

JBMSync Documentation

Juergen Ulbts

<http://www.juergen-ulbts.de/>

June 29, 2007

Contents

1	Package de.juergenulbts.jbookmarksync.interfaces	4
1.1	Interfaces	5
1.1.1	INTERFACE IBookmarkWriter	5
2	Package de.juergenulbts.jbookmarksync.test	6
2.1	Classes	7
2.1.1	CLASS BookmarksLocalTest	7
3	Package de.juergenulbts.jbookmarksync.util	10
3.1	Classes	12
3.1.1	CLASS BookmarkFormatChecker	12
3.1.2	CLASS BookmarkParserAbstract	14
3.1.3	CLASS BookmarkStorageContainer	17
3.1.4	CLASS CentralDataStorage	20
3.1.5	CLASS ComplexBookmark	27
3.1.6	CLASS ComplexBookmarkAndFolderAbstract	37
3.1.7	CLASS ComplexBookmarkAndFolderComparator	41
3.1.8	CLASS ComplexBookmarkFolder	47
3.1.9	CLASS MozillaBookmarkParser	53
3.1.10	CLASS MozillaBookmarkWriter	60
3.1.11	CLASS OperaV2BookmarkParser	66
3.1.12	CLASS OperaV2BookmarkWriter	73
3.1.13	CLASS XBELBookmarkWriter	79
4	Package de.juergenulbts.jbookmarksync.exception	84
4.1	Classes	85
4.1.1	CLASS BookmarkDecompressionException	85
4.1.2	CLASS BookmarkFieldNotSupportedException	87
4.1.3	CLASS BookmarkFieldsNotSetException	89

1 Package

de.juergenulbts.jbookmarksync.interfaces

Package Contents

Page

Interfaces

IBookmarkWriter 5

This interface allows to create classes that need to implement the methods defined in this interface.

1.1 Interfaces

1.1.1 INTERFACE IBookmarkWriter

This interface allows to create classes that need to implement the methods defined in this interface.

Creation date: (09.02.2007 11:54:41)

DECLARATION

```
public interface IBookmarkWriter
```

METHODS

- *writeBookmarkFile*
`public java.lang.String writeBookmarkFile(java.util.Collection
bookmarkdataArray)`
 - **Usage**
 - * This method will construct the bookmark file for the browser and return the file as a String object. The needed data comes from an array that stored the complete bookmark and bookmark folder data in `BookmarkStorageContainer` objects.
 - **Parameters**
 - * `bookmarkdataArray` - The array containing the complete bookmark data within `BookmarkStorageContainer` objects.
 - **Returns** - The created browser bookmark file as String.

2 Package

de.juergenulbts.jbookmarksync.test

Package Contents

Page

Classes

BookmarksLocalTest 7

This class is used to show how the JBMSync bookmark tool works and how it can be extended.

2.1 Classes

2.1.1 CLASS BookmarksLocalTest

This class is used to show how the JBMSync bookmark tool works and how it can be extended. It's also used as simple testcase if the utility classes work as expected.

Most of the work is done using the classes you can find in the package `de.juergenulbts.Bookmark.util!`. All `[Browsertype]BookmarkParser` classes are based upon the abstract class `BookmarkParserAbstract`.

The `BookmarkFormatChecker` was used to check if the bookmark file that should be processed is compressed using GZIP. If it is compressed it should decompress the elements before further processing.

The `BookmarkStorageContainer` will store either one `ComplexBookmark` or `ComplexBookmarkFolder` object. Finally you can store each `BookmarkStorageContainer` object in an `ArrayList`.

I've not used a `HashMap` for the bookmark or bookmark folder because the only usefull key would be the URL and this would prevent to store the same bookmark object in different subfolders as the key needs to be unique.

DECLARATION

```
public class BookmarksLocalTest
extends java.lang.Object
```

FIELDS

- `private int debugMode`
 - This field allows to control the debug output (0=no debug output)
- `private Collection arrayList`
 - `ArrayList` where the parsed data can be stored for the bookmark writers.
- `private Collection arrayList2`
 - Second `ArrayList` where the parsed data can be stored for the bookmark writers. This is needed for the `Comparator` and `CentralDataStorage` test. There we load two different
- `private CentralDataStorage centralDataStorage`
 - The central data storage which will be responsible storing the client/server data as well as the new and changed data.

CONSTRUCTORS

- *BookmarksLocalTest*
`public BookmarksLocalTest()`
 - **Usage**
 - * The default constructor of this class.

METHODS

- *compareBookmarks*
`private void compareBookmarks()`
 - **Usage**
 - * This method will compare two bookmark files. First it will add the data that has been temporary stored in `dataArray` and `dataArray2` to the `CentralDataStorage`. Then the `CentralDataStorage sync-method` will be called to compare the data. After that the new and modified data from the client that has been found will be printed to the screen.
- *getDebugMode*
`private int getDebugMode()`
 - **Usage**
 - * This method returns the current debug setting.
 - **Returns** - Returns the `debugMode`.
- *main*
`public static void main(java.lang.String[] args)`
 - **Parameters**
 - * `args` -
- *readBookmarks*
`private void readBookmarks(java.lang.String fullPathToBookmarkFile, boolean compareMode)`
 - **Usage**
 - * This method will read and process a bookmark file from the given location and finally store the read bookmarks in an `Array`.
 - **Parameters**
 - * `fullPathToBookmarkFile` -
- *writeBookmarks*
`private void writeBookmarks(java.lang.String bookmarkType, java.lang.String fullExportPathToBookmarkFile)`
 - **Usage**
 - * This method will write the stored bookmark data into a file in the format specified by the first parameter.

– **Parameters**

- * `bookmarkType` -
- * `fullExportPathToBookmarkFile` -

3 Package de.juergenulbts.jbookmarksync.util

Package Contents

Page

Classes

BookmarkFormatChecker	12
<i>This class is able to check the format of uploaded data and allows the de-compression of the data.</i>	
BookmarkParserAbstract	14
<i>This abstract class implements some basics the special subclasses for each supported bookmark format need.</i>	
BookmarkStorageContainer	17
<i>This class works as a container to store <code>ComplexBookmark</code> and <code>ComplexBookmarkFolder</code> objects.</i>	
CentralDataStorage	20
<i>This class will function as central data storage for bookmark data uploaded from the client (webbrowser) and server (OX).</i>	
ComplexBookmark	27
<i>This class is able to store bookmark data including the title, description, url, keywords and other data.</i>	
ComplexBookmarkAndFolderAbstract	37
<i>This abstract class is the base for a specialized <code>ComplexBookmark</code> and <code>ComplexFolder</code> class which store the information.</i>	
ComplexBookmarkAndFolderComparator	41
<i>This class allows to compare two <code>ComplexBookmark</code> or <code>ComplexBookmarkFolder</code> objects using the compare function.</i>	
ComplexBookmarkFolder	47
<i>This class is able to store bookmark folder data including the title, description and other data.</i>	
MozillaBookmarkParser	53
<i>This class will process bookmark files based on the Netscape/Mozilla bookmark format used by several different webbrowsers (e.g.</i>	
MozillaBookmarkWriter	60
<i>This class will process previously stored bookmark and bookmark folder data and create a bookmark file out of it that can be used with Mozilla/Netscape based webbrowsers as the output complies with the Netscape bookmark file format.</i>	
OperaV2BookmarkParser	66
<i>This class will process Opera bookmark files using the Opera v2 bookmark format used by severall Opera versions (e.g.</i>	
OperaV2BookmarkWriter	73
<i>This class will process previously stored bookmark and bookmark folder data and create a bookmark file out of it that complies with the OperaV2 bookmark file format used by several Opera versions (e.g.</i>	

XBELBookmarkWriter	79
<i>This class will process previously stored bookmark and bookmark folder data and create a bookmark file out of it that can be used with XML Bookmark Exchange Language (XBEL) supporting services and browsers.</i>	

3.1 Classes

3.1.1 CLASS BookmarkFormatChecker

This class is able to check the format of uploaded data and allows the decompression of the data. Currently only GZip as format is supported.

DECLARATION

```
public class BookmarkFormatChecker
extends java.lang.Object
```

FIELDS

- private int debugMode
 - Debug mode
- private byte zipSignature
 - Signature of a ZIP-Archive
- private final int BUFFER_SIZE
 - Fixed buffer size of a file TODO: Change BUFFER_SIZE to be dynamic

CONSTRUCTORS

- *BookmarkFormatChecker*
public BookmarkFormatChecker()
 - **Usage**
 - * The default constructor of this class.

METHODS

- *checkFormat*

```
public byte[] checkFormat( byte[] bufferArray )
```

- **Usage**

- * This method will check the format of the given byte array and decompress the file if possible and needed (e.g. for GZIP).

- **Parameters**

- * `bufferArray` - Byte array

- **Returns** - byte[] Decompressed content of bufferArray

- **Exceptions**

- *
de.juergenulbts.jbookmarksync.exception.BookmarkDecompressionException
-

- *getBytes*

```
private byte[] getBytes( byte[] sourceArray, int startIndex, int endIndex )
```

- **Usage**

- * This method copies the specified part from the supplied array and returns it as new byte array.

- **Returns** - Returns copied bytes as array[].

- *getDebugMode*

```
private int getDebugMode( )
```

- **Usage**

- * This method returns the current debug setting.

- **Returns** - Returns the debugMode.

- *getFourByteInt*

```
private int getFourByteInt( byte[] bytes, int offset )
```

- **Usage**

- * Returns four bytes from a byte array starting at offset <offset>as integer.

- **Returns** - Returns four bytes as Int (Intel byte order)

- *getTwoByteInt*

```
private int getTwoByteInt( byte[] bytes, int offset )
```

- **Usage**

- * Returns two bytes from a byte array starting at offset <offset>as integer.

- **Returns** - Returns two bytes as Int (Intel byte order)

- *getZipSignature*

```
private byte[] getZipSignature( )
```

- **Usage**

- * This method returns the current zip signature.

- **Returns** - Returns the zipSignature as byte[]

- *processGZipArchive*

```
private byte[] processGZipArchive( byte[] bufferArray )
```

- **Usage**

- * This method will decompress the content of a GZIP archive. The uncompressed data will be returned to the calling method.

- **Parameters**

- * `bufferArray` - Byte array

- **Returns** - byte[] Decompressed content of bufferArray

- **Exceptions**

- *
de.juergenulbts.jbookmarksync.exception.BookmarkDecompressionException
-

- *processZipArchive*

```
private byte[] processZipArchive( byte[] bufferArray )
```

- **Usage**

- * This method will decompress the content of a ZIP archive but is limited to handle one file stored in the archive. The uncompressed data will be returned to the calling method. TODO: EXPERIMENTAL ZIP decompression - no final code here

- **Parameters**

- * `bufferArray` - Byte array

- **Returns** - Content of bufferArray which may have been decompressed.

- **Exceptions**

- *
de.juergenulbts.jbookmarksync.exception.BookmarkDecompressionException
-

- *setZipSignature*

```
private void setZipSignature( byte[] zipSignature )
```

- **Usage**

- * This method allows to set a new zip Signature.

- **Parameters**

- * `zipSignature` - The zipSignature[] to set.

3.1.2 CLASS BookmarkParserAbstract

This abstract class implements some basics the special subclasses for each supported bookmark format need. It is only responsible for the client side.

DECLARATION

```
public abstract class BookmarkParserAbstract  
extends java.lang.Object
```

FIELDS

- private static final String serverObjectIdStartTag
 - The tag to identify the start of the server object id information from the server that is stored in a description.
- private static final String serverObjectIdEndTag
 - The tag to identify the end of the server object id information from the server that is stored in a description.
- private static long supportedBookmarkFolderFields
 - The property allows to identify the bookmark folder fields the implemented bookmark parser does support. Each field has it's own UniquePropertyID (UPI).

This information is needed later to compare the fields of the server bookmark folder to the client bookmark folder

(de.juergenulbts.jbookmarksync.util.ComplexBookmarkAndFolderAbstract (in 3.1.6, page 37) , de.juergenulbts.jbookmarksync.util.ComplexBookmarkFolder (in 3.1.8, page 47)).

- private static long supportedBookmarkFields
 - The property allows to identify the bookmark fields the implemented bookmark parser does support. Each field has it's own UniquePropertyID (UPI).

This information is needed later to compare the fields of the server bookmark to the client bookmark

(de.juergenulbts.jbookmarksync.util.ComplexBookmarkAndFolderAbstract (in 3.1.6, page 37) , de.juergenulbts.jbookmarksync.util.ComplexBookmark (in 3.1.5, page 27)).

CONSTRUCTORS

- *BookmarkParserAbstract*
`public BookmarkParserAbstract()`
 - **Usage**
 - * The default constructor of this class.

METHODS

- *getBookmarks*
`public abstract java.util.Collection getBookmarks(byte[] bookmarkFile)`
 - **Usage**

* This method calls methods to create the bookmark array out of BookmarkStorageContainer objects which contain either a ComplexBookmark or a ComplexBookmarkFolder object. The object type of the stored object inside the BookmarkStorageContainer can be checked with a method.

– **Parameters**

* **bookmarkFile** - The complete bookmark file as byte array.

– **Returns** - Collection containing BookmarkStorageContainer objects

• *getServerObjectIdEndTag*

```
private java.lang.String getServerObjectIdEndTag( )
```

– **Usage**

* The method will return the tag to identify the end of the object id information from the server that is again stored in a description.

– **Returns** - The serverObjectIdEndTag.

• *getServerObjectIdStartTag*

```
private java.lang.String getServerObjectIdStartTag( )
```

– **Usage**

* The method will return the tag to identify the end of the object id information from the server that is again stored in a description.

– **Returns** - The serverObjectIdStartTag.

• *getSupportedBookmarkFields*

```
public abstract long getSupportedBookmarkFields( )
```

– **Usage**

* This method will return the information which fields of a bookmark are supported.

– **Returns** - supportedBookmarkFields as long.

• *getSupportedBookmarkFolderFields*

```
public abstract long getSupportedBookmarkFolderFields( )
```

– **Usage**

* This method will return the information which fields of a bookmark folder are supported.

– **Returns** - supportedBookmarkFolderFields as long.

• *isServerObjectIdIncluded*

```
protected boolean isServerObjectIdIncluded( java.lang.String text )
```

– **Usage**

* This method will check if the text includes the object id from the server which can be stored in description fields. It returns true if the object id could be found otherwise false.

– **Parameters**

* **text** - The String that will be checked if it includes the server object id tag.

– **Returns** - Returns status if the objectId start and end tags can be found in the submitted string.

• *retrieveServerObjectId*

```
protected int retrieveServerObjectId( java.lang.String text )
```

– **Usage**

* This method will look for the object id of the server and return the integer value of it.

– **Parameters**

* **text** - The String that will be checked if it includes the server object id.

– **Returns** - Returns the server object Id as int value.

• *stripServerObjectIdFromDescription*

```
protected java.lang.String stripServerObjectIdFromDescription(  
java.lang.String text )
```

– **Usage**

* This method will remove the object id of the server from the description.

– **Parameters**

* **text** - The String that will be checked if it includes the server object id.

– **Returns** - Returns the description string without the the server object id.

3.1.3 CLASS BookmarkStorageContainer

This class works as a container to store `ComplexBookmark` and `ComplexBookmarkFolder` objects. They are subclasses of `ComplexBookmarkAndFolderAbstract`.

DECLARATION

```
public class BookmarkStorageContainer  
extends java.lang.Object
```

FIELDS

- `public static final int FOLDER`
 - Value of the `complexType` field indicating the stored object is a bookmark folder.
- `public static final int BOOKMARK`
 - Value of the `complexType` field indicating the stored object is a bookmark.
- `private int id`
 - Id of the current bookmark or folder object.
- `private int parentId`
 - `ParentId` which the current bookmark or folder is assigned to.
- `private int folderLevel`

- FolderLevel which describes the depth of the folder inside the hierarchie or that of the bookmark items inside this folder.
- private int complexType
 - The indicator if the ComplexBookmark or ComplexBookmarkFolder is stored in the current ComplexBookmarkAndFolderAbstract. The value of the complexType can be either #FOLDER or #BOOKMARK .
- private ComplexBookmarkAndFolderAbstract complexBkMrk_BkMrkFolder
 - This field can either store a ComplexBookmark or ComplexBookmarkFolder. Which object has been stored can be seen by analysing the complexType field.

CONSTRUCTORS

- *BookmarkStorageContainer*
public **BookmarkStorageContainer**(int id, int parentId, int folderLevel, de.juergenulbts.jbookmarksync.util.ComplexBookmarkAndFolderAbstract complexBkMrk_BkMrkFolder)
 - **Usage**
 - * Constructor to store a ComplexBookmark or ComplexBookmarkFolder object.
 - **Parameters**
 - * id - The unique id of the object.
 - * parentId - The parent of the current object.
 - * folderLevel - The folderLevel of the current object.
 - * complexBkMrk_BkMrkFolder - Subclass of ComplexBookmarkFolder.

