

Document Object Model (DOM)

annexe



La représentation des documents avec DOM

Lorsqu'un document doit être analysé chacun est libre de choisir à la fois la méthode mise en œuvre, et la façon de stocker en mémoire le résultat de l'analyse réalisée.

À l'échelle du Web cette liberté confine au chaos en matière de développement. Le W3C a donc proposé un modèle permettant de représenter et manipuler les documents XML et HTML. Ce modèle se nomme DOM.

DOM ne préjuge en rien de la méthode utilisée pour analyser un document mais se contente de définir une interface et un modèle de représentation des informations qui devra être accessible via l'interface.

Le modèle proposé par DOM repose sur une représentation arborescente de la structure des documents analysés. C'est cette représentation arborescente qui devra être manipulable via l'interface de programmation.

Cependant chaque développeur reste libre de stocker l'information comme bon lui semble (une liste chaînée par exemple) tant que l'implantation de l'interface respecte DOM et expose à l'utilisateur la représentation arborescente définie dans la recommandation. Que celle-ci colle à l'implantation ou pas n'a aucune importance pour l'utilisateur de DOM.

La recommandation officielle du W3C et le copyright associé

Dans cette annexe nous allons reprendre la spécification en IDL (Interface Definition Language) de l'API Document Object Model (DOM) Level 3 Core (Annexe F).

Ces éléments, dont certains sont traduits en français, ne sauraient être confondus avec la recommandation officielle qui demeure la seule référence, téléchargeable librement sur les sites précisés ci-dessous.

Pour chaque interface la mention suivante est applicable :

```
/*
 * Copyright (c) 2004 World Wide Web Consortium,
 *
 * (Massachusetts Institute of Technology, European Research Consortium for
 * Informatics and Mathematics, Keio University). All Rights Reserved. This
 * work is distributed under the W3C(r) Software License [1] in the hope that
 * it will be useful, but WITHOUT ANY WARRANTY; without even the implied
 * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 *
 * [1] http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
 */
```

Par ailleurs, le lecteur est invité à prendre connaissance du texte intégral de la licence mentionnée :

W3C® SOFTWARE NOTICE AND LICENSE

<http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231>

This work (and included software, documentation such as READMEs, or other related items) is being provided by the copyright holders under the following license. By obtaining, using and/or copying this work, you (the licensee) agree that you have read, understood, and will comply with the following terms and conditions.

Permission to copy, modify, and distribute this software and its documentation, with or without modification, for any purpose and without fee or royalty is hereby granted, provided that you include the following on ALL copies of the software and documentation or portions thereof, including modifications:

1. The full text of this NOTICE in a location viewable to users of the redistributed or derivative work.
2. Any pre-existing intellectual property disclaimers, notices, or terms and conditions. If none exist, the W3C Software Short Notice should be included (hypertext is preferred, text is permitted) within the body of any redistributed or derivative code.
3. Notice of any changes or modifications to the files, including the date changes were made. (We recommend you provide URIs to the location from which the code is derived.)

THIS SOFTWARE AND DOCUMENTATION IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE OR DOCUMENTATION WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

COPYRIGHT HOLDERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE SOFTWARE OR DOCUMENTATION.

The name and trademarks of copyright holders may NOT be used in advertising or publicity pertaining to the software without specific, written prior permission. Title to copyright in this software and any associated documentation will at all times remain with copyright holders.

▶ <http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/>

▶ <http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/idl/dom.idl>

Types élémentaires

Au-delà des nœuds qui, dans une représentation arborescente vont constituer l'essentiel des objets manipulés, DOM définit quelques types élémentaires, et en réalité un type clé : `DOMString`.

