ErrDestroy</i>	98
3.5.4	<i>Xml_ErrPrint</i>	99
3.5.5	<i>Xml_EncodeInit</i>	100
3.5.6	<i>Xml_EncodeMsg</i>	100
3.5.7	<i>Xml_DecodeInit</i>	101
3.5.8	<i>Xml_DecodeMsg</i>	102
3.6	XSP INTERFACES.....	104
3.6.1	<i>Interface Structures</i>	104
3.6.1.1	ST_XML_QNAME.....	104
3.6.1.2	ST_XSP_NS	104
3.6.1.3	ST_XML_ELEM.....	104
3.6.1.4	ST_XML_ATTR	105
3.6.1.5	ST_XML_CONTENT_ITEM	105
3.6.2	<i>Map Interfaces</i>	105
3.6.2.1	Xsp_MapGetNsId.....	105
3.6.2.2	Xsp_MapGetNsStr.....	106
3.6.2.3	Xsp_MapGetTknId	106
3.6.2.4	Xsp_MapGetTknStr.....	107
3.6.3	<i>Encoding Interfaces</i>	108
3.6.3.1	Xsp_MsgSaveFile.....	108
3.6.3.2	Xsp_MsgSaveData	109

3.6.3.3	Xsp_DocAddHdr	109
3.6.3.4	Xsp_DocAddRoot	110
3.6.3.5	Xsp_DocAddNsRoot	111
3.6.3.6	Xsp_DocNsAddRoot	112
3.6.3.7	Xsp_ElemAddChild	113
3.6.3.8	Xsp_ElemAddNsChild	114
3.6.3.9	Xsp_ElemNsAddChild	115
3.6.3.10	Xsp_ElemAddAttr	116
3.6.3.11	Xsp_ElemAddAttrVal	117
3.6.3.12	Xsp_ElemAddAttrId	118
3.6.3.13	Xsp_ElemAddAttrIdVal	119
3.6.3.14	Xsp_ElemAddAttrIdValId	120
3.6.3.15	Xsp_ElemAddAttrIdBool	121
3.6.3.16	Xsp_ElemAddAttrIdUIDigit	122
3.6.3.17	Xsp_ElemNsAddAttrId	123
3.6.3.18	Xsp_ElemNsAddAttrIdVal	124
3.6.3.19	Xsp_ElemNsAddAttrIdValId	125
3.6.3.20	Xsp_ElemNsAddAttrIdBool	126
3.6.3.21	Xsp_ElemNsAddAttrIdUIDigit	127
3.6.3.22	Xsp_ElemAddData	128
3.6.3.23	Xsp_ElemAddDataId	129
3.6.3.24	Xsp_ElemNsAddDataId	130
3.6.3.25	Xsp_ElemAddBool	131
3.6.3.26	Xsp_ElemAddUIDigit	132
3.6.3.27	Xsp_ElemAddSIDigit	133
3.6.3.28	Xsp_AttrAddData	133
3.6.3.29	Xsp_AttrAddBool	134
3.6.3.30	Xsp_AttrAddUIDigit	135
3.6.3.31	Xsp_AttrAddSIDigit	136
3.6.4	<i>Decoding Interfaces</i>	137
3.6.4.1	Xsp_MsgLoadFile	137
3.6.4.2	Xsp_MsgLoadData	137
3.6.4.3	Xsp_DocGetRoot	138
3.6.4.4	Xsp_DocGetNs	139
3.6.4.5	Xsp_ElemIsEmpty	139
3.6.4.6	Xsp_ElemGetName	140
3.6.4.7	Xsp_ElemGetNameId	141
3.6.4.8	Xsp_ElemNsGetNameId	142
3.6.4.9	Xsp_ElemGetChild	142
3.6.4.10	Xsp_ElemGetNsChild	143
3.6.4.11	Xsp_ElemNsGetChild	144
3.6.4.12	Xsp_ElemGetFristChild	145
3.6.4.13	Xsp_ElemGetLastChild	146
3.6.4.14	Xsp_ElemGetNextSibling	147

3.6.4.15	Xsp_ElemGetPrevSibling	147
3.6.4.16	Xsp_ElemGetFirstNsChild	148
3.6.4.17	Xsp_ElemGetLastNsChild	149
3.6.4.18	Xsp_ElemGetNextNsSibling	150
3.6.4.19	Xsp_ElemGetPrevNsSibling	151
3.6.4.20	Xsp_ElemNsGetFirstChild	152
3.6.4.21	Xsp_ElemNsGetLastChild	153
3.6.4.22	Xsp_ElemNsGetNextSibling	154
3.6.4.23	Xsp_ElemNsGetPrevSibling	155
3.6.4.24	Xsp_ElemGetFirstAttr	156
3.6.4.25	Xsp_ElemGetLastAttr	156
3.6.4.26	Xsp_ElemGetFirstNsAttr	157
3.6.4.27	Xsp_ElemGetLastNsAttr	158
3.6.4.28	Xsp_ElemNsGetFirstAttr	159
3.6.4.29	Xsp_ElemNsGetLastAttr	160
3.6.4.30	Xsp_ElemGetAttr	161
3.6.4.31	Xsp_ElemGetAttrX	162
3.6.4.32	Xsp_ElemGetAttrId	162
3.6.4.33	Xsp_ElemNsGetAttrId	163
3.6.4.34	Xsp_ElemGetAttrVal	164
3.6.4.35	Xsp_ElemGetAttrValX	165
3.6.4.36	Xsp_ElemGetAttrIdVal	166
3.6.4.37	Xsp_ElemGetAttrIdValId	167
3.6.4.38	Xsp_ElemGetAttrIdBool	168
3.6.4.39	Xsp_ElemGetAttrIdUIDigit	169
3.6.4.40	Xsp_ElemNsGetAttrIdVal	170
3.6.4.41	Xsp_ElemNsGetAttrIdValId	171
3.6.4.42	Xsp_ElemNsGetAttrIdBool	172
3.6.4.43	Xsp_ElemNsGetAttrIdUIDigit	173
3.6.4.44	Xsp_ElemGetData	174
3.6.4.45	Xsp_ElemGetDataId	174
3.6.4.46	Xsp_ElemNsGetDataId	175
3.6.4.47	Xsp_ElemGetBool	176
3.6.4.48	Xsp_ElemGetUIDigit	177
3.6.4.49	Xsp_ElemGetSIDigit	177
3.6.4.50	Xsp_AttrGetNext	178
3.6.4.51	Xsp_AttrGetPrev	179
3.6.4.52	Xsp_AttrGetName	179
3.6.4.53	Xsp_AttrGetNameId	180
3.6.4.54	Xsp_AttrNsGetNameId	181
3.6.4.55	Xsp_AttrGetData	182
3.6.4.56	Xsp_AttrGetDataId	182
3.6.4.57	Xsp_AttrNsGetDataId	183
3.6.4.58	Xsp_AttrGetBool	184

3.6.4.59	Xsp_AttrGetUIDigit	185
3.6.4.60	Xsp_AttrGetSIDigit	185
3.6.5	<i>Utility Interfaces</i>	186
3.6.5.1	Xml_GetContentSize	186
3.6.5.2	Xml_GetContentItem	187
3.6.6	<i>Config Interfaces</i>	187
3.6.6.1	Xml_CfgGetLogLevel	187
3.6.6.2	Xml_CfgSetLogLevel	188

List of Tables

Table 2-1	Basic Data Types	10
Table 2-2	Platform Types	11
Table 3-1	Example table	11

List of Figures

Figure 3-1	DOM representation of the example table	12
------------	---	----

1. Introduction

XML stands for *extensible markup language* and is defined by the World Wide Web Consortium (W3C, www.w3c.org). It's a simple syntax that describes information, a set of technologies that allows you to format and filter information independently of how that information is represented, and the embodiment of an idea that reduces data to its purest form, devoid of formatting and other irrelevant aspects, to attain a very high level of usefulness and flexibility. It's not a markup language. It's a metamarkup specification that lets you create your own markup language.

1.1 Purpose

This document is provided to developers who will develop their own products using DOM (Document Object Model) interfaces and Simple API for XML (SAX).

1.2 Audience

The readers of this document are assumed to have a working knowledge of XML.

1.3 Scope

This document covers DOM interfaces and SAX but not the description about the design and the realization of them.

1.4 Definitions, Acronyms, and Abbreviations

The following definitions, acronyms, and abbreviations are used in this document:

Abbreviation	Description
DOM	Document Object Model
DTD	Data Type Definition
SAX	Simple API for XML
XML	Extensible Markup Language

2. System Environment

2.1 Basic Data Types

There are some basic data types provided by ZOS platform. Table 2-1 lists these types used by the SIP Stack and user agent applications.

Name	Type
ZDOUBLE	double
ZFLOAT	float
ZLONG	long
ZINT	int
ZSHORT	short
ZCHAR	char
ZULONG	unsigned long
ZUINT	unsigned int
ZSIZE_T	unsigned int
ZUSHORT	unsigned short
ZUCHAR	unsigned char
ZBOOL	int
ZVOID	void

Table 2-1 Basic Data Types

2.2 Platform Types

Table 2-3 lists the basic platform types.

Name	Type
ZMUTEX	Mutex
ZSEM	Semaphore
ZTIME_T	Time
ZFUNCPTR	Function Pointer
ZVOIDFUNCPTR	Void Function Pointer
ZLOGID	Log ID
ZMODID	Module ID
ZINSTID	Instance ID
ZTASKID	Task ID
ZTIMERID	Timer ID
ZEVENTID	Event ID

ZPOOLID	Pool ID
---------	---------

Table 2-2 Platform Types

3. User Interfaces

3.1 Basic Concepts

This section gives a brief introduction on DOM and SAX.

3.1.1 DOM

DOM (Document Object Model) is a platform- and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure and style of documents. The Document Object Model provides a standard set of objects for representing HTML and XML documents, a standard model of how these objects can be combined, and a standard interface for accessing and manipulating them. DOM creates a tree of nodes (based on the structure and information in your XML document) and you can access your information by interacting with this tree of nodes.

3.1.2 SAX

SAX (Simple API for XML) like DOM gives access to the information stored in XML documents using any programming language (and a parser for that language). Unlike DOM, with SAX the parser tells the application what is in the document by notifying the application of a stream of parsing events. Application then processes those events to act on data. SAX is very useful when the document is large.

3.2 DOM Interfaces

The DOM presents documents as a hierarchy of node objects that also implement other, more specialized interfaces. Some types of nodes may have child nodes of various types, and others are leaf nodes that cannot have anything below them in the document structure. Look at the table taken from an HTML document:

```
<TABLE>
<TBODY>
<TR>
<TD>Shady Grove</TD>
<TD>Aeolian</TD>
</TR>
<TR>
<TD>Over the River, Charlie</TD>
<TD>Dorian</TD>
</TR>
</TBODY>
</TABLE>
```

Table 3-1 Example table

And the DOM represents this table like this:

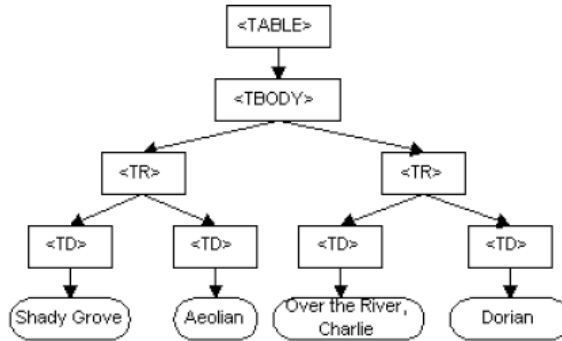


Figure 3-1 DOM representation of the example table

This is a logical structure like a tree of nodes (see the figure above). **But the nodes in the figure above do not represent a data structure.** They are objects which have functions and identity.

3.2.1 Interface Structures

3.2.1.1 *ST_DOM_NODE*

```

typedef struct tagDOM_NODE ST_DOM_NODE;

struct tagDOM_NODE
{
    ZUCHAR ucType;                /* node type EN_DOM_NODE_TYPE */
    ZUCHAR aucSpare[3];          /* for 32 bit alignment */
    ST_DOM_STR stName;           /* node name */
    ST_DOM_STR stVal;            /* node value */
    ST_XML_LIST_NODE stToDoNode; /* todo list node */
    ST_DOM_NODE_LST *pstChildNodes; /* children nodes */
    struct tagDOM_NODE *pstFirstChild; /* the first child node */
    struct tagDOM_NODE *pstLastChild; /* the last child node */
    struct tagDOM_NODE *pstNextSibling; /* the next sibling node */
    struct tagDOM_NODE *pstPrevSibling; /* the previous sibling node */
    struct tagDOM_NODE *pstParentNode; /* parent node */
    ST_DOM_NNODE_MAP stAttrs;    /* attributes */
    ST_DOM_DOC *pstOwnerDoc;     /* document owner */
    ST_DOM_STR stPrefix;         /* namespace prefix */
};

```

3.2.1.2 *EN_DOM_NODE_TYPE*

```
typedef enum EN_DOM_NODE_TYPE
{
    EN_DOM_NODE_NULL = 0,
    EN_DOM_NODE_ELEM = 1,
    EN_DOM_NODE_ATTR = 2,
    EN_DOM_NODE_TXT = 3,
    EN_DOM_NODE_CDATA_SEC = 4,
    EN_DOM_NODE_ENT_REFER = 5,
    EN_DOM_NODE_ENT = 6,
    EN_DOM_NODE_PI = 7,
    EN_DOM_NODE_COMMENT = 8,
    EN_DOM_NODE_DOC = 9,
    EN_DOM_NODE_DOC_TYPE = 10,
    EN_DOM_NODE_DOC_FRAG = 11,
    EN_DOM_NODE_NOTATION = 12
} EN_DOM_NODE_TYPE;
```

3.2.1.3 *ST_DOM_STR*

```
typedef struct tagZOS_USTR
{
    ZUCHAR *pucStr;           /* string pointer */
    ZUSHORT wLen;            /* string length */
    ZUCHAR aucSpare[2];      /* for 32 bit alignment */
} ST_ZOS_USTR;

typedef ST_ZOS_USTR ST_DOM_STR;
```

3.2.1.4 *ST_DOM_DOC*

```
typedef struct tagDOM_DOC ST_DOM_DOC;

struct tagDOM_DOC
{
    ST_DOM_DTYPE stDocType;           /* document type */
    ST_DOM_ELEM stDocElem;           /* document element */
    ST_DOM_STR stInputEncoding;      /* input encoding */
    ST_DOM_STR stXmlEncoding;       /* xml encoding */
    ZBOOL bXmlStandAlone;            /* xml standalone; */
    ST_DOM_STR stXmlVer;             /* xml version */
    ST_DOM_STR stDocUri;             /* document uri */
    ZSBUFID zMemBufId;              /* memory buffer */
};
```

3.2.1.5 *ST_DOM_NNODE_MAP*

```
typedef struct tagDOM_NNODE_MAP
{
    ST_DOM_NAME_LST stNameLst;       /* name list */
} ST_DOM_NNODE_MAP;
```

3.2.1.6 *ST_DOM_CDATA*

```
typedef struct tagDOM_CDATA
{
    ST_DOM_NODE stNode;              /* Comment Node */
    ST_DOM_STR stData;               /* node data */
} ST_DOM_CDATA;
```

3.2.1.7 *ST_DOM_ATTR*

```
typedef struct tagDOM_ATTR
{
    ST_DOM_NODE stNode;              /* Attr Node */
    ST_DOM_STR stName;               /* node name */
    ST_DOM_STR stVal;                /* node value */
    ZBOOL bdQuote;                   /* double quote */
} ST_DOM_ATTR;
```

3.2.1.8 *ST_DOM_ELEM*

```
typedef struct tagDOM_ELEM
{
    ST_DOM_NODE stNode;           /* Element Node */
    ST_DOM_STR stTagName;        /* tag name */
} ST_DOM_ELEM;
```

3.2.1.9 *ST_DOM_DOC*

```
typedef struct tagDOM_DOC ST_DOM_DOC;

struct tagDOM_DOC
{
    ST_DOM_DTYPE stDocType;      /* document type */
    ST_DOM_ELEM stDocElem;      /* document element */
    ST_DOM_STR stInputEncoding; /* input encoding */
    ST_DOM_STR stXmlEncoding;   /* xml encoding */
    ZBOOL bXmlStandAlone;       /* xml standalone */
    ST_DOM_STR stXmlVer;        /* xml version */
    ST_DOM_STR stDocUri;        /* document uri */
    ZSBUFID zMemBufId;         /* memory buffer */
};
```

3.2.1.10 *ST_DOM_DFRAG*

```
typedef ST_DOM_NODE ST_DOM_DFRAG;

typedef struct tagDOM\_NODE ST_DOM_NODE;
```

3.2.1.11 *ST_DOM_TXT*

```
typedef ST\_DOM\_CDATA ST_DOM_TXT;
```

3.2.1.12 *ST_DOM_COMMT*

```
typedef ST\_DOM\_CDATA ST_DOM_COMMT;
```

3.2.1.13 *ST_DOM_CDATA_SEC*

```
ST\_DOM\_TXT ST_DOM_CDATA_SEC;
```

3.2.1.14 ST_DOM_PI

```
typedef struct tagDOM_PI
{
    ST_DOM_NODE stNode;           /* ProcessingInstruction Node */
    ST_DOM_STR stTarget;         /* target */
    ST_DOM_STR stData;          /* data */
} ST_DOM_PI;
```

3.2.1.15 ST_ZOS_DBUF

```
typedef struct tagZOS_DBUF
{
    struct tagZOS_DBUF *pstNext; /* next data buffer */
    ZPOOLID zPoolId;            /* memory pool id for dbuf alloc */
    ZULONG dwBufLen;            /* buffer used length */
    ZULONG dwDftBlkSize;        /* default data block size in buffer */
    ZUCHAR ucBufType;           /* buffer mode ZDBUF_TYPE_BYTE... */
    ZUCHAR ucRefCnt;            /* buffer reference count */
    ZUCHAR aucSpare[2];         /* for 32 bit alignment */
#ifdef ZOS_SUPT_DUMP
    ZDUMPID zDumpId;           /* stack dump */
#endif
    ST_ZOS_DBUF_DATA *pstHead; /* the first data block in buffer */
    ST_ZOS_DBUF_DATA *pstTail; /* the last data block in buffer */
} ST_ZOS_DBUF;
```

3.2.2 Dom_NodeGetType

Retrieves the type of a given node.

```
ZINT Dom_NodeGetType(ST\_DOM\_NODE *pstNode, ZUSHORT *pwType);
```

[Parameters]

Input parameters:

[ST_DOM_NODE](#) *pstNode

A given DOM node.

ZUSHORT *pwType

This will point to the type of the given node.

The enumeration [EN_DOM_NODE_TYPE](#) enumerates all the node types.

Output parameters:

```
ZUSHORT *pwType
```

The pointer to the type of the given node.

[Return value]

Returns ZOK when the type has been obtained or ZFAILED on failure.

[Example]

```
ST\_DOM\_NODE *pstNode;
ZUSHORT wType;
ZINT iRet;
.....
/* Retrieve the type of a given node. */
iRet = Dom_NodeGetType(pstNode, &wType);
```

3.2.3 Dom_NodeGetName

Retrieves the name of a given node.

```
ZINT Dom_NodeGetName(ST\_DOM\_NODE *pstNode, ST\_DOM\_STR **ppstName);
```

[Parameters]

Input parameters:

```
ST\_DOM\_NODE *pstNode
```

A given DOM node.

```
ST\_DOM\_STR **ppstName
```

This pointer will point to the name of the given node.

Output parameters:

```
ST\_DOM\_STR **ppstName
```

The pointer to the name of the given node.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```
ST\_DOM\_NODE *pstNode;
ST\_DOM\_STR *pstName;
ZINT iRet;
.....
/* Retrieve the name of a given node. */
iRet = Dom_NodeGetName(pstNode, &pstName);
```

3.2.4 Dom_NodeSetName

Sets the name of a given node. The name depends on the type of the node.

```
ZINT Dom_NodeSetName(ST_DOM_NODE *pstNode, ST_DOM_STR *pstName);
```

[Parameters]

Input parameters:

ST_DOM_NODE *pstNode

A given DOM node.

ST_DOM_STR *pstName

The name to set to the given node.

Output parameters:

None.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```
ST_DOM_NODE *pstNode;  
ST_DOM_STR *pstName  
ZINT iRet;  
.....  
/* Set the name of a given node. */  
iRet = Dom_NodeSetName(pstNode, pstName);
```

3.2.5 Dom_NodeGetVal

Retrieves the value of a given node. The value depends on the type of the node.

```
INT Dom_NodeGetVal(ST_DOM_NODE *pstNode, ST_DOM_STR **ppstVal);
```

[Parameters]

Input parameters:

ST_DOM_NODE *pstNode

A given DOM node.

ST_DOM_STR **ppstVal

The pointer which will point to the value of the given node.

Output parameters:

[ST_DOM_STR](#) **ppstVal

The pointer to the value of the given node.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```
ST\_DOM\_NODE *pstNode;
ST\_DOM\_STR *pstVal;
ZINT iRet;
.....
/* Retrieve the value of a given node. */
iRet = Dom_NodeGetVal(pstNode, &pstVal);
```

3.2.6 Dom_NodeSetVal

Sets the value of a given node. The value depends on the type of the node.

```
ZINT Dom_NodeSetVal(ST\_DOM\_NODE *pstNode, ST\_DOM\_STR *pstVal);
```

[Parameters]

Input parameters:

[ST_DOM_NODE](#) *pstNode

A given DOM node.

[ST_DOM_STR](#) *pstVal

The pointer to the value of the given node.

Output parameters:

None.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```
ST\_DOM\_NODE *pstNode;
ST\_DOM\_STR *pstVal;
ZINT iRet;
.....
/* Get the value of a given node. */
iRet = Dom_NodeSetVal(pstNode, pstVal);
```

3.2.7 Dom_NodeGetParent

Retrieves the parent node of a given node.

```
ZINT Dom_NodeGetParent(ST\_DOM\_NODE *pstNode, ST\_DOM\_NODE **ppstParent)
```

[Parameters]

Input parameters:

```
ST\_DOM\_NODE *pstNode
```

A given DOM node.

```
ST\_DOM\_NODE **ppstParent
```

The pointer which will point to the parent node of the given node.

Output parameters:

```
ST\_DOM\_NODE **ppstParent
```

The pointer to the parent node of the given node.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```
ST\_DOM\_NODE *pstNode;
```

```
ST\_DOM\_NODE *pstParent;
```

```
ZINT iRet;
```

```
.....
```

```
/* Retrieve the parent node of a given node. */
```

```
iRet = Dom_NodeGetParent();
```

3.2.8 Dom_NodeHasChilds

Checks whether a given node has any child nodes. As mentioned before, some types of nodes may have child nodes of various types, and others are leaf nodes that cannot have anything below them in the document structure.

```
ZBOOL Dom_NodeHasChilds(ST\_DOM\_NODE *pstNode);
```

[Parameters]

Input parameters:

```
ST\_DOM\_NODE *pstNode
```

A given DOM node.

Output parameters:

None.

[Return value]

Returns ZTRUE if it has, or ZFALSE if it does not or the input parameter is ZNULL.

[Example]

```

ST_DOM_NODE *pstNode;
ZBOOL iBool;
.....
/* Check whether a given node has any children. */
iBool = Dom_NodeHasChilDs(pstNode);

```

3.2.9 Dom_NodeGetChilDs

Retrieves a pointer to the list of all child nodes of a given node.

```

ZINT Dom_NodeGetChilDs(ST_DOM_NODE *pstNode,
                       ST_DOM_NODE_LST **ppstChildLst)

```

[Parameters]

Input parameters:

[ST_DOM_NODE](#) *pstNode
A given DOM node.

ST_DOM_NODE_LST **ppstChildLst
A pointer which will point to the pointer to the list.

Output parameters:

ST_DOM_NODE_LST **ppstChildLst
A pointer to the pointer to the list.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```

ST_DOM_NODE *pstNode;
ST_DOM_NODE_LST *pstChildLst;
ZINT iRet;
.....
/* Retrieve a list of child nodes of a given node. */
iRet = Dom_NodeGetChilDs(pstNode, &pstChildLst);

```

3.2.10 Dom_NodeGetFristChild

Retrieves the first child node of a given node. Note that some leaf nodes cannot have anything below them in the document structure.

```
ZINT Dom_NodeGetFristChild(ST\_DOM\_NODE *pstNode,  
                           ST\_DOM\_NODE **ppstChild)
```

[Parameters]

Input parameters:

[ST_DOM_NODE](#) *pstNode

A given DOM node.

[ST_DOM_NODE](#) **ppstChild

Used to point to the pointer to the first child node.

Output parameters:

[ST_DOM_NODE](#) **ppstChild

The pointer to the pointer to the first child node.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```
ST\_DOM\_NODE *pstNode;
```

```
ST\_DOM\_NODE *pstChild;
```

```
ZINT iRet;
```

```
.....
```

```
/* Retrieve the first child node of a given node. */
```

```
iRet = Dom_NodeGetFristChild(pstNode, &pstChild);
```

3.2.11 Dom_NodeGetLastChild

Retrieves the last child node of a given node. Note that some leaf nodes cannot have anything below them in the document structure.