METHODS

- *getComplexBkMrk_BkMrkFolder*
public de.juergenulbts.jbookmarksync.util.ComplexBookmarkAndFolderAbstract **getComplexBkMrk_BkMrkFolder**()
 - **Usage**
 - * This method will return ComplexBookmark or ComplexBookmarkFolder objects. Which one is actually stored can be determined by evaluating the #complexType setting.
 - **Returns** - Returns the complexBkMrk_BkMrkFolder.
- *getComplexType*
public int **getComplexType**()
 - **Usage**
 - * This method will return an indicator (#FOLDER or #BOOKMARK) if an ComplexBookmark or ComplexBookmarkFolder is stored as ComplexBookmarkAndFolderAbstract object.
 - **Returns** - Returns the complexType as int (#FOLDER or #BOOKMARK).

- *getFolderLevel*
public int **getFolderLevel**()
 - **Usage**
 - * This method will return the folderLevel. This is the depth of the folder inside the hierarchie or that of the bookmark items inside this folder.
 - **Returns** - Returns the folderLevel as int.

- *getId*
public int **getId**()
 - **Usage**
 - * This method will return the Id of the current bookmark or folder object.
Notice: This id is normally not equal to the serverObjectId!
 - **Returns** - Returns the id as int.

- *getParentId*
public int **getParentId**()
 - **Usage**
 - * This method will return the parentId. It's the id which the current bookmark or folder is assigned to.
 - **Returns** - Returns the parentId as int.

- *setComplexBkMrk_BkMrkFolder*
protected void **setComplexBkMrk_BkMrkFolder**(
de.juergenulbts.jbookmarksync.util.ComplexBookmarkAndFolderAbstract
complexBkMrk_BkMrkFolder)
 - **Usage**
 - * This method allows to store ComplexBookmark or ComplexBookmarkFolder objects.
 - **Parameters**
 - * complexBkMrk_BkMrkFolder - The complexBkMrk_BkMrkFolder to set.

- *setComplexType*
protected void **setComplexType**(int **complexType**)
 - **Usage**
 - * This method allows to set an indicator (#FOLDER or #BOOKMARK) if the ComplexBookmark or ComplexBookmarkFolder is stored as ComplexBookmarkAndFolderAbstract object.
 - **Parameters**
 - * complexType - The complexType to set (#FOLDER or #BOOKMARK).

- *setFolderLevel*
protected void **setFolderLevel**(int **folderLevel**)
 - **Usage**
 - * This method allows to set the folderLevel. This is the depth of the folder inside the hierarchie or that of the bookmark items inside this folder.
 - **Parameters**
 - * folderLevel - The folderLevel to set.

- *setId*

protected void **setId**(int id)

- **Usage**

- * This method allows to the set Id of the current bookmark or folder object.

- **Parameters**

- * id - The id to set.

- *setParentId*

protected void **setParentId**(int parentId)

- **Usage**

- * This method allows to set the parentId. It's the id which the current bookmark or folder is assigned to.

- **Parameters**

- * parentId - The parentId to set.

- *toString*

public java.lang.String **toString**()

- **Usage**

- * This method will create a string of the content of this bookmark.

- **Returns** - Returns information about the bookmarkStorageContainer as String object.

3.1.4 CLASS *CentralDataStorage*

This class will function as central data storage for bookmark data uploaded from the client (webbrowser) and server (OX). Each side has a add function to put the data into this data storage. The data storage will also store the information of the changed data that has to be stored on the server after the client data has been compared with the server data and vice versa.

Creation date: (06.02.2007 01:24:28)

DECLARATION

```
public class CentralDataStorage
extends java.lang.Object
```

FIELDS

- private int debugMode
 - Debug mode (0 = Debug output turned off)
- private List clientdataArray
 - The clientdataArray list will store the complete bookmark and bookmark folder data from the client.
- private Map clientDataLookup
 - The clientDataLookup is a helper to find entries (server object id) in constant time within the #clientdataArray .
- private List serverdataArray
 - The serverdataArray list will store the complete bookmark and bookmark folder data from the server.
- private Map serverDataLookup
 - The serverDataLookup is a helper to find entries (server object id) in constant time within the #serverdataArray .
- private List newDataArray
 - The newDataArray list will store the new bookmark and bookmark folder data that needs to be stored on the server.
- private List changeddataArray
 - The changeddataArray list will store the changed bookmark and bookmark folder data that needs to be updated on the server.
- private ComplexBookmarkAndFolderComparator comparator
 - The ComplexBookmarkAndFolderComparator class provides methods to compare ComplexBookmark and ComplexBookmarkFolder objects.
- private long supportedClientBookmarkFields
 -
- private long supportedClientBookmarkFolderFields
 -
- private long supportedServerBookmarkFields
 - The property allows to identify the bookmark fields the server does support. Each field has it's own UniquePropertyID (UPI).

This information is needed later to compare the fields of the server bookmark to the client bookmark
(de.juergenulbts.jbookmarksync.util.ComplexBookmarkAndFolderAbstract (in 3.1.6, page 37) , de.juergenulbts.jbookmarksync.util.ComplexBookmark (in 3.1.5, page 27)).
- private long supportedServerBookmarkFolderFields

- The property allows to identify the bookmark folder fields the server does support. Each field has it's own UniquePropertyID (UPI).

This information is needed later to compare the fields of the server bookmark folder to the client bookmark folder

(*de.juergenulbts.jbookmarksync.util.ComplexBookmarkAndFolderAbstract* (in 3.1.6, page 37) , *de.juergenulbts.jbookmarksync.util.ComplexBookmarkFolder* (in 3.1.8, page 47)).

- private boolean `clientOverwritesServerData`
 - This property allows the client to overwrite the existing server data if true (default: false).

CONSTRUCTORS

- *CentralDataStorage*
`public CentralDataStorage()`
 - **Usage**
 - * The default constructor of this class.

METHODS

- *addClientDataEntry*
`public void addClientDataEntry(
de.juergenulbts.jbookmarksync.util.BookmarkStorageContainer
bookmarkStorageContainer)`
 - **Usage**
 - * This method will add bookmark data from the client to the internal storage and does also fill the lookup map.
 - **Parameters**
 - * `bookmarkStorageContainer` - The client bookmark data as `BookmarkStorageContainer`.
- *addServerDataEntry*
`public void addServerDataEntry(
de.juergenulbts.jbookmarksync.util.BookmarkStorageContainer
bookmarkStorageContainer)`
 - **Usage**
 - * This method will add bookmark data from the server to the internal storage and does also fill the lookup map.
 - **Parameters**
 - * `bookmarkStorageContainer` - The server bookmark data as `BookmarkStorageContainer`.
- *clearAll*
`protected void clearAll()`

– Usage

- * This method will delete all data from the internal storage objects.

• *compareClientToServer*

private void **compareClientToServer**()

– Usage

- * This method will iterate through the list of client **BookmarkStorageContainer** objects and compare it to the server **BookmarkStorageContainer** object list. If one **BookmarkStorageContainer** can't be found on the server list it is either a new object on the client side (then it doesn't have a **serverObjectId**) or it has been deleted on the server. If the client object finds a matching server object both objects will be compared field by field (only the fields **client** and **server support**).

– Exceptions

- * **java.lang.ClassCastException** -
- * **BookmarkFieldsNotSetException** -

• *compareServerToClient*

private void **compareServerToClient**()

– Usage

- * This method will iterate through the list of server **BookmarkStorageContainer** objects and compare it to the client **BookmarkStorageContainer** object list. If one **BookmarkStorageContainer** object can't be found on the client list it is either a new object on the server side (**DEFAULT**) or it has been deleted on the client (but that should have been). **TODO**: Put more brain into this issue...maybe mark entries on the server as deleted. The field by field compare has been already done by the client to server object check.

– Exceptions

- * **java.lang.ClassCastException** -
- * **BookmarkFieldsNotSetException** -

• *getChangedDataArray*

public java.util.List **getChangedDataArray**()

– Usage

- * This method will return a list that contains the changed (updated) bookmark and bookmark folder data in **BookmarkStorageContainer** objects. This data needs to be updated on the server.

– **Returns** - Returns the **changedDataArray** as List.

• *getClientDataArray*

private java.util.List **getClientDataArray**()

– Usage

- * This method will return the array containing the client data as **BookmarkStorageContainer** objects.

– **Returns** - Returns the **clientDataArray** as List.

• *getClientDataLookup*

private java.util.Map **getClientDataLookup**()

– **Usage**

- * This method will return the map containing the lookup key (server object id) and the value (position inside the #clientdataArray) for the client data.

– **Returns** - The clientDataLookup as Map.

• *getComparator*

private

de.juergenulbts.jbookmarksync.util.ComplexBookmarkAndFolderComparator

getComparator()

– **Usage**

- * This method will return the ComplexBookmarkAndFolderComparator object which provides methods to compare ComplexBookmark and ComplexBookmarkFolder objects.

– **Returns** - Returns the ComplexBookmarkAndFolderComparator (comparator) object.

• *getNewDataArray*

public java.util.List **getNewDataArray()**

– **Usage**

- * This method will return a list that contains stores the new bookmark and bookmark folder data in BookmarkStorageContainer objects. This data needs to be added (stored) on the server.

– **Returns** - Returns the newDataArray as List.

• *getParentFolderFromClientDataArray*

public int[] **getParentFolderFromClientDataArray(int clientId, int folderLevel, boolean isFolder)**

– **Usage**

- * This method will return the parent folder id for the requested client data item (bookmark or bookmark folder).

– **Parameters**

- * **clientId** - The client id (NOT serverObjectId) to look for.
- * **folderLevel** - The client folderLevel to look for.
- * **isFolder** - If the clientId is from a folder set this value to 'true', otherwise 'false'.

– **Returns** - Returns an int array which stores the id given by the bookmark parser and the serverObjectId (which could be -1 if it's a new item).

• *getServerDataArray*

private java.util.List **getServerDataArray()**

– **Usage**

- * This method will return the array containing the client data as BookmarkStorageContainer objects.

– **Returns** - Returns the serverDataArray as List.

• *getServerDataLookup*

private java.util.Map **getServerDataLookup()**

– **Usage**

- * This method will return the map containing the lookup key (server object id) and the value (position inside the #serverdataArray) for the server data.

- **Returns** - The serverDataLookup as Map.

- *getSupportedClientBookmarkFields*
private long **getSupportedClientBookmarkFields**()
 - **Usage**
 - * This method will return the supported bookmark fields of the client.
 - **Returns** - Returns the supportedClientBookmarkFields as long.

- *getSupportedClientBookmarkFolderFields*
private long **getSupportedClientBookmarkFolderFields**()
 - **Usage**
 - * This method will return the supported bookmark folder fields of the client.
 - **Returns** - Returns the supportedClientBookmarkFolderFields as long.

- *getSupportedServerBookmarkFields*
private long **getSupportedServerBookmarkFields**()
 - **Usage**
 - * This method will return the supported bookmark fields of the server.
 - **Returns** - Returns the supportedServerBookmarkFields as long

- *getSupportedServerBookmarkFolderFields*
private long **getSupportedServerBookmarkFolderFields**()
 - **Usage**
 - * This method will return the supported bookmark folder fields of the server.
 - **Returns** - the supportedServerBookmarkFolderFields as Long

- *isClientOverwritesServerData*
private boolean **isClientOverwritesServerData**()
 - **Usage**
 - * This method will return the value of the clientOverwritesServerData field which allows the client to overwrite the server data.
 - **Returns** - The clientOverwritesServerData setting as boolean

- *setClientDataArray*
private void **setClientDataArray**(java.util.ArrayList **clientDataArray**)
 - **Usage**
 - * This method allows to set the array containing the client data as BookmarkStorageContainer objects.
 - **Parameters**
 - * **clientDataArray** - The clientDataArray to set.

- *setClientDataLookup*
private void **setClientDataLookup**(java.util.Map **clientDataLookup**)
 - **Usage**
 - * This method allows to set a map containing the lookup key (server object id) and the value (position inside the #clientDataArray) for the client data.
 - **Parameters**

* `clientDataLookup` - the `clientDataLookup` to set

• *setClientOverwritesServerData*

```
public void setClientOverwritesServerData( boolean  
clientOverwritesServerData )
```

– Usage

* This method allows to set the option to allow the client to overwrite the server data.

– Parameters

* `clientOverwritesServerData` - The `clientOverwritesServerData` value to set.

• *setComparator*

```
private void setComparator(  
de.juergenulbts.jbookmarksync.util.ComplexBookmarkAndFolderComparator  
comparator )
```

– Usage

* This method allows to set a new `ComplexBookmarkAndFolderComparator` object which provides methods to compare `ComplexBookmark` and `ComplexBookmarkFolder` objects.

– Parameters

* `comparator` - The comparator to set.

• *setServerDataArray*

```
private void setServerDataArray( java.util.ArrayList serverDataArray )
```

– Usage

* This method allows to set the array containing the client data as `BookmarkStorageContainer` objects.

– Parameters

* `serverDataArray` - The `serverDataArray` to set.

• *setServerDataLookup*

```
private void setServerDataLookup( java.util.Map serverDataLookup )
```

– Usage

* This method allows to set a map containing the lookup key (server object id) and the value (position inside the `#serverDataArray`) for the server data.

– Parameters

* `serverDataLookup` - the `serverDataLookup` to set

• *setSupportedClientBookmarkFields*

```
public void setSupportedClientBookmarkFields( long  
supportedClientBookmarkFields )
```

– Usage

* This method allows to set the supported bookmark fields of the client.

– Parameters

* `supportedClientBookmarkFields` - The `supportedClientBookmarkFields` to set.

- *setSupportedClientBookmarkFolderFields*
`public void setSupportedClientBookmarkFolderFields(long supportedClientBookmarkFolderFields)`
 - **Usage**
 - * This method allows to set the supported bookmark folder fields of the client.
 - **Parameters**
 - * `supportedClientBookmarkFolderFields` - The supportedClientBookmarkFolderFields to set.

- *setSupportedServerBookmarkFields*
`public void setSupportedServerBookmarkFields(long supportedServerBookmarkFields)`
 - **Usage**
 - * This method allows to set the supported bookmark fields of the server.
 - **Parameters**
 - * `supportedServerBookmarkFields` - The supportedServerBookmarkFields to set.

- *setSupportedServerBookmarkFolderFields*
`public void setSupportedServerBookmarkFolderFields(long supportedServerBookmarkFolderFields)`
 - **Usage**
 - * This method allows to set the supported bookmark folder fields of the server.
 - **Parameters**
 - * `supportedServerBookmarkFolderFields` - The supportedServerBookmarkFolderFields to set.

- *twoWaySlowSync*
`public boolean twoWaySlowSync()`
 - **Usage**
 - * This method will perform the actual 2-Way-Sync (Slow-Sync) comparing the objects and returning
 - **Returns** - Returns the result of the sync (successfull==true, unsuccessfull==false) as boolean.

3.1.5 CLASS ComplexBookmark

This class is able to store bookmark data including the title, description, url, keywords and other data.

DECLARATION

```
public class ComplexBookmark
extends de.juergenulbts.jbookmarksync.util.ComplexBookmarkAndFolderAbstract
```

FIELDS

- private String url
 - URL of the website.
- private String feedUrl
 - FEEDURL of LiveBookmark or other feed. This should normally be the same as URL
- private String faviconImageData
 - FaviconImageData of the website (Base64 encoded). Example: FireFox stores Favicon as 'ICON="data:<image_mime_type>;base64,<base64_encoded_image_data>"' See also #faviconImageMimeType .
- private String faviconImageMimeType
 - The MimeType of the Favicon used on a website (#faviconImageData).
- private String keyword
 - String of keywords in Firefox/Mozilla/Netscape (SHORTCUTURL="...") or Opera setting (SHORT NAME=...) which can also be used to load the URL.
- private long lastVisitDate
 - Timestamp Firefox/Mozilla/Netscape (LAST_VISIT="<number>") and Opera (VISITED=<number>). The timestamp is in milliseconds.
- private long lastModifiedDate
 - Timestamp Firefox/Mozilla/Netscape (LAST_MODIFIED="<number>"). The timestamp is in milliseconds.
- private boolean loadInSideBar
 - Optional Firefox/Mozilla/Netscape (WEB_PANEL="true") setting to load the URL in the sidebar.
- private boolean onPersonalBar
 - Optional Opera setting (ON PERSONALBAR=YES) to show the URL on the personal bar.
- private int positionInsidePersonalBar
 - Optional Opera setting (PERSONALBAR_POS=<number>) to identify the position of an entry inside the personal bar.
- private boolean inPanel
 - Optional Opera setting (IN PANEL=YES) to show the URL on the panel and open it in another one. It's also an optional Netscape/Mozilla setting to open the URL in a panel (WEB_PANEL="TRUE").
- private int positionInsidePanel
 - Optional Opera setting (PANEL_POS=<number>) to identify the position of an entry inside the panel.