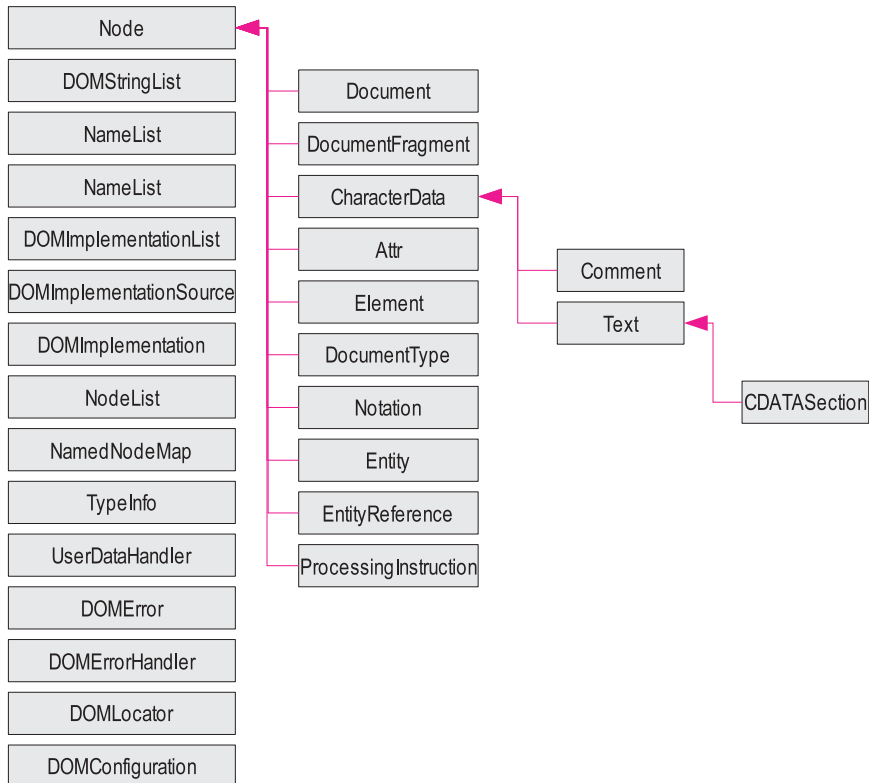
Nombre de méthodes dans l'API DOM vont manipuler des chaînes de caractères. Pour s'assurer la plus large audience, DOM utilise l'encodage UTF-16 pour chaque caractère (qui occupe donc 2 octets). Ce type de caractères est désormais très bien supporté par la plupart des langages, y compris JavaScript.

Par ailleurs DOM propose les types suivants :

- `DOMTimeStamp` pour stocker une date (absolue ou relative). Celle-ci est exprimée en millisecondes ;
- `DOMUserData` qui permet de lier une arborescence DOM à des données utilisateurs extérieures à l'arbre. Ce type de donnée représente un pointeur générique ;
- `DOMObject` pour manipuler des références d'objets DOM.

Interfaces fondamentales

DOM dispose d'implantations dans de nombreux langages, PHP naturellement mais aussi Java et JavaScript entre autres. Le modèle objet n'étant pas nécessairement supporté par tous ces langages, l'API DOM ne définit pas des objets mais des interfaces. Naturellement dans le cas des langages orientés objet, on définira le plus souvent une hiérarchie de classes cohérente avec celles des interfaces DOM.



COMPRENDRE L'OMG IDL

Dans la suite les interfaces de programmation DOM ne seront pas décrites en utilisant la syntaxe de PHP. En effet PHP n'étant pas un langage fortement typé, nos descriptions ne seraient pas assez précises. Toutefois le langage retenu, issu des travaux de l'OMG (Object Management Group) et de CORBA ne devrait pas poser problème.

► http://www.omg.org/technology/documents/formal/corba_2.htm

Dans l'absolu, l'interface Node est la plus importante et est suffisante pour tout faire. Cependant, dans la pratique on utilisera des interfaces plus sophistiquées en fonction de la nature du nœud manipulé comme Element ou Document.

Enfin, il faut noter que DOM est aujourd'hui disponible dans sa troisième mouture, certains attributs ou méthodes sont donc apparus au fil du temps, dans les versions deux ou trois, ces évolutions sont précisées dans les interfaces qui suivent.

Pour plus de détails

Cette annexe reprend les interfaces de l'API DOM de manière succincte. Dans la plupart des cas, les noms des méthodes et des attributs sont caractéristiques du rôle de chacun et se suffisent à eux-mêmes.

Toutefois, certaines interfaces peuvent nécessiter des explications approfondies. Le plus sage est alors de revenir à la recommandation qui explicite pour chaque attribut ou méthode le rôle et les contraintes.

► <http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/>

DOMException

En cas d'échec, lors de situations exceptionnelles, les méthodes de l'API DOM peuvent être amenées à déclencher une exception. DOM ne définit qu'une exception : DOMException (en PHP cette classe dérive naturellement de la classe Exception).

