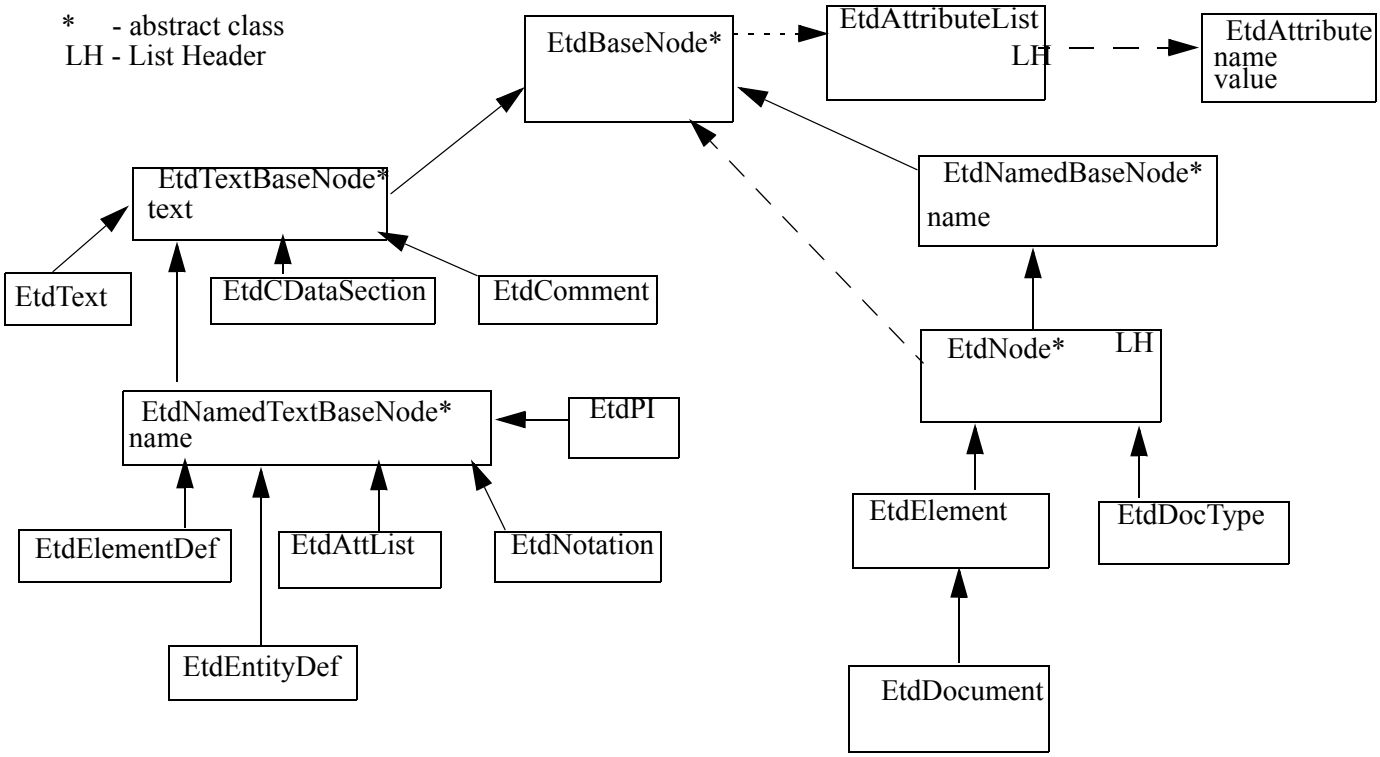
Tagged Data Reference

Tagged Document

The standard library comes with a set of classes that provide a tagged document functionality. A tagged document is similar to an Xml document in structure. The EtdDocument class represents the entire document. An object created from this class has a list which can contain zero ir more objects derived from EtdBaseNode. These are usually EtdElement objects, but they can also contain EtdDocType objects as well. The EtdElement objects can also (optionally) contain a set of attributes. The EtdElement object has a list of (usually) other EtdElement objects. It can also contain EtdText objects as well as EtdPI and EtdComment objects. The EtdElementDef, EtdAttList, EtdNotation and EtdEntityDef are rarely used and they are provided here for a sense of completeness with the Xml model.