```
ZINT Dom_NodeGetLastChild(ST\_DOM\_NODE *pstNode,  
                           ST\_DOM\_NODE **ppstChild)
```

[Parameters]

Input parameters:

[ST_DOM_NODE](#) *pstNode

A given DOM node.

[ST_DOM_NODE](#) **ppstChild

Used to point to the pointer to the last child node.

Output parameters:

[ST_DOM_NODE](#) **ppstChild

The pointer to the pointer to the last child node.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```
ST\_DOM\_NODE *pstNode;
```

```
ST\_DOM\_NODE *pstChild;
```

```
ZINT iRet;
```

```
.....
```

```
/* Retrieve the last child node of a given node. */
```

```
iRet = Dom_NodeGetLastChild(pstNode, &pstChild);
```

3.2.12 Dom_NodeGetPrevSibling

Retrieves the previous sibling node of a given node.

```
ZINT Dom_NodeGetPrevSibling(ST\_DOM\_NODE *pstNode,  
                             ST\_DOM\_NODE **ppstSibling)
```

[Parameters]

Input parameters:

[ST_DOM_NODE](#) *pstNode

A given DOM node.

[ST_DOM_NODE](#) ** ppstSibling

Used to point to the pointer to the previous sibling node.

Output parameters:

[ST_DOM_NODE](#) ** ppstSibling

The pointer to the pointer to the previous sibling node.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```

ST_DOM_NODE *pstNode;
ST_DOM_NODE *pstSibling;
ZINT iRet;
.....
/* Retrieve the previous sibling node of a given node. */
iRet = Dom_NodeGetPrevSibling(pstNode, &pstSibling);

```

3.2.13 Dom_NodeGetNextSibling

Retrieves the next sibling node of a given node.

```

ZINT Dom_NodeGetNextSibling(ST_DOM_NODE *pstNode,
                            ST_DOM_NODE **ppstSibling)

```

[Parameters]

Input parameters:

ST_DOM_NODE *pstNode
A given DOM node.

ST_DOM_NODE ** ppstSibling
Used to point to the pointer to the next sibling node.

Output parameters:

ST_DOM_NODE ** ppstSibling
The pointer to the pointer to the next sibling node.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```

ST_DOM_NODE *pstNode;
ST_DOM_NODE *pstSibling;
ZINT iRet;
.....
/* Retrieve the next sibling node of a given node. */
iRet = Dom_NodeGetNextSibling(pstNode, &pstSibling);

```

3.2.14 Dom_NodeHasAttrs

Checks if a given node has any attributes.

```

ZBOOL Dom_NodeHasAttrs(ST_DOM_NODE *pstNode)

```

[Parameters]**Input parameters:**

[ST_DOM_NODE](#) *pstNode
A given DOM node.

Output parameters:

None.

[Return value]

Returns ZTRUE if the given node has or ZFALSE if it does not have.

[Example]

```
ST\_DOM\_NODE *pstNode;
ZBOOL iBool;

.....

/* Check if a given node has any attributes. */
iBool = Dom_NodeHasAttrs(pstNode);
```

3.2.15 Dom_NodeGetOwnerDoc

Retrieves the owner document of a given node.

```
ZINT Dom_NodeGetOwnerDoc(ST\_DOM\_NODE *pstNode,
ST\_DOM\_DOC **ppstDoc)
```

[Parameters]**Input parameters:**

[ST_DOM_NODE](#) *pstNode
A given DOM node.

[ST_DOM_DOC](#) **ppstDoc
The pointer which will point to the owner document.

Output parameters:

[ST_DOM_DOC](#) **ppstDoc
The pointer to the owner document.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```
ST_DOM_NODE *pstNode;  
ST_DOM_DOC *pstDoc;  
ZINT iRet;  
.....  
/* Retrieve the owner document of a given node. */  
iRet = Dom_NodeGetOwnerDoc(pstNode, &pstDoc);
```

3.2.16 Dom_NodeInsertBefore

Inserts a new child node before an existing child node.

```
ZINT Dom_NodeInsertBefore(ST_DOM_NODE *pstNode,  
                           ST_DOM_NODE *pstNewChild, ST_DOM_NODE *pstRefNode)
```

[Parameters]

Input parameters:

ST_DOM_NODE *pstNode

A given DOM node.

ST_DOM_NODE *pstNewChild

The pointer to the new child node.

ST_DOM_NODE *pstRefNode

The pointer to the reference node before which the new node must be inserted.

Output parameters:

None.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```
ST_DOM_NODE *pstNode;  
ST_DOM_NODE *pstNewChild;  
ST_DOM_NODE *pstRefNode;  
.....  
/* Insert a new child node. */  
iRet = Dom_NodeInsertBefore(pstNode, pstNewChild, pstRefNode);
```

3.2.17 Dom_NodeReplacChild

Replaces an old child node of a given node with a new child node.

```
ZINT Dom_NodeReplacChild(ST\_DOM\_NODE *pstNode,
                          ST\_DOM\_NODE *pstNewChild, ST\_DOM\_NODE *pstOldChild)
```

[Parameters]

Input parameters:

[ST_DOM_NODE](#) *pstNode

A given DOM node.

[ST_DOM_NODE](#) *pstNewChild

The new child node.

[ST_DOM_NODE](#) *pstOldChild

The old child node.

Output parameters:

None.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```
ST\_DOM\_NODE *pstNode;
ST\_DOM\_NODE *pstNewChild;
ST\_DOM\_NODE *pstOldChild;
ZINT iRet;
.....
/* Replace the old child node of given node with a new child node. */
iRet = Dom_NodeReplacChild(pstNode, pstNewChild, pstOldChild);
```

3.2.18 Dom_NodeRemChild

Removes a child node of a given node from the list of children.

```
ZINT Dom_NodeRemChild(ST\_DOM\_NODE *pstNode,
                       ST\_DOM\_NODE *pstOldChild)
```

[Parameters]

Input parameters:

[ST_DOM_NODE](#) *pstNode

A given DOM node.

[ST_DOM_NODE](#) *pstOldChild

The node being removed.

Output parameters:

None.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```
ST_DOM_NODE *pstNode;
ST_DOM_NODE *pstOldChild;
ZINT iRet;
.....
/* Remove a child node. */
iRet = Dom_NodeRemChild(pstNode, pstOldChild);
```

3.2.19 Dom_NodeAppendChild

Adds a new child node to the list of children of a given node. The new node is allowed to be the same as one of the existing child nodes in the list.

```
ZINT Dom_NodeAppendChild(ST_DOM_NODE *pstNode,
                          ST_DOM_NODE *pstNewChild)
```

[Parameters]**Input parameters:**

ST_DOM_NODE *pstNode
A given DOM node.

ST_DOM_NODE *pstNewChild
The node to add.

Output parameters:

None.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```
ST_DOM_NODE *pstNode;
ST_DOM_NODE *pstNewChild;
ZINT iRet;
.....
/* Add a new child node. */
iRet = Dom_NodeAppendChild(pstNode, pstNewChild);
```

3.2.20 Dom_NodeClone

Duplicates a given node.

```
ZINT Dom_NodeClone(ST_DOM_NODE *pstNode, ZBOOL bDeep,
                   ST_DOM_NODE **ppstNewNode)
```

[Parameters]

Input parameters:

ST_DOM_NODE *pstNode

A given DOM node to duplicate.

ZBOOL bDeep

Indicates whether or not duplication is allowed. If it is ZTRUE, it is allowed.

ST_DOM_NODE **ppstNewNode

This pointer will point to the copy of the given node.

Output parameters:

ST_DOM_NODE **ppstNewNode

The pointer to the copy of the given node.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```
ST_DOM_NODE *pstNode;
ST_DOM_NODE *pstNewNode;
ZINT iRet;
ZBOOL bDeep;
.....
/* Duplicate a node. */
iRet = Dom_NodeClone(pstNode, &pstNewNode);
```

3.2.21 Dom_NodeLstGetItem

Retrieves the indexth node in a node list.

```
ZINT Dom_NodeLstGetItem(ST_DOM_NODE_LST *pstNodeLst, ZULONG dwIndex,
                       ST_DOM_NODE **ppstNode)
```

[Parameters]

Input parameters:

ST_DOM_NODE_LST *pstNodeLst

A node list.

ZULONG dwIndex

Index into the list.

[ST_DOM_NODE](#) **ppstNode

This pointer will point to the indexth node.

Output parameters:

[ST_DOM_NODE](#) **ppstNode

The pointer to the indexth node.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```
ST_DOM_NODE_LST *pstNodeLst;
```

```
ZULONG dwIndex;
```

```
ST\_DOM\_NODE *pstNode;
```

```
ZINT iRet;
```

```
.....
```

```
/* Retrieve the indexth node of a node list. */
```

```
iRet = Dom_NodeLstGetItem
```

3.2.22 Dom_NodeLstGetLen

Retrieves the number of nodes in a list.

```
ZINT Dom_NodeLstGetLen(ST_DOM_NODE_LST *pstNodeLst, ZULONG *pdwLen)
```

[Parameters]

Input parameters:

ST_DOM_NODE_LST *pstNodeLst

A node list.

ZULONG *pdwLen

The pointer which will point to the number of nodes.

Output parameters:

ZULONG *pdwLen

The pointer to the number of nodes.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```
ST_DOM_NODE_LST *pstNodeLst;
ZULONG dwLen;
ZINT iRet;
.....
/* Retrieve the number of nodes in a list. */
iRet = Dom_NodeLstGetLen(pstNodeLst, &dwLen);
```

3.2.23 Dom_NNodeMapGetNItem

Retrieves a node specified by name.

```
ZINT Dom_NNodeMapGetNItem(ST\_DOM\_NNODE\_MAP *pstNNodeMap,
                          ST\_DOM\_STR *pstName, ST\_DOM\_NODE **ppstNode)
```

[Parameters]

Input parameters:

[ST_DOM_NNODE_MAP](#) *pstNNodeMap
The map.

[ST_DOM_STR](#) *pstName
Name of a node to retrieve.

[ST_DOM_NODE](#) **ppstNode
Pointer to the node to retrieve.

Output parameters:

[ST_DOM_NODE](#) **ppstNode
Pointer to the retrieved node.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```
ST\_DOM\_NNODE\_MAP *pstNNodeMap;
ST\_DOM\_STR *pstName;
ST\_DOM\_NODE *pstNode;
ZINT iRet;
.....
/* Retrieve a node specified by name. */
iRet = Dom_NNodeMapGetNItem(pstNNodeMap, pstName, &pstNode);
```

3.2.24 Dom_NNodeMapSetNItem

Adds a node to a node map.

```
ZINT Dom_NNodeMapSetNItem(ST\_DOM\_NNODE\_MAP *pstNNodeMap,  
                           ST\_DOM\_NODE *pstArgNode)
```

[Parameters]

Input parameters:

[ST_DOM_NNODE_MAP](#) *pstNNodeMap

The map.

[ST_DOM_NODE](#) *pstArgNode

The node to add.

Output parameters:

None.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```
ST\_DOM\_NNODE\_MAP *pstNNodeMap;  
ST\_DOM\_NODE *pstArgNode;  
ZINT iRet;  
.....  
/* Add a node. */  
iRet = Dom_NNodeMapSetNItem(pstNNodeMap, pstArgNode);
```

3.2.25 Dom_NnodeMapRemNItem

Removes a node specified by name from a node map.

```
ZINT Dom_NNodeMapRemNItem(ST\_DOM\_NNODE\_MAP *pstNNodeMap,  
                           ST\_DOM\_NODE *pstNode)
```

[Parameters]

Input parameters:

[ST_DOM_NNODE_MAP](#) *pstNNodeMap

The map.

[ST_DOM_NODE](#) *pstNode

Node to remove.

Output parameters:

None.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```
ST\_DOM\_NNODE\_MAP *pstNNodeMap;
ST\_DOM\_NODE *pstNode;
ZINT iRet;
.....
/* Remove a node. */
iRet = Dom_NNodeMapRemNItem(pstNNodeMap, pstNode);
```

3.2.26 Dom_NNodeMapGetItem

Retrieves the indexth item in a map.

```
ZINT Dom_NNodeMapGetItem(SST\_DOM\_NNODE\_MAP *pstNNodeMap,
                          ZULONG dwIndex, ST\_DOM\_NODE **ppstNode)
```

[Parameters]

Input parameters:

[ST_DOM_NNODE_MAP](#) *pstNNodeMap

The map.

ZULONG dwIndex

Index into the map.

[ST_DOM_NODE](#) **ppstNode

This will point to the indexth item. It will point to ZNULL on failure.

Output parameters:

[ST_DOM_NODE](#) **ppstNode

The indexth item.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```

ST_DOM_NNODE_MAP *pstNNodeMap;
ST_DOM_NODE *pstNode;
ZULONG dwIndex;
ZINT iRet;
.....
/* Retrieve the indexth item. */
iRet = Dom_NNodeMapGetItem(pstNNodeMap, dwIndex, &pstNode);

```

3.2.27 Dom_NNodeMapGetLen

Returns the number of nodes in a map.

```

ZINT Dom_NNodeMapGetLen(ST_DOM_NNODE_MAP *pstNNodeMap,
                        ZULONG *pdwLen)

```

[Parameters]**Input parameters:**

```

ST_DOM_NNODE_MAP *pstNNodeMap

```

The map.

```

ZULONG *pdwLen

```

This will point to the number of nodes in the map.

Output parameters:

```

ZULONG *pdwLen

```

The number of nodes.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```

ST_DOM_NNODE_MAP *pstNNodeMap;
ZULONG dwLen;
ZINT iRet;
.....
/* Get the number of nodes in a map. */
iRet = Dom_NNodeMapGetLen(pstNNodeMap, &dwLen);

```

3.2.28 Dom_CDataGetSubData

Extracts a range of data from a node.

```
ZINT Dom_CDataGetSubData(ST\_DOM\_CDATA *pstData, ZULONG dwOffset,
                          ZULONG dwCount, ST\_DOM\_STR *pstSubData)
```

[Parameters]

Input parameters:

[ST_DOM_CDATA](#) *pstData

Data of a node.

ZULONG dwOffset

Start offset of substring to extract.

ZULONG dwCount

The number of characters to extract.

[ST_DOM_STR](#) *pstSubData

This will point to the substring.

Output parameters:

[ST_DOM_STR](#) *pstSubData

The substring.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```
ST\_DOM\_CDATA *pstData;
ZULONG dwOffset;
ZULONG dwCount;
ST\_DOM\_STR stSubData;
ZINT iRet;
.....
/* Extract a range of data from a node. */
iRet = Dom_CDataGetSubData(pstData, dwOffset, dwCount, &stSubData);
```

3.2.29 Dom_CDataAppendData

Appends a string to the end of the character data of a node.

```
ZINT Dom_CDataAppendData (ST\_DOM\_CDATA *pstData,  
                           ST\_DOM\_STR *pstAppendData)
```

[Parameters]**Input parameters:**

[ST_DOM_CDATA](#) *pstData
Data of a node.

[ST_DOM_STR](#) *pstAppendData
The string to append.

Output parameters:

None.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```
ST\_DOM\_CDATA *pstData;  
ST\_DOM\_STR *pstAppendData;  
ZINT iRet;  
.....  
/* Append a string to the end of the character data. */  
iRet = Dom_CDataAppendData (pstData, pstAppendData);
```

3.2.30 Dom_CDataInsertData

Inserts a string at the specified character offset.

```
ZINT Dom_CDataInsertData (ST\_DOM\_CDATA *pstData, ZULONG dwOffset,  
                          ST\_DOM\_STR *pstInsertData)
```

[Parameters]**Input parameters:**

[ST_DOM_CDATA](#) *pstData
Data of a node.

ZULONG dwOffset
The character offset at which to insert.

[ST_DOM_STR](#) *pstInsertData
The string to insert.

Output parameters:

None.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```
ST_DOM_CDATA *pstData;  
ZULONG dwOffset;  
ST_DOM_STR *pstInsertData;  
ZINT iRet;  
.....  
/* Insert a string at the specified offset. */  
iRet = Dom_CDataInsertData(pstData, dwOffset, pstInsertData);
```

3.2.31 Dom_CDataDelData

Removes a range of data from a node.

```
ZINT Dom_CDataDelData(ST_DOM_CDATA *pstData, ZULONG dwOffset,  
                      ZULONG dwCount)
```

[Parameters]**Input parameters:**

`ST_DOM_CDATA *pstData`

Data of a node.

`ZULONG dwOffset`

The offset from which to remove characters.

`ZULONG dwCount`

The number of characters to delete.

Output parameters:

None.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```

ST\_DOM\_CDATA *pstData;
ZULONG dwOffset;
ZULONG dwCount;
ZINT iRet;
.....
/* Remove a range of data from a node. */
iRet = Dom_CDataDelData(pstData, dwOffset, dwCount);

```

3.2.32 Dom_CDataReplace

Replaces the whole data of a node with a specified string.

```
ZINT Dom_CDataReplace(ST\_DOM\_CDATA *pstData, ST\_DOM\_STR *pstContent)
```

[Parameters]

Input parameters:

```
ST\_DOM\_CDATA *pstData
```

Data of a node.

```
ST\_DOM\_STR *pstContent
```

The string with the data must be replaced.

Output parameters:

None.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```

ST\_DOM\_CDATA *pstData;
ST\_DOM\_STR *pstContent;
ZINT iRet;
.....
/* Replace the whole data of a node with a specified string. */
iRet = Dom_CDataReplace(pstData, pstContent);

```

3.2.33 Dom_CDataSetData

Sets the character data of a node to a specified string.

```
ZINT Dom_CDataSetData(ST\_DOM\_CDATA *pstData, ST\_DOM\_STR *pstSetData)
```

[Parameters]**Input parameters:**

[ST_DOM_CDATA](#) *pstData
Data of a node.

[ST_DOM_STR](#) *pstSetData
The string to set.

Output parameters:

None.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```
ST\_DOM\_CDATA *pstData;
ST\_DOM\_STR *pstSetData;
ZINT iRet;
.....
/* Set the data of a node to a specified string. */
iRet = Dom_CDataSetData(pstData, pstSetData);
```

3.2.34 Dom_CDataSplit

Breaks a text node into two text nodes at the specified offset and keep both in the tree as siblings. And this node will then have all the content up to the offset point. A new node, which is inserted as the next sibling of this node, contains all the content at and after the offset point.

```
ZINT Dom_CDataSplit(ST_DOM_CDATA *pstData, ZULONG dwOffset,
                   ST_DOM_CDATA **ppstNewData)
```

[Parameters]**Input parameters:**

[ST_DOM_CDATA](#) *pstData
Data of a node.

ULONG dwOffset
The offset at which to split.

[ST_DOM_CDATA](#) **ppstNewData
This will point to the data of the new node.

Output parameters:

[ST_DOM_CDATA](#) **ppstNewData

The pointer to the data of the new node.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```

ST\_DOM\_CDATA *pstData;
ST\_DOM\_CDATA *pstNewData;
ULONG dwOffset;
ZINT iRet;
.....
/* Split a node. */
iRet = Dom_CDataSplit(pstData, dwOffset, &pstNewData);

```

3.2.35 Dom_AttrGetName

Retrieves the name of an attribute.

```
ZINT Dom_AttrGetName(ST\_DOM\_ATTR *pstAttr, ST\_DOM\_STR **ppstName)
```

[Parameters]**Input parameters:**

[ST_DOM_ATTR](#) *pstAttr

The attribute whose name is to be retrieved.

[ST_DOM_STR](#) **ppstName

This will point to the name of the attribute.

Output parameters:

[ST_DOM_STR](#) **ppstName

The name of the attribute on success.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```
ST_DOM_ATTR *pstAttr;  
ST_DOM_STR *pstName;  
ZINT iRet;  
.....  
/* Retrieve the name of an attribute. */  
iRet = Dom_AttrGetName(pstAttr, &pstName);
```

3.2.36 Dom_AttrGetVal

Retrieves the value of an attribute.

```
ZINT Dom_AttrGetVal(ST_DOM_ATTR *pstAttr, ST_DOM_STR **ppstVal)
```

[Parameters]

Input parameters:

```
ST_DOM_ATTR *pstAttr
```

The attribute whose value is to be retrieved.

```
ST_DOM_STR **ppstVal
```

This will point to the value of the attribute.

Output parameters:

```
ST_DOM_STR **ppstVal
```

The value of the attribute on success.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```
ST_DOM_ATTR *pstAttr;  
ST_DOM_STR *pstVal;  
ZINT iRet;  
.....  
/* Retrieve the value of an attribute. */  
iRet = Dom_AttrGetVal(pstAttr, &pstVal);
```

3.2.37 Dom_AttrSetVal

Sets the value of an attribute.

```
ZINT Dom_AttrSetVal(ST\_DOM\_ATTR *pstAttr, ST\_DOM\_STR *pstVal)
```

[Parameters]**Input parameters:**

```
ST\_DOM\_ATTR *pstAttr
```

The attribute whose value is to be set.

```
ST\_DOM\_STR *pstVal
```

The attribute value.

Output parameters:

None.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```
ST\_DOM\_ATTR *pstAttr;
```

```
ST\_DOM\_STR *pstVal;
```

```
ZINT iRet;
```

```
.....
```

```
/* Set the value of an attribute. */
```

```
iRet = Dom_AttrSetVal(pstAttr, pstVal);
```

3.2.38 Dom_AttrGetDQuote

Returns the double quote of an attribute.

```
ZBOOL Dom_AttrGetDQuote(ST\_DOM\_ATTR *pstAttr)
```

[Parameters]**Input parameters:**

```
ST\_DOM\_ATTR *pstAttr
```

The attribute.

Output parameters:

None.

[Return value]

Returns the double quote on success or ZFALSE on failure.

[Example]

```

ST\_DOM\_ATTR *pstAttr;
ZBOOL iBool;
.....
/* Return the double quote of an attribute. */
iBool = Dom_AttrGetDQuote(pstAttr);

```

3.2.39 Dom_AttrSetDQuote

Sets the double of an attribute.

```
ZINT Dom_AttrSetDQuote(ST\_DOM\_ATTR *pstAttr, ZBOOL bDQuote)
```

[Parameters]

Input parameters:

```
ST\_DOM\_ATTR *pstAttr
```

The attribute.

```
ZBOOL bDQuote
```

The double quote.

Output parameters:

None.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```

ST\_DOM\_ATTR *pstAttr;
ZBOOL bDQuote;
ZINT iRet;
.....
/* Set the double quote of an attribute. */
iRet = Dom_AttrSetDQuote(pstAttr, bDQuote);

```

3.2.40 Dom_ElemGetTagName

Retrieves the name of an element.

```
ZINT Dom_ElemGetTagName(ST\_DOM\_ELEM *pstElem,
ST\_DOM\_STR **ppstTagName)
```

[Parameters]

Input parameters:

[ST_DOM_ELEM](#) *pstElem

The element.

[ST_DOM_STR](#) **ppstTagName

This will point to the name of the element on success.

Output parameters:

[ST_DOM_STR](#) **ppstTagName

The element name.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```

ST\_DOM\_ELEM *pstElem;
ST\_DOM\_STR *pstTagName;
ZINT iRet;

.....

/* Retrieve the name of an element. */
iRet = Dom_ElemGetTagName(pstElem, &pstTagName);

```

3.2.41 Dom_ElemGetAttr

Retrieves an attribute value by name.

```

ZINT Dom_ElemGetAttr(ST\_DOM\_ELEM *pstElem, ST\_DOM\_STR *pstName,
                    ST\_DOM\_STR **ppstVal)

```

[Parameters]

Input parameters:

[ST_DOM_ELEM](#) *pstElem

The element on which the attribute is.

[ST_DOM_STR](#) *pstName

Name of the attribute whose value is to be retrieved.

[ST_DOM_STR](#) **ppstVal

This will point to the value of the attribute.

Output parameters:

[ST_DOM_STR](#) **pstVal

Pointer to the value of the attribute.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```
ST\_DOM\_ELEM *pstElem;
ST\_DOM\_STR *pstName;
ST\_DOM\_STR *pstVal;
ZINT iRet;

.....

/* Retrieve an attribute value by name. */
iRet = Dom_ElemGetAttr(pstElem, pstName, &pstVal);
```

3.2.42 Dom_ElemSetAttr

Adds a new attribute.

```
ZINT Dom_ElemSetAttr(ST\_DOM\_ELEM *pstElem, ST\_DOM\_STR *pstName,
                    ST\_DOM\_STR *pstVal)
```

[Parameters]

Input parameters:

[ST_DOM_ELEM](#) *pstElem

The element.

[ST_DOM_STR](#) *pstName

Name of the new attribute.

[ST_DOM_STR](#) *pstVal

Value of the new attribute.

Output parameters:

None.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```

ST_DOM_ELEM *pstElem;
ST_DOM_STR *pstName;
ST_DOM_STR *pstVal;
ZINT iRet;

.....

/* Add a new attribute. */
iRet = Dom_ElemSetAttr(pstElem, pstName, pstVal);

```

3.2.43 Dom_ElemRemAttr

Removes an attribute by name.