Toutefois l'implantation est libre de déclencher des erreurs additionnelles. En outre, les langages non orientés objet peuvent se contenter d'utiliser le mécanisme traditionnel de traitement des erreurs à base de code de retour.

```
exception DOMException {
    unsigned short    code;
};
```

Tableau C-1 Constantes définies dans DOM (niveau 1)

Nom	Valeur	Signification
INDEX_SIZE_ERR	1	Index ou intervalle invalide.
DOMSTRING_SIZE_ERR	2	Le texte proposé n'est pas stockable dans la chaîne.
HIERARCHY_REQUEST_ERR	3	Nœud impossible à insérer à cet endroit.
WRONG_DOCUMENT_ERR	4	Nœud utilisé dans un document incompatible.
INVALID_CHARACTER_ERR	5	Caractère invalide (par exemple dans un nom d'élément).
NO_DATA_ALLOWED_ERR	6	Aucune donnée ne peut être insérée.
NO_MODIFICATION_ALLOWED_ERR	7	Modification interdite
NOT_FOUND_ERR	8	Nœud introuvable dans ce contexte.
NOT_SUPPORTED_ERR	9	Fonctionnalité non prise en charge par l'implantation.
INUSE_ATTRIBUTE_ERR	10	Ajout d'un attribut déjà associé à un élément par ailleurs.

Tableau C-2 Constantes définies dans DOM (niveau 2)

Nom	Valeur	Signification
INVALID_STATE_ERR	11	L'objet n'est plus utilisable.
SYNTAX_ERR	12	La chaîne manipulée est erronée.
INVALID_MODIFICATION_ERR	13	Modification du type de l'objet interdite.
NAMESPACE_ERR	14	Modification apportée à l'objet non conforme aux contraintes de l'espace de noms.
INVALID_ACCESS_ERR	15	Paramètre ou méthode non pris en charge.

Tableau C-3 Constantes définies dans DOM (niveau 3)

Nom	Valeur	Signification
VALIDATION_ERR	16	Le nœud est invalide après utilisation de <code>insertBefore()</code> ou <code>removeChild()</code> .
TYPE_MISMATCH_ERR	17	Type d'objet non conforme à celui attendu.

DOMStringList– DOM③

Cette interface décrit la manipulation d'une liste de chaînes de caractères. La taille de la liste est disponible via l'attribut `length` et chaque chaîne peut être obtenue individuellement avec la fonction `item()`.

```
interface DOMStringList {
  readonly attribute unsigned long length;
  DOMString item (in unsigned long index);
  boolean contains (in DOMString str);
};
```

NameList– DOM③

L'utilisation de listes dans DOM est assez fréquente, en PHP cela peut paraître déroutant tant la notion de tableau est riche. Ici il s'agit de manipuler une liste ordonnée de couples : nom , espace de noms.

```
interface NameList {
  readonly attribute unsigned long length;
  DOMString getName (in unsigned long index);
  DOMString getNamespaceURI (in unsigned long index);
  boolean contains (in DOMString str);
  boolean containsNS (in DOMString namespaceURI,
                    in DOMString name);
};
```

DOMImplementation

L'interface `DOMImplementation` permet de créer ou de manipuler des informations transversales indépendantes des documents manipulés par ailleurs.

En outre, l'interface `DOMImplementation` permet de s'assurer de la disponibilité de certaines fonctionnalités dans l'implantation DOM utilisée.

```
interface DOMImplementation {
    boolean hasFeature (in DOMString feature,
                      in DOMString version);

    // Méthodes disponibles à compter de DOM niveau 2 :
    DocumentType createDocumentType(in DOMString qualifiedName,
                                   in DOMString publicId,
                                   in DOMString systemId) raises(DOMException);

    Document createDocument (in DOMString namespaceURI,
                             in DOMString qualifiedName,
                             in DocumentType doctype) raises(DOMException);

    // Méthodes disponibles à compter de DOM niveau 3 :
    DOMObject getFeature (in DOMString feature,
                          in DOMString version);
};
```

DOMImplementationSource

Dans le prolongement de l'interface `DOMImplementation`, `DOMImplementationSource` permet d'obtenir une ou plusieurs implantations compatibles avec certaines fonctionnalités minimales.

```
interface DOMImplementationSource {
    DOMImplementation getDOMImplementation (in DOMString features);
    DOMImplementationList getDOMImplementationList (in DOMString features);
};
```