However much this resembles Xml, it is not Xml itself. This classes do not provide any validation. In fact, it does not stop you from putting any node into any list. The EtdDocument is used to store complex data structures. Once built the EtdDocument can be saved to a disk file (Text output stream) in a variety of ways, using the ETaggedOutputStream derived classes. It can also be built using the ETaggedInputStream set of classes to read text files (Text Input Streams) formatted in a variety of ways. For example, you could use an EtdDocument to read in an Xml file (using the EXMLTaggedInputStream), make changes, then save it back to a disk file. You also save it in another format (such as INI or JSON or Html or xhtml etc.). Of course, some of these formats are less complex than the Xml format, so data might be lost.

The following diagram illustrates the relationship between the various classes. The solid arrows point to the base class of the specified class. Dotted lines indicate it has an object of that class as a member and dashed lines indicate that the specified class has a list of objects. For example, EtdBaseNode has an instance of EtdAttributeList as a member and the EtdAttrbuteList contains a list (zero or more) of EtdAttribute objects.



These classes are not built-in, so they have be imported. They are all in one imported class file as follows:

```
import('tgio\EtdDocument');
```

The classes are in the `tgio` namespace, so the `tgio` name must be used as a prefix when reference a class name outside the namespace.

class tgio.EtdAttribute

The EtdAttribute class represents an attribute name and value pair. These are kept in a list in those nodes that are allowed attributes.

Instance Methods

EtdAttribute nextAttribute;

This method returns the next attribute in the list, if any. It returns **`null`** if there is no next attribute.

EtdAttribute prevAttribute;

This method returns the previous attribute in the list, if any. It returns **`null`** if there is no previous attribute.

String attrName;

This method returns the attribute name.

String attrValue;

This method returns the attribute value.

String toString;

This method returns a string representation of the attribute.

class tgio.EtdBaseNode

The `EtdBaseNode` class represents the base class of all classes in the EtdDocument system. All the other EtdDocument classes are derived either directly or indirectly from this class. It contains some of properties and methods that are common to all the classes. Most of these deal with attributes and navigation within a node list (EtdNode), if applicable. It is an abstract class, so it cannot be created (instantiated) by itself, only as a base class of other classes.

Instance Methods

EtdBaseNode getNextNode;

This method returns the next node in in the EtdNode list, if any. It returns **`null`** if there is no next node.

EtdBaseNode getPrevNode;

This method returns the previous node in in the EtdNode list, if any. It returns **`null`** if there is no previous node.

EtdBaseNode getParentNode;

This method returns the parent node in in the EtdNode list, if any. It returns **`null`** if there is no parent node.

Boolean hasChildNodes;

This method returns **`true`** if the node has children and **`false`** otherwise.

Boolean hasAttributes;

This method returns **`true`** if the node has attributes and **`false`** otherwise.

Integer attrCount;

This method returns the number of attributes the node contains.

EtdAttribute getFirstAttribute;

This method returns the first attribute in the node.

EtdAttribute setAttribute(pAttrName, pAttrValue);

This method creates an attribute (specified by pAttrName) for this node to the specified value (pAttrValue). If the attribute specified by pAttrName already exists, it sets a new value.

EtdAttribute findAttributeByName(pAttrName);

This method finds the attribute (EtdAttribute) specified by the pAttrName parameter and returns the attribute. If it does not exist, it returns **null**.

String findAttributeValueByName(pAttrName);

This method finds the attribute specified by the pAttrName parameter and returns the **value** of that attribute.. If the attribute does not exist, it returns **null**.

Boolean hasAttributeNameValuePair(pAttrName, pAttrValue);

This method returns **true** if the node contains a matching attribute name and value, otherwise it returns **false**.