```
ZINT Dom_ElemRemAttr(ST_DOM_ELEM *pstElem, ST_DOM_STR *pstName)
```

[Parameters]

Input parameters:

```
ST_DOM_ELEM *pstElem
```

The element.

```
ST_DOM_STR *pstName
```

Name of the attribute to remove.

Output parameters:

None.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```

ST_DOM_ELEM *pstElem;
ST_DOM_STR *pstName;
ZINT iRet;

.....

/* Remove an attribute. */
iRet = Dom_ElemRemAttr(pstElem, pstName);

```

3.2.44 Dom_ElemGetAttrNode

Retrieves an attribute node by name.

```
ZINT Dom_ElemGetAttrNode(ST\_DOM\_ELEM *pstElem, ST\_DOM\_STR *pstName,  
                        ST\_DOM\_ATTR **ppstAttr)
```

[Parameters]

Input parameters:

[ST_DOM_ELEM](#) *pstElem

The element.

[ST_DOM_STR](#) *pstName

Name of the attribute.

[ST_DOM_ATTR](#) **ppstAttr

This will point to the attribute node.

Output parameters:

[ST_DOM_ATTR](#) **ppstAttr

Pointer to the attribute node.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```
ST\_DOM\_ELEM *pstElem;
```

```
ST\_DOM\_STR *pstName;
```

```
ST\_DOM\_ATTR *pstAttr;
```

```
ZINT iRet;
```

```
.....
```

```
/* Retrieve an attribute node by name. */
```

```
iRet = Dom_ElemGetAttrNode(pstElem, pstName, &pstAttr);
```

3.2.45 Dom_ElemSetAttrNode

Adds a new attribute.

```
ZINT Dom_ElemSetAttrNode(ST\_DOM\_ELEM *pstElem,  
                        ST\_DOM\_ATTR *pstNewAttr)
```

[Parameters]

Input parameters:

[ST_DOM_ELEM](#) *pstElem

The element.

[ST_DOM_ATTR](#) *pstNewAttr

The node to add to the attribute list.

Output parameters:

None.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```
ST\_DOM\_ELEM *pstElem;
ST\_DOM\_ATTR *pstNewAttr;
ZINT iRet;
.....
/* Add a new attribute. */
iRet = Dom_ElemSetAttrNode(pstElem, pstNewAttr);
```

3.2.46 Dom_ElemRemAttrNode

```
ZINT Dom_ElemRemAttrNode(ST\_DOM\_ELEM *pstElem,
ST\_DOM\_ATTR *pstOldAttr)
```

[Parameters]

Input parameters:

[ST_DOM_ELEM](#) *pstElem

The element.

[ST_DOM_ATTR](#) *pstNewAttr

The node to remove from the attribute list.

Output parameters:

None.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```

ST_DOM_ELEM *pstElem;
ST_DOM_ATTR *pstNewAttr;
ZINT iRet;
.....
/* Remove a new attribute. */
iRet = Dom_ElemRemAttrNode(pstElem, pstNewAttr);

```

3.2.47 Dom_ElemGetElemsByTagName

Retrieves a node list of all descendant elements with a given tag name, in the order in which the elements would be encountered in a preorder traversal of the Element tree.

```

ZINT Dom_ElemGetElemsByTagName(ST_DOM_ELEM *pstElem,
                               ST_DOM_STR *pstName, ST_DOM_NODE_LST **ppstNodeLst)

```

[Parameters]

Input parameters:

[ST_DOM_ELEM](#) *pstElem

The element.

[ST_DOM_STR](#) *pstName

Name of the element.

ST_DOM_NODE_LST **ppstNodeLst

Pointer which will point to the list.

Output parameters:

ST_DOM_NODE_LST **ppstNodeLst

The node list.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```

ST_DOM_ELEM *pstElem;
ST_DOM_STR *pstName;
ST_DOM_NODE_LST *pstNodeLst;
ZINT iRet;
.....
/* Retrieve a node list of all descendant elements by a given tag name. */
iRet = Dom_ElemGetElemsByTagName(pstElem, pstName, &pstNodeLst);

```

3.2.48 Dom_ElemGetSingleElem

Retrieves a child element by name.

```
ZINT Dom_ElemGetSingleElem(ST\_DOM\_ELEM *pstElem, ST\_DOM\_STR *pstName,  
                           ST\_DOM\_ELEM **ppstChild)
```

[Parameters]

Input parameters:

[ST_DOM_ELEM](#) *pstElem

The element.

[ST_DOM_STR](#) *pstName

Name of the child element.

[ST_DOM_ELEM](#) **ppstChild

This will point to the child element.

Output parameters:

[ST_DOM_ELEM](#) **ppstChild

The child element.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```
ST\_DOM\_ELEM *pstElem;  
ST\_DOM\_STR *pstName;  
ST\_DOM\_ELEM *pstChild;  
ZINT iRet;  
.....  
/* Retrieve a child element by name. */  
iRet = Dom_ElemGetSingleElem(pstElem, pstName, &pstChild);
```

3.2.49 Dom_ElemHasAttr

Checks if a given element has any attributes by name.

```
ZBOOL Dom_ElemHasAttr(ST\_DOM\_ELEM *pstElem, ST\_DOM\_STR *pstName)
```

[Parameters]

Input parameters:

[ST_DOM_ELEM](#) *pstElem

The element.

[ST_DOM_STR](#) *pstName

Name of the element.

Output parameters:

None.

[Return value]

Returns ZTRUE if there is any attribute of ZFALSE if there is not.

[Example]

```
ST\_DOM\_ELEM *pstElem;
ST\_DOM\_STR *pstName;
ZBOOL iBool;
.....
/* Check if a given element has any attributes. */
iBool = Dom_ElemHasAttr(pstElem, pstName);
```

3.2.50 Dom_DocCreateElem

Creates an element of the type specified.

```
ZINT Dom_DocCreateElem(ST\_DOM\_DOC *pstDoc, ST\_DOM\_STR *pstTagName,
ST\_DOM\_ELEM **ppstElem)
```

[Parameters]

Input parameters:

[ST_DOM_DOC](#) *pstDoc

The document.

[ST_DOM_STR](#) *pstTagName

The name of the element type to instantiate.

[ST_DOM_ELEM](#) **ppstElem

This will point to the element to create on success.

Output parameters:

[ST_DOM_ELEM](#) **ppstElem

Pointer to the newly created element.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```

ST\_DOM\_DOC *pstDoc;
ST\_DOM\_STR *pstTagName;
ST\_DOM\_ELEM *pstElem;
ZINT iRet;
.....
/* Create a new element. */
iRet = Dom_DocCreateElem(pstDoc, pstTagName, &pstElem);

```

3.2.51 Dom_DocCreateDocFrag

Creates an empty DocumentFragment object.

```

ZINT Dom_DocCreateDocFrag(ST\_DOM\_DOC *pstDoc,
                          ST\_DOM\_DFRAG **ppstDocFrag)

```

[Parameters]

Input parameters:

[ST_DOM_DOC](#) *pstDoc
The document.

[ST_DOM_DFRAG](#) **ppstDocFrag
This will point to the DocumentFragment to create on success.

Output parameters:

[ST_DOM_DFRAG](#) **ppstDocFrag
Pointer to the newly created DocumentFragment object.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```

ST\_DOM\_DOC *pstDoc;
ST\_DOM\_DFRAG *pstDocFrag;
ZINT iRet;
.....
/* Create an empty DocumentFragment object. */
iRet = Dom_DocCreateDocFrag(pstDoc, &pstDocFrag);

```

3.2.52 Dom_DocCreateTxt

Creates a text node given the specified string.

```
ZINT Dom_DocCreateTxt(ST\_DOM\_DOC *pstDoc, ST\_DOM\_STR *pstData,
                     ST\_DOM\_TXT **ppstTxt)
```

[Parameters]

Input parameters:

[ST_DOM_DOC](#) *pstDoc

The document.

[ST_DOM_STR](#) *pstData

The string for the node.

[ST_DOM_TXT](#) **ppstTxt

This will point to the text node on success.

Output parameters:

[ST_DOM_TXT](#) **ppstTxt

Pointer to the text node.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```
ST\_DOM\_DOC *pstDoc;
ST\_DOM\_STR *pstData;
ST\_DOM\_TXT *pstTxt;
ZINT iRet;

.....
/* Create a text node. */
iRet = Dom_DocCreateTxt(pstDoc, pstData, &pstTxt);
```

3.2.53 Dom_DocCreateCommt

Creates a comment node given the specified string.

```
ZINT Dom_DocCreateCommt(ST\_DOM\_DOC *pstDoc, ST\_DOM\_STR *pstData,
                       ST\_DOM\_COMMT **ppstCommt)
```

[Parameters]

Input parameters:

[ST_DOM_DOC](#) *pstDoc

The document.

[ST_DOM_STR](#) *pstData

The string for the node.

[ST_DOM_COMMT](#) **ppstCommt

This will point to the comment node on success.

Output parameters:

[ST_DOM_COMMT](#) **ppstCommt

Pointer to the comment node.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```
ST\_DOM\_DOC *pstDoc;
ST\_DOM\_STR *pstData;
ST\_DOM\_COMMT *pstCommt;
ZINT iRet;
.....
/* Create a comment node. */
iRet = Dom_DocCreateCommt(pstDoc, pstData, &pstCommt);
```

3.2.54 Dom_DocCreateCDATAsec

Creates a CDATASection node whose value is the given string.

```
ZINT Dom_DocCreateCDATAsec(ST\_DOM\_DOC *pstDoc, ST\_DOM\_STR *pstData,
ST\_DOM\_CDATA\_SEC **ppstCDATAsec)
```

[Parameters]

Input parameters:

[ST_DOM_DOC](#) *pstDoc

The document.

[ST_DOM_STR](#) *pstData

The string for the node.

[ST_DOM_CDATA_SEC](#) **ppstCDATAsec

This will point to the CDATASection node on success.

Output parameters:

[ST_DOM_CDATA_SEC](#) **ppstCDataSec
 Pointer to the CDATASection node.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```
ST\_DOM\_DOC *pstDoc;
ST\_DOM\_STR *pstData;
ST\_DOM\_CDATA\_SEC *pstCDataSec;
ZINT iRet;
.....
/* Create a CDATASection node. */
iRet = Dom_DocCreateCDataSec(pstDoc, pstData, &pstCDataSec);
```

3.2.55 Dom_DocCreatePi

Creates a ProcessingInstruction node given the specified name and data.

```
ZINT Dom_DocCreatePi(ST\_DOM\_DOC *pstDoc, ST\_DOM\_STR *pstTarget,
ST\_DOM\_STR *pstData, ST\_DOM\_PI **ppstPi)
```

[Parameters]

Input parameters:

[ST_DOM_DOC](#) *pstDoc
 The document.

[ST_DOM_STR](#) *pstTarget
 The target part of the processing instruction.

[ST_DOM_STR](#) *pstData
 The data for the node.

[ST_DOM_PI](#) **ppstPi
 This will point to the ProcessingInstruction node on success.

Output parameters:

[ST_DOM_PI](#) **ppstPi
 Pointer to the ProcessingInstruction node.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```

    ST\_DOM\_DOC *pstDoc;
    ST\_DOM\_STR *pstData;
    ST\_DOM\_STR *pstTarget
    ST\_DOM\_PI *pstPi;
    ZINT iRet;
    .....
    /* Create a ProcessingInstruction node. */
    iRet = Dom_DocCreatePi(pstDoc, pstTarget , pstData, &pstPi);

```

3.2.56 Dom_DocCreateAttr

Creates an attribute of the given name.

```

    ZINT Dom_DocCreateAttr(ST\_DOM\_DOC *pstDoc, ST\_DOM\_STR *pstName,
                          ST\_DOM\_ATTR **ppstAttr)

```

[Parameters]

Input parameters:

[ST_DOM_DOC](#) *pstDoc
The document.

[ST_DOM_STR](#) *pstName
Name for the node.

[ST_DOM_ATTR](#) **ppstAttr
This will point to the attribute on success.

Output parameters:

[ST_DOM_ATTR](#) **ppstAttr
Pointer to the attribute node.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```

    ST\_DOM\_DOC *pstDoc;
    ST\_DOM\_ATTR *pstAttr;
    ST\_DOM\_STR *pstName;
    ZINT iRet;
    .....
    /* Create an attribute of a given name. */
    iRet = Dom_DocCreateAttr(pstDoc, pstName, &pstName);

```

3.2.57 Dom_DocGetElemsByTagName

Retrieves a node list of all the elements within a given tag name in the order in which the elements would be encountered in a preorder traversal of the Document tree.

```
ZINT Dom_DocGetElemsByTagName(ST_DOM_DOC *pstDoc,
                               ST_DOM_STR *pstTagName, ST_DOM_NODE_LST **ppstNodeLst)
```

[Parameters]

Input parameters:

ST_DOM_DOC *pstDoc

The document.

ST_DOM_STR *pstTagName

The name of the tag to match on. The special value "*" matches all tags.

ST_DOM_NODE_LST **ppstNodeLst

This will point to the node list on success.

Output parameters:

ST_DOM_NODE_LST **ppstNodeLst

Pointer to the node list.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```
ST_DOM_DOC *pstDoc;
ST_DOM_STR *pstTagName;
ST_DOM_NODE_LST *pstNodeLst
ZINT iRet;
.....
/* Retrieve a node list of all elements within a given tag name. */
iRet = Dom_DocGetElemsByTagName(pstDoc, pstTagName, &pstNodeLst);
```

3.2.58 Dom_DocGetElem

Retrieves the root element of a document.

```
ZINT Dom_DocGetElem(ST_DOM_DOC *pstDoc, ST_DOM_ELEM **ppstElem)
```

[Parameters]

Input parameters:

[ST_DOM_DOC](#) *pstDoc

The document.

[ST_DOM_ELEM](#) **ppstElem

This will point to the root element.

Output parameters:

[ST_DOM_ELEM](#) **ppstElem

Pointer to the root element.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```
ST\_DOM\_DOC *pstDoc;
ST\_DOM\_ELEM *pstElem;
ZINT iRet;
.....
/* Retrieve the root element of a document. */
iRet = Dom_DocGetElem(pstDoc, &pstElem);
```

3.2.59 Dom_DocLoad

Loads an XML document from the specified location.

```
ZINT Dom_DocLoad(ST\_DOM\_DOC *pstDoc, ZCHAR *pcXmlUri)
```

[Parameters]

Input parameters:

[ST_DOM_DOC](#) *pstDoc

The document to load.

ZCHAR *pcXmlUri

A string containing a URL that specifies the location of the XML document.

Output parameters:

None.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```

ST\_DOM\_DOC *pstDoc;
ZCHAR *pcXmlUri;
ZINT iRet;
.....
/* Load an XML document from the specified location. */
iRet = Dom_DocLoad(pstDoc, pcXmlUri);

```

3.2.60 Dom_DocLoadXml

Loads an XML document using the supplied string.

```
ZINT Dom_DocLoadXml(ST\_DOM\_DOC *pstDoc, ST\_DOM\_STR *pstXmlStr)
```

[Parameters]

Input parameters:

[ST_DOM_DOC](#) *pstDoc
The document to load.

[ST_DOM_STR](#) *pstXmlStr
A string containing the XML string to load into this XML document object. This string can contain an entire XML document or a well-formed fragment.

Output parameters:

None.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```

ST\_DOM\_DOC *pstDoc;
ST\_DOM\_STR *pstXmlStr;
ZINT iRet;
.....
/* Load an XML document using the supplied string. */
iRet = Dom_DocLoadXml(pstDoc, pstXmlStr);

```

3.2.61 Dom_DocSave

Saves an XML document to the specified location.

```
ZINT Dom_DocSave(ST\_DOM\_DOC *pstDoc, ZCHAR *pcXmlUri)
```

[Parameters]

Input parameters:

[ST_DOM_DOC](#) *pstDoc

The document to save.

ZCHAR *pcXmlUri

The specified location.

Output parameters:

None.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```
ST\_DOM\_DOC *pstDoc;
ZCHAR *pcXmlUri;
ZINT iRet;
.....
/* Save an XML docment to the specified location. */
iRet = Dom_DocSave(pstDoc, pcXmlUri);
```

3.2.62 Dom_DocSaveXml

Saves an XML document to the supplied message buffer.

```
ZINT Dom_DocSaveXml(ST\_DOM\_DOC *pstDoc, ST\_ZOS\_DBUF *pstMsgBuf)
```

[Parameters]**Input parameters:**

[ST_DOM_DOC](#) *pstDoc

The document to save.

[ST_ZOS_DBUF](#) *pstMsgBuf

The supplied message buffer.

Output parameters:

None.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```

ST\_DOM\_DOC *pstDoc;
ST\_ZOS\_DBUF *pstMsgBuf;
ZINT iRet;
.....
/* Save an XML document using supplied message buffer. */
iRet = Dom_DocSaveXml(pstDoc, pstMsgBuf);

```

3.2.63 Dom_DocCreate

Creates a document.

```
ZINT Dom_DocCreate(ZUCHAR ucEncodingType, ST\_DOM\_DOC **ppstDoc);
```

[Parameters]

Input parameters:

```
ZUCHAR ucEncodingType
Type of the document to create.
```

```
ST\_DOM\_DOC **ppstDoc
This will point to the document on success.
```

Output parameters:

```
ST\_DOM\_DOC **ppstDoc
Pointer to the document on success.
```

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```

ZUCHAR ucEncodingType
ST\_DOM\_DOC *pstDoc;
ZINT iRet;
.....
/* Create a document with a specified type. */
iRet = Dom_DocCreate(ucEncodingType, &pstDoc);

```

3.2.64 Dom_DocDelete

Deletes a document.

```
ZINT Dom_DocDelete(ST\_DOM\_DOC *pstDoc);
```

[Parameters]

Input parameters:

[ST_DOM_DOC](#) *pstDoc

The document to delete.

Output parameters:

None.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```
ST\_DOM\_DOC *pstDoc;
ZINT iRet;
.....
/* Delete a document. */
iRet = Dom_DocDelete(pstDoc);
```

3.2.65 Dom_DocImportNode

Imports the specified node to a document.

```
ZINT Dom_DocImportNode(ST\_DOM\_DOC *pstDoc,
                       ST\_DOM\_NODE *pstImportedNode, ZBOOL bDeep)
```

[Parameters]

Input parameters:

[ST_DOM_DOC](#) *pstDoc

The document.

[ST_DOM_NODE](#) *pstImportedNode

The node to import.

ZBOOL bDeep

If it is ZTRUE, the child tree of this node will be cloned; If it is not, the child tree will not be cloned.

Output parameters:

None.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```

ST_DOM_DOC *pstDoc;
ST_DOM_NODE *pstImportedNode;
ZBOOL bDeep;
ZINT iRet;
.....
/* Import a node. */
iRet = Dom_DocImportNode(pstDoc, pstImportedNode, bDeep);

```

3.3 DOM Utility Interfaces

Here are some DOM utility interfaces for accessing XML data.

3.3.1 Dom_SecGetKey

Gets the first section key element of a specific name.

```

ZINT Dom_SecGetKey(ST_DOM_ELEM *pstSecElem, ZCHAR *pcKeyName,
                  ST_DOM_ELEM **ppstKeyElem);

```

[Parameters]

Input parameters:

```

ST_DOM_ELEM *pstSecElem
The element that contains the key element.

ZCHAR *pcKeyName
The key element name.

```

Output parameters:

```

ST_DOM_ELEM **ppstKeyElem
The key element on success.

```

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```

ST_DOM_ELEM *pstSecElem;
ZCHAR *pcKeyName;
ST_DOM_ELEM *pstKeyElem;
ZINT iRet;
.....
/* Get the first section key element of a specific name. */
iRet = Dom_SecGetKey(pstSecElem, pcKeyName,
                    &pstKeyElem);

```

3.3.2 Dom_SecGetKeyX

Gets the indexth section key element of a specific name.

```
ZINT Dom_SecGetKeyX(ST_DOM_ELEM *pstSecElem, ZCHAR *pcKeyName,
                    ZINT iIdx, ST_DOM_ELEM **ppstKeyElem);
```

[Parameters]

Input parameters:

[ST_DOM_ELEM](#) *pstSecElem

The element that contains the key element.

ZCHAR *pcKeyName

The key element name.

ZINT iIdx

The index.

Output parameters:

[ST_DOM_ELEM](#) **ppstKeyElem

The key element on success.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```
ST\_DOM\_ELEM *pstSecElem;
ZCHAR *pcKeyName;
ZINT iIdx;
ST\_DOM\_ELEM *pstKeyElem;
ZINT iRet;
.....
/* Get the indexth section key element of a specific name. */
iRet = Dom_SecGetKeyX(pstSecElem, pcKeyName,
                    ZINT iIdx, &pstKeyElem);
```

3.3.3 Dom_SecGetKeyS

Gets all the section elements of a specific name.

```
ZINT Dom_SecGetKeyS(ST_DOM_ELEM *pstSecElem, ZCHAR *pcKeyName,
                   ST_DOM_NODE_LST **ppstNodeLst);
```

[Parameters]**Input parameters:**

[ST_DOM_ELEM](#) *pstSecElem

The element that contains the elements to get.

ZCHAR *pcKeyName

The element name.

Output parameters:

[ST_DOM_NODE_LST](#) **ppstNodeLst

List of the elements of a specific name on success.

[Return value]

Returns ZOK on success or ZFAILED on failure on success.

[Example]

```
ST\_DOM\_ELEM *pstSecElem;
```

```
ZCHAR *pcKeyName;
```

```
ST\_DOM\_NODE\_LST *ppstNodeLst;
```

```
ZINT iRet;
```

```
.....
```

```
/* Get all the section elements of a specific name. */
```

```
iRet = ZINT Dom_SecGetKeyS(pstSecElem, pcKeyName, ppstNodeLst);
```

3.3.4 Dom_SecGetVal

Gets the first child node of a key element of a specific name.

```
ZINT Dom_SecGetVal(ST\_DOM\_ELEM *pstSecElem, ZCHAR *pcKeyName,
ST\_DOM\_NODE **ppstNode);
```

[Parameters]**Input parameters:**

[ST_DOM_ELEM](#) *pstSecElem

The element that contains the key element.

ZCHAR *pcKeyName

The key element name.

Output parameters:

[ST_DOM_NODE](#) **ppstNode

The first child node of the key element on success.

[Return value]

Returns ZOK on success or ZFAILED on failure on success.

[Example]

```

ST_DOM_ELEM *pstSecElem;
ZCHAR *pcKeyName;
ST_DOM_NODE *pstNode;
ZINT iRet;
.....
/* Get the first child node of a key element of a specific name. *
iRet = Dom_SecGetVal(pstSecElem, pcKeyName, &pstNode);

```

3.3.5 Dom_SecGetValX

Gets the first child node of the indexth key element of a specific name.

```

ZINT Dom_SecGetValX(ST_DOM_ELEM *pstSecElem, ZCHAR *pcKeyName,
                   ZINT iIdx, ST_DOM_NODE **ppstNode);

```

[Parameters]

Input parameters:

[ST_DOM_ELEM](#) *pstSecElem
The element that contains the key element.

ZCHAR *pcKeyName
The key element name.

ZINT iIdx
Then index.

Output parameters:

[ST_DOM_NODE](#) **ppstNode
The first child node of the indexth key element on success.

[Return value]

Returns ZOK on success or ZFAILED on failure on success.

[Example]

```

ST_DOM_ELEM *pstSecElem;
ZCHAR *pcKeyName;
ZINT iIdx;
ST_DOM_NODE *pstNode;
ZINT iRet;
.....
/* Get the first child node of the indexth key element of a specific name.*/
iRet = Dom_SecGetValX(pstSecElem, pcKeyName, iIdx, &pstNode);

```

3.3.6 Dom_SecGetStr

Gets a string contained in an element.

```

ZINT Dom_SecGetStr(ST_DOM_ELEM *pstSecElem, ZCHAR *pcKeyName,
                  ZCHAR *pcDftStr, ZCHAR *pcDstStr, ZUSHORT wDstSize);

```

[Parameters]

Input parameters:

[ST_DOM_ELEM](#) *pstSecElem

The element that contains the string.

ZCHAR *pcKeyName

The name of a key element contained in the element pstSecElem. The name can be ZNULL.

ZCHAR *pcDftStr

The default string.

ZUSHORT wDstSize

The maximum size of the string to get.

Output parameters:

ZCHAR *pcDstStr

The string on success.

[Return value]

Returns ZOK on success or ZFAILED on failure on success.

[Example]

```

    ST\_DOM\_ELEM *pstSecElem;
    ZCHAR *pcKeyName;
    ZCHAR *pcDftStr;
    ZUSHORT wDstSize;
    ZCHAR *pcDstStr;
    ZINT iRet;
    .....
    /* Get a string contained in an element. */
    iRet = Dom_SecGetStr(pstSecElem, pcKeyName, pcDftStr, pcDstStr, wDstSize);

```

3.3.7 Dom_SecGetUI

Gets the key value of unsigned long.