DOMImplementationList

Encore une liste, mais consacrée aux objets `DOMImplementation`, et notamment utilisée dans le cadre de `DOMImplementationSource`.

```
interface DOMImplementationList {
    readonly attribute unsigned long length;
    DOMImplementation item(in unsigned long index);
};
```

Node

L'interface `Node` est naturellement l'interface fondamentale de l'API DOM. Elle fournit les fonctions essentielles pour parcourir et manipuler l'arbre représentant chaque document analysé.

```
interface Node {
    readonly attribute DOMString nodeName;
    attribute DOMString nodeValue;
    // l'affectation et la lecture peuvent lever une
    // exception DOMException
};
```

```

readonly attribute unsigned short nodeType;
readonly attribute Node           parentNode;
readonly attribute NodeList       childNodes;
readonly attribute Node           firstChild;
readonly attribute Node           lastChild;
readonly attribute Node           previousSibling;
readonly attribute Node           nextSibling;
readonly attribute NamedNodeMap   attributes;
// Attributs modifiés pour DOM niveau 2
readonly attribute Document       ownerDocument;
// Attributs disponibles compter de DOM niveau 2
readonly attribute DOMString      namespaceURI;
    attribute DOMString           prefix;
// l'affectation peut lever une exception DOMException
readonly attribute DOMString      localName;
// Attributs disponibles compter de DOM niveau 3
readonly attribute DOMString      baseURI;
    attribute DOMString           textContent;
// l'affectation et la lecture peuvent lever une
// exception DOMException
boolean          hasChildNodes      ();
Node            cloneNode           (in boolean deep);
// Méthodes modifiées pour DOM niveau 3
Node            insertBefore        (in Node newChild,
    in Node refChild) raises(DOMException);
Node            replaceChild        (in Node newChild,
    in Node oldChild) raises(DOMException);
Node            removeChild         (in Node oldChild) raises(DOMException);
Node            appendChild         (in Node newChild) raises(DOMException);
void            normalize           ();
// Méthodes disponibles à compter de DOM niveau 2
boolean         isSupported         (in DOMString feature,
    in DOMString version);
boolean         hasAttributes        ();

// Méthodes disponibles à compter de DOM niveau 3
unsigned short compareDocumentPosition (in Node other) raises(DOMException);
boolean         isSameNode           (in Node other);
DOMString       lookupPrefix         (in DOMString namespaceURI);
boolean         isDefaultNamespace   (in DOMString namespaceURI);
DOMString       lookupNamespaceURI   (in DOMString prefix);
boolean         isEqualNode          (in Node arg);
DOMObject       getFeature           (in DOMString feature,
    in DOMString version);
DOMUserData     setData              (in DOMString key,
    in DOMUserData data,
    in UserDataHandler handler);
DOMUserData     getUserData          (in DOMString key);
};

```

Tableau C-4 Constantes associées aux différents types de nœuds DOM

Type	Code associé
ELEMENT_NODE	1
ATTRIBUTE_NODE	2
TEXT_NODE	3
CDATA_SECTION_NODE	4
ENTITY_REFERENCE_NODE	5
ENTITY_NODE	6
PROCESSING_INSTRUCTION_NODE	7
COMMENT_NODE	8
DOCUMENT_NODE	9
DOCUMENT_TYPE_NODE	10
DOCUMENT_FRAGMENT_NODE	11
NOTATION_NODE	12

Tableau C-5

Nom	Valeur associée
DOCUMENT_POSITION_DISCONNECTED	0x01
DOCUMENT_POSITION_PRECEDING	0x02
DOCUMENT_POSITION_FOLLOWING	0x04
DOCUMENT_POSITION_CONTAINS	0x08
DOCUMENT_POSITION_CONTAINED_BY	0x10
DOCUMENT_POSITION_IMPLEMENTATION_SPECIFIC	0x20

NodeList

NodeList définit une liste ordonnée de nœuds. Ce type de liste est souvent utilisée comme valeur de retour par les méthodes DOM.

```
interface NodeList {
    Node          item(in unsigned long index);
    readonly attribute unsigned long length;
};
```