EtdAttribute removeAttribute(pAttrName);

This method removes the attribute (EtdAttribute) specified by the pAttrName parameter and returns the attribute. If it does not exist, it returns **null**.

null removeAllAttributes;

This method removes all of the attributes from the node.

null loadFromTaggedStream(pTaggedInputStream);

This method loads the data for this node (the derived node) from the specified tagged input stream.

This method is usually called from an EtdDocument object to load an entire document from a tagged input stream, which will create all the child objects and call the loadFromTaggedStream method recursively for each child created. If the tagged input stream is well formed, then the EtdDocument will make sure that the tagged input stream is positioned correctly, before it calls the loadFromTaggedStream for an child object.

Note: The loadFromTaggedStream can be used independent of an EtdDocument, but the tagged input stream must be positioned to the correct place for it to work properly.

null saveToTaggedStream(pTaggedOutputStream);

This method saves the data for this node (the derived node) into the specified tagged output stream. It is assumed that the tagged output stream has been positioned at the correct place.

This method is usually called from an EtdDocument object to save an entire document to a tagged output stream, which will call the saveToTaggedStream method recursively for each child object in the EtdDocument.

Note: An indiviual node can be saved to a tagged output stream independent of an EtdDocument object, as long as the state of the tagged output stream is expecting it.

String toString;

This method returns a string representation of the object.

class tgio.EtdTextBaseNode<tgio.EtdBaseNode

The `EtdTextBaseNode` class is derived from the EtdBaseNode and represents a base class of all classes in the EtdDocument system that contains unformatted text (such as EtdText and EtdComment). It is an abstract class, so it cannot be created (instantiated) by itself, only as a base class of other derived classes.

Instance Methods

String text[(pTextString)];

This method gets and optionally sets the text value of this node.

`null` appendText[(pTextString)];

This method appends the specifed text to the text value of this node.

class tgio.EtdText<tgio.EtdTextBaseNode

The `EtdText` class is derived from the EtdTextBaseNode and represents a text block. This is typically an object that is a child of an EtdElement node.

class Methods

`tgio.EtdText.`make[(pTextString)];

This creates an instance of a `EtdText` object and optionally adds the string value.

class tgio.EtdCDataSection<tgio.EtdTextBaseNode

The EtdCDataSection class is derived from the EtdTextBaseNode and represents a text block similar to the Xml CData Section. This is typically an object that is a child of an EtdElement node.

class Methods

`tgio.EtdCDataSection.`make[(pTextString)];

This creates an instance of a `EtdCDataSection` object and optionally adds the string value.

class tgio.EtdComment<tgio.EtdTextBaseNode

The EtdComment class is derived from the EtdTextBaseNode and represents a text block similar to the Xml comment. This is typically an object that is a child of an EtdElement node.

class Methods

`tgio.EtdComment.`make[(pTextString)];

This creates an instance of a EtdComment object and optionally adds the string value.

class tgio.EtdNamedTextBaseNode<tgio.EtdTextBaseNode

The `EtdNamedTextBaseNode` class is derived from the EtdTextBaseNode and represents a base class of all classes in the EtdDocument system that contains unformatted text and also a name (such as EtdPI and EtdNotation). It is an abstract class, so it cannot be created (instantiated) by itself, only as a base class of other derived classes.

Instance Methods

String name(pName);

This method gets and optionally sets the name value of this node.

Boolean isNamed(pName);

This method returns **`true`** if the node's name is equal to the pName parameter, other is returns **`false`**.

Boolean isNamedI(pName);

This method returns **`true`** if the node's name is equal to the pName parameter, other is returns **`false`**. The comparison is case insensitive.

class tgio.EtdElementDef<tgio.EtdNamedTextBaseNode

The EtdElementDef class is derived from the EtdNamedTextBaseNode and represents a named text block similar to the Xml ElementDef. The name is the name of the element and the text is the definition. This is typically an object that is a child of an EtdDocType node.

class Methods

`tgio.EtdElementDef.`make[(pName,pText)];

This creates an instance of a `EtdElementDef` object and optionally adds the name and string value.