```

    ZINT Dom_SecGetUI(ST\_DOM\_ELEM *pstSecElem, ZCHAR *pcKeyName,
                     ZULONG dwDftVal, ZULONG *pdwVal);

```

[Parameters]

Input parameters:

```

    ST\_DOM\_ELEM *pstSecElem
    The element that contains the key element.

```

```

    ZCHAR *pcKeyName
    The key element name.

```

```

    ZULONG dwDftVal
    The default value.

```

Output parameters:

```

    ZULONG *pdwVal
    The value.

```

[Return value]

Returns ZOK.

[Example]

```

    ST\_DOM\_ELEM *pstSecElem;
    ZCHAR *pcKeyName;
    ZULONG dwDftVal;
    ZULONG dwVal;
    .....
    /* Gets the key value of unsigned long. */
    Dom_SecGetUI(pstSecElem, pcKeyName, dwDftVal, &dwVal);

```

3.3.8 Dom_SecGetUc

Gets the key value of unsigned char.

```
ZINT Dom_SecGetUc(ST_DOM_ELEM *pstSecElem, ZCHAR *pcKeyName,
                  ZUCHAR ucDftVal, ZUCHAR *pucVal);
```

[Parameters]

Input parameters:

[ST_DOM_ELEM](#) *pstSecElem

The element that contains the key element.

ZCHAR *pcKeyName

The key element name.

ZUCHAR ucDftVal

The default value.

Output parameters:

ZUCHAR *pucVal

The key value.

[Return value]

Returns ZOK.

[Example]

```
ST\_DOM\_ELEM *pstSecElem;
ZCHAR *pcKeyName;
ZUCHAR ucDftVal;
ZUCHAR ucVal;
.....
/* Gets the key value of unsigned char. */
Dom_SecGetUc(pstSecElem, pcKeyName, ucDftVal, &ucVal);
```

3.3.9 Dom_SecGetUs

Gets the key value of unsigned short.

```
ZINT Dom_SecGetUs(ST_DOM_ELEM *pstSecElem, ZCHAR *pcKeyName,
                  ZUSHORT wDftVal, ZUSHORT *pwVal);
```

[Parameters]

Input parameters:

[ST_DOM_ELEM](#) *pstSecElem

The element that contains the key element.

ZCHAR *pcKeyName

The key element name.

ZUSHORT wDftVal

The default value.

Output parameters:

ZUSHORT *pwVal

The key value.

[Return value]

Returns ZOK.

[Example]

```

ST\_DOM\_ELEM *pstSecElem;
ZCHAR *pcKeyName;
ZUSHORT wDftVal;
ZUSHORT wVal;

.....

/* Get the key value of unsigned short. */
Dom_SecGetUs(pstSecElem, pcKeyName, wDftVal, &wVal);

```

3.3.10 Dom_SecGetBool

Gets the key value of Boolean.

```

ZINT Dom_SecGetBool(ST\_DOM\_ELEM *pstSecElem, ZCHAR *pcKeyName,
                    ZBOOL bDftTrueFalse, ZBOOL *pbTrueFalse);

```

[Parameters]

Input parameters:

[ST_DOM_ELEM](#) *pstSecElem

The element that contains the key element.

ZCHAR *pcKeyName

The key element name.

ZBOOL bDftTrueFalse

The default value.

Output parameters:

```
ZBOOL *pbTrueFalse
The key value.
```

[Return value]

Returns ZOK.

[Example]

```
ST\_DOM\_ELEM *pstSecElem;
ZCHAR *pcKeyName;
ZBOOL bDftTrueFalse;
ZBOOL bTrueFalse;
.....
/* Get the key value of Boolean. */
Dom_SecGetBool(pstSecElem, pcKeyName, bDftTrueFalse, &bTrueFalse);
```

3.3.11 Dom_KeyGetVal

Gets the first child node of an element.

```
ZINT Dom_KeyGetVal(ST\_DOM\_ELEM *pstKeyElem, ST\_DOM\_NODE **ppstNode);
```

[Parameters]**Input parameters:**

```
ST\_DOM\_ELEM *pstKeyElem
The element.
```

Output parameters:

```
ST\_DOM\_NODE **ppstNode
The first child node on success.
```

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```
ST\_DOM\_ELEM *pstKeyElem;
ST\_DOM\_NODE *pstNode;
ZINT iRet;
.....
/* Get the first child node of an element. */
iRet = Dom_KeyGetVal(pstKeyElem, &pstNode);
```

3.3.12 Dom_KeyGetStr

Gets the string contained in an element.

```
ZINT Dom_KeyGetStr(ST\_DOM\_ELEM *pstKeyElem, ZCHAR *pcDftStr,
                  ZCHAR *pcDstStr, ZUSHORT wDstSize);
```

[Parameters]

Input parameters:

[ST_DOM_ELEM](#) *pstKeyElem

The element.

ZCHAR *pcDftStr

The default string.

ZUSHORT wDstSize

The maximum size of the string.

Output parameters:

ZCHAR *pcDstStr

The string contained in the element.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```
ST\_DOM\_ELEM *pstKeyElem;
ZCHAR *pcDftStr;
ZCHAR *pcDstStr;
ZUSHORT wDstSize;
ZINT iRet;
.....
/* Get the string contained in an element. */
iRet = Dom_KeyGetStr(pstKeyElem, pcDftStr, pcDstStr, wDstSize);
```

3.3.13 Dom_KeyGetUl

Gets the value of unsigned long contained in a key element.

```
ZINT Dom_KeyGetUl(ST\_DOM\_ELEM *pstKeyElem, ZULONG dwDftVal,
                  ZULONG *pdwVal);
```

[Parameters]

Input parameters:

```
ST_DOM_ELEM *pstKeyElem
```

The key element.

```
ZULONG dwDftVal
```

The default value.

Output parameters:

```
ZULONG *pdwVal
```

The value of unsigned long.

[Return value]

Returns ZOK.

[Example]

```
ST_DOM_ELEM *pstKeyElem;
```

```
ULONG dwDftVal;
```

```
ZULONG dwVal;
```

```
.....
```

```
/* Get the value of unsigned long contained in a key element. */
```

```
Dom_KeyGetUl(pstKeyElem, dwDftVal, &dwVal);
```

3.3.14 Dom_KeyGetUs

Gets the value of unsigned short contained in a key element.

```
ZINT Dom_KeyGetUs(ST_DOM_ELEM *pstKeyElem, ZUSHORT wDftVal,  
                  ZUSHORT *pwVal);
```

[Parameters]

Input parameters:

```
ST_DOM_ELEM *pstKeyElem
```

The key element.

```
ZUSHORT wDftVal
```

The default value.

Output parameters:

```
ZUSHORT *pwVal
```

The value of unsigned short.

[Return value]

Returns ZOK.

[Example]

```

ST_DOM_ELEM *pstKeyElem;
ZUSHORT wDftVal;
ZUSHORT wVal;
.....
/* Get the value of unsigned short contained in a key element. */
Dom_KeyGetUs (pstKeyElem, wDftVal, &wVal);

```

3.3.15 Dom_KeyGetUc

Gets the value of unsigned char contained in the key element.

```

ZINT Dom_KeyGetUc (ST_DOM_ELEM *pstKeyElem, ZUCHAR ucDftVal,
                  ZUCHAR *pucVal);

```

[Parameters]

Input parameters:

```

ST_DOM_ELEM *pstKeyElem

```

The key element.

```

ZUCHAR ucDftVal

```

The default value.

Output parameters:

```

ZUCHAR *pucVal

```

The value of unsigned char.

[Return value]

Returns ZOK.

[Example]

```

ST_DOM_ELEM *pstKeyElem;
ZUCHAR ucDftVal;
ZUCHAR ucVal;
.....
/* Get the value of unsigned char contained in the key element. */
Dom_KeyGetUc (pstKeyElem, ucDftVal, &ucVal);

```

3.3.16 Dom_KeyGetBool

Gets the value of Boolean.

```

ZINT Dom_KeyGetBool (ST_DOM_ELEM *pstKeyElem, ZBOOL bDftTrueFalse,
                    ZBOOL *pbTrueFalse);

```

[Parameters]**Input parameters:**

[ST_DOM_ELEM](#) *pstKeyElem

The key element.

ZBOOL bDftTrueFalse

The default value.

Output parameters:

ZBOOL *pbTrueFalse

The Boolean value.

[Return value]

Returns ZOK.

[Example]

```
ST\_DOM\_ELEM *pstKeyElem;
```

```
ZBOOL bDftTrueFalse;
```

```
ZBOOL bTrueFalse;
```

```
.....
```

```
/* Get the value of Boolean. */
```

```
Dom_KeyGetBool(pstKeyElem, bDftTrueFalse, &bTrueFalse);
```

3.3.17 Dom_KeyGetAttrStr

Gets the attribute string contained in the key element.

```
ZINT Dom_KeyGetAttrStr(ST\_DOM\_ELEM *pstKeyElem, ZCHAR *pcName,
                       ZCHAR *pcVal, ZUSHORT wMaxSize);
```

[Parameters]**Input parameters:**

[ST_DOM_ELEM](#) *pstKeyElem

The key element.

ZCHAR *pcName

Name of the node where the string is located.

ZUSHORT wMaxSize

The maximum size of the string to get.

Output parameters:

ZCHAR *pcVal
The attribute string.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST_DOM_ELEM *pstKeyElem;
ZCHAR *pcName;
ZUSHORT wMaxSize;
ZCHAR *pcVal;
ZINT iRet;
.....
/* Get the attribute string contained in the key element. */
iRet = Dom_KeyGetAttrStr(pstKeyElem, pcName, pcVal, wMaxSize);

```

3.3.18 Dom_KeyGetAttrUl

Gets the attribute of unsigned long contained in a key element.

```

ZINT Dom_KeyGetAttrUl(ST_DOM_ELEM *pstKeyElem, ZCHAR *pcName,
                     ZULONG dwDftVal, ZULONG *pdwVal);

```

[Parameters]

Input parameters:

[ST_DOM_ELEM](#) *pstKeyElem
The key element.

ZCHAR *pcName
Name of the node where the attribute is located.

ZULONG dwDftVal
The default value.

Output parameters:

ZULONG *pdwVal
The attribute value.

[Return value]

Returns ZOK.

[Example]

```

ST\_DOM\_ELEM *pstKeyElem;
ZCHAR *pcName;
ZULONG dwDftVal;
ZULONG dwVal;
.....
/* Get the attribute of unsigned long contained in a key element. */
Dom_KeyGetAttrUl(pstKeyElem, pcName, dwDftVal, &dwVal);

```

3.3.19 Dom_KeyGetAttrUs

Gets the attribute of unsigned short contained in a key element.

```

ZINT Dom_KeyGetAttrUs(ST\_DOM\_ELEM *pstKeyElem, ZCHAR *pcName,
                     ZUSHORT wDftVal, ZUSHORT *pwVal);

```

[Parameters]

Input parameters:

```
ST\_DOM\_ELEM *pstKeyElem
```

The key element.

```
ZCHAR *pcName
```

Name of the node where the attribute is located.

```
ZUSHORT wDftVal
```

The default value.

Output parameters:

```
ZUSHORT *pwVal
```

The attribte value.

[Return value]

Returns ZOK.

[Example]

```

ST\_DOM\_ELEM *pstKeyElem;
ZCHAR *pcName;
ZUSHORT wDftVal;
ZUSHORT wVal;
.....
/* Get the attribute of unsigned short contained in a key element. */
Dom_KeyGetAttrUs(pstKeyElem, pcName, wDftVal, &wVal);

```

3.3.20 Dom_KeyGetAttrUc

Gets the attribute of unsigned char contained in a key element.

```
ZINT Dom_KeyGetAttrUc(ST_DOM_ELEM *pstKeyElem, ZCHAR *pcName,  
                     ZUCHAR ucDftVal, ZUCHAR *pucVal);
```

[Parameters]

Input parameters:

[ST_DOM_ELEM](#) *pstKeyElem
The key element.

ZCHAR *pcName
Name of the node where the attribute is located.

ZUCHAR ucDftVal
The default value.

Output parameters:

ZUCHAR *pucVal
The attribute value.

[Return value]

Returns ZOK.

[Example]

```
ST\_DOM\_ELEM *pstKeyElem;  
ZCHAR *pcName;  
ZUCHAR ucDftVal;  
ZUCHAR ucVal;  
.....  
/* Get the attribute of unsigned char contained in a key element. */  
Dom_KeyGetAttrUc(pstKeyElem, pcName, ucDftVal, &ucVal);
```

3.3.21 Dom_DocPutRoot

Creates a root element for a document tree.

```
ZINT Dom_DocPutRoot(ST_DOM_DOC *pstDoc, ZCHAR *pcRootName,  
                   ST\_DOM\_ELEM **ppstRootElem);
```

[Parameters]

Input parameters:

[ST_DOM_DOC](#) *pstDoc
Document structure.

ZCHAR *pcRootName
Name of the root element to create.

Output parameters:

[ST_DOM_ELEM](#) **ppstRootElem
The newly created root element.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```
ST\_DOM\_DOC *pstDoc;
ZCHAR *pcRootName;
ST\_DOM\_ELEM *pstRootElem;
ZINT iRet;
.....
/* Create a root element for a document tree. */
iRet = Dom_DocPutRoot(pstDoc, pcRootName, &pstRootElem);
```

3.3.22 Dom_SecPutKey

Creates a key element and put it into another element.

```
ZINT Dom_SecPutKey(ST\_DOM\_ELEM *pstSecElem, ZCHAR *pcKeyName,
ST\_DOM\_ELEM **ppstKeyElem);
```

[Parameters]

Input parameters:

[ST_DOM_ELEM](#) *pstSecElem
The element that shall contain the key element to create.

ZCHAR *pcKeyName
Name of the key element.

Output parameters:

[ST_DOM_ELEM](#) **ppstKeyElem
The newly created key element on success.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST_DOM_ELEM *pstSecElem;
ZCHAR *pcKeyName;
ST_DOM_ELEM *pstKeyElem;
ZINT iRet;
.....
/* Create a key element and put it into another element. */
iRet = Dom_SecPutKey(pstSecElem, pcKeyName, &pstKeyElem);

```

3.3.23 Dom_SecSetStr

Sets the value of the string contained in a key element.

```

ZINT Dom_SecSetStr(ST_DOM_ELEM *pstSecElem, ZCHAR *pcKeyName,
                  ZCHAR *pcValStr);

```

[Parameters]**Input parameters:**

```

ST_DOM_ELEM *pstSecElem
The element that contains the key element.

ZCHAR *pcKeyName
Name of the key element.

ZCHAR *pcValStr
The string value.

```

Output parameters:

None.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST_DOM_ELEM *pstSecElem;
ZCHAR *pcKeyName;
ZCHAR *pcValStr;
ZINT iRet;
.....
/* Set the value of the string contained in a key element. */
iRet = Dom_SecSetStr(pstSecElem, pcKeyName, pcValStr);

```

3.3.24 Dom_SecSetUl

Sets the key value of unsigned long of a key element.

```
ZINT Dom_SecSetUl(ST_DOM_ELEM *pstSecElem, ZCHAR *pcKeyName,
                  ZULONG dwVal);
```

[Parameters]

Input parameters:

ST_DOM_ELEM *pstSecElem
The element that contains the key element.

ZCHAR *pcKeyName
Name of the key element.

ZULONG dwVal
The value of unsigned long.

Output parameters:

None.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```
ST_DOM_ELEM *pstSecElem;
ZCHAR *pcKeyName;
ZULONG dwVal;
ZINT iRet;
.....
/* Sets the key value of unsigned long of a key element. */
iRet = Dom_SecSetUl(pstSecElem, pcKeyName, dwVal);
```

3.3.25 Dom_SecSetUs

Sets the key value of unsigned short of a key element.

```
ZINT Dom_SecSetUs(ST_DOM_ELEM *pstSecElem, ZCHAR *pcKeyName,
                  ZUSHORT wVal);
```

[Parameters]

Input parameters:

[ST_DOM_ELEM](#) *pstSecElem
The element that contains the key element.

ZCHAR *pcKeyName
Name of the key element.

ZUSHORT wVal
The value of unsigned short.

Output parameters:

None.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```
ST\_DOM\_ELEM *pstSecElem;
ZCHAR *pcKeyName;
ZUSHORT wVal;
ZINT iRet;
.....
/* Set the key value of unsigned short of a key element. */
iRet = Dom_SecSetUs(pstSecElem, pcKeyName, wVal);
```

3.3.26 Dom_SecSetUc

Sets the key value of unsigned char of a key element.

```
ZINT Dom_SecSetUc(ST\_DOM\_ELEM *pstSecElem, ZCHAR *pcKeyName,
                  ZUCHAR ucVal);
```

[Parameters]

Input parameters:

[ST_DOM_ELEM](#) *pstSecElem
The element that contains the key element.

ZCHAR *pcKeyName
Name of the key element.

ZUCHAR ucVal
The value of unsigned char.

Output parameters:

None.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST_DOM_ELEM *pstSecElem;
ZCHAR *pcKeyName;
ZUCHAR ucVal;
ZINT iRet;
.....
/* Set the key value of unsigned char of a key element. */
iRet = Dom_SecSetUc(pstSecElem, pcKeyName, ucVal);

```

3.3.27 Dom_SecSetBool

Sets the key value of Boolean of a key element.

```

ZINT Dom_SecSetBool(ST_DOM_ELEM *pstSecElem, ZCHAR *pcKeyName,
                    ZBOOL bTrueFalse);

```

[Parameters]

Input parameters:

```

ST_DOM_ELEM *pstSecElem
The element that contains the key element.

```

```

ZCHAR *pcKeyName
Name of the key element.

```

```

ZBOOL bTrueFalse
The Boolean value.

```

Output parameters:

None.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```
ST_DOM_ELEM *pstSecElem;
ZCHAR *pcKeyName;
ZBOOL bTrueFalse;
ZINT iRet;
.....
/* Set the key value of Boolean of a key element. */
iRet = Dom_SecSetBool(pstSecElem, pcKeyName, bTrueFalse);
```

3.3.28 Dom_KeyPutVal

Adds a string of characters into a key element.

```
ZINT Dom_KeyPutVal(ST_DOM_ELEM *pstKeyElem, ZCHAR *pcVal);
```

[Parameters]

Input parameters:

```
ST_DOM_ELEM *pstKeyElem
```

The key element.

```
ZCHAR *pcVal
```

The string of characters.

Output parameters:

None.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```
ST_DOM_ELEM *pstKeyElem;
ZCHAR *pcVal;
ZINT iRet;
.....
/* Add a string of characters into a key element. */
iRet = Dom_KeyPutVal(pstKeyElem, pcVal);
```

3.3.29 Dom_KeySetUI

Sets the value of unsigned long contained in a key element.

```
ZINT Dom_KeySetUI(ST_DOM_ELEM *pstKeyElem, ZULONG dwVal);
```

[Parameters]

Input parameters:

```
ST\_DOM\_ELEM *pstKeyElem
```

The key element.

```
ZULONG dwVal
```

The value of unsigned long.

Output parameters:

None.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```
ST\_DOM\_ELEM *pstKeyElem;
ZULONG dwVal;
ZINT iRet;
.....
/* Set the value of unsigned long contained in a key element. */
iRet = Dom_KeySetUl(pstKeyElem, dwVal);
```

3.3.30 Dom_KeySetUs

Sets the value of unsigned short contained in a key element.

```
ZINT Dom_KeySetUs(ST\_DOM\_ELEM *pstKeyElem, ZUSHORT wVal);
```

[Parameters]**Input parameters:**

```
ST\_DOM\_ELEM *pstKeyElem
```

The key element.

```
ZUSHORT wVal
```

The value of unsigned long.

Output parameters:

None.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST_DOM_ELEM *pstKeyElem;
ZUSHORT wVal;
ZINT iRet;
.....
/* Set the value of unsigned short contained in a key element. */
iRet = Dom_KeySetUs(pstKeyElem, wVal);

```

3.3.31 Dom_KeySetUc

Sets the value of unsigned char contained in a key element.

```
ZINT Dom_KeySetUc(ST_DOM_ELEM *pstKeyElem, ZUCHAR ucVal);
```

[Parameters]

Input parameters:

```
ST_DOM_ELEM *pstKeyElem
```

The key element.

```
ZUCHAR ucVal
```

The value of unsigned char.

Output parameters:

None.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST_DOM_ELEM *pstKeyElem;
ZUCHAR ucVal;
ZINT iRet;
.....
/* Set the value of unsigned char contained in a key element. */
iRet = Dom_KeySetUc(pstKeyElem, ucVal);

```

3.3.32 Dom_KeySetBool

Sets the value of Boolean contained in a key element.

```
ZINT Dom_KeySetBool(ST_DOM_ELEM *pstKeyElem, ZBOOL bTrueFalse);
```

[Parameters]

Input parameters:

[ST_DOM_ELEM](#) *pstKeyElem

The key element.

ZBOOL bTrueFalse

The Boolean value.

Output parameters:

None.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```
ST\_DOM\_ELEM *pstKeyElem;
ZBOOL bTrueFalse;
ZINT iRet;
.....
/* Set the value of Boolean contained in a key element. */
iRet = Dom_KeySetBool(pstKeyElem, bTrueFalse);
```

3.3.33 Dom_KeySetAttrStr

Sets an attribute string contained in a key element. A node to hold the string will be created first and then be added in to the key element.

```
ZINT Dom_KeySetAttrStr(ST\_DOM\_ELEM *pstKeyElem, ZCHAR *pcName,
                      ZCHAR *pcVal);
```

[Parameters]

Input parameters:

[ST_DOM_ELEM](#) *pstKeyElem

The key element.

ZCHAR *pcName

Name of the string node.

ZCHAR *pcVal

The string value.

Output parameters:

None.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```
ST_DOM_ELEM *pstKeyElem;
ZCHAR *pcName;
ZCHAR *pcVal;
ZINT iRet;
.....
/* Set the value of a string contained in a key element. */
iRet = Dom_KeySetAttrStr(pstKeyElem, pcName, pcVal);
```

3.3.34 Dom_KeySetAttrUl

Sets the attribute value of unsigned long contained in a key element.

```
ZINT Dom_KeySetAttrUl(ST_DOM_ELEM *pstKeyElem, ZCHAR *pcName,
                     ZULONG dwVal);
```

[Parameters]

Input parameters:

`ST_DOM_ELEM *pstKeyElem`
The key element.

`ZCHAR *pcName`
Name of the attribute.

`ZULONG dwVal`
The attribute value of unsigned long.

Output parameters:

None.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```
ST_DOM_ELEM *pstKeyElem;
ZCHAR *pcName;
ZULONG dwVal;
ZINT iRet;
.....
/* Set the attribute value of unsigned long contained in a key element. */
iRet = Dom_KeySetAttrUl(pstKeyElem, pcName, dwVal);
```

3.3.35 Dom_KeySetAttrUs

Sets the attribute value of unsigned short contained in a key element.

```
ZINT Dom_KeySetAttrUs(ST_DOM_ELEM *pstKeyElem, ZCHAR *pcName,
                      ZUSHORT wVal);
```

[Parameters]

Input parameters:

[ST_DOM_ELEM](#) *pstKeyElem

The key element.

ZCHAR *pcName

Name of the attribute.

ZUSHORT wVal

The attribute value of unsigned short.

Output parameters:

None.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```
ST\_DOM\_ELEM *pstKeyElem;
ZCHAR *pcName;
ZUSHORT wVal;
ZINT iRet;
.....
/* Set the attribute value of unsigned short contained in a key element. */
iRet = Dom_KeySetAttrUs(pstKeyElem, pcName, wVal);
```

3.3.36 Dom_KeySetAttrUc

Sets the attribute value of unsigned char contained in a key element.

```
ZINT Dom_KeySetAttrUc(ST_DOM_ELEM *pstKeyElem, ZCHAR *pcName,
                      ZUCHAR ucVal);
```

[Parameters]

Input parameters:

[ST_DOM_ELEM](#) *pstKeyElem

The key element.

ZCHAR *pcName

Name of the attribute.

ZUCHAR ucVal

The attribute value of unsigned char.

Output parameters:

None.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```
ST\_DOM\_ELEM *pstKeyElem;
```

```
ZCHAR *pcName;
```

```
ZUCHAR ucVal;
```

```
ZINT iRet;
```

```
.....
```

```
/* Set the attribute value of unsigned char contained in a key element. */
```

```
iRet = Dom_KeySetAttrUc(pstKeyElem, pcName, ucVal);
```

3.3.37 Dom_KeySetAttrBool

Sets the attribute value of Boolean contained in a key element.

```
ZINT Dom_KeySetAttrBool(ST\_DOM\_ELEM *pstKeyElem, ZCHAR *pcName,  
                        ZBOOL bTrueFalse);
```

[Parameters]

Input parameters:

[ST_DOM_ELEM](#) *pstKeyElem

The key element.

ZCHAR *pcName

Name of the attribute.

ZBOOL bTrueFalse

The attribute value of Boolean.