Document

Autre interface fondamentale de DOM, Document étend l'interface Node pour proposer des méthodes de plus au niveau (comme la création d'éléments ou de sections Text).

```
interface Document : Node {
    readonly attribute DOMImplementation implementation;
    readonly attribute Element documentElement;
    // Attributs modifiés pour DOM niveau 3
    readonly attribute DocumentType doctype;
    // Attributs disponibles à compter de DOM niveau 3
    readonly attribute DOMString inputEncoding;
    readonly attribute DOMString xmlEncoding;
    attribute boolean xmlStandalone;
    // l'affectation peut lever une exception DOMException
    attribute DOMString xmlVersion;
    // l'affectation peut lever une exception DOMException
    attribute boolean strictErrorChecking;
    attribute DOMString documentURI;
    readonly attribute DOMConfiguration domConfig;
    Element createElement (in DOMString tagName) raises(DOMException);
    DocumentFragment createDocumentFragment ();
    Text createTextNode (in DOMString data);
    Comment createComment (in DOMString data);
    CDATASection createCDATASection (in DOMString data) raises(DOMException);
    ProcessingInstruction createProcessingInstruction(in DOMString target,
        in DOMString data) raises(DOMException);
    Attr createAttribute (in DOMString name) raises(DOMException);
    EntityReference createEntityReference (in DOMString name) raises(DOMException);
    NodeList getElementsByTagName (in DOMString tagName);
    // Méthodes disponibles à compter de DOM niveau 2
    Node importNode (in Node importedNode,
        in boolean deep) raises(DOMException);
    Element createElementNS (in DOMString namespaceURI,
        in DOMString qualifiedName) raises(DOMException);
    Attr createAttributeNS (in DOMString namespaceURI,
        in DOMString qualifiedName) raises(DOMException);
    NodeList getElementsByTagNameNS (in DOMString namespaceURI,
        in DOMString localName);
    Element getElementById (in DOMString elementId);
    // Méthodes disponibles à compter de DOM niveau 3 :
    Node adoptNode (in Node source) raises(DOMException);
    void normalizeDocument ();
    Node renameNode (in Node n,
        in DOMString namespaceURI,
        in DOMString qualifiedName) raises(DOMException);
};
```

DocumentFragment

L'interface DocumentFragment permet de manipuler des morceaux de document. Il serait bien sûr possible d'utiliser l'interface Document classique. Cependant, en fonction de l'implantation, celle-ci peut s'avérer gourmande en ressources.

DocumentFragment laisse donc à l'implantation la possibilité de proposer une version allégée, utilisée spécifiquement pour composer des morceaux de documents entre eux.

Cette interface ne définit aucune méthode complémentaire à celles définies dans l'interface Node.

```
interface DocumentFragment : Node {
};
```

NamedNodeMap

Outre les listes, qui permettent d'accéder à des collections d'objets ordonnés, l'interface NamedNodeMap permet de manipuler une collection de nœuds en proposant un accès par leur nom (y compris l'espace de noms).

```
interface NamedNodeMap {
    readonly attribute unsigned long    length;
    Node                               getNamedItem    (in DOMString name);
    Node                               setNamedItem    (in Node arg) raises(DOMException);
    Node                               removeNamedItem (in DOMString name) raises(DOMException);
    Node                               item            (in unsigned long index);
    // Méthodes disponibles à compter de DOM niveau 2
    Node                               getNamedItemNS  (in DOMString namespaceURI,
                                                    in DOMString localName) raises(DOMException);
    Node                               setNamedItemNS  (in Node arg) raises(DOMException);
    Node                               removeNamedItemNS (in DOMString namespaceURI,
                                                         in DOMString localName) raises(DOMException);
};
```

CharacterData

CharacterData représente une interface qu'on pourrait qualifier d'abstraite. En effet, dans un document DOM aucun nœud de ce type n'existe. Par contre, cette interface sera complétée de manière à définir les interfaces Text et Comment notamment.

```
interface CharacterData : Node {
    attribute DOMString    data;
    // l'affectation et la lecture peuvent lever une
    // exception DOMException
    readonly attribute unsigned long    length;
    DOMString                       substringData (in unsigned long offset,
                                                    in unsigned long count) raises(DOMException);
    void                             appendData  (in DOMString arg) raises(DOMException);
    void                             insertData  (in unsigned long offset,
                                                    in DOMString arg) raises(DOMException);
    void                             deleteData  (in unsigned long offset,
                                                    in unsigned long count) raises(DOMException);
    void                             replaceData (in unsigned long offset,
                                                    in unsigned long count,
                                                    in DOMString arg) raises(DOMException);
};
```