class tgio.EtdAttList<tgio.EtdNamedTextBaseNode

The EtdAttList class is derived from the EtdNamedTextBaseNode and represents a named text block similar to the Xml ATTLIST. The name is the name of the attribute and the text is the definition. This is typically an object that is a child of an EtdDocType node.

class Methods

`tgio.EtdAttList.`make[(pName,pText)];

This creates an instance of a EtdAttList object and optionally adds the name and string value.

class tgio.EtdNotation<tgio.EtdNamedTextBaseNode

The EtdNotation class is derived from the EtdNamedTextBaseNode and represents a named text block similar to the Xml Notation. The name is the name of the notation and the text is the data. This is typically an object that is a child of an EtdDocType node.

class Methods

`tgio.EtdNotation.`make[(pName,pText)];

This creates an instance of a EtdNotation object and optionally adds the name and string value.

class tgio.EtdEntityDef<tgio.EtdNamedTextBaseNode

The EtdEntityDef class is derived from the EtdNamedTextBaseNode and represents a named text block similar to the Xml Entity. The name is the name of the entity and the text is the data. This is typically an object that is a child of an EtdDocType node.

class Methods

`tgio.EtdEntityDef.`make[(pName,pText[,pParmEntity)];

This creates an instance of a EtdEntityDef object and optionally adds the name and string value and a flag indicating that it is a parameter entity.

class tgio.EtdPI<tgio.EtdNamedTextBaseNode

The EtdPI class is derived from the EtdNamedTextBaseNode and represents a named text block similar to the Xml Processing instruction. The name is the name of the PI and the text is the data. This is typically an object that can be a child of an EtdDocType or EtdElement node.

class Methods

`tgio.EtdPI.`make[(pName,pText)];

This creates an instance of a EtdPI object and optionally adds the name and string value.

class tgio.EtdNamedBaseNode<tgio.EtdBaseNode

The `EtdNamedBaseNode` class is derived from the EtdBaseNode and represents a base class of all classes in the EtdDocument system that contains a name but no text (such as EtdElement). It is an abstract class, so it cannot be created (instantiated) by itself, only as a base class of other derived classes.

Instance Methods

String name(pName);

This method gets and optionally sets the name value of this node.

Boolean isNamed(pName);

This method returns **`true`** if the node's name is equal to the pName parameter, other is returns **`false`**.

Boolean isNamedI(pName);

This method returns **`true`** if the node's name is equal to the pName parameter, other is returns **`false`**. The comparison is case insensitive.

class tgio.EtdNode<tgio.EtdNamedBaseNode

The EtdNode class is derived from the EtdNamedBaseNode and represents a base class of all classes in the EtdDocument system that contains a name and has the ability to have child nodes (such as EtdElement). It is an abstract class, so it cannot be created (instantiated) by itself, only as a base class of other derived classes.

Instance Methods

null addChildNode(pNode);

This method adds the specified node to the end of the node list for this node.

null insertAtHead(pNode);

This method inserts the specified node to the head of the node list for this node.

null insertBefore(pNode,pBeforeNode);

This method inserts the specified node before the pBeforeNode for this node.

null insertAfter(pNode,pAfterNode);

This method inserts the specified node after the pAfterNode for this node.

EtdBaseNode getFirstChild;

This method returns the first node in in the EtdNode list, if any. It returns **null** if there is no first node.

EtdBaseNode getLastChild;

This method returns the last node in in the EtdNode list, if any. It returns **null** if there is no last node.

Integer getChildCount;

This method returns the number of child nodes this node contains.

Boolean isEmpty;

This method returns **true** if the node has no children, otherwise it returns **false**.

String getAllText;

This method returns a string value of all the text items (EtdText objects) that are children of this node.

EtdElementNode findChildElement(pElementNameExpression);

This method locates an element (EtdElement) node based on the pElementNameExpression and the optional pNumber. The pElementNameExpression is a list of one or more element names separated by the '/' character. Each occurrence of the '/' indicates a level in the hierarchy. If the first character is the '/' character, then the search starts at the beginning of the document. It returns the found EtdElement node or **null** if not found.