CONSTRUCTORS

- *ComplexBookmark*

```
public ComplexBookmark( int serverObjectId, java.lang.String title,
java.lang.String url, java.lang.String description, long createdDate, long
lastModifiedDate )
```

- Usage

- * Constructor to create a simple Bookmark that will store the information stored by Open-Xchange. This constructor can be used to store bookmarks from OX 0.8.0/0.8.2/0.8.6!

- Parameters

- * `serverObjectId` - The unique ObjectID used on the server to identify the bookmark object.
- * `title` - The title of the bookmark.
- * `url` - The URL of the bookmark.
- * `description` - The description of the bookmark.
- * `createdDate` - Timestamp when the bookmark has been created or added.
- * `lastModifiedDate` - Timestamp when the bookmark has been modified the last time.

- *ComplexBookmark*

```
public ComplexBookmark( java.lang.String title, java.lang.String url,
java.lang.String description, java.lang.String keyword, long createdDate,
long lastVisitDate, boolean onPersonalBar, int positionInsidePersonalBar,
boolean inPanel, int positionInsidePanel, java.lang.String
clientLineBreakString )
```

- Usage

- * Constructor to create a more complete Bookmark that will store the most basic informations including the favicon (encoded Base64). This constructor can be used to store bookmarks from the Opera webbrowser!

- Parameters

- * `title` - The title of the bookmark.
- * `url` - The url of the bookmark.
- * `description` - The description of the bookmark.
- * `keyword` - The keyword of the bookmark.
- * `createdDate` - Timestamp when the bookmark has been created or added.
- * `lastVisitDate` - Timestamp when the bookmark has been visited the last time.
- * `onPersonalBar` - The boolean value if the entry is shown on the personal bar or not.
- * `positionInsidePersonalBar` - Setting to identify the position of an entry inside the personal bar.
- * `inPanel` - The boolean value if the entry is shown in the panel or not.
- * `positionInsidePanel` - Setting to identify the position of an entry inside the personal bar.
- * `clientLineBreakString` - Line break string used by the client.

- *ComplexBookmark*

```
public ComplexBookmark( java.lang.String title, java.lang.String url,
java.lang.String feedUrl, java.lang.String description, java.lang.String
keyword, long createdDate, long lastModifiedDate, long lastVisitDate,
```

java.lang.String **favIconImageData**, java.lang.String **favIconImageMimeType**, boolean **loadInSideBar**, boolean **onPersonalBar**, int **positionInsidePersonalBar**, boolean **inPanel**, int **positionInsidePanel**)

– Usage

- * Constructor to create a more complete Bookmark that will store the most basic informations including the favicon (encoded Base64). This constructor can be used to store bookmarks from Firefox/Mozilla/Netscape/Opera and other webbrowsers!

– Parameters

- * **title** - The title of the bookmark.
- * **url** - The URL of the bookmark.
- * **feedUrl** - The feed URL of the bookmark.
- * **description** - The description of the bookmark.
- * **keyword** - The keyword of the bookmark.
- * **createdDate** - Timestamp when the bookmark has been created or added.
- * **lastModifiedDate** - Timestamp when the bookmark has been modified the last time.
- * **lastVisitDate** - Timestamp when the bookmark has been visited the last time.
- * **favIconImageData** - The FavIcon image data of the website (Base64 encoded - see `#favIconImageData`).
- * **favIconImageMimeType** - The mime type of the FavIcon image (see `#favIconImageMimeType`).
- * **loadInSideBar** - The boolean value if the entry will be loaded in the sidebar.
- * **onPersonalBar** - The boolean value if the entry is shown on the personal bar or not.
- * **positionInsidePersonalBar** - Setting to identify the position of an entry inside the personal bar.
- * **inPanel** - The boolean value if the entry is shown in the panel or not.
- * **positionInsidePanel** - Setting to identify the position of an entry inside the personal bar.

• *ComplexBookmark*

public **ComplexBookmark**(java.lang.String **title**, java.lang.String **url**, java.lang.String **feedUrl**, java.lang.String **description**, java.lang.String **keyword**, long **createdDate**, long **lastModifiedDate**, long **lastVisitDate**, java.lang.String **favIconImageData**, java.lang.String **favIconImageMimeType**, boolean **loadInSideBar**, java.lang.String **clientLineBreakString**)

– Usage

- * Constructor to create a more complete Bookmark that will store the most basic informations including the favicon (encoded Base64). This constructor can be used to store bookmarks from the Firefox/Mozilla/Netscape webbrowser!

– Parameters

- * **title** - The title of the bookmark.
- * **url** - The URL of the bookmark.
- * **feedUrl** - The feed URL of the bookmark.
- * **description** - The description of the bookmark.
- * **keyword** - The keyword of the bookmark.
- * **createdDate** - Timestamp when the bookmark has been created or added.
- * **lastModifiedDate** - Timestamp when the bookmark has been modified the last time.
- * **lastVisitDate** - Timestamp when the bookmark has been visited the last time.

- * `favIconImageData` - The FavIcon image data of the website (Base64 encoded - see `#favIconImageData`).
- * `favIconImageMimeType` - The mime type of the FavIcon image (see `#favIconImageMimeType`).
- * `loadInSideBar` - The boolean value if the entry will be loaded in the sidebar.
- * `clientLineBreakString` - Line break string used by the client.

METHODS

- *getFavIconImageData*

```
public java.lang.String getFavIconImageData( )
```

- **Usage**

- * This method returns the image data of a FavIcon as Base64 encoded string or null if there is no image data.

- **Returns** - Returns the favIconImageData.

- *getFavIconImageMimeType*

```
public java.lang.String getFavIconImageMimeType( )
```

- **Usage**

- * This method returns the mime type of the FavIcon image. It will return null if there is no image data.

- **Returns** - Returns the favIconImageMimeType.

- *getFeedUrl*

```
public java.lang.String getFeedUrl( )
```

- **Usage**

- * This method will return the FEEDURL of the Live-Bookmark or other feed and should normally be the same as URL.

- **Returns** - Returns the feedUrl as String.

- *getKeyword*

```
public java.lang.String getKeyword( )
```

- **Usage**

- * This method will return the keyword.

- **Returns** - Returns the keyword as String.

- *getLastModifiedDate*

```
public long getLastModifiedDate( )
```

- **Usage**

- * This method will return the timestamp when the bookmark has been modified the last time. The timestamp is in milliseconds.

- **Returns** - Returns the lastModifiedDate as long.

- *getLastVisitDate*

```
public long getLastVisitDate( )
```

- **Usage**

* This method will return the timestamp when the bookmark has been visited the last time. The timestamp is in milliseconds.

– **Returns** - Returns the lastVisitDate as long.

• *getPositionInsidePanel*

public int **getPositionInsidePanel**()

– **Usage**

* This method will return the position inside the panel (used e.g. by Mozilla).

– **Returns** - Returns the positionInsidePanel.

• *getPositionInsidePersonalBar*

public int **getPositionInsidePersonalBar**()

– **Usage**

* This method will return the position inside the personal bar (used e.g. by Opera).

– **Returns** - Returns the positionInsidePersonalBar.

• *getUrl*

public java.lang.String **getUrl**()

– **Usage**

* This method will return the URL of the bookmark.

– **Returns** - Returns the url as String.

• *isInPanel*

public boolean **isInPanel**()

– **Usage**

* This method will return the setting of this entry for the panel (used e.g. by Mozilla). It returns true if the entry is visible in the panel otherwise false.

– **Returns** - Returns status if the entry is shown in the panel or not.

• *isLoadInSideBar*

public boolean **isLoadInSideBar**()

– **Usage**

* This method will return the setting of this entry for the sidebar. It returns true if the entry will be loaded in the sidebar otherwise false.

– **Returns** - Returns the status if the entry will be loaded in the sidebar.

• *isOnPersonalBar*

public boolean **isOnPersonalBar**()

– **Usage**

* This method will return the setting of this entry for the personal bar (used e.g. by Opera). It returns true if the entry is visible on the personal bar otherwise false.

– **Returns** - Returns status if the entry is shown on the personal bar or not.

• *setFavIconImageData*

protected void **setFavIconImageData**(java.lang.String **favIconImageData**)

– **Usage**

* This method will store the image data of a FavIcon as string which is encoded as Base64. If it is null there is no image data.

– **Parameters**

* `favIconImageData` - The `favIconImageData` to set.

• *setFavIconImageMimeType*

protected void **setFavIconImageMimeType**(java.lang.String
favIconImageMimeType)

– **Usage**

* This method will store the mime type of the FavIcon. It is null if there is no image data.

– **Parameters**

* `favIconImageMimeType` - The `favIconImageMimeType` to set.

• *setFeedUrl*

protected void **setFeedUrl**(java.lang.String **feedUrl**)

– **Usage**

* This method allows to set the FEEDURL of the Live-Bookmark or other feed and should normally be the same as URL.

– **Parameters**

* `feedUrl` - The `feedUrl` to set.

• *setInPanel*

protected void **setInPanel**(boolean **inPanel**)

– **Usage**

* This method allows to set the status of this entry for the panel (used e.g. by Mozilla).

– **Parameters**

* `inPanel` - The boolean value if the entry is shown in the panel or not.

• *setKeyword*

protected void **setKeyword**(java.lang.String **keyword**)

– **Usage**

* This method allows to set the keyword.

– **Parameters**

* `keyword` - The keyword to set.

• *setLastModifiedDate*

protected void **setLastModifiedDate**(long **lastModifiedDate**)

– **Usage**

* This method allows to set the timestamp when the bookmark has been modified the last time. The timestamp needs to be in milliseconds.

– **Parameters**

* `lastModifiedDate` - The `lastModifiedDate` to set.

• *setLastVisitDate*

protected void **setLastVisitDate**(long **lastVisitDate**)

– **Usage**

- * This method allows to set the timestamp when the bookmark has been visited the last time. The timestamp needs to be in milliseconds.

- **Parameters**

- * `lastVisitDate` - The `lastVisitDate` to set.
-

- *setLoadInSideBar*

protected void **setLoadInSideBar**(boolean **loadInSideBar**)

- **Usage**

- * This method allows to set the status of this entry for the sidebar.

- **Parameters**

- * `loadInSideBar` - The boolean value if the entry will be loaded in the sidebar.
-

- *setOnPersonalBar*

protected void **setOnPersonalBar**(boolean **onPersonalBar**)

- **Usage**

- * This method allows to set the status of this entry for the personal bar (used e.g. by Opera).

- **Parameters**

- * `onPersonalBar` - The boolean value if the entry is shown on the personal bar or not.
-

- *setPositionInsidePanel*

protected void **setPositionInsidePanel**(int **positionInsidePanel**)

- **Usage**

- * This method allows to set the position inside the panel (used e.g. by Mozilla).

- **Parameters**

- * `positionInsidePanel` - The `positionInsidePanel` to set.
-

- *setPositionInsidePersonalBar*

protected void **setPositionInsidePersonalBar**(int **positionInsidePersonalBar**)

- **Usage**

- * This method allows to set the position inside the personal bar (used e.g. by Opera).

- **Parameters**

- * `positionInsidePersonalBar` - The `positionInsidePersonalBar` to set.
-

- *setUrl*

protected void **setUrl**(java.lang.String **url**)

- **Usage**

- * This method allows to set the URL of the bookmark.

- **Parameters**

- * `url` - The url to set.
-

- *toString*

public java.lang.String **toString**()

- **Usage**

- * This method will create a string of the content of this bookmark. All date and time calculations are based on the TimeZone 'GMT'!
- **Returns** - Returns the bookmark as String object.

METHODS INHERITED FROM CLASS

de.juergenulbts.jbookmarksync.util.ComplexBookmarkAndFolderAbstract

(in 3.1.6, page 37)

- *getCreatedDate*
public long **getCreatedDate**()
 - **Usage**
 - * This method will return the timestamp when the bookmark folder or bookmark has been created/added. The timestamp is in milliseconds.
 - **Returns** - Returns the createdDate as long.

- *getDescription*
public java.lang.String **getDescription**()
 - **Usage**
 - * This method will return the description of a bookmark or bookmark folder. Line breaks use CR+LF (CHR(13)+CHR(10)) which may not compatible to the format where you may want to use the data. See also {link #getDescription(String)}.
 - **Returns** - Returns the description as String.

- *getDescription*
public java.lang.String **getDescription**(java.lang.String **lineBreakString**)
 - **Usage**
 - * This method will return the description of a bookmark or bookmark folder. The line break string which is CR+LF (CHR(13)+CHR(10)) will be replaced by the specified linebreak.
 - **Parameters**
 - * **lineBreakString** - Line break string used by the client.
 - **Returns** - Returns the description as String.

- *getDescriptionWithoutServerObjectId*
public java.lang.String **getDescriptionWithoutServerObjectId**()
 - **Usage**
 - * This method will return the description of a bookmark or bookmark folder without the serverObjectId information. Line breaks use CR+LF (CHR(13)+CHR(10)) which may not compatible to the format where you may want to use the data. See also {link #getDescriptionWithoutServerObjectId(String)}.
 - **Returns** - Returns the description as String without server object id.

- *getDescriptionWithoutServerObjectId*
public java.lang.String **getDescriptionWithoutServerObjectId**(java.lang.String **lineBreakString**)
 - **Usage**
 - * This method will return the description of a bookmark or bookmark folder without the serverObjectId information. The line break string which is CR+LF (CHR(13)+CHR(10)) will be replaced by the specified linebreak.
 - **Parameters**

- * `lineBreakString` - Line break string used by the client.
 - **Returns** - Returns the description as String without server object id.

- *getServerObjectId*
`public int getServerObjectId()`
 - **Usage**
 - * The unique ObjectID normally used on a server to identify the object (in a database table).
 - **Returns** - Returns the unique ObjectId as int.

- *getServerObjectIdEndTag*
`private java.lang.String getServerObjectIdEndTag()`
 - **Usage**
 - * The method will return the tag to identify the end of the object id information from the server that is again stored in a description.
 - **Returns** - The serverObjectIdEndTag.

- *getServerObjectIdStartTag*
`private java.lang.String getServerObjectIdStartTag()`
 - **Usage**
 - * The method will return the tag to identify the end of the object id information from the server that is again stored in a description.
 - **Returns** - The serverObjectIdStartTag.

- *getTitle*
`public java.lang.String getTitle()`
 - **Usage**
 - * This method will return the title of a bookmark or bookmark folder.
 - **Returns** - Returns the title as String.

- *replaceAll*
`private java.lang.String replaceAll(java.lang.String inputString, java.lang.String searchString, java.lang.String replacement)`
 - **Usage**
 - * This method is an replacement for the replaceAll function available since Java 1.4 because this code should be compatible with Java 1.3 (or even older).
The given string will checked for the searchString that will be replaced by the replacement.
Finally the new String will be returned.
 - **Parameters**
 - * `inputString` - The string to work with.
 - * `searchString` - The string that will be searched for.
 - * `replacement` - The replacement string used instead of searchString.
 - **Returns** - The new string where all searched substrings have been replaced.

- *setCreatedDate*
`protected void setCreatedDate(long createdDate)`
 - **Usage**
 - * This method allows to set the timestamp when the bookmark folder or bookmark has been created/added. The timestamp needs to be in milliseconds.
 - **Parameters**
 - * `createdDate` - The createdDate to set.

- *setDescription*
`protected void setDescription(java.lang.String description)`

- **Usage**
 - * This method allows to store the description of a bookmark or bookmark folder.
 - **Parameters**
 - * `description` - The description to set.
-

- *setDescription*

```
protected void setDescription( java.lang.String description, java.lang.String  
lineBreakString )
```

- **Usage**
 - * This method allows to store the description of a bookmark or bookmark folder. The line break string used by the client (`lineBreakString`) will be replaced by CR+LF (CHR(13)+CHR(10)) which is used internally by JBMSync.
 - **Parameters**
 - * `description` - The description to set
 - * `lineBreakString` - Line break string used by the client.
-

- *setServerObjectId*

```
protected void setServerObjectId( int serverObjectId )
```

- **Usage**
 - * The unique ObjectID normally used on a server to identify the object (in a database table).
 - **Parameters**
 - * `serverObjectId` - The objectId to set.
-

- *setTitle*

```
protected void setTitle( java.lang.String title )
```

- **Usage**
 - * This method allows to store the title of a bookmark or bookmark folder. If the parameter is null or empty the title will be automatically set to 'Unknown title'.
- **Parameters**
 - * `title` - The title to set.

3.1.6 CLASS ComplexBookmarkAndFolderAbstract

This abstract class is the base for a specialized ComplexBookmark and ComplexFolder class which store the information.

DECLARATION

```
public abstract class ComplexBookmarkAndFolderAbstract  
extends java.lang.Object
```

FIELDS

- private static final String serverObjectIdStartTag
 - The tag to identify the start of the server object id information from the server that is stored in a description.
- private static final String serverObjectIdEndTag
 - The tag to identify the end of the server object id information from the server that is stored in a description.
- private long createDate
 - Timestamp Firefox/Mozilla/Netscape (ADD_DATE="<number>") and Opera (CREATED=<number>) when the URL has been created (added). The timestamp is in milliseconds.
- private String title
 - The title of a bookmark or bookmark folder.
- private String description
 - The description of a bookmark or bookmark folder.
- private int serverObjectId
 - The unique ObjectID normally used on a server to identify the object (in a database table).