Output parameters:

None.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST_DOM_ELEM *pstKeyElem;
ZCHAR *pcName;
ZBOOL bTrueFalse;
ZINT iRet;
.....
/* Set the attribute value of Boolean contained in a key element. */
iRet = Dom_KeySetAttrBool(pstKeyElem, pcName, bTrueFalse);

```

3.4 SAX Interfaces

3.4.1 Interface Structures

3.4.1.1 ST_SAX1_ACTION

```

typedef struct tagSAX1_ACTION
{
    ST_SAX1_DTD_ACTION stDtd;           /* DTD action */
    ST_SAX1_DOC_ACTION stDoc;          /* Document action */
    ST_SAX1_ERR_ACTION stErr;         /* Error action */
    ST_SAX1_REV_ACTION stRev;         /* resolving entities action */
} ST_SAX1_ACTION;

```

3.4.1.2 ST_SAX1_DTD_ACTION

```

typedef struct tagSAX1_DTD_ACTION
{
    PFN_SAX1NOTAIONDECL pfnNotationDecl; /* notation declaration */
    PFN_SAX1UPENTDECL pfnUnparsedEntDecl; /* unparsed entity declaration */
} ST_SAX1_DTD_ACTION;

```

3.4.1.3 ST_ZOS_USTR

```

typedef struct tagZOS_USTR
{
    ZUCHAR *pucStr;                    /* string pointer */
    ZUSHORT wLen;                      /* string length */
    ZUCHAR aucSpare[2];                /* for 32 bit alignment */
} ST_ZOS_USTR;

```

3.4.2 Sax1_SetDftAction

This interface allows applications to initialize a SAX action.

```
ZINT Sax1_SetDftAction(ST SAX1 ACTION *pstAct)
```

[Parameters]

Input parameters:

```
ST SAX1 ACTION *pstAct
```

The SAX action.

Output parameters:

None.

[Return value]

Returns ZOK.

[Example]

```
ST SAX1 ACTION *pstAct;  
.....  
/* Allow applications to initialize a SAX action. */  
Sax1_SetDftAction(pstAct);
```

3.4.3 Sax1_SetDtdAction

Allows applications to register a specified DTD action with a SAX action.

```
ZINT Sax1_SetDtdAction(ST SAX1 ACTION *pstAct,  
                       ST SAX1 DTD ACTION *pstDtd);
```

[Parameters]

Input parameters:

```
ST SAX1 ACTION *pstAct
```

The SAX action.

```
ST SAX1 DTD ACTION *pstDtd
```

The DTD action to register.

Output parameters:

None.

[Return value]

Returns ZOK.

[Example]

```

ST\_SAX1\_ACTION *pstAct;
ST\_SAX1\_DTD\_ACTION *pstDtd;
.....
/* Register a DTD action with a SAX action. */
Sax1_SetDtdAction(pstAct, pstDtd);

```

3.4.4 Sax1_SetDocAction

Allows applications to register a specified document action with a SAX action.

```

ZINT Sax1_SetDocAction(SST\_SAX1\_ACTION *pstAct,
                      ST\_SAX1\_DTD\_ACTION *pstDoc);

```

[Parameters]

Input parameters:

[ST_SAX1_ACTION](#) *pstAct
The SAX action.

[ST_SAX1_DTD_ACTION](#) *pstDoc
The document action to register.

Output parameters:

None.

[Return value]

Returns ZOK.

[Example]

```

ST\_SAX1\_ACTION *pstAct;
ST\_SAX1\_DTD\_ACTION *pstDoc;
.....
/* Register a document action with a SAX action. */
Sax1_SetDocAction(pstAct, pstDoc);

```

3.4.5 Sax1_SetResolveAction

Allows applications to register a specified resolving entities action with a SAX action.

```
ZINT Sax1_SetResolveAction(ST SAX1 ACTION *pstAct,  
                           ST SAX1 DTD ACTION *pstRev);
```

[Parameters]

Input parameters:

[ST SAX1 ACTION](#) *pstAct
The SAX action.

[ST SAX1 DTD ACTION](#) *pstRev
The resolving entities action to register.

Output parameters:

None.

[Return value]

Returns ZOK.

[Example]

```
ST SAX1 ACTION *pstAct;  
ST SAX1 DTD ACTION *pstRev;  
.....  
/* Register a resolving entities action with a SAX action. */  
Sax1_SetResolveAction(pstAct, pstRev);
```

3.4.6 Sax1_SetErrAction

Allows applications to register a specified error action with a SAX action.

```
ZINT Sax1_SetErrAction(ST SAX1 ACTION *pstAct,  
                       ST SAX1 DTD ACTION *pstErr);
```

[Parameters]

Input parameters:

[ST SAX1 ACTION](#) *pstAct
The SAX action.

[ST SAX1 DTD ACTION](#) * pstErr
The error action to register.

Output parameters:

None.

[Return value]

Returns ZOK.

[Example]

```

ST SAX1 ACTION *pstAct;
ST SAX1 DTD ACTION *pstErr;
.....
/* Register an error action with a SAX action. */
Sax1_SetErrAction(pstAct, pstErr);

```

3.4.7 Sax1_ParseData

Parses XML data.

```
ZINT Sax1_ParseData(ST ZOS USTR *pstData, ST SAX1 ACTION *pstAct);
```

[Parameters]

Input parameters:

[ST ZOS USTR](#) *pstData
The data to parse.

[ST SAX1 ACTION](#) *pstAct
The SAX action.

Output parameters:

None.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```

ST SAX1 ACTION *pstAct;
ST ZOS USTR *pstData
ZINT iRet;
.....
/* Parse XML data. */
iRet = Sax1_ParseData(pstData, pstAct);

```

3.4.8 Sax1_ParseFile

Parses an XML file.

```
ZINT Sax1_ParseFile(ZCHAR *pcFileName, ST\_SAX1\_ACTION *pstAct);
```

[Parameters]

Input parameters:

ZCHAR *pcFileName
Name of the file to parse.

[ST_SAX1_ACTION](#) *pstAct
The SAX action.

Output parameters:

None.

[Return value]

Returns ZOK on success or ZFAILED on failure.

[Example]

```
ST\_SAX1\_ACTION *pstAct;
ZCHAR *pcFileName;
ZINT iRet;
.....
/* Parse an XML file. */
iRet = Sax1_ParseFile(pcFileName, pstAct);
```

3.5 Codec Interfaces

3.5.1 Interface Structures

3.5.1.1 *ST_XML_ENCODE_MSG*

```
typedef struct tagXML_ENCODE_MSG
{
    ZBOOL bIndentFmt;           /* indent format */
    ZULONG dwElemDeep;         /* element deep */
    ST_ZOS_EBUF *pstMsgBuf;     /* data message buffer */
    ST_XML_ERR_INFO *pstErr;    /* error info */
    ST_XML_UCS_EACTION *pstAction; /* ucs encode action */
} ST_XML_ENCODE_MSG;
```

3.5.1.2 *ST_XML_ERR_INFO*

```
typedef struct tagXML_ERR_INFO
{
    ZUCHAR *pucFirstErrStr;        /* the first error string */
    ZULONG dwCode;                 /* error code */
    ZULONG dwErrNum;              /* xml user error number */
    ZDUMPID zDumpId;              /* dump stack */
} ST_XML_ERR_INFO;
```

3.5.1.3 *ST_XML_MSG*

```
typedef struct tagXML_MSG
{
    ZUCHAR ucPres;                 /* present flag */
    ZUCHAR aucSpare[3];           /* for 32 bit alignment */
    ZSBUFID zMemBufId;           /* memory buffer */
    ST_ZOS_EBUF *pstMsgBuf;       /* message buffer */
    ST_ZOS_USTR stMsgStr;         /* message string(decode used) */
    ST_XML_DOC stDoc;             /* xml document */
} ST_XML_MSG;
```

3.5.1.4 *ST_ZOS_EBUF*

```
typedef struct tagZOS_EBUF
{
    ZULONG dwMagic;               /* buffer magic */
    ZULONG dwPageSize;           /* page size */
    ZULONG dwMemSize;            /* memory size */
    ZUCHAR *pucMemBegin;         /* memory begin */
    ZBOOL bIsUsrBuf;             /* is user message buffer */
    ST_ZOS_DBUF *pstMsgBuf;       /* message buffer */
} ST_ZOS_EBUF;
```

3.5.1.5 ST_XML_DECODE_MSG

```
typedef struct tagXML_DECODE_MSG
{
    ZSBUFID zSBufId;           /* structure buffer id */
    ST_ZOS_PBUF *pstPBuf;     /* pipe buffer */
    ST_XML_ERR_INFO *pstErr;  /* error info */
    ST_XML_BUF_INFO stBuf;    /* data buffer */
    ST_XML_BUF_INFO stSavedBuf; /* saved data buffer */
    ST_XML_UCS_DACTION *pstAction; /* ucs decode action */
    ZVOID *pSaxCtx;          /* sax context */
} ST_XML_DECODE_MSG;
```

3.5.2 Xml_ErrInit

Initializes an error information structure.

```
ZINT Xml_ErrInit(ST\_XML\_ERR\_INFO *pstErrInfo)
```

[Parameters]

Input parameters:

None.

Output parameters:

```
ST\_XML\_ERR\_INFO *pstErrInfo
```

The initialized error information structure.

[Return value]

Returns XML_OK on success or XML_FAIL on failure.

[Example]

See the example of [Xml_errDestroy](#).

3.5.3 Xml_ErrDestroy

Destroys error information.

```
ZVOID Xml_ErrDestroy(ST\_XML\_ERR\_INFO *pstErrInfo);
```

[Parameters]

Input parameters:

[ST_XML_ERR_INFO](#) *pstErrInfo

The error information to destroy.

Output parameters:

None.

[Return value]

None.

[Example]

```
ST\_XML\_ERR\_INFO stErrInfo;
ZINT iRet;
.....
/* Initialize an error information structure.*/
iRet = Xml_ErrInit(&stErrInfo);
.....
/* Destroy the error information. */
Xml_ErrDestroy((&stErrInfo);
```

3.5.4 Xml_ErrPrint

Prints out the error information generated when a message is being decoded.

```
ZINT Xml_ErrPrint(ST\_XML\_DECODE\_MSG *pstDecodeMsg);
```

[Parameters]**Input parameters:**

[ST_XML_DECODE_MSG](#) *pstDecodeMsg

The decoding message structure which contains the error information.

Output parameters:

None.

[Return value]

Returns XML_OK on success or XML_FAIL on failure.

[Example]

See the example of [Xml DecodeMsg](#).

3.5.5 Xml_EncodeInit

Encoding initialization which should be completed before encoding a message.

```
ZINT Xml_EncodeInit(ST\_XML\_ENCODE\_MSG *pstEncodeMsg, ZBOOL bIndentFmt,  
                   ST\_ZOS\_EBUF *pstMsgBuf, ST\_XML\_ERR\_INFO *pstErrInfo);
```

[Parameters]

Input parameters:

[ST_XML_ENCODE_MSG](#) *pstEncodeMsg
Some encoding settings to initialize.

ZBOOL bIndentFmt
Whether or not to allow automatic indention.

[ST_ZOS_EBUF](#) *pstMsgBuf
ZOS buffer.

[ST_XML_ERR_INFO](#) *pstErrInfo
The error information. This can be ZNULL.

Output parameters:

None.

[Return value]

Returns XML_OK on success or XML_FAIL on failure.

[Example]

See [Xml_EncodeMsg](#).

3.5.6 Xml_EncodeMsg

Encodes an XML tree, transforming it into a message.

```
ZINT Xml_EncodeMsg(ST\_XML\_ENCODE\_MSG *pstEncodeMsg,  
                  ST\_XML\_MSG *pstXmlMsg)
```

[Parameters]

Input parameters:

[ST_XML_ENCODE_MSG](#) *pstEncodeMsg

The XML tree to encode.

[ST_XML_MSG](#) *pstXmlMsg

This will point to the encoded message.

Output parameters:

[ST_XML_MSG](#) *pstXmlMsg

The encoded message.

[Return value]

Returns XML_OK on success or XML_FAIL on failure.

[Example]

```

ST\_XML\_ENCODE\_MSG stEncodeMsg;
ST\_XML\_MSG stXmlMsg;
ST\_ZOS\_EBUF *pstEbuf;
ZINT iRet;
.....
/* Encoding initialization. */
Xml_EncodeInit(&stEncodeMsg, ZTRUE, pstEbuf, ZNULL);

iRet = Xml_EncodeMsg(&stEncodeMsg, &stXmlMsg);

```

3.5.7 Xml_DecodeInit

Decoding initialization.

```

ZINT Xml_DecodeInit(ST\_XML\_DECODE\_MSG *pstDecodeMsg, ST\_ZOS\_USTR *pstStr,
                   ZSBUFID zSBufId, ZVOID *pSaxCtx, ST\_XML\_ERR\_INFO *pstErr);

```

[Parameters]

Input parameters:

[ST_XML_DECODE_MSG](#) *pstDecodeMsg

Message to initialize.

[ST_ZOS_USTR](#) *pstStr

Message to decode.

ZSBUFID zSBufId

Buffer ID.

ZVOID *pSaxCtx

The SAX context.

[ST_XML_ERR_INFO](#) *pstErr

The error information.

Output parameters:

None.

[Return value]

Returns XML_OK on success or XML_FAIL on failure.

[Example]

See [Xml DecodeMsg](#).

3.5.8 Xml_DecodeMsg

Decodes a message, transforming it into an XML tree.

```
ZINT Xml_DecodeMsg(ST\_XML\_DECODE\_MSG *pstDecodeMsg,  
                  ST\_XML\_MSG *pstXmlMsg)
```

[Parameters]

Input parameters:

[ST_XML_DECODE_MSG](#) *pstDecodeMsg

The message to decode.

[ST_XML_MSG](#) *pstXmlMsg

This will point to the decoded tree.

Output parameters:

[ST_XML_MSG](#) *pstXmlMsg

The decoded tree.

[Return value]

Returns XML_OK on success or XML_FAIL on failure.

[Example]

```
ST\_XML\_DECODE\_MSG stDecodeMsg;
ST\_XML\_ERR\_INFO stErrInfo;
ST\_XML\_MSG stXmlMsg;
ZSBUFID zMemBufId;
ZVOID *pSaxCtx;
ZINT iRet;
.....
/* Decoding initialization. */
Xml_DecodeInit(&stDecodeMsg, pstXmlStr, zMemBufId, pSaxCtx, &stErrInfo);

/* XML decoding. */
iRet = Xml_DecodeMsg(&stDecodeMsg, &stXmlMsg);
if (iRet != XML_OK)
{
    XML_LOG_ERROR((XML_LOGID, "DocLoadXml decode xml message.));

    /* print error info */
    Xml_ErrPrint(&stDecodeMsg);
    Xml_ErrDestroy(&stErrInfo);

    /* delete memory buffer */
    Zos_SbufDelete(zMemBufId);
}
```

3.6 XSP Interfaces

3.6.1 Interface Structures

3.6.1.1 *ST_XML_QNAME*

```
typedef struct tagXML_QNAME
{
    ZUCHAR ucPrefixPres;           /* PrefixedName present flag */
    ZUCHAR aucSpare[3];           /* for 32 bit alignment */
    ST_XML_NCNAME stPrefix;       /* Prefix */
    ST_XML_NCNAME stLocalPart;    /* LocalPart */
} ST_XML_QNAME;
```

3.6.1.2 *ST_XSP_NS*

```
typedef struct tagXSP_NS
{
    ZULONG dwDftNsId;             /* default namespace id EN_XSP_NS_TYPE */
    ZULONG dwNsAttrNum;          /* namespace attributes number */
    ST_XSP_NS_ATTR astNsAttr[XSP_NS_MAX_NUM]; /* namespace attributes */
} ST_XSP_NS;
```

3.6.1.3 *ST_XML_ELEM*

```
/* xml element */
struct tagXML_ELEM
{
    ZUCHAR ucPres;               /* present flag */
    ZUCHAR ucEmptyPres;         /* EmptyElemTag present flag */
    ZUCHAR aucSpare[2];         /* for 32 bit alignment */
    union
    {
        ST_XML_EMPTY_ELEM_TAG stEmptyElemTag; /* EmptyElemTag */
        ST_XML_ELEM_TAG stElemTag; /* element tag */
    } u;
};

typedef struct tagXML_ELEM ST_XML_ELEM;
```

3.6.1.4 ST_XML_ATTR

```
typedef struct tagXML_ATTR
{
    ST_XML_QNAME stQName;          /* QName */
    ST_XML_ATT_VAL stVal;         /* AttValue */
} ST_XML_ATTR;
```

3.6.1.5 ST_XML_CONTENT_ITEM

```
typedef struct tagXML_CONTENT_ITEM
{
    ZUCHAR ucPres;                /* present flag */
    ZUCHAR ucType;               /* content type EN_XML_CONTENT_ITEM_TYPE */
    ZUCHAR aucSpare[2];          /* for 32 bit alignment */
    union
    {
        ST_XML_ELEM *pstElem;    /* element */
        ST_ZOS_USTR *pstChrData; /* CharData */
        ST_XML_REF *pstRef;      /* Reference */
        ST_XML_CD_SECT *pstCdSect; /* CDsect */
        ST_XML_PI *pstPi;        /* PI */
        ST_XML_COMMENT *pstComment; /* Comment */
        ST_ZOS_USTR *pstIgnWS;   /* ignorable whitespace */
    } u;
} ST_XML_CONTENT_ITEM;
```

3.6.2 Map Interfaces

3.6.2.1 Xsp_MapGetNsId

Gets the namespace ID from a string.

```
ZINT Xsp_MapGetNsId(ST_ZOS_USTR *pstStr, ZULONG *pdwNsId);
```

[Parameters]

Input parameters:

[ST_ZOS_USTR](#) *pstStr
The string.

Output parameters:

ZULONG *pdwNsId
The namespace ID.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```
ST\_ZOS\_USTR *pstStr;
ZULONG dwNsId;
ZINT iRet;
.....
/* Get the namespace ID from a string. */
iRet = Xsp_MapGetNsId(pstStr, &dwNsId);
```

3.6.2.2 *Xsp_MapGetNsStr*

Gets the namespace string from an ID.

```
ZINT Xsp_MapGetNsStr(ZULONG dwNsId, ST\_ZOS\_USTR *pstStr);
```

[Parameters]**Input parameters:**

```
ZULONG dwNsId
The ID.
```

Output parameters:

```
ST\_ZOS\_USTR *pstStr
The namespace string.
```

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```
ZULONG dwNsId;
ST\_ZOS\_USTR *pstStr;
ZINT iRet;
.....
/* Get the namespace string from an ID. */
iRet = Xsp_MapGetNsStr(dwNsId, pstStr);
```

3.6.2.3 *Xsp_MapGetTknId*

Gets a token ID from a string.

```
ZINT Xsp_MapGetTknId(ZULONG dwNsId, ST\_ZOS\_USTR *pstStr,  
                    ZULONG *pdwTknId);
```

[Parameters]

Input parameters:

ZULONG dwNsId
The namespace ID.

[ST_ZOS_USTR](#) *pstStr
The string.

Output parameters:

ZULONG *pdwTknId
The token ID.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```
ZULONG dwNsId;  
ST\_ZOS\_USTR *pstStr;  
ZULONG dwTknId;  
ZINT iRet;  
.....  
/* Get a token ID from a string. */  
iRet = Xsp_MapGetTknId(dwNsId, pstStr, &dwTknId);
```

3.6.2.4 *Xsp_MapGetTknStr*

Gets the token string from an ID.

```
ZINT Xsp_MapGetTknStr(ZULONG dwNsId, ZULONG dwTknId,  
                     ST\_ZOS\_USTR *pstStr);
```

[Parameters]

Input parameters:

ZULONG dwNsId
The namespace ID.

ZULONG dwTknId
The token ID.

Output parameters:

[ST_ZOS_USTR](#) *pstStr
The token string.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```
ZULONG dwNsId;
ZULONG dwTknId;
ST\_ZOS\_USTR *pstStr;
ZINT iRet;
.....
/* Get the token string from an ID. */
iRet = Xsp_MapGetTknStr(dwNsId, dwTknId, pstStr);
```

3.6.3 Encoding Interfaces

3.6.3.1 Xsp_MsgSaveFile

Saves a message to a file.

```
ZINT Xsp_MsgSaveFile(ST\_XML\_MSG *pstMsg, ZCHAR *pcFileName);
```

[Parameters]**Input parameters:**

[ST_XML_MSG](#) *pstMsg
The message to save.

ZCHAR *pcFileName
The file name.

Output parameters:

None.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST_XML_MSG *pstMsg;
ZCHAR *pcFileName;
ZINT iRet;
.....
/* Save a message to a file. */
iRet = Xsp_MsgSaveFile(pstMsg, pcFileName);

```

3.6.3.2 Xsp_MsgSaveData

Encodes a message and creates a data buffer to hold the encoded information.

```
ZINT Xsp_MsgSaveData(ST_XML_MSG *pstMsg, ST_ZOS_DBUF **ppstData);
```

[Parameters]**Input parameters:**

[ST_XML_MSG](#) *pstMsg

The message.

Output parameters:

[ST_ZOS_DBUF](#) **ppstData

The data buffer which holds the encoded information.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST_XML_MSG *pstMsg;
ST_ZOS_DBUF *pstData;
ZINT iRet;
.....
/* Encode a message and create a data buffer to hold the encoded information. */
iRet = Xsp_MsgSaveData(pstMsg, &pstData);

```

3.6.3.3 Xsp_DocAddHdr

Adds a document header into an XML message.

```
ZINT Xsp_DocAddHdr(ST\_XML\_MSG *pstMsg);
```

[Parameters]

Input parameters:

[ST_XML_MSG](#) *pstMsg

The message.

Output parameters:

None.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```
ST\_XML\_MSG *pstMsg;  
ZINT iRet;  
.....  
/* Add a document header into an XML message. */  
iRet = Xsp_DocAddHdr(pstMsg);
```

3.6.3.4 Xsp_DocAddRoot

Adds the root element into a document.

```
ZINT Xsp_DocAddRoot(ST\_XML\_MSG *pstMsg, ST\_XML\_QNAME *pstQName,  
                   ST\_XSP\_NS *pstNs, ST\_XML\_ELEM **ppstRoot);
```

[Parameters]

Input parameters:

[ST_XML_MSG](#) *pstMsg

The message which includes the XML document.

[ST_XML_QNAME](#) *pstQName

The XML QName.

[ST_XSP_NS](#) *pstNs

The namespace.

Output parameters:

[ST_XML_ELEM](#) **ppstRoot
The added root element.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST\_XML\_MSG *pstMsg;
ST\_XML\_QNAME *pstQName;
ST\_XSP\_NS *pstNs;
ST\_XML\_ELEM *pstRoot;
ZINT iRet;
.....
/* Add the root element into a document. */
iRet = Xsp_DocAddRoot(pstMsg, pstQName, pstNs, &pstRoot);

```

3.6.3.5 Xsp_DocAddNsRoot

Adds the root element of the default namespace into a document.

```

ZINT Xsp_DocAddNsRoot(ST\_XML\_MSG *pstMsg, ST\_XSP\_NS *pstNs,
                     ZULONG dwNameId, ST\_XML\_ELEM **ppstRoot);

```

[Parameters]**Input parameters:**

[ST_XML_MSG](#) *pstMsg
The message which includes the XML document.

[ST_XSP_NS](#) *pstNs
The namespace.

ZULONG dwNameId
The token ID.

Output parameters:

[ST_XML_ELEM](#) **ppstRoot
The added namespace element.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST_XML_MSG *pstMsg;
ST_XSP_NS *pstNs;
ZULONG dwNameId
ST_XML_ELEM *pstRoot;
ZINT iRet;
.....
/* Add the root namespace element into a document. */
iRet = Xsp_DocAddNsRoot(pstMsg, pstNs, dwNameId, &pstRoot);

```

3.6.3.6 Xsp_DocNsAddRoot

Adds the root element of a specific namespace into a document.

```

ZINT Xsp_DocNsAddRoot(ST_XML_MSG *pstMsg, ST_XSP_NS *pstNs,
                      ZULONG dwNsId, ZULONG dwNameId, ST_XML_ELEM **ppstRoot);

```

[Parameters]

Input parameters:

[ST_XML_MSG](#) *pstMsg

The message which includes the XML document.

[ST_XSP_NS](#) *pstNs

The namespace.

ZULONG dwNsId

The namespace ID.

ZULONG dwNameId

The token ID.

Output parameters:

[ST_XML_ELEM](#) **ppstRoot

The added namespace element.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST\_XML\_MSG *pstMsg;
ST\_XSP\_NS *pstNs;
ZULONG dwNsId;
ZULONG dwNameId;
ST\_XML\_ELEM *pstRoot;
ZINT iRet;
.....
/* Add the root element of a specific namespace into a document. */
iRet = Xsp_DocNsAddRoot(pstMsg, pstNs, dwNsId, dwNameId, &pstRoot);

```

3.6.3.7 Xsp_ElemAddChild

Adds a child element to an element.

```

ZINT Xsp_ElemAddChild(ZSBUFID zMemBufId, ST\_XML\_ELEM *pstElem,
                    ST\_XML\_QNAME *pstQName, ST\_XML\_ELEM **ppstChild);

```

[Parameters]

Input parameters:

[ZSBUFID](#) zMemBufId
The memory buffer ID.

[ST_XML_ELEM](#) *pstElem
The element which will have the new element as its child element.

[ST_XML_QNAME](#) *pstQName
The XML QName.