The following code loads a document, then searches for the first 'Level2' element under the first 'Level1' element.

```
var vCfgDoc=tgio.EtdDocument.make;
var vRetCode=vCfgDoc.loadFromFile('c:\MyTestFolder\MyTestFile.xml','xml');
if vRetCode==0
   var vElement=findChildElement('Level1\Level2);
end;
```

EtdElementNode makeElementNode(pNodeName);

This method creates a new EtdElement node and adds it to the end of the node list. It returns the new EtdElement node.

EtdDocType makeDocTypeNode(pNodeName);

This method creates a new EtdDocType node and adds it to the end of the node list. It returns the new EtdDocType node.

EtdText makeTextNode(pNodeName);

This method creates a new EtdText node and adds it to the end of the node list. It returns the new EtdText node.

EtdPI makePINode(pNodeName);

This method creates a new EtdPI node and adds it to the end of the node list. It returns the EtdPI node.

EtdComment makeCommentNode;

This method creates a new EtdComment node and adds it to the end of the node list. It returns the EtdComment node.

EtdNotation makeNotationNode(pNodeName);

This method creates a new EtdNotation node and adds it to the end of the node list. It returns the EtdNotation node.

EtdCDataSection makeCDataSectionNode;

This method creates a new EtdCDataSection node and adds it to the end of the node list. It returns the EtdCDataSection node.

EtdElementDef makeElementDefNode(pNodeName);

This method creates a new EtdElementDef node and adds it to the end of the node list. It returns the EtdElementDef node.

EtdAttList makeAttListNode(pNodeName);

This method creates a new EtdAttList node and adds it to the end of the node list. It returns the EtdAttList node.

EtdEntityDef makeEntityNode(pNodeName[,pText][,pParmEntity]);

This method creates a new EtdEntityDef node and adds it to the end of the node list. It returns the EtdEntityDef node. Setting the pParmEntity to **`true`** means that this is a parm entity.

EtdBaseNode removeFirstChild;

This method removes the first child node (EtdBaseNode) and returns the node. If there are no nodes, it returns **`null`**.

EtdBaseNode removeChild(pChildNode);

This method removes the node (EtdBaseNode) specified by the pChildNode parameter and returns the node. If the node is not present, it returns **null**.

null removeAllChildNodes;

This method removes al the child nodes.

class tgio.EtdElement<tgio.EtdNode

The EtdElement class is derived from the EtdNode and represents a named node similar to the Xml Element. The name is the name of the element. This is typically an object that is a child of an EtdElement or an EtdDocument node.

class Methods

`tgio.EtdElement.`make(pName);

This creates an instance of a EtdElement object with the specified name.

class tgio.EtdDocType<tgio.EtdNode

The EtdDocType class is derived from the EtdNode and represents a named node similar to the Xml DocType. It also can contain some unformatted text. The name is the name of the doctype. This is typically an object that is a child of an EtdDocument node.

Instance Methods

String text[(pTextString)];

This method gets and optionally sets the text value of this node.

null appendText[(pTextString)];

This method appends the specifed text to the text value of this node.

class Methods

`tgio.EtdDocType.`make(pName);

This creates an instance of a EtdDocType object with the specified name.

class tgio.EtdDocument<tgio.EtdElement

The EtdDocument class is derived from the EtdElement and represents an entire tagged document. The name is the name of the highest element. The EtdDocument is the highest parent of all the child nodes.

Instance Methods

Integer loadFromFile(pFileName,pType,pEncoding='utf8');

This method loads the EtdDocument object from the specified file. This is a convenience method in that it will create a tagged input stream, call the loadFromTaggedStream method of the EtdDocument object, then close the stream. The same thing can be accomplished by creating the tagged input stream yourself. The pType indicates the type of tagged input stream to create. The pEncoding indicates the source file encoding.

A return value of zero, indicates a successful operation. Otherwise, it returns an error code.