CONSTRUCTORS

- *ComplexBookmarkAndFolderAbstract*
ComplexBookmarkAndFolderAbstract()
 - **Usage**
 - * The default constructor of this class.

METHODS

- *getCreateDate*
public long **getCreateDate()**
 - **Usage**
 - * This method will return the timestamp when the bookmark folder or bookmark has been created/added. The timestamp is in milliseconds.
 - **Returns** - Returns the createDate as long.
- *getDescription*
public java.lang.String **getDescription()**
 - **Usage**

- * This method will return the description of a bookmark or bookmark folder. Line breaks use CR+LF (CHR(13)+CHR(10)) which may not compatible to the format where you may want to use the data. See also {link #getDescription(String)}.

– **Returns** - Returns the description as String.

- *getDescription*

```
public java.lang.String getDescription( java.lang.String lineBreakString )
```

– **Usage**

- * This method will return the description of a bookmark or bookmark folder. The line break string which is CR+LF (CHR(13)+CHR(10)) will be replaced by the specified linebreak.

– **Parameters**

- * **lineBreakString** - Line break string used by the client.

– **Returns** - Returns the description as String.

- *getDescriptionWithoutServerObjectId*

```
public java.lang.String getDescriptionWithoutServerObjectId( )
```

– **Usage**

- * This method will return the description of a bookmark or bookmark folder without the serverObjectId information. Line breaks use CR+LF (CHR(13)+CHR(10)) which may not compatible to the format where you may want to use the data. See also {link #getDescriptionWithoutServerObjectId(String)}.

– **Returns** - Returns the description as String without server object id.

- *getDescriptionWithoutServerObjectId*

```
public java.lang.String getDescriptionWithoutServerObjectId( java.lang.String lineBreakString )
```

– **Usage**

- * This method will return the description of a bookmark or bookmark folder without the serverObjectId information. The line break string which is CR+LF (CHR(13)+CHR(10)) will be replaced by the specified linebreak.

– **Parameters**

- * **lineBreakString** - Line break string used by the client.

– **Returns** - Returns the description as String without server object id.

- *getServerObjectId*

```
public int getServerObjectId( )
```

– **Usage**

- * The unique ObjectId normally used on a server to identify the object (in a database table).

– **Returns** - Returns the unique ObjectId as int.

- *getServerObjectIdEndTag*

```
private java.lang.String getServerObjectIdEndTag( )
```

– **Usage**

- * The method will return the tag to identify the end of the object id information from the server that is again stored in a description.

– **Returns** - The serverObjectIdEndTag.

• *getServerObjectIdStartTag*

```
private java.lang.String getServerObjectIdStartTag( )
```

– **Usage**

* The method will return the tag to identify the end of the object id information from the server that is again stored in a description.

– **Returns** - The serverObjectIdStartTag.

• *getTitle*

```
public java.lang.String getTitle( )
```

– **Usage**

* This method will return the title of a bookmark or bookmark folder.

– **Returns** - Returns the title as String.

• *replaceAll*

```
private java.lang.String replaceAll( java.lang.String inputString,  
java.lang.String searchString, java.lang.String replacement )
```

– **Usage**

* This method is an replacement for the replaceAll function available since Java 1.4 because this code should be compatible with Java 1.3 (or even older). The given string will checked for the searchString that will be replaced by the replacement. Finally the new String will be returned.

– **Parameters**

* **inputString** - The string to work with.

* **searchString** - The string that will be searched for.

* **replacement** - The replacement string used instead of searchString.

– **Returns** - The new string where all searched substrings have been replaced.

• *setCreatedDate*

```
protected void setCreatedDate( long createdDate )
```

– **Usage**

* This method allows to set the timestamp when the bookmark folder or bookmark has been created/added. The timestamp needs to be in milliseconds.

– **Parameters**

* **createdDate** - The createdDate to set.

• *setDescription*

```
protected void setDescription( java.lang.String description )
```

– **Usage**

* This method allows to store the description of a bookmark or bookmark folder.

– **Parameters**

* **description** - The description to set.

• *setDescription*

```
protected void setDescription( java.lang.String description, java.lang.String  
lineBreakString )
```

– **Usage**

- * This method allows to store the description of a bookmark or bookmark folder. The line break string used by the client (lineBreakString) will be replaced by CR+LF (CHR(13)+CHR(10)) which is used internally by JBMSync.

– **Parameters**

- * **description** - The description to set
- * **lineBreakString** - Line break string used by the client.

- *setServerObjectId*

protected void **setServerObjectId**(int **serverObjectId**)

– **Usage**

- * The unique ObjectID normally used on a server to identify the object (in a database table).

– **Parameters**

- * **serverObjectId** - The objectId to set.

- *setTitle*

protected void **setTitle**(java.lang.String **title**)

– **Usage**

- * This method allows to store the title of a bookmark or bookmark folder. If the parameter is null or empty the title will be automatically set to 'Unknown title'.

– **Parameters**

- * **title** - The title to set.

3.1.7 CLASS ComplexBookmarkAndFolderComparator

This class allows to compare two `ComplexBookmark` or `ComplexBookmarkFolder` objects using the compare function.

This class is using several static `areEqual(..)` methods to simplify comparing the values. `Arrays` are not handled by these helper methods``. This is because the `Arrays.equals` methods should be used for array fields.

DECLARATION

```
public class ComplexBookmarkAndFolderComparator
extends java.lang.Object
```

FIELDS

- private int debugMode
 - Debug mode (0 = Debug output turned off)
- private static final int BEFORE
 - Helper property of the compare methods. If the first value is smaller than the second value BEFORE is returned. If boolean values are compared BEFORE is returned when the first value is not true (!) and the second is true.
- private static final int EQUAL
 - Helper property of the compare methods. EQUAL is used if the compared values are the same.
- private static final int AFTER
 - Helper property of the compare methods. If the first value is bigger than the second value AFTER is returned. If boolean values are compared AFTER is returned when the first value is true and the second is not true (!).
- private long supportedBookmarkFolderFields
 - This property stores the intersecting fields of the client and server bookmark folder. This information will be used by the specialized compare methods.
- private long supportedBookmarkFields
 - This property stores the intersecting fields of the client and server bookmark. This information will be used by the specialized compare methods.

CONSTRUCTORS

- *ComplexBookmarkAndFolderComparator*
public ComplexBookmarkAndFolderComparator(long supportedClientBookmarkFields, long supportedClientBookmarkFolderFields, long supportedServerBookmarkFields, long supportedServerBookmarkFolderFields)
 - **Usage**
 - * This constructor will create the comparator object and set the intersecting UniquePropertyID (UPI) of the client and server bookmark and bookmark folder.
 - **Parameters**
 - * **supportedClientBookmarkFields** - Supported UniquePropertyID (UPI) for client bookmark fields.
 - * **supportedClientBookmarkFolderFields** - Supported UniquePropertyID (UPI) for client bookmark folder fields.
 - * **supportedServerBookmarkFields** - Supported UniquePropertyID (UPI) for server bookmark fields.
 - * **supportedServerBookmarkFolderFields** - Supported UniquePropertyID (UPI) for server bookmark folder fields.

METHODS

- *compare*

```
public int compare( java.lang.Object obj0, java.lang.Object obj1 )
```

- **Usage**

- * This method will be responsible for the basic compare of the given objects and will then call the special `ComplexBookmark` or `ComplexBookmarkFolder` compare methods.

Both objects have to be from the same type and the supported bookmark folder and bookmarks fields have to be set before this method can be used (see Constructor).

- **Parameters**

- * `obj0` - The first object (`ComplexBookmark` or `ComplexBookmarkFolder`) to compare. Both objects have to be from the same type.

- * `obj1` - The second object (`ComplexBookmark` or `ComplexBookmarkFolder`) to compare. Both objects have to be from the same type.

- **Returns** - Returns an integer to indicate if the compared objects are the same (0) nor not (-1 or 1).

- **Exceptions**

- * `java.lang.ClassCastException` -

- * `de.juergenulbts.jbookmarksync.exception.BookmarkFieldsNotSetException`

-

- *compareComplexBookmark*

```
private int compareComplexBookmark(  
de.juergenulbts.jbookmarksync.util.ComplexBookmark obj0,  
de.juergenulbts.jbookmarksync.util.ComplexBookmark obj1 )
```

- **Usage**

- * This method is responsible to comparing the fields of the two `ComplexBookmarkFolder` objects passed to this method. It's using the `#supportedBookmarkFolderFields` information to decide which fields need to be compared.

It is using the using the static helper methods `areEqual(..)` to compare the fields.

- **Parameters**

- * `obj0` - The first `ComplexBookmark` object to compare.

- * `obj1` - The second `ComplexBookmark` object to compare.

- **Returns** - Returns an integer to indicate if the compared objects are the same (0) nor not (-1 or 1).

- *compareComplexBookmarkFolder*

```
private int compareComplexBookmarkFolder(  
de.juergenulbts.jbookmarksync.util.ComplexBookmarkFolder obj0,  
de.juergenulbts.jbookmarksync.util.ComplexBookmarkFolder obj1 )
```

- **Usage**

- * This method is responsible to comparing the fields of the two `ComplexBookmark` objects passed to this method. It's using the `#supportedBookmarkFields` information to decide which fields need to be compared.

It is using the using the static helper methods `areEqual(..)` to compare the fields.

– **Parameters**

- * obj0 - The first ComplexBookmarkFolder object to compare.
- * obj1 - The second ComplexBookmarkFolder object to compare.

– **Returns** - Returns an integer to indicate if the compared objects are the same (0) nor not (-1 or 1).

• *compareLastModifiedDate*

```
public int compareLastModifiedDate( java.lang.Object obj0,  
java.lang.Object obj1 )
```

– **Usage**

- * This method will be responsible for the basic compare of the given objects and will then call the special ComplexBookmark or ComplexBookmarkFolder compare methods.

Both objects have to be from the same type and the supported bookmark folder and bookmarks fields have to be set before this method can be used (see Constructor).

– **Parameters**

- * obj0 - The first object (ComplexBookmark or ComplexBookmarkFolder) to compare. Both objects have to be from the same type.
- * obj1 - The second object (ComplexBookmark or ComplexBookmarkFolder) to compare. Both objects have to be from the same type.

– **Returns** - Returns an integer to indicate if the compared objects are the same (0) nor not (-1 or 1).

– **Exceptions**

- * java.lang.ClassCastException -
- *
de.juergenulbts.jbookmarksync.exception.BookmarkFieldNotSupportedException
-

• *compareModifiedDateComplexBookmark*

```
private int compareModifiedDateComplexBookmark(  
de.juergenulbts.jbookmarksync.util.ComplexBookmark obj0,  
de.juergenulbts.jbookmarksync.util.ComplexBookmark obj1 )
```

– **Usage**

- * This method will compare the last date information of both ComplexBookmark objects.

– **Parameters**

- * obj0 - The first ComplexBookmark object to compare.
- * obj1 - The second ComplexBookmark object to compare.

– **Returns** - Returns an integer to indicate if the compared objects are the same (0) nor not (-1 or 1). It returns 1 if obj0 has a newer date then obj1, otherwise -1.

• *compareModifiedDateComplexBookmarkFolder*

```
private int compareModifiedDateComplexBookmarkFolder(  
de.juergenulbts.jbookmarksync.util.ComplexBookmarkFolder obj0,  
de.juergenulbts.jbookmarksync.util.ComplexBookmarkFolder obj1 )
```

– **Usage**

- * This method will compare the last date information of both ComplexBookmarkFolder objects.

– **Parameters**

- * obj0 - The first `ComplexBookmarkFolder` object to compare.
- * obj1 - The second `ComplexBookmarkFolder` object to compare.

– **Returns** - Returns an integer to indicate if the compared objects are the same (0) nor not (-1 or 1). It returns 1 if obj0 has a newer date then obj1, otherwise -1.

• *compareValue*

```
private int compareValue( boolean value0, boolean value1 )
```

– **Usage**

- * This method will compare two boolean values and return the result.

– **Parameters**

- * value0 - The first boolean value to compare.
- * value1 - The second boolean value to compare.

– **Returns** - The result of the comparison as int (0 if equal or -1 or +1 if not equal).

• *compareValue*

```
private int compareValue( char value0, char value1 )
```

– **Usage**

- * This method will compare two char values and return the result.

– **Parameters**

- * value0 - The first char value to compare.
- * value1 - The second char value to compare.

– **Returns** - The result of the comparison as int (0 if equal or -1 or +1 if not equal).

• *compareValue*

```
private int compareValue( double value0, double value1 )
```

– **Usage**

- * This method will compare two double values and return the result.

– **Parameters**

- * value0 - The first double value to compare.
- * value1 - The second double value to compare.

– **Returns** - The result of the comparison as int (0 if equal or -1 or +1 if not equal).

• *compareValue*

```
private int compareValue( float value0, float value1 )
```

– **Usage**

- * This method will compare two float values and return the result.

– **Parameters**

- * value0 - The first float value to compare.
- * value1 - The second float value to compare.

– **Returns** - The result of the comparison as int (0 if equal or -1 or +1 if not equal).

• *compareValue*

```
private int compareValue( long value0, long value1 )
```

– **Usage**

- * This method will compare two long (byte, short, int) values and return the result.

– **Parameters**

- * **value0** - The first long (byte, short, int) value to compare.
 - * **value1** - The second long (byte, short, int) value to compare.
 - **Returns** - The result of the comparison as int (0 if equal or -1 or +1 if not equal).

- *getSupportedBookmarkFields*
private long getSupportedBookmarkFields()
 - **Usage**
 - * This method will return the intersecting fields of the client and server bookmark. It is used to decide which fields of the **ComplexBookmark** object have to be compared.
 - **Returns** - Returns the supportedBookmarkFields value as long.

- *getSupportedBookmarkFolderFields*
private long getSupportedBookmarkFolderFields()
 - **Usage**
 - * This method will return the intersecting fields of the client and server bookmark folder. It is used to decide which fields of the **ComplexBookmarkFolder** object have to be compared.
 - **Returns** - Returns the supportedBookmarkFolderFields value as long.

- *setIntersectingBookmarkFields*
private void setIntersectingBookmarkFields(long supportedClientBookmarkFields, long supportedServerBookmarkFields)
 - **Usage**
 - * This method will calculate and store the unique property id for bookmarks supported by both, the client and the server.
 - **Parameters**
 - * **supportedClientBookmarkFields** - Supported UniquePropertyID (UPI) for client bookmark fields.
 - * **supportedServerBookmarkFields** - Supported UniquePropertyID (UPI) for server bookmark fields.

- *setIntersectingBookmarkFolderFields*
private void setIntersectingBookmarkFolderFields(long supportedClientBookmarkFolderFields, long supportedServerBookmarkFolderFields)
 - **Usage**
 - * This method will calculate and store the unique property id for bookmark folders supported by both, the client and the server.
 - **Parameters**
 - * **supportedClientBookmarkFolderFields** - Supported UniquePropertyID (UPI) for client bookmark folder fields.
 - * **supportedServerBookmarkFolderFields** - Supported UniquePropertyID (UPI) for server bookmark folder fields.

- *setSupportedBookmarkFields*
private void setSupportedBookmarkFields(long supportedBookmarkFields)
 - **Usage**

- * This method will save the intersecting fields of the client and server bookmark. It is used to decide which fields of the `ComplexBookmark` object have to be compared.

– **Parameters**

- * `supportedBookmarkFields` - The `supportedBookmarkFields` to set.

• *setSupportedBookmarkFolderFields*

```
private void setSupportedBookmarkFolderFields( long supportedBookmarkFolderFields )
```

– **Usage**

- * This method will save the intersecting fields of the client and server bookmark folder. It is used to decide which fields of the `ComplexBookmarkFolder` object have to be compared.

– **Parameters**

- * `supportedBookmarkFolderFields` - The `supportedBookmarkFolderFields` to set.

3.1.8 CLASS `ComplexBookmarkFolder`

This class is able to store bookmark folder data including the title, description and other data.

DECLARATION

```
public class ComplexBookmarkFolder
extends de.juergenulbts.jbookmarksync.util.ComplexBookmarkAndFolderAbstract
```

FIELDS

- private `String` `keyword`
 - Optional Opera keyword setting (`SHORT_NAME=...`).
- private `long` `lastModifiedDate`
 - Optional timestamp used by Firefox/Mozilla/Netscape (`LAST_MODIFIED="<number>"`). The timestamp is in milliseconds.
- private `boolean` `onPersonalBar`
 - Optional Opera setting (`ON_PERSONALBAR=YES`) to show the folder on the personal.
- private `int` `positionInsidePersonalBar`
 - Optional Opera setting (`PERSONALBAR_POS=<number>`) to identify the position of an entry inside the personal bar.
- private `boolean` `personalToolbarFolder`

- Optional Mozilla setting (PERSONAL_TOOLBAR_FOLDER) is only allowed on one folder while Opera allows multiple folders to be used in it's personal bar.