Output parameters:

[ST_XML_ELEM](#) **ppstChild
The added children element.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ZSBUFID zMemBufId;
ST_XML_ELEM *pstElem;
ST_XML_QNAME *pstQName;
ST_XML_ELEM *pstChild;
ZINT iRet;
.....
/* Add a child element to an element. */
iRet = Xsp_ElemAddChild(zMemBufId, pstElem, pstQName, &pstChild);

```

3.6.3.8 Xsp_ElemAddNsChild

Adds the default namespace as a child element to an element.

```

ZINT Xsp_ElemAddNsChild(ZSBUFID zMemBufId, ST_XML_ELEM *pstElem,
                        ST_XSP_NS *pstNs, ZULONG dwNameId, ST_XML_ELEM **ppstChild);

```

[Parameters]

Input parameters:

ZSBUFID zMemBufId
The memory buffer ID.

ST_XML_ELEM *pstElem
The element which will have the new element as its child element.

ST_XSP_NS *pstNs
The namespace.

ZULONG dwNameId
The token ID.

Output parameters:

ST_XML_ELEM **ppstChild
The added children element.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ZSBUFID zMemBufId;
ST_XML_ELEM *pstElem;
ST_XSP_NS *pstNs;
ZULONG dwNameId;
ST_XML_ELEM *pstChild;
ZINT iRet;
.....
/* Add the default namespace as a child element to an element. */
iRet = Xsp_ElemAddNsChild(zMemBufId, pstElem, pstNs, dwNameId, &pstChild);

```

3.6.3.9 Xsp_ElemNsAddChild

Adds a specific namespace as a child element to an element.

```

ZINT Xsp_ElemNsAddChild(ZSBUFID zMemBufId, ST_XML_ELEM *pstElem,
                        ST_XSP_NS *pstNs, ZULONG dwNsId, ZULONG dwNameId,
                        ST_XML_ELEM **ppstChild);

```

[Parameters]

Input parameters:

ZSBUFID zMemBufId
The memory buffer ID.

ST_XML_ELEM *pstElem
The element which will have the new element as its child element.

ST_XSP_NS *pstNs
The namespace.

ZULONG dwNsId
The namespace ID.

ZULONG dwNameId
The token ID.

Output parameters:

ST_XML_ELEM **ppstChild
The added children element.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ZSBUFID zMemBufId;
ST_XML_ELEM *pstElem;
ST_XSP_NS *pstNs;
ZULONG dwNsId;
ZULONG dwNameId;
ST_XML_ELEM *pstChild;
ZINT iRet;
.....
/* Add a specific namespace as a child element to an element. */
iRet = Xsp_ElemNsAddChild(zMemBufId, pstElem,
                          pstNs, dwNsId, dwNameId, &pstChild);

```

3.6.3.10 Xsp_ElemAddAttr

Adds an attribute into an element.

```

ZINT Xsp_ElemAddAttr(ZSBUFID zMemBufId, ST_XML_ELEM *pstElem,
                    ST_XML_QNAME *pstQName, ST_XML_ATTR **ppstAttr);

```

[Parameters]

Input parameters:

ZSBUFID zMemBufId
The memory buffer ID.

ST_XML_ELEM *pstElem
The element.

ST_XML_QNAME *pstQName
The QName.

Output parameters:

ST_XML_ATTR **ppstAttr
The added attribute.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ZSBUFID zMemBufId;
ST_XML_ELEM *pstElem;
ST_XML_QNAME *pstQName;
ST_XML_ATTR *pstAttr;
ZINT iRet;
.....
/* Add an attribute into an element. */
iRet = Xsp_ElemAddAttr(zMemBufId, pstElem, pstQName, &pstAttr);

```

3.6.3.11 Xsp_ElemAddAttrVal

Adds an attribute with a structure string to an element.

```

ZINT Xsp_ElemAddAttrVal(ZSBUFID zMemBufId, ST_XML_ELEM *pstElem,
                        ST_ZOS_USTR *pstName, ST_ZOS_USTR *pstVal);

```

[Parameters]

Input parameters:

ZSBUFID zMemBufId
The memory buffer ID.

ST_XML_ELEM *pstElem
The element.

ST_ZOS_USTR *pstName
The QName of the attribute.

ST_ZOS_USTR *pstVal
The value string of the attribute.

Output parameters:

None.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ZSBUFID zMemBufId;
ST_XML_ELEM *pstElem;
ST_ZOS_USTR *pstName;
ST_ZOS_USTR *pstVal;
ZINT iRet;
.....
/* Add an attribute with a structure string to an element. */
iRet = Xsp_ElemAddAttrVal(zMemBufId, pstElem, pstName, pstVal);

```

3.6.3.12 Xsp_ElemAddAttrId

Adds the attribute ID into an element. A new attribute will be created.

```

ZINT Xsp_ElemAddAttrId(ZSBUFID zMemBufId, ST_XML_ELEM *pstElem,
                      ST_XSP_NS *pstNs, ZULONG dwNameId, ST_XML_ATTR **ppstAttr);

```

[Parameters]

Input parameters:

ZSBUFID zMemBufId
The memory buffer ID.

ST_XML_ELEM *pstElem
The element.

ST_XSP_NS *pstNs
The namespace.

ZULONG dwNameId
The token ID.

Output parameters:

ST_XML_ATTR **ppstAttr
The newly created attribute.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ZSBUFID zMemBufId;
ST_XML_ELEM *pstElem;
ST_XSP_NS *pstNs;
ZULONG dwNameId;
ST_XML_ATTR *pstAttr;
ZINT iRet;
.....
/* Add the attribute ID into an element. A new attribute will be created. */
iRet = Xsp_ElemAddAttrId(zMemBufId, pstElem, pstNs, dwNameId, &pstAttr);

```

3.6.3.13 Xsp_ElemAddAttrIdVal

Adds an attribute with its name ID and value into an element.

```

ZINT Xsp_ElemAddAttrIdVal(ZSBUFID zMemBufId, ST_XML_ELEM *pstElem,
                          ST_XSP_NS *pstNs, ZULONG dwNameId, ST_ZOS_USTR *pstVal);

```

[Parameters]

Input parameters:

ZSBUFID zMemBufId
The memory buffer ID.

ST_XML_ELEM *pstElem
The element.

ST_XSP_NS *pstNs
The namespace.

ZULONG dwNameId
The token ID.

ST_ZOS_USTR *pstVal
The value.

Output parameters:

None.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ZSBUFID zMemBufId;
ST_XML_ELEM *pstElem;
ST_XSP_NS *pstNs;
ZULONG dwNameId;
ST_ZOS_USTR *pstVal;
ZINT iRet;
.....
/* Add an attribute with its name ID and value into an element. */
iRet = Xsp_ElemAddAttrIdVal(zMemBufId, pstElem, pstNs, dwNameId, pstVal);

```

3.6.3.14 Xsp_ElemAddAttrIdValId

Adds an attribute with its name ID and value ID into an element.

```

ZINT Xsp_ElemAddAttrIdValId(ZSBUFID zMemBufId, ST_XML_ELEM *pstElem,
                             ST_XSP_NS *pstNs, ZULONG dwNameId, ZULONG dwValId);

```

[Parameters]

Input parameters:

ZSBUFID zMemBufId
The memory buffer ID.

ST_XML_ELEM *pstElem
The element.

ST_XSP_NS *pstNs
The namespace.

ZULONG dwNameId
The name ID.

ZULONG dwValId
The value ID.

Output parameters:

None.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ZSBUFID zMemBufId;
ST_XML_ELEM *pstElem;
ST_XSP_NS *pstNs;
ZULONG dwNameId;
ZULONG dwValId;
ZINT iRet;
.....
/* Add an attribute with its name ID and value ID into an element. */
iRet = Xsp_ElemAddAttrIdValId(zMemBufId, pstElem, pstNs, dwNameId, dwValId);

```

3.6.3.15 Xsp_ElemAddAttrIdBool

Adds an attribute with a Boolean value into an element.

```

ZINT Xsp_ElemAddAttrIdBool(ZSBUFID zMemBufId, ST_XML_ELEM *pstElem,
                           ST_XSP_NS *pstNs, ZULONG dwNameId, ZBOOL bTrueFalse);

```

[Parameters]

Input parameters:

ZSBUFID zMemBufId
The memory buffer ID.

ST_XML_ELEM *pstElem
The element.

ST_XSP_NS *pstNs
The namespace.

ZULONG dwNameId
The token ID.

ZBOOL bTrueFalse
The Boolean value.

Output parameters:

None.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ZSBUFID zMemBufId;
ST_XML_ELEM *pstElem;
ST_XSP_NS *pstNs;
ZULONG dwNameId;
ZBOOL bTrueFalse;
ZINT iRet;
.....
/* Add an attribute with a Boolean value into an element. */
iRet = Xsp_ElemAddAttrIdBool(zMemBufId, pstElem, pstNs, dwNameId, bTrueFalse);

```

3.6.3.16 Xsp_ElemAddAttrIdULDigit

Adds an attribute with a unsigned long value into an element.

```

ZINT Xsp_ElemAddAttrIdULDigit(ZSBUFID zMemBufId, ST_XML_ELEM *pstElem,
                               ST_XSP_NS *pstNs, ZULONG dwNameId, ZULONG dwData);

```

[Parameters]

Input parameters:

ZSBUFID zMemBufId
The memory buffer ID.

ST_XML_ELEM *pstElem
The element.

ST_XSP_NS *pstNs
The namespace.

ZULONG dwNameId
The token ID.

ZULONG dwData
The unsigned long value.

Output parameters:

None.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ZSBUFID zMemBufId;
ST_XML_ELEM *pstElem;
ST_XSP_NS *pstNs;
ZULONG dwNameId;
ZULONG dwData
ZINT iRet;
.....
/* Add an attribute with a unsigned long value into an element. */
iRet = Xsp_ElemAddAttrIdULDigit(zMemBufId, pstElem, pstNs, dwNameId, dwData);

```

3.6.3.17 Xsp_ElemNsAddAttrId

Adds an attribute ID into an element. This will cause the creation of a new attribute.

```

ZINT Xsp_ElemNsAddAttrId(ZSBUFID zMemBufId, ST_XML_ELEM *pstElem,
                        ST_XSP_NS *pstNs, ZULONG dwNsId, ZULONG dwNameId,
                        ST_XML_ATTR **ppstAttr);

```

[Parameters]

Input parameters:

ZSBUFID zMemBufId
The memory buffer ID.

ST_XML_ELEM *pstElem
The element.

ST_XSP_NS *pstNs
The namespace.

ZULONG dwNameId
The token ID.

Output parameters:

ST_XML_ATTR **ppstAttr
The newly created attribute.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ZSBUFID zMemBufId;
ST_XML_ELEM *pstElem;
ST_XSP_NS *pstNs;
ZULONG dwNameId;
ST_XML_ATTR *pstAttr;
ZINT iRet;
.....
/* Add an attribute ID into an element. This will cause the creation of a new attribute.
*/
iRet = Xsp_ElemNsAddAttrId(zMemBufId, pstElem, pstNs, dwNameId, &pstAttr);

```

3.6.3.18 Xsp_ElemNsAddAttrIdVal

Adds an attribute with its name ID and value into an element.

```

ZINT Xsp_ElemNsAddAttrIdVal(ZSBUFID zMemBufId, ST_XML_ELEM *pstElem,
                             ST_XSP_NS *pstNs, ZULONG dwNsId, ZULONG dwNameId,
                             ST_ZOS_USTR *pstVal);

```

[Parameters]

Input parameters:

ZSBUFID zMemBufId
The memory buffer ID.

ST_XML_ELEM *pstElem
The element.

ST_XSP_NS *pstNs
The namespace.

ZULONG dwNameId
The name ID.

ST_ZOS_USTR *pstVal
The value.

Output parameters:

None.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ZSBUFID zMemBufId;
ST XML ELEM *pstElem;
ST XSP\_NS *pstNs;
ZULONG dwNameId;
ST ZOS USTR *pstVal;
ZINT iRet;
.....

/* Add an attribute with its name ID and value into an element. */
iRet = Xsp_ElemNsAddAttrIdVal(zMemBufId, pstElem, pstNs, dwNameId, pstVal);

```

3.6.3.19 Xsp_ElemNsAddAttrIdValId

Adds an attribute with its name ID and value ID into an element.

```

ZINT Xsp_ElemNsAddAttrIdValId(ZSBUFID zMemBufId, ST XML ELEM *pstElem,
ST XSP\_NS *pstNs, ZULONG dwNsId, ZULONG dwNameId,
ZULONG dwValId);

```

[Parameters]**Input parameters:**

ZSBUFID zMemBufId
The memory buffer ID.

[ST XML ELEM](#) *pstElem
The element.

[ST XSP_NS](#) *pstNs
The namespace.

ZULONG dwNameId
The name ID.

ZULONG dwValId
The value ID.

Output parameters:

None.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```
ZSBUFID zMemBufId;
ST_XML_ELEM *pstElem;
ST_XSP_NS *pstNs;
ZULONG dwNameId;
ZULONG dwValId;
ZINT iRet;
.....
/* Add an attribute with its name ID and value ID into an element. */
iRet = Xsp_ElemNsAddAttrIdValId(zMemBufId, pstElem, pstNs, dwNameId, dwValId);
```

3.6.3.20 Xsp_ElemNsAddAttrIdBool

Adds an attribute with a Boolean value into an element.

```
ZINT Xsp_ElemNsAddAttrIdBool(ZSBUFID zMemBufId, ST_XML_ELEM *pstElem,
                             ST_XSP_NS *pstNs, ZULONG dwNsId, ZULONG dwNameId,
                             ZBOOL bTrueFalse);
```

[Parameters]

Input parameters:

ZSBUFID zMemBufId
The memory buffer ID.

ST_XML_ELEM *pstElem
The element.

ST_XSP_NS *pstNs
The namespace.

ZULONG dwNameId
The name ID.

ZBOOL bTrueFalse
The Boolean value.

Output parameters:

None.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```
ZSBUFID zMemBufId;
ST_XML_ELEM *pstElem;
ST_XSP_NS *pstNs;
ZULONG dwNameId;
ZBOOL bTrueFalse;
ZINT iRet;

.....
/* Add an attribute with a Boolean value into an element. */
iRet = Xsp_ElemNsAddAttrIdBool(zMemBufId, pstElem, pstNs, dwNameId, bTrueFalse);
```

3.6.3.21 Xsp_ElemNsAddAttrIdUlDigit

Adds an attribute with an unsigned long value into an element.

```
ZINT Xsp_ElemNsAddAttrIdUlDigit(ZSBUFID zMemBufId,
                                ST_XML_ELEM *pstElem, SST_XSP_NS *pstNs, ZULONG dwNsId,
                                ZULONG dwNameId, ZULONG dwData);
```

[Parameters]

Input parameters:

ZSBUFID zMemBufId
The memory buffer ID.

ST_XML_ELEM *pstElem
The element.

ST_XSP_NS *pstNs
The namespace.

ZULONG dwNameId
The name ID.

ZULONG dwData
The unsigned long value.

Output parameters:

None.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```
ZSBUFID zMemBufId;
ST_XML_ELEM *pstElem;
ST_XSP_NS *pstNs;
ZULONG dwNameId;
ZULONG dwData;
ZINT iRet;
.....
/* Add an attribute with an unsigned long value into an element. */
iRet = Xsp_ElemNsAddAttrIdUldigit(zMemBufId, pstElem, pstNs, dwNameId, dwData);
```

3.6.3.22 Xsp_ElemAddData

Adds character data into an element.

```
ZINT Xsp_ElemAddData(ZSBUFID zMemBufId, ST_XML_ELEM *pstElem,
                    ST_ZOS_USTR *pstData);
```

[Parameters]

Input parameters:

ZSBUFID zMemBufId
The memory buffer ID.

ST_XML_ELEM *pstElem
The element.

ST_ZOS_USTR *pstData
The character data.

Output parameters:

None.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ZSBUFID zMemBufId;
ST_XML_ELEM *pstElem;
ST_ZOS_USTR *pstData
ZINT iRet;

.....
/* Add character data into an element. */
iRet = Xsp_ElemAddData(zMemBufId, pstElem, pstData);

```

3.6.3.23 Xsp_ElemAddDataId

Uses a data ID to get data from a default namespace and adds the data ID into an element.

```

ZINT Xsp_ElemAddDataId(ZSBUFID zMemBufId, ST_XML_ELEM *pstElem,
                       ST_XSP_NS *pstNs, ZULONG dwDataId);

```

[Parameters]

Input parameters:

ZSBUFID zMemBufId
The memory buffer ID.

ST_XML_ELEM *pstElem
The element.

ST_XSP_NS *pstNs
The namespace.

ZULONG dwDataId
The character data ID.

Output parameters:

None.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```
ZSBUFID zMemBufId;
ST_XML_ELEM *pstElem;
ST_XSP_NS *pstNs;
ZULONG dwDataId;
ZINT iRet;
.....
/* Uses a data ID to get data from a default namespace and adds the data ID into an
   element. */
iRet = Xsp_ElemAddDataId(zMemBufId, pstElem, pstNs, dwDataId);
```

3.6.3.24 *Xsp_ElemNsAddDataId*

Uses a data ID to get data from a specific namespace and adds the data ID into an element.

```
ZINT Xsp_ElemNsAddDataId(ZSBUFID zMemBufId, ST_XML_ELEM *pstElem,
                          ZULONG dwNsId, ZULONG dwDataId);
```

[Parameters]

Input parameters:

ZSBUFID zMemBufId
The memory buffer ID.

ST_XML_ELEM *pstElem
The element.

ZULONG dwNsId
The namespace ID.

ZULONG dwDataId
The data ID.

Output parameters:

None.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```
ZSBUFID zMemBufId;
ST_XML_ELEM *pstElem;
ZULONG dwNsId;
ZULONG dwDataId;
ZINT iRet;
.....
/* Use a data ID to get data from a specific namespace and adds the data ID into an
   element. */
iRet = Xsp_ElemNsAddDataId(zMemBufId, pstElem, dwNsId, dwDataId);
```

3.6.3.25 Xsp_ElemAddBool

Adds a Boolean value into an element.

```
ZINT Xsp_ElemAddBool(ZSBUFID zMemBufId, ST_XML_ELEM *pstElem,
                    ZBOOL bTrueFalse);
```

[Parameters]

Input parameters:

ZSBUFID zMemBufId
The memory buffer ID.

ST_XML_ELEM *pstElem
The element.

ZBOOL bTrueFalse
The Boolean value.

Output parameters:

None.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ZSBUFID zMemBufId;
ST_XML_ELEM *pstElem;
ZBOOL bTrueFalse;
ZINT iRet;
.....
/* Add a Boolean value into an element. */
iRet = Xsp_ElemAddBool(zMemBufId, pstElem, bTrueFalse);

```

3.6.3.26 Xsp_ElemAddUldigit

Adds an unsigned long value into an element.

```

ZINT Xsp_ElemAddUldigit(ZSBUFID zMemBufId, ST_XML_ELEM *pstElem,
                        ZULONG dwData);

```

[Parameters]

Input parameters:

ZSBUFID zMemBufId
The memory buffer ID.

ST_XML_ELEM *pstElem
The element.

ZULONG dwData
The unsigned long value.

Output parameters:

None.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ZSBUFID zMemBufId;
ST_XML_ELEM *pstElem;
ZULONG dwData;
ZINT iRet;
.....
/* Add an unsigned long value into an element. */
iRet = Xsp_ElemAddUldigit(zMemBufId, pstElem, dwData);

```

3.6.3.27 *Xsp_ElemAddSlDigit*

Adds a signed long value into an element.

```
ZINT Xsp_ElemAddSlDigit(ZSBUFID zMemBufId, ST\_XML\_ELEM *pstElem,
                        ZLONG dwData);
```

[Parameters]

Input parameters:

ZSBUFID zMemBufId
The memory buffer ID.

[ST_XML_ELEM](#) *pstElem
The element.

ZLONG dwData
The signed long value.

Output parameters:

None.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```
ZSBUFID zMemBufId;
ST\_XML\_ELEM *pstElem;
ZLONG dwData;
ZINT iRet;
.....
/* Add a signed long value into an element. */
iRet = Xsp_ElemAddSlDigit(zMemBufId, pstElem, dwData);
```

3.6.3.28 *Xsp_AttrAddData*

Adds a value into an attribute.

```
ZINT Xsp_AttrAddData(ZSBUFID zMemBufId, ST\_XML\_ATTR *pstAttr,
                    ST\_ZOS\_USTR *pstData);
```

[Parameters]

Input parameters:

ZSBUFID zMemBufId
The memory buffer ID.

[ST_XML_ATTR](#) *pstAttr
The attribute.

[ST_ZOS_USTR](#) *pstData
The data.

Output parameters:

None.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```
ZSBUFID zMemBufId;
ST\_XML\_ATTR *pstAttr;
ST\_ZOS\_USTR *pstData;
ZINT iRet;
.....
/* Add a value into an attribute. */
iRet = Xsp_AttrAddData(zMemBufId, pstAttr, pstData);
```

3.6.3.29 *Xsp_AttrAddBool*

Adds a Boolean value into an attribute.

```
ZINT Xsp_AttrAddBool(ZSBUFID zMemBufId, ST\_XML\_ATTR *pstAttr,
                    ZBOOL bTrueFalse);
```

[Parameters]

Input parameters:

ZSBUFID zMemBufId
The memory buffer ID.

[ST_XML_ATTR](#) *pstAttr
The attribute.

ZBOOL bTrueFalse
The Boolean value.

Output parameters:

None.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```
ZSBUFID zMemBufId;
ST_XML_ATTR *pstAttr;
ZBOOL bTrueFalse;
ZINT iRet;
.....
/* Add a Boolean value into an attribute. */
iRet = Xsp_AttrAddBool(zMemBufId, pstAttr, bTrueFalse);
```

3.6.3.30 Xsp_AttrAddUIDigit

Adds an unsigned long value into an element.

```
ZINT Xsp_AttrAddUIDigit(ZSBUFID zMemBufId, ST_XML_ATTR *pstAttr,
                        ZULONG dwData);
```

[Parameters]**Input parameters:**

ZSBUFID zMemBufId
The memory buffer ID.

ST_XML_ATTR *pstAttr
The attribute.

ZULONG dwData
The unsigned long value.

Output parameters:

None.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ZSBUFID zMemBufId;
ST_XML_ATTR *pstAttr;
ZULONG dwData;
ZINT iRet;
.....
/* Add an unsigned long value into an element. */
iRet = Xsp_AttrAddUldigit(zMemBufId, pstAttr, dwData);

```

3.6.3.31 Xsp_AttrAddSlDigit

Adds a signed long value into an attribute.

```

ZINT Xsp_AttrAddSlDigit(ZSBUFID zMemBufId, ST_XML_ATTR *pstAttr,
                        ZLONG dwData);

```

[Parameters]

Input parameters:

ZSBUFID zMemBufId
The memory buffer ID.

ST_XML_ATTR *pstAttr
The attribute.

ZLONG dwData
The signed long value.

Output parameters:

None.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ZSBUFID zMemBufId;
ST_XML_ATTR *pstAttr;
ZLONG dwData;
ZINT iRet;
.....
/* Add an unsigned long value into an element. */
iRet = Xsp_AttrAddSlDigit(zMemBufId, pstAttr, dwData);

```

3.6.4 Decoding Interfaces

3.6.4.1 Xsp_MsgLoadFile

Loads an XML message from a file.

```
ZINT Xsp_MsgLoadFile(ZCHAR *pcFileName, ST\_ZOS\_DBUF **ppstMsgBuf,
                    ST\_XML\_MSG **ppstXmlMsg);
```

[Parameters]

Input parameters:

ZCHAR *pcFileName
The file name.

Output parameters:

[ST_ZOS_DBUF](#) **ppstMsgBuf
The message buffer.

[ST_XML_MSG](#) **ppstXmlMsg
The XML message.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```
ZCHAR *pcFileName;
ST\_ZOS\_DBUF *pstMsgBuf;
ST\_XML\_MSG *pstXmlMsg;
ZINT iRet;
.....
/* Load an XML message from a file. */
iRet = Xsp_MsgLoadFile(pcFileName, &pstMsgBuf, &pstXmlMsg);
```

3.6.4.2 Xsp_MsgLoadData

Loads an XML message from a string.

```
ZINT Xsp_MsgLoadData(ST\_ZOS\_USTR *pstData, ST\_XML\_MSG **ppstXmlMsg);
```

[Parameters]

Input parameters:

[ST_ZOS_USTR](#) *pstData

The string.

Output parameters:

[ST_ZOS_DBUF](#) **ppstMsgBuf

The message buffer.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```
ST\_ZOS\_USTR *pstData;
ST\_ZOS\_DBUF *pstMsgBuf;
ZINT iRet;

.....
/* Load an XML message from a string. */
iRet = MsgLoadData(pstData, pstMsgBuf);
```

3.6.4.3 Xsp_DocGetRoot

Gets the root element of a document.