Integer saveToFile(pFileName,pType='xml',pEncoding='utf8'[,pLFAfter=false]);

This method saves the EtdDocument object to the specified file. This is a convenience method in that it will create a tagged output stream, call the saveToTaggedStream method of the EtdDocument object, then close the stream. The same thing can be accomplished by creating the tagged output stream yourself. The pType indicates the type of tagged output stream to create. The pEncoding indicates the output source file encoding. If the pLFAfter is set to true, then a line feed is written after each closed element.

A return value of zero, indicates a successful operation. Otherwise, it returns an error code.

class Methods

`tgio.EtdDocument.`make(pName);

This creates an instance of a EtdDocument object with the specified highest element name.

Tagged Streams

In addition to the standard streams (byte and text) there can be specialized stream types. Tagged streams are a companion set of classes for the tagged document classes. They can be used in conjunction with the EtdDocument classes or they can be used independently of them.

Just as the EtdDocument objects mimic the structure of an xml document, tagged streams work in a similar fashion as Xml/Sax. Unlike Xml/Sax however, Tagged streams can be used as input or output. Also, the source and/or destination does not have to be an Xml file. It can be a file (or text stream) of almost any type.

A tagged input stream returns a sequence of type indicators (Integer) along with some optional extra data. Some items have extra data and some do not. See “List of TaggedInputStream tag types” on page 15. The `getNextTaggedToken` method of the tagged input stream fetches the next tagged item and returns the type of the

item. The `currTag` method returns the optional data, if any. For example, a tagged input stream might return a type of `tgio.ETaggedInputStream.TGTnewElement` when it encounters the following in an xml file:

```
<eltname
```

The `currTag` method would return the 'eltname' as a string value. If the tagged input stream were an ini it would return the same code and tag when the following was encountered:

```
[eltname]
```

Other types of tagged input streams might return the same values when it parses other types of text (e.g as Css file or a Json file).

A tagged output stream contains a set of methods that output the various types of data corresponding Xml-like items (e.g. `putStartElement` and `putEndElement`). The `putStartElement` of an Xml type tagged output stream would put out something like the following:

```
<eltname
```

However, for another type of tagged output stream such as an ini file, it might something like the following:

```
[eltname]
```

In other words, tagged output streams can create files of various types as long as they write out their own type of markup in response to the standard Xml type output methods.

class ETaggedInputStream

The `ETaggedInputStream` class represents the base class of all tagged input streams. It is a class that provides the basic functionality for a stream of tagged types and tagged values that correspond to the basic tagged architecture. It requires a ETextInputStream object (or one with the same methods) to provide a stream of text characters, which it parses to create the tagged items. This base class provides the tag type codes and a few utility methods. Derived classes usually supply the ETextInputStream required and all the parsing code to parse the incoming text into the currTagType and the currTag value. The tagType values are defined in the class properties, see the List of TaggedInputStream tag types on page 15.

There are also some class methods

This class is not built-in, so it has be imported, as follows:

```
import('tgio\ETaggedInputStream');
```

Instance Methods

`Integer` getNextTaggedToken;

This method retrieves and returns the next tag type and stores the curr tag value, so it can be retrieved by the currTag method. This method should be overridden in derived classes. The one in the base class only returns the end of stream tag type.

A typical program would get each tag and type and then process them in turn. Here is an example

```
var vDone=false;
var vStream=tgio.ETaggedInputStream.makeTaggedInputStream(
      'c:\MyFolder\MyFile.xml','xml');
while !vDone
   var vTagType=vStream.getNextTaggedToken;
   var vCurrTag=vStream.currTag;
   if vTagType==tgio.ETaggedInputStream.TGTeos
      vDone=true;
   else
      var vTagName=tgio.ETaggedInputStream.getTokenName(vTagType);
      writeln('Tag code='+vTagType.toS+'   Tag Name='+vTagName);
      // DO SOMETHING HERE
   end;
 end
```

This code loops through each tagged item and writes the code and name to the output console. In a real program, you would insert your own code after the comment line to process each type (or each type that you want to process).

`String` currTag;

This method returns the current tag value.