Attr

Cette interface permet de manipuler les propriétés d'un nœud correspondant à un attribut dans le document original.

```
interface Attr : Node {
    readonly attribute DOMString      name;
    readonly attribute boolean        specified;
    attribute DOMString              value;
                                     // l'affectation peut lever une exception DOMException
    // Attributs disponibles à partir de DOM niveau 2
    readonly attribute Element        ownerElement;
    // Attributs disponibles à partir de DOM niveau 3
    readonly attribute TypeInfo       schemaTypeInfo;
    readonly attribute boolean        isId;
};
```

Element

Cette interface étend l'interface Node de manière à proposer des fonctions de plus haut niveau pour les nœuds qui s'avèrent être des éléments dans le document original.

```
interface Element : Node {
    readonly attribute DOMString      tagName;
    DOMString                        getAttribute(in DOMString name);
    // Attributs disponibles à partir de DOM niveau 3
    readonly attribute TypeInfo       schemaTypeInfo;
    void                             setAttribute      (in DOMString name,
                                                         in DOMString value) raises(DOMException);
    void                             removeAttribute   (in DOMString name) raises(DOMException);
    Attr                             getAttributeNode  (in DOMString name);
    Attr                             setAttributeNode  (in Attr newAttr) raises(DOMException);
    Attr                             removeAttributeNode(in Attr oldAttr) raises(DOMException);
    NodeList                         getElementsByTagName(in DOMString name);
    // Méthodes disponibles à compter de DOM niveau 2
    DOMString                        getAttributeNS    (in DOMString namespaceURI,
                                                         in DOMString localName) raises(DOMException);
    void                             setAttributeNS    (in DOMString namespaceURI,
                                                         in DOMString qualifiedName,
                                                         in DOMString value) raises(DOMException);
    void                             removeAttributeNS (in DOMString namespaceURI,
                                                         in DOMString localName) raises(DOMException);
    Attr                             getAttributeNodeNS(in DOMString namespaceURI,
                                                         in DOMString localName) raises(DOMException);
    Attr                             setAttributeNodeNS(in Attr newAttr) raises(DOMException);
    NodeList                         getElementsByTagNameNS(in DOMString namespaceURI,
                                                         in DOMString localName) raises(DOMException);
    boolean                          hasAttribute      (in DOMString name);
    boolean                          hasAttributeNS    (in DOMString namespaceURI,
                                                         in DOMString localName) raises(DOMException);
};
```

```
// Méthodes disponibles à compter de DOM niveau 3
void          setIdAttribute      (in DOMString name,
                                in boolean isId) raises(DOMException);
void          setIdAttributeNS   (in DOMString namespaceURI,
                                in DOMString localName,
                                in boolean isId) raises(DOMException);
void          setIdAttributeNode (in Attr idAttr,
                                in boolean isId) raises(DOMException);
};
```

Text

Cette interface permet de manipuler les contenus texte des éléments d'un document.

```
interface Text : CharacterData {
    // Attributs disponibles à partir de DOM niveau 3
    readonly attribute boolean    isElementContentWhitespace;
    readonly attribute DOMString  wholeText;
    // Méthodes disponibles à compter de DOM niveau 3 :
    Text          splitText      (in unsigned long offset) raises(DOMException);
    Text          replaceWholeText (in DOMString content) raises(DOMException);
};
```

Comment

```
interface Comment : CharacterData {
};
```

TypeInfo– DOM^③

L'interface TypeInfo permet de manipuler les types associés aux attributs ou éléments lorsqu'un schéma est associé au document analysé.

```
interface TypeInfo {
    readonly attribute DOMString  typeName;
    readonly attribute DOMString  typeNamespace;
    boolean          isDerivedFrom(in DOMString typeNamespaceArg,
                                in DOMString typeNameArg,
                                in unsigned long derivationMethod);
};
```

Nom	Valeur
DERIVATION_RESTRICTION	0x00000001
DERIVATION_EXTENSION	0x00000002
DERIVATION_UNION	0x00000004
DERIVATION_LIST	0x00000008