```
ZINT Xsp_DocGetRoot(ST\_XML\_MSG *pstMsg, ST\_XSP\_NS *pstNs,
                   ST\_XML\_ELEM **ppstRoot);
```

[Parameters]

Input parameters:

[ST_XML_MSG](#) *pstMsg

An XML message which contains the XML document.

[ST_XSP_NS](#) *pstNs

The namespace.

Output parameters:

[ST_XML_ELEM](#) **ppstRoot

The root element.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST\_XML\_MSG *pstMsg;
ST\_XSP\_NS *pstNs;
ST\_XML\_ELEM *pstRoot;
ZINT iRet;
.....
/* Get the root element of a document. */
iRet = Xsp_DocGetRoot(pstMsg, pstNs, &pstRoot);

```

3.6.4.4 Xsp_DocGetNs

Gets the namespace of a document.

```
ZINT Xsp_DocGetNs(ST\_XML\_MSG *pstMsg, ST\_XSP\_NS *pstNs);
```

[Parameters]**Input parameters:**

```
ST\_XML\_MSG *pstMsg
```

An XML message which contains the XML document.

Output parameters:

```
ST\_XSP\_NS *pstNs
```

The namespace.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST\_XML\_MSG *pstMsg;
ST\_XSP\_NS *pstNs;
.....
/* Get the namespace of a document. */
iRet = Xsp_DocGetNs(pstMsg, pstNs);

```

3.6.4.5 Xsp_ElemIsEmpty

Checks if an element is empty.

```
ZBOOL Xsp_ElemIsEmpty(ST\_XML\_ELEM *pstElem);
```

[Parameters]**Input parameters:**

[ST_XML_ELEM](#) *pstElem
The element to check.

Output parameters:

None.

[Return value]

Returns ZTRUE if it is empty, or ZFALSE if it is not.

[Example]

```
ST\_XML\_ELEM *pstElem;  
ZBOOL ibool;  
  
.....  
/* Check if an element is empty. */  
ibool = Xsp_ElemIsEmpty(pstElem);
```

3.6.4.6 Xsp_ElemGetName

Gets the QName from an element.

```
ZINT Xsp_ElemGetName(ST\_XML\_ELEM *pstElem, ST\_XML\_QNAME **ppstQName);
```

[Parameters]**Input parameters:**

[ST_XML_ELEM](#) *pstElem
The element.

Output parameters:

[ST_XML_QNAME](#) **ppstQName
The QName.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST_XML_ELEM *pstElem;
ST_XML_QNAME *pstQName;
ZINT iRet;
.....
/* Get the QName from an element. */
iRet = Xsp_ElemGetName(pstElem, &pstQName);

```

3.6.4.7 Xsp_ElemGetNameId

Gets the namespace and name ID of an element.

```

ZINT Xsp_ElemGetNameId(ST_XML_ELEM *pstElem, ST_XSP_NS *pstNs,
                      ZULONG *pdwNameId);

```

[Parameters]

Input parameters:

```

ST_XML_ELEM *pstElem

```

The element.

```

ST_XSP_NS *pstNs

```

The namespace.

Output parameters:

```

ZULONG *pdwNameId

```

The name ID.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST_XML_ELEM *pstElem;
ST_XSP_NS *pstNs;
ZULONG dwNameId;
ZINT iRet;
.....
/* Get the name ID of an element namespace. */
iRet = Xsp_ElemGetNameId(pstElem, pstNs, &dwNameId);

```

3.6.4.8 *Xsp_ElemNsGetNameId*

Gets the namespace ID and name ID of an element.

```
ZBOOL Xsp_ElemNsGetNameId(ST_XML_ELEM *pstElem, ST_XSP_NS *pstNs,  
                          ZULONG *pdwNsId, ZULONG *pdwNameId);
```

[Parameters]

Input parameters:

ST_XML_ELEM *pstElem

The element.

ST_XSP_NS *pstNs

The namespace.

Output parameters:

ZULONG *pdwNsId

The namespace ID.

ZULONG *pdwNameId

The name ID.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```
ST_XML_ELEM *pstElem;  
ST_XSP_NS *pstNs;  
ZULONG dwNsId;  
ZULONG dwNameId;  
ZINT iRet;  
.....  
/* Get the namespace ID and name ID of an element. */  
iRet = Xsp_ElemNsGetNameId(pstElem, pstNs, &dwNsId, &dwNameId);
```

3.6.4.9 *Xsp_ElemGetChild*

Finds an element's child element by its name.

```
ZINT Xsp_ElemGetChild(ST_XML_ELEM *pstElem, ST_ZOS_USTR *pstName,
                     ST_XML_ELEM **ppstChild);
```

[Parameters]

Input parameters:

ST_XML_ELEM *pstElem

The element.

ST_ZOS_USTR *pstName

The name of the child element.

Output parameters:

ST_XML_ELEM **ppstChild

The child element.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```
ST_XML_ELEM *pstElem;
ST_ZOS_USTR *pstName;
ST_XML_ELEM *pstChild;
ZINT iRet;
.....
/* Find an element's child element by its name. */
iRet = Xsp_ElemGetChild(pstElem, pstName, &pstChild);
```

3.6.4.10 Xsp_ElemGetNsChild

Gets a child element of an element by its name. Here the namespace is the default one.

```
ZINT Xsp_ElemGetNsChild(ST_XML_ELEM *pstElem, ST_XSP_NS *pstNs,
                       ZULONG dwNameId, ST_XML_ELEM **ppstChild);
```

[Parameters]

Input parameters:

[ST_XML_ELEM](#) *pstElem

The element.

[ST_XSP_NS](#) *pstNs

The namespace.

ZULONG dwNameId

The name ID of the child element.

Output parameters:

[ST_XML_ELEM](#) **ppstChild

The child element.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST\_XML\_ELEM *pstElem;
ST\_XSP\_NS *pstNs;
ZULONG dwNameId;
ST\_XML\_ELEM *pstChild;
ZINT iRet;
.....
/* Get the first child element of an element. */
iRet = Xsp_ElemGetNsChild(pstElem, pstNs, dwNameId, &pstChild);

```

3.6.4.11 Xsp_ElemNsGetChild

Gets a child element of an element by its name. Here the namespace is specified by the namespace ID.

```

ZINT Xsp_ElemNsGetChild(ST\_XML\_ELEM *pstElem, ST\_XSP\_NS *pstNs,
                        ZULONG dwNsId, ZULONG dwNameId, ST\_XML\_ELEM **ppstChild);

```

[Parameters]

Input parameters:

[ST_XML_ELEM](#) *pstElem

The element.

[ST_XSP_NS](#) *pstNs

The namespace.

ZULONG dwNsId

The namespace ID.

ZULONG dwNameId

The name ID of the child element.

Output parameters:

[ST_XML_ELEM](#) **ppstChild

The child element.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST\_XML\_ELEM *pstElem;
ST\_XSP\_NS *pstNs;
ZULONG dwNsId;
ZULONG dwNameId;
ST\_XML\_ELEM *pstChild;
ZINT iRet;
.....
/* Get a child element of an element by its name. */
iRet = Xsp_ElemNsGetChild(pstElem, pstNs, dwNsId, dwNameId, &pstChild);

```

3.6.4.12 Xsp_ElemGetFristChild

Gets the current element's first child element.

```

ZINT Xsp_ElemGetFristChild(ST\_XML\_ELEM *pstElem,
                          ST\_XML\_ELEM **ppstChild);

```

[Parameters]

Input parameters:

[ST_XML_ELEM](#) *pstElem

The current element.

Output parameters:

[ST_XML_ELEM](#) **ppstChild

The first child element.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST\_XML\_ELEM *pstElem;
ST\_XML\_ELEM *ppstChild;
ZINT iRet;

.....
/* Get the current element's first child element. */
iRet = Xsp_ElemGetFristChild(pstElem, &ppstChild);

```

3.6.4.13 *Xsp_ElemGetLastChild*

Gets the current element's last child element.

```

ZINT Xsp_ElemGetLastChild(ST\_XML\_ELEM *pstElem,
                          ST\_XML\_ELEM **ppstChild);

```

[Parameters]

Input parameters:

[ST_XML_ELEM](#) *pstElem

The current element.

Output parameters:

[ST_XML_ELEM](#) **ppstChild

The last child element.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST_XML_ELEM *pstElem;
ST_XML_ELEM *pstChild;
ZINT iRet;
.....
/* Get the current element's last child element. */
iRet = Xsp_ElemGetLastChild(pstElem, &pstChild);

```

3.6.4.14 *Xsp_ElemGetNextSibling*

Gets the current element's next sibling element.

```

ZINT Xsp_ElemGetNextSibling(ST_XML_ELEM *pstElem,
                            ST_XML_ELEM **ppstSibling);

```

[Parameters]

Input parameters:

```

ST_XML_ELEM *pstElem

```

The current element.

Output parameters:

```

ST_XML_ELEM **ppstSibling

```

The next sibling element.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST_XML_ELEM *pstElem;
ST_XML_ELEM *pstSibling;
ZINT iRet;
.....
/* Get the current element's next sibling element. */
iRet = Xsp_ElemGetNextSibling(pstElem, &pstSibling);

```

3.6.4.15 *Xsp_ElemGetPrevSibling*

Gets the current element's previous sibling.

```

ZINT Xsp_ElemGetPrevSibling(ST_XML_ELEM *pstElem,
                             ST_XML_ELEM **ppstSibling);

```

[Parameters]

Input parameters:

[ST_XML_ELEM](#) *pstElem
The current element.

Output parameters:

[ST_XML_ELEM](#) **ppstSibling
The previous sibling element.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST\_XML\_ELEM *pstElem;
ST\_XML\_ELEM *pstSibling;
ZINT iRet;
.....
/* Get the current element's previous sibling. */
iRet = Xsp_ElemGetPrevSibling(pstElem, &pstSibling);

```

3.6.4.16 Xsp_ElemGetFirstNsChild

Gets the first namespace child element. Here the namespace is the default one.

```

ZINT Xsp_ElemGetFirstNsChild(ST\_XML\_ELEM *pstElem, ST\_XSP\_NS *pstNs,
                             ZULONG dwNameId, ST\_XML\_ELEM **ppstChild);

```

[Parameters]

Input parameters:

[ST_XML_ELEM](#) *pstElem
The current element.

[ST_XSP_NS](#) *pstNs
The namespace.

ZULONG dwNameId
The name ID.

Output parameters:

[ST_XML_ELEM](#) **ppstChild
 The first namespace child element.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST\_XML\_ELEM *pstElem;
ST\_XSP\_NS *pstNs;
ZULONG dwNameId;
ST\_XML\_ELEM *pstChild;
ZINT iRet;
.....
/* Get the first namespace child element. */
iRet = Xsp_ElemGetFirstNsChild(pstElem, pstNs, dwNameId, &pstChild);

```

3.6.4.17 Xsp_ElemGetLastNsChild

Gets the last namespace child element. Here the namespace is the default one.

```

ZINT Xsp_ElemGetLastNsChild(ST\_XML\_ELEM *pstElem, ST\_XSP\_NS *pstNs,
                           ZULONG dwNameId, ST\_XML\_ELEM **ppstChild);

```

[Parameters]

Input parameters:

[ST_XML_ELEM](#) *pstElem
 The current element.

[ST_XSP_NS](#) *pstNs
 The namespace.

ZULONG dwNameId
 The name ID of the namespace child element.

Output parameters:

[ST_XML_ELEM](#) **ppstChild
 The last namespace child element.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST\_XML\_ELEM *pstElem;
ST\_XSP\_NS *pstNs;
ZULONG dwNameId;
ST\_XML\_ELEM *pstChild;
ZINT iRet;
.....
/* Get the last namespace child element. */
iRet = Xsp_ElemGetLastNsChild(pstElem, pstNs, dwNameId, &pstChild);

```

3.6.4.18 Xsp_ElemGetNextNsSibling

Gets the next namespace sibling element of the current element. Here the namespace is the default one.

```

ZINT Xsp_ElemGetNextNsSibling(ST\_XML\_ELEM *pstElem, ST\_XSP\_NS *pstNs,
                             ZULONG dwNameId, ST\_XML\_ELEM **ppstSibling);

```

[Parameters]

Input parameters:

[ST_XML_ELEM](#) *pstElem

The current element.

[ST_XSP_NS](#) *pstNs

The namespace.

ZULONG dwNameId

The name ID of the next namespace sibling element.

Output parameters:

[ST_XML_ELEM](#) **ppstSibling

The next namespace sibling element.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST_XML_ELEM *pstElem;
ST_XSP_NS *pstNs;
ZULONG dwNameId;
ST_XML_ELEM *pstSibling;
ZINT iRet;
.....
/* Get the next namespace sibling element of the current element. */
iRet = Xsp_ElemGetNextNsSibling(pstElem, pstNs, dwNameId, &pstSibling);

```

3.6.4.19 Xsp_ElemGetPrevNsSibling

Gets the previous namespace sibling element of the current element. Here the namespace is the default one.

```

ZINT Xsp_ElemGetPrevNsSibling(ST_XML_ELEM *pstElem, ST_XSP_NS *pstNs,
                             ZULONG dwNameId, ST_XML_ELEM **ppstSibling);

```

[Parameters]

Input parameters:

[ST_XML_ELEM](#) *pstElem
The current element.

[ST_XSP_NS](#) *pstNs
The namespace.

ZULONG dwNameId
The name ID of the previous namespace sibling element.

Output parameters:

[ST_XML_ELEM](#) **ppstSibling
The previous namespace sibling element.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST_XML_ELEM *pstElem;
ST_XSP_NS *pstNs;
ZULONG dwNameId;
ST_XML_ELEM *pstSibling;
ZINT iRet;
.....
/* Get the previous namespace sibling element of the current element. */
iRet = Xsp_ElemGetPrevNsSibling(pstElem, pstNs, dwNameId, &pstSibling);

```

3.6.4.20 Xsp_ElemNsGetFirstChild

Gets the first namespace child element of the current element. Here the namespace is specified by the namespace ID.

```

ZINT Xsp_ElemNsGetFirstChild(ST_XML_ELEM *pstElem, ST_XSP_NS *pstNs,
                            ZULONG dwNsId, ZULONG dwNameId, ST_XML_ELEM **ppstChild);

```

[Parameters]

Input parameters:

[ST_XML_ELEM](#) *pstElem
The current element.

[ST_XSP_NS](#) *pstNs
The namespace.

ZULONG dwNsId
The namespace ID.

ZULONG dwNameId
The name ID of the first namespace child element.

Output parameters:

[ST_XML_ELEM](#) **ppstChild
The first namespace child element.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST\_XML\_ELEM *pstElem;
ST\_XSP\_NS *pstNs;
ZULONG dwNsId;
ZULONG dwNameId;
ST\_XML\_ELEM *pstChild;
ZINT iRet;
.....
/* Get the first namespace child element of the current element. */
iRet = Xsp_ElemNsGetFirstChild(pstElem, pstNs, dwNsId, dwNameId, &pstChild);

```

3.6.4.21 Xsp_ElemNsGetLastChild

Gets the last namespace child element of the current element. Here the namespace is specified by the namespace ID.

```

ZINT Xsp_ElemNsGetLastChild(ST\_XML\_ELEM *pstElem, ST\_XSP\_NS *pstNs,
                            ZULONG dwNsId, ZULONG dwNameId, ST\_XML\_ELEM **ppstChild);

```

[Parameters]

Input parameters:

[ST_XML_ELEM](#) *pstElem
The current element.

[ST_XSP_NS](#) *pstNs
The namespace.

ZULONG dwNsId
The namespace ID.

ZULONG dwNameId
The name ID of the last namespace child element.

Output parameters:

[ST_XML_ELEM](#) **ppstChild
The last namespace child element.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

    ST\_XML\_ELEM *pstElem;
    ST\_XSP\_NS *pstNs;
    ZULONG dwNsId;
    ZULONG dwNameId;
    ST\_XML\_ELEM *pstChild;
    ZINT iRet;
    .....
    /* Get the last namespace child element of the current element. */
    iRet = Xsp_ElemNsGetLastChild(pstElem, pstNs, dwNsId, dwNameId, &pstChild);

```

3.6.4.22 Xsp_ElemNsGetNextSibling

Gets the next namespace sibling element of the current element. Here the namespace is specified by the namespace ID.

```

ZINT Xsp_ElemNsGetNextSibling(ST\_XML\_ELEM *pstElem, ST\_XSP\_NS *pstNs,
                             ZULONG dwNsId, ZULONG dwNameId, ST\_XML\_ELEM **ppstSibling);

```

[Parameters]

Input parameters:

[ST_XML_ELEM](#) *pstElem
The current element.

[ST_XSP_NS](#) *pstNs
The namespace.

ZULONG dwNsId
The namespace ID.

ZULONG dwNameId
The name ID of the next namespace sibling element.

Output parameters:

[ST_XML_ELEM](#) **ppstChild
The next namespace sibling element.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST\_XML\_ELEM *pstElem;
ST\_XSP\_NS *pstNs;
ZULONG dwNsId;
ZULONG dwNameId;
ST\_XML\_ELEM *pstChild;
ZINT iRet;
.....
/* Get the next namespace sibling element of the current element. */
iRet = Xsp_ElemNsGetNextSibling(pstElem, pstNs, dwNsId, dwNameId, &pstChild);

```

3.6.4.23 Xsp_ElemNsGetPrevSibling

Gets the previous namespace sibling element of the current element. Here the namespace is specified by the namespace ID.

```

ZINT Xsp_ElemNsGetPrevSibling(ST\_XML\_ELEM *pstElem, ST\_XSP\_NS *pstNs,
                             ZULONG dwNsId, ZULONG dwNameId, ST\_XML\_ELEM **ppstSibling);

```

[Parameters]

Input parameters:

[ST_XML_ELEM](#) *pstElem
The current element.

[ST_XSP_NS](#) *pstNs
The namespace.

ZULONG dwNsId
The namespace ID.

ZULONG dwNameId
The name ID of the previous namespace sibling element.

Output parameters:

[ST_XML_ELEM](#) **ppstChild
The previous namespace sibling element.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST_XML_ELEM *pstElem;
ST_XSP_NS *pstNs;
ZULONG dwNsId;
ZULONG dwNameId;
ST_XML_ELEM *pstChild;
ZINT iRet;
.....
/* Get the previous namespace sibling element of the current element. */
iRet = Xsp_ElemNsGetPrevSibling(pstElem, pstNs, dwNsId, dwNameId, &pstChild);

```

3.6.4.24 Xsp_ElemGetFirstAttr

Gets the first attribute of the current element.

```
ZINT Xsp_ElemGetFirstAttr(ST_XML_ELEM *pstElem, ST_XML_ATTR **ppstAttr);
```

[Parameters]

Input parameters:

```
ST_XML_ELEM *pstElem
```

The current element.

Output parameters:

```
ST_XML_ATTR **ppstAttr
```

The first attribute.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST_XML_ELEM *pstElem;
ST_XML_ATTR *pstAttr;
ZINT iRet;
.....
/* Get the first attribute of the current element. */
iRet = Xsp_ElemGetFirstAttr(pstElem, &pstAttr);

```

3.6.4.25 Xsp_ElemGetLastAttr

Gets the last attribute of the current element.

```
ZINT Xsp_ElemGetLastAttr(ST\_XML\_ELEM *pstElem, ST\_XML\_ATTR **ppstAttr);
```

[Parameters]

Input parameters:

[ST_XML_ELEM](#) *pstElem

The current element.

Output parameters:

[ST_XML_ATTR](#) **ppstAttr

The last attribute.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```
ST\_XML\_ELEM *pstElem;
ST\_XML\_ATTR *pstAttr;
ZINT iRet;
.....
/* Get the last attribute of the current element. */
iRet = Xsp_ElemGetLastAttr(pstElem, &pstAttr);
```

3.6.4.26 *Xsp_ElemGetFirstNsAttr*

Gets the first namespace attribute of the current element. Here the namespace is the default one.

```
ZINT Xsp_ElemGetFirstNsAttr(ST\_XML\_ELEM *pstElem, ST\_XSP\_NS *pstNs,
                             ZULONG dwNameId, ST\_XML\_ATTR **ppstAttr);
```

[Parameters]

Input parameters:

[ST_XML_ELEM](#) *pstElem

The current element.

[ST_XSP_NS](#) *pstNs

The namespace.

ZULONG dwNameId

The name ID of the attribute.

Output parameters:

[ST_XML_ATTR](#) **ppstAttr
The first namespace attribute.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST\_XML\_ELEM *pstElem;
ST\_XSP\_NS *pstNs;
ZULONG dwNameId;
ST\_XML\_ATTR *pstAttr;
ZINT iRet;
.....
/* Get the first namespace attribute of the current element. *
iRet = Xsp_ElemGetFirstNsAttr(pstElem, pstNs, dwNameId, &pstAttr);

```

3.6.4.27 Xsp_ElemGetLastNsAttr

Gets the last namespace attribute of the current element. Here the namespace is the default one.

```

ZINT Xsp_ElemGetLastNsAttr(ST\_XML\_ELEM *pstElem, ST\_XSP\_NS *pstNs,
                          ZULONG dwNameId, ST\_XML\_ATTR **ppstAttr);

```

[Parameters]**Input parameters:**

[ST_XML_ELEM](#) *pstElem
The current element.

[ST_XSP_NS](#) *pstNs
The namespace.

ZULONG dwNameId
The name ID of the attribute.

Output parameters:

[ST_XML_ATTR](#) **ppstAttr
The last namespace attribute.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST\_XML\_ELEM *pstElem;
ST\_XSP\_NS *pstNs;
ZULONG dwNameId;
ST\_XML\_ATTR *pstAttr;
ZINT iRet;
.....
/* Get the last namespace attribute of the current element. */
iRet = Xsp_ElemGetLastNsAttr(pstElem, pstNs, dwNameId, &pstAttr);

```

3.6.4.28 Xsp_ElemNsGetFirstAttr

Gets the first namespace attribute of the current element. Here the namespace is specified by the namespace ID.

```

ZINT Xsp_ElemNsGetFirstAttr(ST\_XML\_ELEM *pstElem, ZULONG dwNsId,
                            ZULONG dwNameId, ST\_XML\_ATTR **ppstAttr);

```

[Parameters]

Input parameters:

```

ST\_XML\_ELEM *pstElem
The current element.

ZULONG dwNsId
The namespace ID.

ZULONG dwNameId
Name ID of the attribute.

```

Output parameters:

```

ST\_XML\_ATTR **ppstAttr
The first namespace attribute.

```

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST\_XML\_ELEM *pstElem;
ZULONG dwNsId;
ZULONG dwNameId;
ST\_XML\_ATTR *pstAttr;
ZINT iRet;
.....
/* Get the first namespace attribute of the current element. */
iRet = Xsp_ElemNsGetFirstAttr(pstElem, dwNsId, dwNameId, &pstAttr);

```

3.6.4.29 Xsp_ElemNsGetLastAttr

Gets the last namespace attribute of the current element. Here the namespace is specified by the namespace ID.

```

ZINT Xsp_ElemNsGetLastAttr(ST\_XML\_ELEM *pstElem, ZULONG dwNsId,
                           ZULONG dwNameId, ST\_XML\_ATTR **ppstAttr);

```

[Parameters]

Input parameters:

[ST_XML_ELEM](#) *pstElem
The current element.

ZULONG dwNsId
The namespace ID.

ZULONG dwNameId
Name ID of the attribute.

Output parameters:

[ST_XML_ATTR](#) **ppstAttr
The last namespace attribute.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

    ST\_XML\_ELEM *pstElem;
    ZULONG dwNsId;
    ZULONG dwNameId;
    ST\_XML\_ATTR *pstAttr;
    ZINT iRet;
    .....
    /* Get the last namespace attribute of the current element. */
    iRet = Xsp_ElemNsGetLastAttr(pstElem, dwNsId, dwNameId, &pstAttr);

```

3.6.4.30 Xsp_ElemGetAttr

Finds an attribute of the current element by its name.

```

    ZINT Xsp_ElemGetAttr(ST\_XML\_ELEM *pstElem, ST\_ZOS\_USTR *pstName,
                        ST\_XML\_ATTR **ppstAttr);

```

[Parameters]

Input parameters:

[ST_XML_ELEM](#) *pstElem
The current element.

[ST_ZOS_USTR](#) *pstName
Name of the attribute.

Output parameters:

[ST_XML_ATTR](#) **ppstAttr
The attribute.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

    ST\_XML\_ELEM *pstElem;
    ST\_ZOS\_USTR *pstName;
    ST\_XML\_ATTR *pstAttr;
    ZINT iRet;
    .....
    /* Find an attribute of the current element by its name. */
    iRet = Xsp_ElemGetAttr(pstElem, pstName, &pstAttr);

```

3.6.4.31 *Xsp_ElemGetAttrX*

Finds an attribute of the current element by its QName.

```
ZINT Xsp_ElemGetAttrX(ST_XML_ELEM *pstElem, ST_XML_QNAME *pstQName,
                     ST_XML_ATTR **ppstAttr);
```

[Parameters]

Input parameters:

ST_XML_ELEM *pstElem

The current element.

ST_XML_QNAME *pstQName

The QName.

Output parameters:

ST_XML_ATTR **ppstAttr

The attribute.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```
ST_XML_ELEM *pstElem;
ST_XML_QNAME *pstQName;
ST_XML_ATTR *pstAttr;
ZINT iRet;
.....
/* Find an attribute of the current element by its QName. */
iRet = Xsp_ElemGetAttrX(pstElem, pstQName, &pstAttr);
```

3.6.4.32 *Xsp_ElemGetAttrId*

Gets an attribute of the current element by its name ID. Here the namespace is the default one.