CONSTRUCTORS

- *ComplexBookmarkFolder*

```
public ComplexBookmarkFolder( int serverObjectId, java.lang.String title,
java.lang.String description, java.lang.String keyword, long createdAt,
long lastModifiedDate )
```

- **Usage**

- * Constructor to save the folder settings from Open-Xchange or other server stored bookmark data.

- **Parameters**

- * **serverObjectId** - The unique ObjectID used on the server to identify the bookmark object.
- * **title** - The title of the bookmark folder.
- * **description** - The description of the bookmark folder.
- * **keyword** - The keyword of the bookmark folder.
- * **createdAt** - Timestamp when the bookmark folder has been created or added.
- * **lastModifiedDate** - Timestamp when the bookmark has been modified the last time.

- *ComplexBookmarkFolder*

```
public ComplexBookmarkFolder( java.lang.String title, java.lang.String
description, long createdAt, long lastModifiedDate, boolean
personalToolbarFolder, java.lang.String clientLineBreakString )
```

- **Usage**

- * Constructor to save the folder settings from Mozilla/Firefox/Netscape.

- **Parameters**

- * **title** - The title of the bookmark folder.
- * **description** - The description of the bookmark folder.
- * **createdAt** - Timestamp when the bookmark folder has been created or added.
- * **lastModifiedDate** - Timestamp of the last modification
- * **personalToolbarFolder** - The boolean value if the folder is the personal toolbar folder or not.
- * **clientLineBreakString** - Line break string used by the client.

- *ComplexBookmarkFolder*

```
public ComplexBookmarkFolder( java.lang.String title, java.lang.String
description, java.lang.String keyword, long createdAt, boolean
onPersonalBar, int positionInsidePersonalBar, java.lang.String
clientLineBreakString )
```

- **Usage**

- * Constructor to save the folder settings from Opera.

- **Parameters**

- * **title** - The title of the bookmark folder.
- * **description** - The description of the bookmark folder.
- * **keyword** - The keyword of the bookmark folder.

- * **createdDate** - Timestamp when the bookmark folder has been created or added.
- * **onPersonalBar** - The boolean value if the entry is shown on the personal bar or not.
- * **positionInsidePersonalBar** - Setting to identify the position of an entry inside the personal bar.
- * **clientLineBreakString** - Line break string used by the client.

METHODS

- *getKeyword*
public java.lang.String **getKeyword**()
 - **Usage**
 - * This method will return the keyword.
 - **Returns** - Returns the keyword as String.

- *getLastModifiedDate*
public long **getLastModifiedDate**()
 - **Usage**
 - * This method will return the timestamp when the bookmark has been modified the last time. The timestamp is in milliseconds.
 - **Returns** - Returns the lastModifiedDate as long.

- *getPositionInsidePersonalBar*
public int **getPositionInsidePersonalBar**()
 - **Usage**
 - * This method will return the position inside the personal bar (used e.g. by Opera).
 - **Returns** - Returns the positionInsidePersonalBar.

- *isOnPersonalBar*
public boolean **isOnPersonalBar**()
 - **Usage**
 - * This method will return the setting of this entry for the personal bar (used e.g. by Opera). It returns true if the entry is visible on the personal bar otherwise false.
 - **Returns** - Returns status if the entry is shown on the personal bar or not.

- *isPersonalToolbarFolder*
public boolean **isPersonalToolbarFolder**()
 - **Usage**
 - * This method will return the setting of this entry for the personal toolbar folder (only used by Netscape/Mozilla). It returns true if the folder is the personal toolbar folder otherwise false.
 - **Returns** - Returns status if the folder is the personal toolbar folder or not.

- *setKeyword*
protected void **setKeyword**(java.lang.String keyword)
 - **Usage**

* This method allows to set the keyword.

– **Parameters**

* keyword - The keyword to set.

• *setLastModifiedDate*

protected void **setLastModifiedDate**(long lastModifiedDate)

– **Usage**

* This method allows to set the timestamp when the bookmark folder has been modified the last time. The timestamp needs to be in milliseconds.

– **Parameters**

* lastModifiedDate - The lastModifiedDate to set.

• *setOnPersonalBar*

protected void **setOnPersonalBar**(boolean onPersonalBar)

– **Usage**

* This method allows to set the status of this entry for the personal bar (used e.g. by Opera).

– **Parameters**

* onPersonalBar - The boolean value if the entry is shown on the personal bar or not.

• *setPersonalToolbarFolder*

protected void **setPersonalToolbarFolder**(boolean personalToolbarFolder)

– **Usage**

* This method allows to set the status of this entry for the personal bar (only used by Netscape/Mozilla).

– **Parameters**

* personalToolbarFolder - The boolean value if the folder is the personal toolbar folder or not.

• *setPositionInsidePersonalBar*

protected void **setPositionInsidePersonalBar**(int positionInsidePersonalBar)

– **Usage**

* This method allows to set the position inside the personal bar (used e.g. by Opera).

– **Parameters**

* positionInsidePersonalBar - The positionInsidePersonalBar to set.

• *toString*

public java.lang.String **toString**()

– **Usage**

* This method will create a string of the content of this bookmark. All date and time calculations are based on the TimeZone 'GMT'!

– **Returns** - Returns the bookmark as String object.

METHODS INHERITED FROM CLASS

de.juergenulbts.jbookmarksync.util.ComplexBookmarkAndFolderAbstract

(in 3.1.6, page 37)

- *getCreatedDate*
public long getCreatedDate()
 - **Usage**
 - * This method will return the timestamp when the bookmark folder or bookmark has been created/added. The timestamp is in milliseconds.
 - **Returns** - Returns the createdDate as long.

- *getDescription*
public java.lang.String getDescription()
 - **Usage**
 - * This method will return the description of a bookmark or bookmark folder. Line breaks use CR+LF (CHR(13)+CHR(10)) which may not compatible to the format where you may want to use the data. See also {link #getDescription(String)}.
 - **Returns** - Returns the description as String.

- *getDescription*
public java.lang.String getDescription(java.lang.String lineBreakString)
 - **Usage**
 - * This method will return the description of a bookmark or bookmark folder. The line break string which is CR+LF (CHR(13)+CHR(10)) will be replaced by the specified linebreak.
 - **Parameters**
 - * **lineBreakString** - Line break string used by the client.
 - **Returns** - Returns the description as String.

- *getDescriptionWithoutServerObjectId*
public java.lang.String getDescriptionWithoutServerObjectId()
 - **Usage**
 - * This method will return the description of a bookmark or bookmark folder without the serverObjectId information. Line breaks use CR+LF (CHR(13)+CHR(10)) which may not compatible to the format where you may want to use the data. See also {link #getDescriptionWithoutServerObjectId(String)}.
 - **Returns** - Returns the description as String without server object id.

- *getDescriptionWithoutServerObjectId*
public java.lang.String getDescriptionWithoutServerObjectId(java.lang.String lineBreakString)
 - **Usage**
 - * This method will return the description of a bookmark or bookmark folder without the serverObjectId information. The line break string which is CR+LF (CHR(13)+CHR(10)) will be replaced by the specified linebreak.
 - **Parameters**
 - * **lineBreakString** - Line break string used by the client.
 - **Returns** - Returns the description as String without server object id.

- *getServerObjectId*
public int getServerObjectId()

- **Usage**
 - * The unique ObjectID normally used on a server to identify the object (in a database table).
 - **Returns** - Returns the unique ObjectId as int.

- *getServerObjectIdEndTag*
private java.lang.String **getServerObjectIdEndTag**()
 - **Usage**
 - * The method will return the tag to identify the end of the object id information from the server that is again stored in a description.
 - **Returns** - The serverObjectIdEndTag.

- *getServerObjectIdStartTag*
private java.lang.String **getServerObjectIdStartTag**()
 - **Usage**
 - * The method will return the tag to identify the end of the object id information from the server that is again stored in a description.
 - **Returns** - The serverObjectIdStartTag.

- *getTitle*
public java.lang.String **getTitle**()
 - **Usage**
 - * This method will return the title of a bookmark or bookmark folder.
 - **Returns** - Returns the title as String.

- *replaceAll*
private java.lang.String **replaceAll**(java.lang.String **inputString**, java.lang.String **searchString**, java.lang.String **replacement**)
 - **Usage**
 - * This method is an replacement for the replaceAll function available since Java 1.4 because this code should be compatible with Java 1.3 (or even older).
The given string will checked for the searchString that will be replaced by the replacement.
Finally the new String will be returned.
 - **Parameters**
 - * **inputString** - The string to work with.
 - * **searchString** - The string that will be searched for.
 - * **replacement** - The replacement string used instead of searchString.
 - **Returns** - The new string where all searched substrings have been replaced.

- *setCreatedDate*
protected void **setCreatedDate**(long **createdDate**)
 - **Usage**
 - * This method allows to set the timestamp when the bookmark folder or bookmark has been created/added. The timestamp needs to be in milliseconds.
 - **Parameters**
 - * **createdDate** - The createdDate to set.

- *setDescription*
protected void **setDescription**(java.lang.String **description**)
 - **Usage**
 - * This method allows to store the description of a bookmark or bookmark folder.
 - **Parameters**
 - * **description** - The description to set.

- *setDescription*
protected void **setDescription**(java.lang.String **description**, java.lang.String **lineBreakString**)
 - **Usage**
 - * This method allows to store the description of a bookmark or bookmark folder. The line break string used by the client (lineBreakString) will be replaced by CR+LF (CHR(13)+CHR(10)) which is used internally by JBMSync.
 - **Parameters**
 - * **description** - The description to set
 - * **lineBreakString** - Line break string used by the client.

- *setServerObjectId*
protected void **setServerObjectId**(int **serverObjectId**)
 - **Usage**
 - * The unique ObjectID normally used on a server to identify the object (in a database table).
 - **Parameters**
 - * **serverObjectId** - The objectId to set.

- *setTitle*
protected void **setTitle**(java.lang.String **title**)
 - **Usage**
 - * This method allows to store the title of a bookmark or bookmark folder. If the parameter is null or empty the title will be automatically set to 'Unknown title'.
 - **Parameters**
 - * **title** - The title to set.

3.1.9 CLASS MozillaBookmarkParser

This class will process bookmark files based on the Netscape/Mozilla bookmark format used by several different webbrowsers (e.g. Mozilla-Suite, Firefox, SeaMonkey, Netscape, Camino and other).

DECLARATION

```
public class MozillaBookmarkParser
extends de.juergenulbts.jbookmarksync.util.BookmarkParserAbstract
```

FIELDS

- private static final char LF
 - LF character found at ASCII/UNICODE table position 10 (0x0a).
- private int debugMode
 - Debug mode (0 = Debug output turned off)

- private Pattern linePattern
 - Pattern to look for (reading line by line).
- private Matcher matcher
 - Matcher which will return the results found by the given pattern.
- private final String bookmarkTag
 - Tag to identify the bookmark.
- private final String urlTag
 - Tag to identify the url.
- private final String feedUrlTag
 - Tag to identify the feedUrl of a live bookmark.
This should normally be the equal to the value found on the urlTag but sometimes only the feedUrl is given.
- private final String bookmarkFolderTag
 - Tag to identify the bookmark folder.
- private final String descriptionTag
 - Tag to identify the description.
- private final String descriptionLineBreakString
 - String used for linebreaks inside the description.
- private final String addDateTag
 - Tag to identify the date the bookmark or bookmark folder has been added.
- private final String lastVisitDateTag
 - Tag to identify the last date (timestamp) the bookmark (url) has been visited.
- private final String lastModificationDateTag
 - Tag to identify the last modification date (timestamp) of the bookmark or bookmark folder.
- private final String favIconTag
 - Tag to identify the FavIcon of the url. Further processing of the FavIcon data is done with the methods `#getFavIconImageMimeType(String)` and `#getFavIconImageData(String)` .
- private final String characterSetTag
 - Tag to identify the used character set ->currently unused. TODO: Maybe useful to support LAST_CHARSET setting
- private final String keywordTag
 - Tag to identify the keywords.
- private final String sideBarTag
 - Tag to identify if the url should be added to the sidebar.
- private final String personalToolbarFolderTag
 - Tag to identify the personal toolbar folder setting.
- private long supportedBookmarkFolderFields

-
- private long supportedBookmarkFields
-

CONSTRUCTORS

- *MozillaBookmarkParser*
`public MozillaBookmarkParser()`
 - **Usage**
 - * The default constructor of this class.

METHODS

- *getAddDateTag*
`private java.lang.String getAddDateTag()`
 - **Usage**
 - * This method returns the tag to identify the date the bookmark or bookmark folder has been added.
 - **Returns** - Returns the addDateTag.

- *getBookmark*
`private de.juergenulbts.jbookmarksync.util.ComplexBookmark getBookmark(java.lang.String line)`
 - **Usage**
 - * This method will parse the bookmark and gather all information which is finally stored inside a `ComplexBookmark` object. If there is a problem it returns null.
 - **Parameters**
 - * `line` - One line of the bookmark file which has to be parsed.
 - **Returns** - `ComplexBookmark`

- *getBookmarkFolder*
`private de.juergenulbts.jbookmarksync.util.ComplexBookmarkFolder getBookmarkFolder(java.lang.String line)`
 - **Usage**
 - * This method will parse the bookmark folder and gather all information which is finally stored inside a `ComplexBookmarkFolder` object. If there is a problem it returns null.
 - **Parameters**
 - * `line` - One line of the bookmark file which has to be parsed.
 - **Returns** - `ComplexBookmarkFolder`

- *getBookmarkFolderTag*
`private java.lang.String getBookmarkFolderTag()`

– **Usage**

* This method returns the tag to identify a bookmark folder.

– **Returns** - Returns the bookmarkFolderTag.

• *getBookmarks*

public java.util.Collection **getBookmarks**(byte[] bookmarkFile)

– **Usage**

* This method calls methods to create the bookmark array out of BookmarkStorageContainer objects which contain either a ComplexBookmark or a ComplexBookmarkFolder object. The object type of the stored object inside the BookmarkStorageContainer can checked with a method.

– **Parameters**

* bookmarkFile - The complete bookmark file as byte array.

– **Returns** - Collection containing BookmarkStorageContainer objects

• *getBookmarkTag*

private java.lang.String **getBookmarkTag**()

– **Usage**

* This method returns the tag to identify a bookmark.

– **Returns** - Returns the bookmarkTag.

• *getCharacterSetTag*

private java.lang.String **getCharacterSetTag**()

– **Usage**

* This method returns the character set used in the bookmark file. TODO: CharacterSet (encoding) of mozilla bookmark file is not used.

– **Returns** - Returns the characterSetTag.

• *getDescriptionLineBreakString*

private java.lang.String **getDescriptionLineBreakString**()

– **Usage**

* This method will return the string used for linebreaks inside the description.

– **Returns** - Returns the descriptionLineBreakString.

• *getDescriptionTag*

private java.lang.String **getDescriptionTag**()

– **Usage**

* This method returns the tag to identify a description used on a bookmark and bookmark folder.

– **Returns** - Returns the descriptionTag.

• *getFavIconImageData*

private java.lang.String **getFavIconImageData**(java.lang.String favIconData)

– **Usage**

* This method will only return the image data as base64 encoded string of the FavIcon data given by a parameter.

– **Parameters**

* **favIconData** - The data of the favIcon as it has been retrieved from the bookmark.

– **Returns** - Returns the image data as base64 encoded string or null if there is no image data.

• *getFavIconImageMimeType*

```
private java.lang.String getFavIconImageMimeType( java.lang.String  
favIconData )
```

– **Usage**

* This method will only return the mime type of the FavIcon data given by a parameter.

– **Parameters**

* **favIconData** - The data of the favIcon as it has been retrieved from the bookmark.

– **Returns** - Returns the mime type of the image as string or null if there is no mime type.

• *getFavIconTag*

```
private java.lang.String getFavIconTag( )
```

– **Usage**

* This method returns the tag to identify the FavIcon of the url.

– **Returns** - Returns the favIconTag.

• *getFeedUrlTag*

```
private java.lang.String getFeedUrlTag( )
```

– **Usage**

* This method returns the tag to identify the feedUrl of a live bookmark. This should normally be the equal to the value found on the urlTag but sometimes only the feedUrl is given.

– **Returns** - Returns the urlTag.

• *getKeywordTag*

```
private java.lang.String getKeywordTag( )
```

– **Usage**

* This method returns the tag to identify the keyword.

– **Returns** - Returns the keywordTag.

• *getLastModificationDateTag*

```
private java.lang.String getLastModificationDateTag( )
```

– **Usage**

* This method returns the tag to identify the last modification date (timestamp) of a bookmark or bookmark folder.

– **Returns** - Returns the lastModificationDateTag.

• *getLastVisitDateTag*

```
private java.lang.String getLastVisitDateTag( )
```

– **Usage**

* This method returns the tag to identify the last date (timestamp) a bookmark (the url) has been visited.

– **Returns** - Returns the lastVisitDateTag.

• *getLinePattern*

private java.util.regex.Pattern **getLinePattern**()

– **Usage**

* This method returns the stored pattern to look for (reading line by line).