UserDataHandler– DOM^③

On a vu avec le type `DOMUserData` que l'API DOM permet d'associer à un nœud une référence vers des informations externes. Cette interface permet de définir un handler qui sera invoqué en cas de clonage, renommage ou importation et plus généralement une modification sur un nœud comportant une donnée utilisateur.

```
interface UserDataHandler {
    void handle(in unsigned short operation,
               in DOMString key,
               in DOMUserData data,
               in Node src,
               in Node dst);
};
```

Tableau C-6 Types d'opération définis

Nom	Valeur
NODE_CLONED	1
NODE_IMPORTED	2
NODE_DELETED	3
NODE_RENAMED	4
NODE_ADOPTED	5

DOMError– DOM^③

Cette interface décrit une erreur au sein de l'API DOM.

```
interface DOMError {
    readonly attribute unsigned short severity;
    readonly attribute DOMString message;
    readonly attribute DOMString type;
    readonly attribute DOMObject relatedException;
    readonly attribute DOMObject relatedData;
    readonly attribute DOMLocator location;
};
```

Tableau C-7 Niveaux d'erreur définis

Nom	Valeur
SEVERITY_WARNING	1;
SEVERITY_ERROR	2;
SEVERITY_FATAL_ERROR	3;

DOMErrorHandler– DOM③

Cette interface décrit un handler qui pourra être appelé automatiquement par l'implantation DOM en cas d'erreur.

```
interface DOMErrorHandler {
    boolean      handleError(in DOMError error);
};
```

DOMLocator– DOM③

Toujours dans le cadre du traitement des erreurs cette interface permet de déterminer une position dans un document, notamment le lieu d'une erreur.

```
interface DOMLocator {
    readonly attribute long      lineNumber;
    readonly attribute long      columnNumber;
    readonly attribute long      byteOffset;
    readonly attribute long      utf16offset;
    readonly attribute Node      relatedNode;
    readonly attribute DOMString uri;
};
```

DOMConfiguration– DOM③

DOMConfiguration définit une interface permettant d'obtenir et de modifier les paramètres d'analyse propres à un document donné.

```
interface DOMConfiguration {
    readonly attribute DOMStringList parameterNames;
    void      setParameter      (in DOMString name,
                                in DOMUserData value) raises(DOMException);
    DOMUserData      getParameter      (in DOMString name) raises(DOMException);
    boolean      canSetParameter      (in DOMString name,
                                        in DOMUserData value);
};
```

Interfaces étendues pour XML

L'API DOM est utilisable tant pour les documents HTML que les documents XML. Les interfaces suivantes permettent de manipuler plus précisément des nœuds spécifiques aux documents XML.

CDATASection

Une section CDATA dans un document XML permet de manipuler du texte qui ne respecte pas la syntaxe XML (par exemple un extrait de code ou du HTML mal formé).

```
interface CDATASection : Text {
};
```

DocumentType

Cette interface permet de manipuler la définition de type d'un document XML.

```
interface DocumentType : Node {
  readonly attribute DOMString      name;
  readonly attribute NamedNodeMap    entities;
  readonly attribute NamedNodeMap    notations;
  // Attributs disponibles à partir de DOM niveau 2
  readonly attribute DOMString      publicId;
  readonly attribute DOMString      systemId;
  readonly attribute DOMString      internalSubset;
};
```

Notation

Cette interface est utilisée pour obtenir la notation associée à une DTD.

```
interface Notation : Node {
  readonly attribute DOMString      publicId;
  readonly attribute DOMString      systemId;
};
```

Entity

Cette interface permet d'obtenir le détail d'une entité dans un document (par exemple `&` ou `´`).

```
interface Entity : Node {
    readonly attribute DOMString    publicId;
    readonly attribute DOMString    systemId;
    readonly attribute DOMString    notationName;
    // Attributs disponibles à partir de DOM niveau 3
    readonly attribute DOMString    inputEncoding;
    readonly attribute DOMString    xmlEncoding;
    readonly attribute DOMString    xmlVersion;
};
```

EntityReference

```
interface EntityReference : Node {
};
```

ProcessingInstruction

Cette interface permet d'obtenir ou de modifier le code d'une instruction de traitement. Ainsi cette interface pourrait permettre d'obtenir le code PHP enfoui dans un document HTML ou XML.

```
interface ProcessingInstruction : Node {
    readonly attribute DOMString    target;
    attribute DOMString             data;
    // l'affectation peut lever une exception DOMException
};
```