```
ZINT Xsp_ElemGetAttrId(ST_XML_ELEM *pstElem, ST_XSP_NS *pstNs,
                       ZULONG dwNameId, ST_XML_ATTR **ppstAttr);
```

[Parameters]

Input parameters:

[ST_XML_ELEM](#) *pstElem

The current element.

[ST_XSP_NS](#) *pstNs

The namespace.

ZULONG dwNameId

Name ID of the attribute.

Output parameters:

[ST_XML_ATTR](#) **ppstAttr

The attribute.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST\_XML\_ELEM *pstElem;
ST\_XSP\_NS *pstNs;
ZULONG dwNameId;
ST\_XML\_ATTR *pstAttr;
ZINT iRet;
.....
/* Get an attribute of the current element by its name ID. */
iRet = Xsp_ElemGetAttrId(pstElem, pstNs, dwNameId, &pstAttr);

```

3.6.4.33 Xsp_ElemNsGetAttrId

Finds an attribute of the current element by its name. Here the namespace is specified by the namespace ID.

```

ZINT Xsp_ElemNsGetAttrId(ST\_XML\_ELEM *pstElem, ZULONG dwNsId,
                        ZULONG dwNameId, ST\_XML\_ATTR **ppstAttr);

```

[Parameters]

Input parameters:

[ST_XML_ELEM](#) *pstElem

The current element.

ZULONG dwNsId

The namespace ID.

ZULONG dwNameId

The name ID of the attribute.

Output parameters:

[ST_XML_ATTR](#) **ppstAttr

The attribute.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST\_XML\_ELEM *pstElem;
ZULONG dwNsId;
ZULONG dwNameId;
ST\_XML\_ATTR *pstAttr;
ZINT iRet;
.....
/* Find an attribute of the current element by its name. */
iRet = Xsp_ElemNsGetAttrId(pstElem, dwNsId, dwNameId, &pstAttr);

```

3.6.4.34 Xsp_ElemGetAttrVal

Gets the attribute value by its name from an element.

```

ZINT Xsp_ElemGetAttrVal(ST\_XML\_ELEM *pstElem, ST\_ZOS\_USTR *pstName,
ST\_ZOS\_USTR **ppstVal);

```

[Parameters]

Input parameters:

[ST_XML_ELEM](#) *pstElem

The element.

[ST_ZOS_USTR](#) *pstName

Name of the attribute.

Output parameters:

[ST_ZOS_USTR](#) **ppstVal
The attribute value.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST\_XML\_ELEM *pstElem;
ST\_ZOS\_USTR *pstName;
ST\_ZOS\_USTR *pstVal;
ZINT iRet;

.....

/* Get the attribute value from an element. */
iRet = Xsp_ElemGetAttrVal(pstElem, pstName, &pstVal);

```

3.6.4.35 Xsp_ElemGetAttrValX

Gets the attribute value by its QName from the an element.

```

ZINT Xsp_ElemGetAttrValX(ST\_XML\_ELEM *pstElem, ST\_XML\_QNAME *pstQName,
ST\_ZOS\_USTR **ppstVal);

```

[Parameters]**Input parameters:**

[ST_XML_ELEM](#) *pstElem
The element.

[ST_XML_QNAME](#) *pstQName
The QName of the attribute.

Output parameters:

[ST_ZOS_USTR](#) **ppstVal
The attribute value.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST_XML_ELEM *pstElem;
ST_XML_QNAME *pstQName;
ST_ZOS_USTR *pstVal;
ZINT iRet;

.....

/* Get the attribute value by its QName from the an element. */
iRet = Xsp_ElemGetAttrValX(pstElem, pstQName, &pstVal);

```

3.6.4.36 Xsp_ElemGetAttrIdVal

Gets an attribaute value by its name ID from an element.

```

ZINT Xsp_ElemGetAttrIdVal(ST_XML_ELEM *pstElem, ST_XSP_NS *pstNs,
                          ZULONG dwNameId, ST_ZOS_USTR **ppstVal);

```

[Parameters]

Input parameters:

[ST_XML_ELEM](#) *pstElem

The element.

[ST_XSP_NS](#) *pstNs

The namespace

ZULONG dwNameId

The name ID.

Output parameters:

[ST_ZOS_USTR](#) **ppstVal

The attribute value.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST_XML_ELEM *pstElem;
ST_XSP_NS *pstNs;
ZULONG dwNameId;
ST_ZOS_USTR *pstVal;
ZINT iRet;
.....
/* Get an attribaute value by its name ID from an element. */
iRet = Xsp_ElemGetAttrIdVal(pstElem, pstNs, dwNameId, &pstVal);

```

3.6.4.37 Xsp_ElemGetAttrIdValId

Gets an attribute value ID by its name ID from an element.

```

ZINT Xsp_ElemGetAttrIdValId(ST_XML_ELEM *pstElem, ST_XSP_NS *pstNs,
                           ZULONG dwNameId, ZULONG *pdwValId);

```

[Parameters]

Input parameters:

[ST_XML_ELEM](#) *pstElem

The element.

[ST_XSP_NS](#) *pstNs

The namespace

ZULONG dwNameId

The name ID.

Output parameters:

ZULONG *pdwValId

The value ID.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST_XML_ELEM *pstElem;
ST_XSP_NS *pstNs;
ZULONG dwNameId;
ZULONG dwValId;
ZINT iRet;
.....
/* Get an attribute value ID by its name ID from an element. */
iRet = Xsp_ElemGetAttrIdValId(pstElem, pstNs, dwNameId, &dwValId);

```

3.6.4.38 Xsp_ElemGetAttrIdBool

Gets an attribute Boolean value by its name ID from an element.

```

ZINT Xsp_ElemGetAttrIdBool(ST_XML_ELEM *pstElem, ST_XSP_NS *pstNs,
                          ZULONG dwNameId, ZBOOL *pbTrueFalse);

```

[Parameters]

Input parameters:

[ST_XML_ELEM](#) *pstElem

The element.

[ST_XSP_NS](#) *pstNs

The namespace

ZULONG dwNameId

The name ID.

Output parameters:

ZBOOL *pbTrueFalse

The Boolean value.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST_XML_ELEM *pstElem;
ST_XSP_NS *pstNs;
ZULONG dwNameId;
ZBOOL bTrueFalse;
ZINT iRet;
.....
/* Get an attribute Boolean value by its name ID from an element. */
iRet = Xsp_ElemGetAttrIdBool(pstElem, pstNs, dwNameId, &bTrueFalse);

```

3.6.4.39 Xsp_ElemGetAttrIdUldigit

Gets an attribute value of unsigned long by its name ID from an element.

```

ZINT Xsp_ElemGetAttrIdUldigit(ST_XML_ELEM *pstElem, ST_XSP_NS *pstNs,
                             ZULONG dwNameId, ZULONG *pdwData);

```

[Parameters]

Input parameters:

[ST_XML_ELEM](#) *pstElem

The element.

[ST_XSP_NS](#) *pstNs

The namespace

ZULONG dwNameId

The name ID.

Output parameters:

ZULONG *pdwData

The unsigned long value.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST_XML_ELEM *pstElem;
ST_XSP_NS *pstNs;
ZULONG dwNameId;
ZULONG dwData;
ZINT iRet;
.....
/* Get an attribute value of unsigned long by its name ID from an element. */
iRet = Xsp_ElemGetAttrIdU1Digit(pstElem, pstNs, dwNameId, &dwData);

```

3.6.4.40 Xsp_ElemNsGetAttrIdVal

Gets an attribute value by its name ID from an element. Here the namespace is specified by the namespace ID.

```

ZINT Xsp_ElemNsGetAttrIdVal(ST_XML_ELEM *pstElem, ZULONG dwNsId,
                           ZULONG dwNameId, ST_ZOS_USTR **ppstVal);

```

[Parameters]

Input parameters:

[ST_XML_ELEM](#) *pstElem

The element.

ZULONG dwNsId

The namespace ID.

ZULONG dwNameId

The name ID.

Output parameters:

[ST_ZOS_USTR](#) **ppstVal

The attribute value.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

    ST\_XML\_ELEM *pstElem;
    ZULONG dwNsId;
    ZULONG dwNameId;
    ST\_ZOS\_USTR *pstVal;
    ZINT iRet;
    .....
    /* Get an attribute value by its name ID from an element. */
    iRet = Xsp_ElemNsGetAttrIdVal(pstElem, dwNsId, dwNameId, &pstVal);

```

3.6.4.41 *Xsp_ElemNsGetAttrIdValId*

Gets the attribute value ID by its name ID from an element. Here the namespace is specified by the namespace ID.

```

    ZINT Xsp_ElemNsGetAttrIdValId(ST\_XML\_ELEM *pstElem, ZULONG dwNsId,
                                   ZULONG dwNameId, ZULONG *pdwValId);

```

[Parameters]

Input parameters:

[ST_XML_ELEM](#) *pstElem

The element.

ZULONG dwNsId

The namespace ID.

ZULONG dwNameId

The name ID.

Output parameters:

ZULONG *pdwValId

The value ID.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST_XML_ELEM *pstElem;
ZULONG dwNsId;
ZULONG dwNameId;
ZULONG dwValId;
ZINT iRet;
.....
/* Get the attribute value ID by its name ID from an element. */
iRet = Xsp_ElemNsGetAttrIdValId(pstElem, dwNsId, dwNameId, &dwValId);

```

3.6.4.42 Xsp_ElemNsGetAttrIdBool

Gets an attribute Boolean value by its name ID from an element Here the namespace is specified by the namespace ID.

```

ZINT Xsp_ElemNsGetAttrIdBool(ST_XML_ELEM *pstElem, ZULONG dwNsId,
                             ZULONG dwNameId, ZBOOL *pbTrueFalse);

```

[Parameters]

Input parameters:

```
ST_XML_ELEM *pstElem
```

The element.

```
ZULONG dwNsId
```

The namespace ID.

```
ZULONG dwNameId
```

The name ID.

Output parameters:

```
ZBOOL *pbTrueFalse
```

The attribute Boolean value.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST_XML_ELEM *pstElem;
ZULONG dwNsId;
ZULONG dwNameId;
ZBOOL bTrueFalse;
ZINT iRet;
.....
/* Get an attribute Boolean value by its name ID from an element */
iRet = Xsp_ElemNsGetAttrIdBool(pstElem, dwNsId, dwNameId, &bTrueFalse);

```

3.6.4.43 Xsp_ElemNsGetAttrIdUIDigit

Gets an attribute value of unsigned long by its name ID long from an element. Here the namespace is specified by the namespace ID.

```

ZINT Xsp_ElemNsGetAttrIdUIDigit(ST_XML_ELEM *pstElem, ZULONG dwNsId,
                                ZULONG dwNameId, ZULONG *pdwData);

```

[Parameters]

Input parameters:

[ST_XML_ELEM](#) *pstElem

The element.

ZULONG dwNsId

The namespace ID.

ZULONG dwNameId

The name ID.

Output parameters:

ZULONG *pdwData

The attribute value of unsigned long.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST_XML_ELEM *pstElem;
ZULONG dwNsId;
ZULONG dwNameId;
ZULONG dwData;
ZINT iRet;
.....
/* Get an attribute value of unsigned long by its name ID long from an element. */
iRet = Xsp_ElemNsGetAttrIdUldigit(pstElem, dwNsId, dwNameId, &dwData);

```

3.6.4.44 Xsp_ElemGetData

Gets some char data from an element.

```
ZINT Xsp_ElemGetData(ST_XML_ELEM *pstElem, ST_ZOS_USTR **ppstData);
```

[Parameters]

Input parameters:

[ST_XML_ELEM](#) *pstElem
The element.

Output parameters:

[ST_ZOS_USTR](#) **ppstData
The char data.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST_XML_ELEM *pstElem;
ST_ZOS_USTR *pstData;
ZINT iRet;
.....
/* Get some char data from an element. */
iRet = Xsp_ElemGetData(pstElem, &pstData);

```

3.6.4.45 Xsp_ElemGetDataId

Gets an element char data ID. Here the namespace is the default one.

```
ZINT Xsp_ElemGetDataId(ST\_XML\_ELEM *pstElem, ST\_XSP\_NS *pstNs,  
                      ZULONG *pdwDataId);
```

[Parameters]

Input parameters:

[ST_XML_ELEM](#) *pstElem

The element.

[ST_XSP_NS](#) *pstNs

The namespace.

Output parameters:

ZULONG *pdwDataId

The data ID.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```
ST\_XML\_ELEM *pstElem;  
ST\_XSP\_NS *pstNs;  
ZULONG dwDataId;  
ZINT iRet;  
.....  
/* Get an element char data ID. */  
iRet = Xsp_ElemGetDataId(pstElem, pstNs, &dwDataId);
```

3.6.4.46 Xsp_ElemNsGetDataId

Gets an element char data ID. Here the namespace is specified by the namespace ID.

```
ZINT Xsp_ElemNsGetDataId(ST\_XML\_ELEM *pstElem, ZULONG dwNsId,  
                        ZULONG *pdwDataId);
```

[Parameters]

Input parameters:

[ST_XML_ELEM](#) *pstElem

The element.

ZULONG dwNsId

The namespacef ID.

Output parameters:

ZULONG *pdwDataId

The data ID.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```
ST\_XML\_ELEM *pstElem;
ZULONG dwNsId;
ZULONG dwDataId;
ZINT iRet;
.....
/* Get an element char data ID. */
iRet = Xsp_ElemNsGetDataId(pstElem, dwNsId, &dwDataId);
```

3.6.4.47 *Xsp_ElemGetBool*

Gets a Boolean value from an element.

```
ZINT Xsp_ElemGetBool(ST\_XML\_ELEM *pstElem, ZBOOL *pbTrueFalse);
```

[Parameters]

Input parameters:

[ST_XML_ELEM](#) *pstElem

The element.

Output parameters:

ZBOOL *pbTrueFalse

The Boolean value.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST_XML_ELEM *pstElem;
ZBOOL bTrueFalse;
ZINT iRet;
.....
/* Get a Boolean value from an element. */
iRet = Xsp_ElemGetBool(pstElem, &bTrueFalse);

```

3.6.4.48 Xsp_ElemGetUlDigit

Gets an unsigned long value from an element.

```
ZINT Xsp_ElemGetUlDigit(ST_XML_ELEM *pstElem, ZULONG *pdwData);
```

[Parameters]**Input parameters:**

[ST_XML_ELEM](#) *pstElem
The element.

Output parameters:

ZULONG *pdwData
The unsigned long value.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST_XML_ELEM *pstElem;
ZULONG dwData;
ZINT iRet;
.....
/* Get an unsigned long value from an element. */
iRet = Xsp_ElemGetUlDigit(pstElem, &dwData);

```

3.6.4.49 Xsp_ElemGetSlDigit

Gets a signed long value from an element.

```
ZINT Xsp_ElemGetSlDigit(ST_XML_ELEM *pstElem, ZLONG *pdwData);
```

[Parameters]**Input parameters:**

[ST_XML_ELEM](#) *pstElem
The element.

Output parameters:

ZLONG *pdwData
The signed long value.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```
ST\_XML\_ELEM *pstElem;  
ZLONG dwData;  
ZINT iRet;  
.....  
/* Get a signed long value from an element. */  
iRet = Xsp_ElemGetS1Digit(pstElem, &dwData);
```

3.6.4.50 *Xsp_AttrGetNext*

Gets the next attribute.

```
ZINT Xsp_AttrGetNext(ST\_XML\_ATTR *pstAttr, ST\_XML\_ATTR **ppstNext);
```

[Parameters]**Input parameters:**

[ST_XML_ATTR](#) *pstAttr
The current attribute.

Output parameters:

[ST_XML_ATTR](#) **ppstNext
The next attribute.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST\_XML\_ATTR *pstAttr;
ST\_XML\_ATTR *pstNext;
ZINT iRet;
.....
/* Get the next attribute. */
iRet = Xsp_AttrGetNext(pstAttr, &pstNext);

```

3.6.4.51 *Xsp_AttrGetPrev*

Gets the previous attribute.

```
ZINT Xsp_AttrGetPrev(ST\_XML\_ATTR *pstAttr, ST\_XML\_ATTR **ppstPrev);
```

[Parameters]

Input parameters:

```
ST\_XML\_ATTR *pstAttr
```

The current attribute.

Output parameters:

```
ST\_XML\_ATTR **ppstPrev
```

The previous attribute.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST\_XML\_ATTR *pstAttr;
ST\_XML\_ATTR *pstPrev;
ZINT iRet;
.....
/* Get the previous attribute. */
iRet = Xsp_AttrGetPrev(pstAttr, &pstPrev);

```

3.6.4.52 *Xsp_AttrGetName*

Gets the QName from an attribute.

```
ZINT Xsp_AttrGetName(ST\_XML\_ATTR *pstAttr, ST\_XML\_QNAME **ppstQName);
```

[Parameters]

Input parameters:

[ST_XML_ATTR](#) *pstAttr

The attribute.

Output parameters:

[ST_XML_QNAME](#) **ppstQName

The QName.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST\_XML\_ATTR *pstAttr;
ST\_XML\_QNAME *pstQName;
ZINT iRet;
.....
/* Get the QName from an attribute. */
iRet = Xsp_AttrGetName(pstAttr, pstQName);

```

3.6.4.53 Xsp_AttrGetNameId

Gets the name ID from an attribute. Here the namespace is the default one.

```

ZINT Xsp_AttrGetNameId(ST\_XML\_ATTR *pstAttr, ST\_XSP\_NS *pstNs,
                      ZULONG *pdwNameId);

```

[Parameters]

Input parameters:

[ST_XML_ATTR](#) *pstAttr

The attribute.

[ST_XSP_NS](#) *pstNs

The namespace

Output parameters:

ZULONG *pdwNameId

The name ID.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST\_XML\_ATTR *pstAttr;
ST\_XSP\_NS *pstNs;
ZULONG dwNameId;
ZINT iRet;
.....
/* Get the name ID from an attribute. */
iRet = Xsp_AttrGetNameId(pstAttr, pstNs, &dwNameId);

```

3.6.4.54 Xsp_AttrNsGetNameId

Gets the name ID from an attribute. Here the namespace is specified by the namespace ID.

```

ZINT Xsp_AttrNsGetNameId(ST\_XML\_ATTR *pstAttr, ZULONG dwNsId,
                        ZULONG *pdwNameId);

```

[Parameters]

Input parameters:

[ST_XML_ATTR](#) *pstAttr
The attribute.

ZULONG dwNsId
The namespace ID.

Output parameters:

ZULONG *pdwNameId
The name ID.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

    ST\_XML\_ATTR *pstAttr;
    ZULONG dwNsId;
    ZULONG dwNameId;
    ZINT iRet;
    .....
    /* Get the name ID from an attribute. */
    iRet = Xsp_AttrNsGetNameId(pstAttr, dwNsId, &dwNameId);

```

3.6.4.55 *Xsp_AttrGetData*

Gets data from an attribute.

```

ZINT Xsp_AttrGetData(ST\_XML\_ATTR *pstAttr, ST\_ZOS\_USTR **ppstData);

```

[Parameters]

Input parameters:

```

ST\_XML\_ATTR *pstAttr
    The attribute.

```

Output parameters:

```

ST\_ZOS\_USTR **ppstData
    The data.

```

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST\_XML\_ATTR *pstAttr;
ST\_ZOS\_USTR *pstData;
ZINT iRet;
.....
/* Get data from an attribute. */
iRet = Xsp_AttrGetData(pstAttr, &pstData);

```

3.6.4.56 *Xsp_AttrGetDataId*

Gets the data ID from an attribute. Here the namespace is the default one.

```

ZINT Xsp_AttrGetDataId(ST\_XML\_ATTR *pstAttr, ST\_XSP\_NS *pstNs,
    ZULONG *pdwDataId);

```

[Parameters]**Input parameters:**ST_XML_ATTR *pstAttr

The attribute.

ST_XSP_NS *pstNs

The namespace.

Output parameters:

ZULONG *pdwDataId

The data ID.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST_XML_ATTR *pstAttr;
ST_XSP_NS *pstNs;
ZULONG dwDataId;
ZINT iRet;
.....
/* Get the data ID from an attribute. */
iRet = Xsp_AttrGetDataId(pstAttr, pstNs, &dwDataId);

```

3.6.4.57 Xsp_AttrNsGetDataId

Gets the data ID from an attribute. Here the namespace is specified by the namespace ID.

```

ZINT Xsp_AttrNsGetDataId(ST_XML_ATTR *pstAttr, ZULONG dwNsId,
ZULONG *pdwDataId);

```

[Parameters]**Input parameters:**ST_XML_ATTR *pstAttr

The attribute.

ZULONG dwNsId

The namespace ID.

Output parameters:

```
ZULONG *pdwDataId
```

The data ID.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```
ST\_XML\_ATTR *pstAttr;  
ZULONG dwNsId;  
ZULONG dwDataId;  
ZINT iRet;  
.....  
/* Get the data ID from an attribute. */  
iRet = Xsp_AttrNsGetDataId(pstAttr, dwNsId, &dwDataId);
```

3.6.4.58 *Xsp_AttrGetBool*

Gets a Boolean value from an attribute.

```
ZINT Xsp_AttrGetBool(ST\_XML\_ATTR *pstAttr, ZBOOL *pbTrueFalse);
```

[Parameters]**Input parameters:**

```
ST\_XML\_ATTR *pstAttr
```

The attribute.

Output parameters:

```
ZBOOL *pbTrueFalse
```

The Boolean value.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST_XML_ATTR *pstAttr;
ZBOOL bTrueFalse;
ZINT iRet;
.....
/* Get a Boolean value from an attribute. */
iRet = Xsp_AttrGetBool(pstAttr, &bTrueFalse);

```

3.6.4.59 Xsp_AttrGetUIDigit

Gets an unsigned long value from an attribute.

```
ZINT Xsp_AttrGetUIDigit(ST_XML_ATTR *pstAttr, ZULONG *pdwData);
```

[Parameters]

Input parameters:

```
ST_XML_ATTR *pstAttr
```

The attribute.

Output parameters:

```
ZULONG *pdwData
```

The unsigned long value.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```

ST_XML_ATTR *pstAttr;
ZULONG dwData;
ZINT iRet;
.....
/* Get an unsigned long value from an attribute. */
iRet = Xsp_AttrGetUIDigit(pstAttr, &dwData);

```

3.6.4.60 Xsp_AttrGetSIDigit

Gets a signed long value from an attribute.

```
ZINT Xsp_AttrGetSIDigit(ST_XML_ATTR *pstAttr, ZLONG *pdwData);
```

[Parameters]

Input parameters:

[ST_XML_ATTR](#) *pstAttr
The attribute.

Output parameters:

ZLONG *pdwData
The signed long value.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```
ST\_XML\_ATTR *pstAttr;
ZLONG dwData;
ZINT iRet;
.....
/* Get a signed long value from an attribute. */
iRet = Xsp_AttrGetSlDigit(pstAttr, &dwData);
```

3.6.5 Utility Interfaces

These interfaces are included in xml_util.h.

3.6.5.1 *Xml_GetContentSize*

Gets the content item size by the item type.

```
ZINT Xml_GetContentSize(ZUCHAR ucType);
```

[Parameters]**Input parameters:**

ZUCHAR ucType
The item type.

Output parameters:

None.

[Return value]

Returns the size, or 0.

[Example]

```
ZUCHAR ucType;
.....
/* Get the content item size by the item type. */
Xml_GetContentSize(ucType);
```

3.6.5.2 *Xml_GetContentItem*

Gets the content item from an element.

```
ZINT Xml_GetContentItem(ST_XML_ELEM *pstElem,
                        ST_XML_CONTENT_ITEM **ppstItem);
```

[Parameters]

Input parameters:

[ST_XML_ELEM](#) *pstElem
The element.

Output parameters:

[ST_XML_CONTENT_ITEM](#) **ppstItem
The content item.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```
ST\_XML\_ELEM *pstElem;
ST\_XML\_CONTENT\_ITEM *pstItem;
ZINT iRet;
.....
/* Get the content item from an element. */
iRet = Xml_GetContentItem(pstElem, &pstItem);
```

3.6.6 Config Interfaces

These interfaces are included in xml_cfg.h.

3.6.6.1 *Xml_CfgGetLogLevel*

Gets the log level.

```
ZINT Xml_CfgGetLogLevel (ZULONG *pdwLevel);
```

[Parameters]

Input parameters:

None.

Output parameters:

ZULONG *pdwLevel

The log level.

[Return value]

Returns ZOK.

[Example]

```
ZULONG dwLevel;  
.....  
/* Get the log level. */  
Xml_CfgGetLogLevel (&dwLevel);
```

3.6.6.2 *Xml_CfgSetLogLevel*

Sets the log level.

```
ZINT Xml_CfgSetLogLevel (ZULONG dwLevel);
```

[Parameters]

Input parameters:

ZULONG dwLevel

The log level.

Output parameters:

None.

[Return value]

Returns ZOK.

[Example]

```
ZULONG dwLevel;  
  
.....  
/* Set the log level. */  
Xml_CfgSetLogLevel(dwLevel);
```