– **Returns** - Returns the linePattern.

• *getMatcher*

private java.util.regex.Matcher **getMatcher**()

– **Usage**

* This method will return the matcher which uses a pattern to parse the bookmark file and return results found by the pattern.

– **Returns** - Returns the matcher.

• *getPersonalToolbarFolderTag*

private java.lang.String **getPersonalToolbarFolderTag**()

– **Usage**

* This method will return the tag to identify if the processed bookmark folder should be shown as personalToolbarFolder.
Creation date: (30.01.2007 20:36:21)

– **Returns** - Returns the personalToolbarFolderTag.

• *getSideBarTag*

private java.lang.String **getSideBarTag**()

– **Usage**

* This method returns the tag to identify if the url should be added to the sidebar.

– **Returns** - Returns the sideBarTag.

• *getSupportedBookmarkFields*

public long **getSupportedBookmarkFields**()

– **Returns** - supportedBookmarkFields as long.

– **See Also**

* de.juergenulbts.jbookmarksync.util.BookmarkParserAbstract.
getSupportedBookmarkFields (in 3.1.2, page 16)

• *getSupportedBookmarkFolderFields*

public long **getSupportedBookmarkFolderFields**()

– **Returns** - supportedBookmarkFolderFields as long.

– **See Also**

* de.juergenulbts.jbookmarksync.util.BookmarkParserAbstract.
getSupportedBookmarkFolderFields (in 3.1.2, page 16)

• *getUrlTag*

private java.lang.String **getUrlTag**()

– **Usage**

- * This method returns the tag to identify the url of a bookmark.
 - **Returns** - Returns the urlTag.
-

- *setMatcher*

```
private void setMatcher( java.util.regex.Matcher matcher )
```

- **Usage**

- * This method will set a matcher to be used to parse the bookmark file.

- **Parameters**

- * **matcher** - The matcher to set.

METHODS INHERITED FROM CLASS

de.juergenulbts.jbookmarksync.util.BookmarkParserAbstract

(in 3.1.2, page 14)

- *getBookmarks*

```
public abstract java.util.Collection getBookmarks( byte[] bookmarkFile )
```

- **Usage**

- * This method calls methods to create the bookmark array out of BookmarkStorageContainer objects which contain either a ComplexBookmark or a ComplexBookmarkFolder object. The object type of the stored object inside the BookmarkStorageContainer can checked with a method.

- **Parameters**

- * **bookmarkFile** - The complete bookmark file as byte array.

- **Returns** - Collection containing BookmarkStorageContainer objects

- *getServerObjectIdEndTag*

```
private java.lang.String getServerObjectIdEndTag( )
```

- **Usage**

- * The method will return the tag to identify the end of the object id information from the server that is again stored in a description.

- **Returns** - The serverObjectIdEndTag.

- *getServerObjectIdStartTag*

```
private java.lang.String getServerObjectIdStartTag( )
```

- **Usage**

- * The method will return the tag to identify the end of the object id information from the server that is again stored in a description.

- **Returns** - The serverObjectIdStartTag.

- *getSupportedBookmarkFields*

```
public abstract long getSupportedBookmarkFields( )
```

- **Usage**

- * This method will return the information which fields of a bookmark are supported.

- **Returns** - supportedBookmarkFields as long.

- *getSupportedBookmarkFolderFields*

```
public abstract long getSupportedBookmarkFolderFields( )
```

- **Usage**

- * This method will return the information which fields of a bookmark folder are supported.
 - **Returns** - supportedBookmarkFolderFields as long.

- *isServerObjectIdIncluded*
protected boolean **isServerObjectIdIncluded**(java.lang.String text)
 - **Usage**
 - * This method will check if the text includes the object id from the server which can be stored in description fields. It returns true if the object id could be found otherwise false.
 - **Parameters**
 - * text - The String that will be checked if it includes the server object id tag.
 - **Returns** - Returns status if the objectId start and end tags can be found in the submitted string.

- *retrieveServerObjectId*
protected int **retrieveServerObjectId**(java.lang.String text)
 - **Usage**
 - * This method will look for the object id of the server and return the integer value of it.
 - **Parameters**
 - * text - The String that will be checked if it includes the server object id.
 - **Returns** - Returns the server object Id as int value.

- *stripServerObjectIdFromDescription*
protected java.lang.String **stripServerObjectIdFromDescription**(java.lang.String text)
 - **Usage**
 - * This method will remove the object id of the server from the description.
 - **Parameters**
 - * text - The String that will be checked if it includes the server object id.
 - **Returns** - Returns the description string without the the server object id.

3.1.10 CLASS MozillaBookmarkWriter

This class will process previously stored bookmark and bookmark folder data and create a bookmark file out of it that can be used with Mozilla/Netscape based webbrowsers as the output complies with the Netscape bookmark file format.

The inner tags used for on bookmarks tags have been prefixed with a space as optimization (this saved one append() method call on a StringBuffer that will be called quite often).

Creation date: (09.02.2007 20:01:04)

DECLARATION

```
public class MozillaBookmarkWriter
extends java.lang.Object
implements de.juergenulbts.jbookmarksync.interfaces.IBookmarkWriter
```

FIELDS

- private static final String CRLF
 - Carriage Return + LineFeed - not platform dependent
- private static final String INDENT
 - Netscape Mozilla indent always by four blanks
- private static final char LF
 - LF character found at ASCII/UNICODE table position 10 (0x0a).
- private final String openListSectionTag
 - Tag to identify the opening of a list section.
- private final String closeListSectionTag
 - Tag to identify the closing of a list section.
- private final String openBookmarkTag
 - Tag to identify the opening of a bookmark.
- private final String closeBookmarkTag
 - Tag to identify the closing of a bookmark.
- private final String urlTag
 - Tag to identify the url.
- private final String feedUrlTag
 - Tag to identify the feedUrl of a live bookmark.
This should normally be the equal to the value found on the urlTag but sometimes only the feedUrl is given.
- private final String openBookmarkFolderTag
 - Tag to identify the opening bookmark folder.
- private final String closeBookmarkFolderTag
 - Tag to identify the opening bookmark folder.
- private final String descriptionTag
 - Tag to identify the description.
- private final String descriptionLineBreakString
 - String used for linebreaks inside the description.
- private final String addDateTag
 - Tag to identify the date the bookmark or bookmark folder has been added.
- private final String lastVisitDateTag
 - Tag to identify the last date (timestamp) the bookmark (url) has been visited.
- private final String lastModificationDateTag
 - Tag to identify the last modification date (timestamp) of the bookmark or bookmark folder.
- private final String favIconTag

- Tag to identify the FavIcon of the url. Further processing of the FavIcon data is done with the methods `#getFavIconImageMimeType(String)` and `#getFavIconImageData(String)` .
- private final String characterSetTag
 - Tag to identify the used character set ->currently unused. TODO: Maybe useful to support LAST_CHARSET setting
- private final String keywordTag
 - Tag to identify the keywords.
- private final String sideBarTag
 - Tag to identify if the url should be added to the sidebar.
- private final String personalToolbarFolderTag
 - Tag to identify the personal toolbar folder setting.
- private int objectCounter
 - Counter that shows how many bookmark and bookmark folders have been processed.

CONSTRUCTORS

- *MozillaBookmarkWriter*
`public MozillaBookmarkWriter()`

METHODS

- *constructHeader*
`private java.lang.String constructHeader()`
 - **Usage**
 - * This method constructs the basic header for the Netscape/Mozilla bookmark file format.
 - **Returns** - Returns the default header of a Netscape/Mozilla bookmark file as String.
- *getAddDateTag*
`private java.lang.String getAddDateTag()`
 - **Usage**
 - * This method returns the tag to identify the date the bookmark or bookmark folder has been added.
 - **Returns** - Returns the addDateTag.
- *getCharacterSetTag*
`private java.lang.String getCharacterSetTag()`
 - **Usage**
 - * This method returns the character set used in the bookmark file. TODO: CharacterSet (encoding) of mozilla bookmark file is not used.

– **Returns** - Returns the characterSetTag.

• *getCloseBookmarkFolderTag*

private java.lang.String **getCloseBookmarkFolderTag**()

– **Usage**

* This method returns the tag to identify the closing of a bookmark folder.

– **Returns** - Returns the closeBookmarkFolderTag.

• *getCloseBookmarkTag*

private java.lang.String **getCloseBookmarkTag**()

– **Usage**

* This method returns the tag to identify the closing of a bookmark.

– **Returns** - Returns the closeBookmarkTag as String.

• *getCloseListSectionTag*

private java.lang.String **getCloseListSectionTag**()

– **Usage**

* This method returns the tag to identify the closing of a list section.

– **Returns** - Returns the closings tag of a list section as String.

• *getDescriptionLineBreakString*

private java.lang.String **getDescriptionLineBreakString**()

– **Usage**

* This method will return the string used for linebreaks inside the description.

– **Returns** - Returns the descriptionLineBreakString.

• *getDescriptionTag*

private java.lang.String **getDescriptionTag**()

– **Usage**

* This method returns the tag to identify a description used on a bookmark and bookmark folder.

– **Returns** - Returns the descriptionTag.

• *getFavIconTag*

private java.lang.String **getFavIconTag**()

– **Usage**

* This method returns the tag to identify the FavIcon of the url.

– **Returns** - Returns the favIconTag.

• *getFeedUrlTag*

private java.lang.String **getFeedUrlTag**()

– **Usage**

* This method returns the tag to identify the feedUrl of a live bookmark. This should normally be the equal to the value found on the urlTag but sometimes only the feedUrl is given.

– **Returns** - Returns the feedUrlTag as String.

- *getKeywordTag*
private java.lang.String **getKeywordTag**()
 - **Usage**
 - * This method returns the tag to identify the keyword.
 - **Returns** - Returns the keywordTag.

- *getLastModificationDateTag*
private java.lang.String **getLastModificationDateTag**()
 - **Usage**
 - * This method returns the tag to identify the last modification date (timestamp) of a bookmark or bookmark folder.
 - **Returns** - Returns the lastModificationDateTag.

- *getLastVisitDateTag*
private java.lang.String **getLastVisitDateTag**()
 - **Usage**
 - * This method returns the tag to identify the last date (timestamp) a bookmark (the url) has been visited.
 - **Returns** - Returns the lastVisitDateTag.

- *getObjectCounter*
private int **getObjectCounter**()
 - **Usage**
 - * This method will return the current number of objects processed (bookmark or bookmark folder)
 - **Returns** - Returns the objectCounter

- *getOpenBookmarkFolderTag*
private java.lang.String **getOpenBookmarkFolderTag**()
 - **Usage**
 - * This method returns the tag to identify the opening of a bookmark folder.
 - **Returns** - Returns the openBookmarkFolderTag.

- *getOpenBookmarkTag*
private java.lang.String **getOpenBookmarkTag**()
 - **Usage**
 - * This method returns the tag to identify the opening of a bookmark.
 - **Returns** - Returns the openBookmarkTag as String.

- *getOpenListSectionTag*
private java.lang.String **getOpenListSectionTag**()
 - **Usage**
 - * This method returns the tag to identify the opening of a list section.
 - **Returns** - Return the opening tag of a list section as String.

- *getPersonalToolbarFolderTag*
private java.lang.String **getPersonalToolbarFolderTag**()

– **Usage**

- * This method will return the tag to identify if the processed bookmark folder should be shown as personalToolbarFolder.

Creation date: (30.01.2007 20:36:21)

– **Returns** - Returns the personalToolbarFolderTag.

• *getSideBarTag*

```
private java.lang.String getSideBarTag( )
```

– **Usage**

- * This method returns the tag to identify if the url should be added to the sidebar.

– **Returns** - Returns the sideBarTag.

• *getUrlTag*

```
private java.lang.String getUrlTag( )
```

– **Usage**

- * This method returns the tag to identify the url of a bookmark.

– **Returns** - Returns the urlTag as String.

• *indentByFolderLevel*

```
private java.lang.String indentByFolderLevel( int indentLevel, boolean isFolder )
```

– **Usage**

- * This method will create the indent for the specified level and type.

– **Parameters**

- * **indentLevel** - Hierarchie level to indent.
- * **isFolder** - Indicator if it's a folder (true) or a bookmark (false).

– **Returns** - Returns string containing the needed spaces.

• *setObjectCounter*

```
private void setObjectCounter( int objectCounter )
```

– **Usage**

- * This method allows to the the new objectCounter value which indicates the number of bookmark and bookmark folder data has been processed.

– **Parameters**

- * **objectCounter** - the objectCounter to set
-

• *writeBookmarkData*

```
private void writeBookmarkData( int currentFolderLevel,  
java.lang.StringBuffer stringBuffer,  
de.juergenulbts.jbookmarksync.util.BookmarkStorageContainer  
myClientBookmarkData )
```

– **Usage**

- * This method will write the bookmark data into stringBuffer (parameter).

– **Parameters**

- * **currentFolderLevel** -
 - * **stringBuffer** -
 - * **myClientBookmarkData** -
-

- *writeBookmarkFile*

```
public java.lang.String writeBookmarkFile( java.util.Collection  
bookmarkdataArray )
```

- *writeBookmarkFolderData*

```
private boolean writeBookmarkFolderData( int currentFolderLevel, int  
oldFolderLevel, boolean isFolderInCollection, java.lang.StringBuffer  
stringBuffer, de.juergenulbts.jbookmarksync.util.BookmarkStorageContainer  
myClientBookmarkData )
```

- **Usage**

- * This method will write the bookmark folder data into stringBuffer (parameter).

- **Parameters**

- * `currentFolderLevel` -
 - * `oldFolderLevel` -
 - * `isFolderInCollection` -
 - * `stringBuffer` -
 - * `myClientBookmarkData` -

- **Returns** - Returns true if there is a bookmark folder, else false.

3.1.11 CLASS OperaV2BookmarkParser

This class will process Opera bookmark files using the Opera v2 bookmark format used by several Opera versions (e.g. Opera 8.50, 9.00, 9.10).

DECLARATION

```
public class OperaV2BookmarkParser  
extends de.juergenulbts.jbookmarksync.util.BookmarkParserAbstract
```

FIELDS

- private static final char STX
 - STX character found at ASCII/UNICODE table position 2 (0x02).
- private int debugMode
 - Debug mode
- private Pattern linePattern
 - Pattern to look for (reading line by line).
- private Matcher matcher
 - Matcher which will return the results found by the given pattern.
- private final String bookmarkTag

- Tag to identify the bookmark.
- private final String bookmarkFolderTag
 - Tag to identify the bookmark folder.
- private final String idTag
 - Tag to identify the id of the bookmark/bookmark folder →currently ignored.
- private final String descriptionTag
 - Tag to identify the description.
- private final String descriptionLineBreakString
 - String used for linebreaks inside the description.
- private final String titleTag
 - Tag to identify the title.
- private final String urlTag
 - Tag to identify the url.
- private final String addDateTag
 - Tag to identify the date the bookmark or bookmark folder has been added.
- private final String lastVisitDateTag
 - Tag to identify the last date (timestamp) the bookmark (url) has been visited.
- private final String favIconTag
 - Tag to identify the favicon of the url →Opera uses an internal icon, so this is ignored!
- private final String characterSetTag
 - Tag to identify the used character set →currently unused.
- private final String keywordTag
 - Tag to identify the keywords.
- private final String personalBarTag
 - Tag to identify the personal bar setting.
- private final String personalBarPosTag
 - Tag to identify the personal bar position setting.
- private final String panelTag
 - Tag to identify if the url should be added to the Opera Panel.
- private final String panelPosTag
 - Tag to identify the sidebar position setting.
- private final String trashFolderTag
 - Tag to identify the trash folder that will be skipped when found.
- private boolean processingTrashFolder
 - This values indicates if the parser has already found the trash folder and is processing it's content or not. Entries inside the trash folder will be skipped as they are not usefull for the synchronization process!

- private long supportedBookmarkFolderFields
–
- private long supportedBookmarkFields
–

CONSTRUCTORS

- *OperaV2BookmarkParser*
public OperaV2BookmarkParser()
 - **Usage**
 - * The default constructor of this class.

METHODS

- *getAddDateTag*
private java.lang.String getAddDateTag()
 - **Usage**
 - * This method returns the tag to identify the date the bookmark or bookmark folder has been added.
 - **Returns** - Returns the addDateTag.

 - *getBookmark*
private de.juergenulbts.jbookmarksync.util.ComplexBookmark getBookmark(java.lang.String line)
 - **Usage**
 - * This method will parse the bookmark and gather all information which is finally stored inside a **ComplexBookmark** object. If there is a problem it returns null.
 - **Parameters**
 - * **line** - One line of the bookmark file which has to be parsed.
 - **Returns** - **ComplexBookmark**

 - *getBookmarkFolder*
private de.juergenulbts.jbookmarksync.util.ComplexBookmarkFolder getBookmarkFolder(java.lang.String line)
 - **Usage**
 - * This method will parse the bookmark folder and gather all information which is finally stored inside a **ComplexBookmarkFolder** object. If there is a problem it returns null.
 - **Parameters**
 - * **line** - One line of the bookmark file which has to be parsed.
 - **Returns** - **ComplexBookmarkFolder**
-

- *getBookmarkFolderTag*

```
private java.lang.String getBookmarkFolderTag( )
```

- **Usage**

- * This method returns the tag to identify a bookmark folder.

- **Returns** - Returns the bookmarkFolderTag.

- *getBookmarks*

```
public java.util.Collection getBookmarks( byte[] bookmarkFile )
```

- **Usage**

- * This method calls methods to create the bookmark array out of BookmarkStorageContainer objects which contain either a ComplexBookmark or a ComplexBookmarkFolder object. The object type of the stored object inside the BookmarkStorageContainer can be checked with a method.

- **Parameters**

- * bookmarkFile - The complete bookmark file as byte array.

- **Returns** - Collection containing BookmarkStorageContainer objects

- *getBookmarkTag*

```
private java.lang.String getBookmarkTag( )
```

- **Usage**

- * This method returns the tag to identify the bookmark.

- **Returns** - Returns the bookmarkTag.

- *getCharacterSetTag*

```
private java.lang.String getCharacterSetTag( )
```

- **Usage**

- * This method returns the character set used in the bookmark file. TODO: CharacterSet (encoding) of opera bookmark file is not used.

- **Returns** - Returns the characterSetTag.

- *getDescriptionLineBreakString*

```
private java.lang.String getDescriptionLineBreakString( )
```

- **Usage**

- * This method will return the string used for linebreaks inside the description.

- **Returns** - Returns the descriptionLineBreakString.

- *getDescriptionTag*

```
private java.lang.String getDescriptionTag( )
```

- **Usage**

- * This method returns the tag to identify a description used on a bookmark and bookmark folder.

- **Returns** - Returns the descriptionTag.

- *getFavIconTag*

```
private java.lang.String getFavIconTag( )
```

- **Usage**

- * This method returns the tag to identify the fav icon.

– **Returns** - Returns the favIconTag.

• *getIdTag*

private java.lang.String **getIdTag**()

– **Usage**

* This method returns the tag to identify the internal id of the entry used by this browser.

– **Returns** - Returns the idTag.

• *getKeywordTag*

private java.lang.String **getKeywordTag**()

– **Usage**

* This method returns the tag to identify the keyword.

– **Returns** - Returns the keywordTag.

• *getLastVisitDateTag*

private java.lang.String **getLastVisitDateTag**()

– **Usage**

* This method returns the tag to identify the last date (timestamp) a bookmark (the url) has been visited.

– **Returns** - Returns the lastVisitDateTag.

• *getLinePattern*

private java.util.regex.Pattern **getLinePattern**()

– **Usage**

* This method returns the stored pattern to look for (reading line by line).

– **Returns** - Returns the linePattern.

• *getMatcher*

private java.util.regex.Matcher **getMatcher**()

– **Usage**

* This method will return the matcher which uses a pattern to parse the bookmark file and return results found by the pattern.

– **Returns** - Returns the matcher.

• *getPanelPosTag*

private java.lang.String **getPanelPosTag**()

– **Usage**

* This method returns the tag to identify the sidebar position setting.

– **Returns** - Returns the panelPosTag.

• *getPanelTag*

private java.lang.String **getPanelTag**()

– **Usage**

* This method returns the tag to identify if the url should be added to the Opera Panel.

– **Returns** - Returns the panelTag.

- *getPersonalBarPosTag*
private java.lang.String **getPersonalBarPosTag**()
 - **Usage**
 - * This method returns the tag to identify the internal id of the browser.
 - **Returns** - Returns the personalBarPosTag.

- *getPersonalBarTag*
private java.lang.String **getPersonalBarTag**()
 - **Usage**
 - * This method returns the tag to identify the personal bar setting.
 - **Returns** - Returns the personalBarTag.

- *getSupportedBookmarkFields*
public long **getSupportedBookmarkFields**()
 - **Returns** - supportedBookmarkFields as long.
 - **See Also**
 - * de.juergenulbts.jbookmarksync.util.BookmarkParserAbstract.
getSupportedBookmarkFields (in 3.1.2, page 16)

- *getSupportedBookmarkFolderFields*
public long **getSupportedBookmarkFolderFields**()
 - **Returns** - supportedBookmarkFolderFields as long.
 - **See Also**
 - * de.juergenulbts.jbookmarksync.util.BookmarkParserAbstract.
getSupportedBookmarkFolderFields()

- *getTitleTag*
private java.lang.String **getTitleTag**()
 - **Usage**
 - * This method returns the tag to identify the title of a bookmark or bookmark folder.
 - **Returns** - Returns the titleTag.

- *getTrashFolderTag*
private java.lang.String **getTrashFolderTag**()
 - **Usage**
 - * This method will return the tag that identifies the trash folder that will be skipped including bookmarks and folders within.
 - **Returns** - Returns the trashFolderTag.

- *getUrlTag*
private java.lang.String **getUrlTag**()
 - **Usage**
 - * This method returns the tag to identify the url of a bookmark.
 - **Returns** - Returns the urlTag.

- *isProcessingTrashFolder*
private boolean **isProcessingTrashFolder**()

– **Usage**

* This method will return the status of the setting of the trash folder. If the trashFolder is processed is will return true otherwise false.

– **Returns** - Returns status if it's processing the trash folder or not.

• *setMatcher*

```
private void setMatcher( java.util.regex.Matcher matcher )
```

– **Usage**

* This method will set a matcher to be used to parse the bookmark file.

– **Parameters**

* **matcher** - The matcher to set.

• *setProcessingTrashFolder*

```
private void setProcessingTrashFolder( boolean processingTrashFolder )
```

– **Usage**

* This method allows to set the status for the trash folder. If a trash folder has been found and will be processed it should be set to true otherwise false.

– **Parameters**

* **processingTrashFolder** - The boolean value if the trash folder is currently processed or not.

METHODS INHERITED FROM CLASS

de.juergenulbts.jbookmarksync.util.BookmarkParserAbstract

(in 3.1.2, page 14)

• *getBookmarks*

```
public abstract java.util.Collection getBookmarks( byte[] bookmarkFile )
```

– **Usage**

* This method calls methods to create the bookmark array out of BookmarkStorageContainer objects which contain either a ComplexBookmark or a ComplexBookmarkFolder object. The object type of the stored object inside the BookmarkStorageContainer can checked with a method.

– **Parameters**

* **bookmarkFile** - The complete bookmark file as byte array.

– **Returns** - Collection containing BookmarkStorageContainer objects

• *getServerObjectIdEndTag*

```
private java.lang.String getServerObjectIdEndTag( )
```

– **Usage**

* The method will return the tag to identify the end of the object id information from the server that is again stored in a description.

– **Returns** - The serverObjectIdEndTag.

• *getServerObjectIdStartTag*

```
private java.lang.String getServerObjectIdStartTag( )
```

– **Usage**

- * The method will return the tag to identify the end of the object id information from the server that is again stored in a description.
 - **Returns** - The serverObjectIdStartTag.

- *getSupportedBookmarkFields*
public abstract long **getSupportedBookmarkFields**()
 - **Usage**
 - * This method will return the information which fields of a bookmark are supported.
 - **Returns** - supportedBookmarkFields as long.

- *getSupportedBookmarkFolderFields*
public abstract long **getSupportedBookmarkFolderFields**()
 - **Usage**
 - * This method will return the information which fields of a bookmark folder are supported.
 - **Returns** - supportedBookmarkFolderFields as long.

- *isServerObjectIdIncluded*
protected boolean **isServerObjectIdIncluded**(java.lang.String text)
 - **Usage**
 - * This method will check if the text includes the object id from the server which can be stored in description fields. It returns true if the object id could be found otherwise false.
 - **Parameters**
 - * **text** - The String that will be checked if it includes the server object id tag.
 - **Returns** - Returns status if the objectId start and end tags can be found in the submitted string.

- *retrieveServerObjectId*
protected int **retrieveServerObjectId**(java.lang.String text)
 - **Usage**
 - * This method will look for the object id of the server and return the integer value of it.
 - **Parameters**
 - * **text** - The String that will be checked if it includes the server object id.
 - **Returns** - Returns the server object Id as int value.

- *stripServerObjectIdFromDescription*
protected java.lang.String **stripServerObjectIdFromDescription**(java.lang.String text)
 - **Usage**
 - * This method will remove the object id of the server from the description.
 - **Parameters**
 - * **text** - The String that will be checked if it includes the server object id.
 - **Returns** - Returns the description string without the the server object id.

3.1.12 CLASS OperaV2BookmarkWriter

This class will process previously stored bookmark and bookmark folder data and create a bookmark file out of it that complies with the OperaV2 bookmark file format used by several Opera versions (e.g. Opera 8.50, 9.00, 9.10).

Creation date: (15.02.2007 22:07:09)

DECLARATION

```
public class OperaV2BookmarkWriter
extends java.lang.Object
implements de.juergenulbts.jbookmarksync.interfaces.IBookmarkWriter
```

FIELDS

- private static final String CRLF
 - Carriage Return + LineFeed - not platform dependent
- private static final String INDENT
 - Opera indent is always one TAB character.
- private static final String TRASH_FOLDER_NAME
 - The opera trash folder name (HARDCODED). TODO: Make trash folder name language independent!
- private static final char STX
 - STX character found at ASCII/UNICODE table position 2 (0x02).
- private final String closeBookmarkFolderTag
 - Tag to identify the closing of a bookmark folder.
- private final String openBookmarkTag
 - Tag to identify the opening of a bookmark.
- private final String openBookmarkFolderTag
 - Tag to identify the opening bookmark folder.
- private final String idTag
 - Tag to identify the id of the bookmark/bookmark folder
- private final String urlTag
 - Tag to identify the url.
- private final String titleTag
 - Tag to identify the title.
- private final String descriptionTag
 - Tag to identify the description.
- private final String descriptionLineBreakString
 - String used for linebreaks inside the description.
- private final String addDateTag
 - Tag to identify the date the bookmark or bookmark folder has been added.
- private final String lastVisitDateTag

- Tag to identify the last date (timestamp) the bookmark (url) has been visited.
- private final String favIconTag
 - Tag to identify the favicon of the url ->Opera uses an internal icon, so this is ignored!
- private final String characterSetTag
 - Tag to identify the used character set ->currently unused.
- private final String keywordTag
 - Tag to identify the keywords.
- private final String personalBarTag
 - Tag to identify the personal bar setting.
- private final String personalBarPosTag
 - Tag to identify the personal bar position setting.
- private final String panelTag
 - Tag to identify if the url should be added to the Opera Panel.
- private final String panelPosTag
 - Tag to identify the sidebar position setting.
- private final String trashFolderTag
 - Tag to identify the trash folder that will be skipped when found.
- private int objectCounter
 - Counter that shows how many bookmark and bookmark folders have been processed.

CONSTRUCTORS

- *OperaV2BookmarkWriter*
public **OperaV2BookmarkWriter**()

METHODS

- *constructHeader*
private java.lang.String **constructHeader**()
 - **Usage**
 - * This method constructs the basic header for the Opera bookmark file format.
 - **Returns** - Returns the default header of a Opera bookmark file as String.
- *constructTrashFolder*
private java.lang.String **constructTrashFolder**()
 - **Usage**
 - * This method constructs the trash folder for the Opera bookmark file format. It's the first bookmark folder that is created. TODO: Trash folder title has to be language independent. Currently it's hardcoded as class variable!

– **Returns** - Returns the trash folder of a Opera bookmark file as String.

• *getAddDateTag*

private java.lang.String **getAddDateTag**()

– **Usage**

* This method returns the tag to identify the date the bookmark or bookmark folder has been added.

– **Returns** - Returns the addDateTag.

• *getCharacterSetTag*

private java.lang.String **getCharacterSetTag**()

– **Usage**

* This method returns the character set used in the bookmark file.

– **Returns** - Returns the characterSetTag.

• *getCloseBookmarkFolderTag*

private java.lang.String **getCloseBookmarkFolderTag**()

– **Usage**

* This method returns the tag to identify the closing of a bookmark folder.

– **Returns** - Returns the closings tag of a bookmark folder as String.

• *getDescriptionLineBreakString*

private java.lang.String **getDescriptionLineBreakString**()

– **Usage**

* This method will return the string used for linebreaks inside the description.

– **Returns** - Returns the descriptionLineBreakString.

• *getDescriptionTag*

private java.lang.String **getDescriptionTag**()

– **Usage**

* This method returns the tag to identify a description used on a bookmark and bookmark folder.

– **Returns** - Returns the descriptionTag.

• *getIdTag*

private java.lang.String **getIdTag**()

– **Usage**

* This method returns the tag to identify the internal id of the entry used by this browser.

– **Returns** - Returns the idTag.

• *getKeywordTag*

private java.lang.String **getKeywordTag**()

– **Usage**

* This method returns the tag to identify the keyword.

– **Returns** - Returns the keywordTag.

- *getLastVisitDateTag*
private java.lang.String **getLastVisitDateTag**()
 - **Usage**
 - * This method returns the tag to identify the last date (timestamp) a bookmark (the url) has been visited.
 - **Returns** - Returns the lastVisitDateTag.

- *getObjectCounter*
private int **getObjectCounter**()
 - **Usage**
 - * This method will return the current number of objects processed (bookmark or bookmark folder)
 - **Returns** - Returns the objectCounter

- *getOpenBookmarkFolderTag*
private java.lang.String **getOpenBookmarkFolderTag**()
 - **Usage**
 - * This method returns the tag to identify the opening of a bookmark folder.
 - **Returns** - Returns the openBookmarkFolderTag.

- *getOpenBookmarkTag*
private java.lang.String **getOpenBookmarkTag**()
 - **Usage**
 - * This method returns the tag to identify the opening of a bookmark.
 - **Returns** - Returns the openBookmarkTag as String.

- *getPanelPosTag*
private java.lang.String **getPanelPosTag**()
 - **Usage**
 - * This method returns the tag to identify the sidebar position setting.
 - **Returns** - Returns the panelPosTag.

- *getPanelTag*
private java.lang.String **getPanelTag**()
 - **Usage**
 - * This method returns the tag to identify if the url should be added to the Opera Panel.
 - **Returns** - Returns the panelTag.

- *getPersonalBarPosTag*
private java.lang.String **getPersonalBarPosTag**()
 - **Usage**
 - * This method returns the tag to identify the internal id of the browser.
 - **Returns** - Returns the personalBarPosTag.

- *getPersonalBarTag*
private java.lang.String **getPersonalBarTag**()

- **Usage**
 - * This method returns the tag to identify the personal bar setting.
 - **Returns** - Returns the personalBarTag.
-

- *getTitleTag*

```
private java.lang.String getTitleTag( )
```

- **Usage**
 - * This method returns the tag to identify the title of a bookmark or bookmark folder.
 - **Returns** - Returns the titleTag.
-

- *getTrashFolderTag*

```
private java.lang.String getTrashFolderTag( )
```

- **Usage**
 - * This method will return the tag that identifies the trash folder that will be skipped including bookmarks and folders within.
 - **Returns** - Returns the trashFolderTag.
-

- *getUrlTag*

```
private java.lang.String getUrlTag( )
```

- **Usage**
 - * This method returns the tag to identify the url of a bookmark.
 - **Returns** - Returns the urlTag as String.
-

- *setObjectCounter*

```
private void setObjectCounter( int objectCounter )
```

- **Usage**
 - * This method allows to the the new objectCounter value which indicates the number of bookmark and bookmark folder data has been processed.
 - **Parameters**
 - * **objectCounter** - the objectCounter to set
-

- *writeBookmarkData*

```
private void writeBookmarkData( java.lang.StringBuffer stringBuffer,  
de.juergenulbts.jbookmarksync.util.BookmarkStorageContainer  
myClientBookmarkData )
```

- **Usage**
 - * This method will write the bookmark data into stringBuffer (parameter).
 - **Parameters**
 - * **stringBuffer** -
 - * **myClientBookmarkData** -
-

- *writeBookmarkFile*

```
public java.lang.String writeBookmarkFile( java.util.Collection  
bookmarkDataArray )
```

- *writeBookmarkFolderData*

```
private boolean writeBookmarkFolderData( int currentFolderLevel, int
oldFolderLevel, boolean isFolderInCollection, java.lang.StringBuffer
stringBuffer, de.juergenulbts.jbookmarksync.util.BookmarkStorageContainer
myClientBookmarkData )
```

- **Usage**

- * This method will write the bookmark folder data into stringBuffer (parameter).

- **Parameters**

- * `currentFolderLevel` -
 - * `oldFolderLevel` -
 - * `isFolderInCollection` -
 - * `stringBuffer` -
 - * `myClientBookmarkData` -

- **Returns** - Returns true if there is a bookmark folder, else false.

3.1.13 CLASS XBELBookmarkWriter

This class will process previously stored bookmark and bookmark folder data and create a bookmark file out of it that can be used with XML Bookmark Exchange Language (XBEL) supporting services and browsers. The output complies to the XML based XBEL bookmark file format.

Creation date: (09.02.2007 20:01:04)

DECLARATION

```
public class XBELBookmarkWriter
extends java.lang.Object
implements de.juergenulbts.jbookmarksync.interfaces.IBookmarkWriter
```

FIELDS

- private static final String CRLF
 - Carriage Return + LineFeed - not platform dependent
- private static final String INDENT
 - Indent always by two blanks
- private SimpleDateFormat sdf
 - SimpleDateFormat function to be used to generate the correct ISO8601 date String
- private final String openBookmarkTag
 - Tag to identify the opening of a bookmark.
- private final String closeBookmarkTag

- Tag to identify the closing of a bookmark.
- private final String urlTag
 - Tag to identify the url.
- private final String openBookmarkFolderTag
 - Tag to identify the opening bookmark folder.
- private final String closeBookmarkFolderTag
 - Tag to identify the opening bookmark folder.
- private final String descriptionLineBreakString
 - String used for linebreaks inside the description.
- private int objectCounter
 - Counter that shows how many bookmark and bookmark folders have been processed.

CONSTRUCTORS

- *XBELBookmarkWriter*
`public XBELBookmarkWriter()`
 - **Usage**
 - * Default constructor which will also store the default timezone (GMT) of the `SimpleDateFormatter` (`{ling java.text.SimpleDateFormat}`) which is needed for the XBEL output.

METHODS

- *constructHeader*
`private java.lang.String constructHeader()`
 - **Usage**
 - * This method constructs the basic header for the Netscape/Mozilla bookmark file format.
 - **Returns** - Returns the default header of a Netscape/Mozilla bookmark file as String.
- *getCloseBookmarkFolderTag*
`private java.lang.String getCloseBookmarkFolderTag()`
 - **Usage**
 - * This method returns the tag to identify the closing of a bookmark folder.
 - **Returns** - Returns the closeBookmarkFolderTag.
- *getCloseBookmarkTag*
`private java.lang.String getCloseBookmarkTag()`
 - **Usage**
 - * This method returns the tag to identify the closing of a bookmark.
 - **Returns** - Returns the closeBookmarkTag as String.

- *getDescriptionLineBreakString*

```
private java.lang.String getDescriptionLineBreakString( )
```

- **Usage**

- * This method will return the string used for linebreaks inside the description.

- **Returns** - Returns the descriptionLineBreakString.

- *getISO8601Date*

```
private java.lang.String getISO8601Date( java.util.Date dateTime )
```

- **Usage**

- * This method will return the specified date from the date object as date string using the ISO8601 format.

- **Parameters**

- * **dateTime** - The date as date object.

- **Returns** - Returns date as ISO8601 formatted String or null.

- *getISO8601Date*

```
private java.lang.String getISO8601Date( long timestamp )
```

- **Usage**

- * This method will return the specified timestamp as date string using the ISO8601 format.

- **Parameters**

- * **timestamp** - The date as timestamp.

- **Returns** - Returns date as ISO8601 formatted String or null.

- *getObjectCounter*

```
private int getObjectCounter( )
```

- **Usage**

- * This method will return the current number of objects processed (bookmark or bookmark folder)

- **Returns** - Returns the objectCounter

- *getOpenBookmarkFolderTag*

```
private java.lang.String getOpenBookmarkFolderTag( )
```

- **Usage**

- * This method returns the tag to identify the opening of a bookmark folder.

- **Returns** - Returns the openBookmarkFolderTag.

- *getOpenBookmarkTag*

```
private java.lang.String getOpenBookmarkTag( )
```

- **Usage**

- * This method returns the tag to identify the opening of a bookmark.

- **Returns** - Returns the openBookmarkTag as String.

- *getUrlTag*

```
private java.lang.String getUrlTag( )
```

- **Usage**

- * This method returns the tag to identify the url of a bookmark.
 - **Returns** - Returns the urlTag as String.

- *indentByFolderLevel*
private java.lang.String **indentByFolderLevel**(int **indentLevel**, boolean **isFolder**)
 - **Usage**
 - * This method will create the indent for the specified level and type.
 - **Parameters**
 - * **indentLevel** - Hierarchie level to indent.
 - * **isFolder** - Indicator if it's a folder (true) or a bookmark (false).
 - **Returns** - Returns string containing the needed spaces.

- *setObjectCounter*
private void **setObjectCounter**(int **objectCounter**)
 - **Usage**
 - * This method allows to the the new objectCounter value which indicates the number of bookmark and bookmark folder data has been processed.
 - **Parameters**
 - * **objectCounter** - the objectCounter to set

- *writeBookmarkData*
private void **writeBookmarkData**(int **currentFolderLevel**, java.lang.StringBuffer **stringBuffer**, de.juergenulbts.jbookmarksync.util.BookmarkStorageContainer **myClientBookmarkData**)
 - **Usage**
 - * This method will write the bookmark data into stringBuffer (parameter).
 - **Parameters**
 - * **currentFolderLevel** -
 - * **stringBuffer** -
 - * **myClientBookmarkData** -

- *writeBookmarkFile*
public java.lang.String **writeBookmarkFile**(java.util.Collection **bookmarkdataArray**)

- *writeBookmarkFolderData*
private boolean **writeBookmarkFolderData**(int **currentFolderLevel**, int **oldFolderLevel**, boolean **isFolderInCollection**, java.lang.StringBuffer **stringBuffer**, de.juergenulbts.jbookmarksync.util.BookmarkStorageContainer **myClientBookmarkData**)
 - **Usage**
 - * This method will write the bookmark folder data into stringBuffer (parameter).
 - **Parameters**
 - * **currentFolderLevel** -
 - * **oldFolderLevel** -
 - * **isFolderInCollection** -
 - * **stringBuffer** -

* myClientBookmarkData -

– **Returns** - Returns true if there is a bookmark folder, else false.

4 Package

de.juergenulbts.jbookmarksync.exception

Package Contents

Page

Classes

BookmarkDecompressionException	85
<i>The class <code>BookmarkDecompressionException</code> are a form of <code>Throwable</code> that indicates conditions that a reasonable application might want to catch.</i>	
BookmarkFieldNotSupportedException	87
<i>The class <code>BookmarkFieldNotSupportedException</code> are a form of <code>Throwable</code> that indicates conditions that a reasonable application might want to catch.</i>	
BookmarkFieldsNotSetException	89
<i>The class <code>BookmarkFieldsNotSetException</code> are a form of <code>Throwable</code> that indicates conditions that a reasonable application might want to catch.</i>	

4.1 Classes

4.1.1 CLASS BookmarkDecompressionException

The class `BookmarkDecompressionException` are a form of `Throwable` that indicates conditions that a reasonable application might want to catch.

This exception will be thrown if there are problems with unsupported compression formats or when the decompression does fail.

DECLARATION

```
public class BookmarkDecompressionException
extends java.lang.Exception
```

CONSTRUCTORS

- *BookmarkDecompressionException*

```
public BookmarkDecompressionException( )
```

- **Usage**

- * Constructs a new exception with null as it's detail message.

- *BookmarkDecompressionException*

```
public BookmarkDecompressionException( java.lang.String message )
```

- **Usage**

- * Constructs a new exception with the specified detail message.

- **Parameters**

- * **message** - The detail message. The detail message is saved for later retrieval by the `Throwable.getMessage()` method.

- *BookmarkDecompressionException*

```
public BookmarkDecompressionException( java.lang.String message,
java.lang.Throwable cause )
```

- **Usage**

- * Constructs a new exception with the specified detail message and cause.

- **Parameters**

- * **message** - The detail message. The detail message is saved for later retrieval by the `Throwable.getMessage()` method.

- * **cause** - The cause (which is saved for later retrieval by the `Throwable.getCause()` method). A null value is permitted, and indicates that the cause is nonexistent or unknown.

- *BookmarkDecompressionException*

```
public BookmarkDecompressionException( java.lang.Throwable cause )
```

– **Usage**

- * Constructs a new exception with the specified cause and a detail message of (cause==null ? null : cause.toString()) (which typically contains the class and detail message of cause).

– **Parameters**

- * **cause** - The cause (which is saved for later retrieval by the Throwable.getCause() method). A null value is permitted and indicates that the cause is nonexistent or unknown.

METHODS INHERITED FROM CLASS java.lang.Exception

METHODS INHERITED FROM CLASS java.lang.Throwable

- *fillInStackTrace*

```
public synchronized native java.lang.Throwable fillInStackTrace( )
```
- *getCause*

```
public java.lang.Throwable getCause( )
```
- *getLocalizedMessage*

```
public java.lang.String getLocalizedMessage( )
```
- *getMessage*

```
public java.lang.String getMessage( )
```
- *getOurStackTrace*

```
private synchronized java.lang.StackTraceElement[] getOurStackTrace( )
```
- *getStackTrace*

```
public java.lang.StackTraceElement[] getStackTrace( )
```
- *getStackTraceDepth*

```
private native int getStackTraceDepth( )
```
- *getStackTraceElement*

```
private native java.lang.StackTraceElement getStackTraceElement( int arg0 )
```
- *initCause*

```
public synchronized java.lang.Throwable initCause( java.lang.Throwable arg0 )
```
- *printStackTrace*

```
public void printStackTrace( )
```
- *printStackTrace*

```
public void printStackTrace( java.io.PrintStream arg0 )
```
- *printStackTrace*

```
public void printStackTrace( java.io.PrintWriter arg0 )
```

- *printStackTraceAsCause*
`private void printStackTraceAsCause(java.io.PrintStream arg0,
java.lang.StackTraceElement[] arg1)`
- *printStackTraceAsCause*
`private void printStackTraceAsCause(java.io.PrintWriter arg0,
java.lang.StackTraceElement[] arg1)`
- *setStackTrace*
`public void setStackTrace(java.lang.StackTraceElement[] arg0)`
- *toString*
`public java.lang.String toString()`
- *writeObject*
`private synchronized void writeObject(java.io.ObjectOutputStream arg0)`

4.1.2 CLASS BookmarkFieldNotSupportedException

The class `BookmarkFieldNotSupportedException` are a form of `Throwable` that indicates conditions that a reasonable application might want to catch.

The bookmark and bookmark folder comparator object `ComplexBookmarkAndFolderComparator` will check if certain fields are available/supported by the client and server stored bookmarks. It is used to compare the value stored on the client object with the server object.

Creation date: (28.05.2007 14:19:59)

DECLARATION

```
public class BookmarkFieldNotSupportedException  
extends java.lang.Exception
```

CONSTRUCTORS

- *BookmarkFieldNotSupportedException*
`public BookmarkFieldNotSupportedException()`
 - **Usage**
 - * Constructs a new exception with null as it's detail message.
- *BookmarkFieldNotSupportedException*
`public BookmarkFieldNotSupportedException(java.lang.String message)`
 - **Usage**
 - * Constructs a new exception with the specified detail message.
 - **Parameters**

* **message** - The detail message. The detail message is saved for later retrieval by the `Throwable.getMessage()` method.

• *BookmarkFieldNotSupportedException*

public BookmarkFieldNotSupportedException(java.lang.String message, java.lang.Throwable cause)

– **Usage**

* Constructs a new exception with the specified detail message and cause.

– **Parameters**

* **message** - The detail message. The detail message is saved for later retrieval by the `Throwable.getMessage()` method.

* **cause** - The cause (which is saved for later retrieval by the `Throwable.getCause()` method). A null value is permitted, and indicates that the cause is nonexistent or unknown.

• *BookmarkFieldNotSupportedException*

public BookmarkFieldNotSupportedException(java.lang.Throwable cause)

– **Usage**

* Constructs a new exception with the specified cause and a detail message of `(cause==null ? null : cause.toString())` (which typically contains the class and detail message of cause).

– **Parameters**

* **cause** - The cause (which is saved for later retrieval by the `Throwable.getCause()` method). A null value is permitted and indicates that the cause is nonexistent or unknown.

METHODS INHERITED FROM CLASS `java.lang.Exception`

METHODS INHERITED FROM CLASS `java.lang.Throwable`

• *fillInStackTrace*

public synchronized native java.lang.Throwable fillInStackTrace()

• *getCause*

public java.lang.Throwable getCause()

• *getLocalizedMessage*

public java.lang.String getLocalizedMessage()

• *getMessage*

public java.lang.String getMessage()

• *getOurStackTrace*

private synchronized java.lang.StackTraceElement[] getOurStackTrace()

• *getStackTrace*

public java.lang.StackTraceElement[] getStackTrace()

- *getStackTraceDepth*
private native int **getStackTraceDepth**()

- *getStackTraceElement*
private native java.lang.StackTraceElement **getStackTraceElement**(int **arg0**)

- *initCause*
public synchronized java.lang.Throwable **initCause**(java.lang.Throwable **arg0**)

- *printStackTrace*
public void **printStackTrace**()

- *printStackTrace*
public void **printStackTrace**(java.io.PrintStream **arg0**)

- *printStackTrace*
public void **printStackTrace**(java.io.PrintWriter **arg0**)

- *printStackTraceAsCause*
private void **printStackTraceAsCause**(java.io.PrintStream **arg0**,
java.lang.StackTraceElement[] **arg1**)

- *printStackTraceAsCause*
private void **printStackTraceAsCause**(java.io.PrintWriter **arg0**,
java.lang.StackTraceElement[] **arg1**)

- *setStackTrace*
public void **setStackTrace**(java.lang.StackTraceElement[] **arg0**)

- *toString*
public java.lang.String **toString**()

- *writeObject*
private synchronized void **writeObject**(java.io.ObjectOutputStream **arg0**)

4.1.3 CLASS BookmarkFieldsNotSetException

The class BookmarkFieldsNotSetException are a form of Throwable that indicates conditions that a reasonable application might want to catch.

The bookmark and bookmark folder comparator object ComplexBookmarkAndFolderComparator needs the information which fields are supported by the client and server stored bookmarks, so that only the intersection will be compared.

Creation date: (01.05.2007 10:19:59)

DECLARATION

```
public class BookmarkFieldsNotSetException
extends java.lang.Exception
```

CONSTRUCTORS

- *BookmarkFieldsNotSetException*

```
public BookmarkFieldsNotSetException( )
```

- **Usage**

- * Constructs a new exception with null as it's detail message.

- *BookmarkFieldsNotSetException*

```
public BookmarkFieldsNotSetException( java.lang.String message )
```

- **Usage**

- * Constructs a new exception with the specified detail message.

- **Parameters**

- * **message** - The detail message. The detail message is saved for later retrieval by the `Throwable.getMessage()` method.

- *BookmarkFieldsNotSetException*

```
public BookmarkFieldsNotSetException( java.lang.String message,  
java.lang.Throwable cause )
```

- **Usage**

- * Constructs a new exception with the specified detail message and cause.

- **Parameters**

- * **message** - The detail message. The detail message is saved for later retrieval by the `Throwable.getMessage()` method.

- * **cause** - The cause (which is saved for later retrieval by the `Throwable.getCause()` method). A null value is permitted, and indicates that the cause is nonexistent or unknown.

- *BookmarkFieldsNotSetException*

```
public BookmarkFieldsNotSetException( java.lang.Throwable cause )
```

- **Usage**

- * Constructs a new exception with the specified cause and a detail message of `(cause==null ? null : cause.toString())` (which typically contains the class and detail message of cause).

- **Parameters**

- * **cause** - The cause (which is saved for later retrieval by the `Throwable.getCause()` method). A null value is permitted and indicates that the cause is nonexistent or unknown.

METHODS INHERITED FROM CLASS `java.lang.Exception`

METHODS INHERITED FROM CLASS java.lang.Throwable

- *fillInStackTrace*
public synchronized native java.lang.Throwable fillInStackTrace()
- *getCause*
public java.lang.Throwable getCause()
- *getLocalizedMessage*
public java.lang.String getLocalizedMessage()
- *getMessage*
public java.lang.String getMessage()
- *getOurStackTrace*
private synchronized java.lang.StackTraceElement[] getOurStackTrace()
- *getStackTrace*
public java.lang.StackTraceElement[] getStackTrace()
- *getStackTraceDepth*
private native int getStackTraceDepth()
- *getStackTraceElement*
private native java.lang.StackTraceElement getStackTraceElement(int arg0)
- *initCause*
public synchronized java.lang.Throwable initCause(java.lang.Throwable arg0)
- *printStackTrace*
public void printStackTrace()
- *printStackTrace*
public void printStackTrace(java.io.PrintStream arg0)
- *printStackTrace*
public void printStackTrace(java.io.PrintWriter arg0)
- *printStackTraceAsCause*
private void printStackTraceAsCause(java.io.PrintStream arg0,
java.lang.StackTraceElement[] arg1)
- *printStackTraceAsCause*
private void printStackTraceAsCause(java.io.PrintWriter arg0,
java.lang.StackTraceElement[] arg1)
- *setStackTrace*
public void setStackTrace(java.lang.StackTraceElement[] arg0)
- *toString*
public java.lang.String toString()
- *writeObject*
private synchronized void writeObject(java.io.ObjectOutputStream arg0)