`Integer` currTagType;

This method returns the current tag type.

`ETextInputStream` inputStream[(pInputStream)];

This method gets and optionally sets the text input stream object. This is usually set by a derived method.

`null` putBackToken(String pTag, Integer pTagType);

This method returns a token and tag type combination to the tagged input stream. Theses values are stacked and will be returned the next tiem the getNextTaggedToken method is called. When reading from a stream sometimes it is convenient to know what the next token type will be. Since you can only do this by getting it, you might want to return it, if it is not what you want at the moment.

Class Properties

The following values indicate the input tagged type event id, a description of the event and the data returned in the currTag method

Table 1: List of TaggedInputStream tag types

currTagType	Description	currTag
TGTbeginDocType	Beginning of a doc type	Doc Type Name
TGTdocType	A doc type entry	Doc type text
TGTendDocType	End of a doc type	<none>
TGTbeginComment	Beginning of a comment	<none>
TGTcomment	A comment	Comment Text
TGTendComment	End of a comment	<none>
TGTbeginNotation	Beginning of a notation	<none>
TGTnotation	A notation	Notation information
TGTendNotation	End of a notation	<none>
TGTbeginEntity	Beginning of an entity	Entity name
TGTbeginParmEntity	Beginning of an Parm entity	Entity name
TGTentity	An Entity	Entity Value
TGTendEntity	End of an entity	<none>
TGTbeginElementDef	Beginning of a comment	Element Def Name
TGTelementDef	Element Def	Element Def info
TGTendElementDef	End of a comment	<none>
TGTbeginAttList	Beginning of a attribute def list	AttList name
TGTattList	Attribute Def List	AttList info
TGTendAttList	End of a attribute def list	<none>
TGTbeginPI	Beginning of a PI	PI name
TGTPI	A Processing instruction	PI Text
TGTendPI	End of a PI	<none>
TGTtext	A Text string	Text
TGTnewElement	Start of an element	Element Name
TGTattrName	An Attribute Name	Attribute name
TGTattrValue	An Attribute Value	Attribute Value
TGTendElement	End of an Element	Element Name(if specified)
TGTbeginCDataSection	Beginning of a CData section	<none>
TGTCDataSection	A CData section	CData Section info
TGTendCDataSection	End of a CData section	<none>
TGTeos	End Of Tagged Stream	<none>

Class Methods

The following are class methods. They can be called without creating an object.

`ETaggedInputStream` makeTaggedInputStream(pInput,pType,pEncoding);

This method creates a tagged input stream based on the information provided. This is a convenience method to make it easier to create a standard tagged input stream. You can do these steps manually by creating a text input stream and then the appropriate tagged input stream, then putting them together, but this does it all in one method. If you have a non-standard tagged input stream, then you will need to do these steps separately.

The pInput parameter can be a file name, a byte input stream or a text input stream. The pType parameter is a string value that indicates the type of tagged input stream that you wish to create. The possible values are 'xml', 'html', 'xhtml' or 'ini'. The 'xml', 'html' and 'xhtml' types create an EXMLTaggedInputStream. The 'ini' type creates an INITaggedInputStream. The pEncoding parameter indicates the type of encoding to be used with the text input stream.

If the pInput parameter is a string value, then it is assumed to be a file name. A ETextInputFile will be created using this file name and the specified encoding. If the pInput is a byte input stream, then a ETextInputStream will be created using this byte input stream.

Once we have an ETextInputStream, an ETaggedInputStream object will be created (based on the pType parameter). The input text stream will be set and the new tagged input stream will be returned.

`String` getTokenName(pTagType);

This method returns a string name for the specifed integer tag type value. This can be used as a debugging tool for displaying the name of the tag type.

class EXMLTaggedInputStream<ETaggedInputStream

The `EXMLTaggedInputStream` class is derived from the base class and has all its data and methods. This class parses input files that follow the basic Xml/Sgml/Html/XHtml tagging scheme.

This class is not built-in, so it has be imported, as follows:


```
import('tgio\ETaggedInputStream');
```

Instance Methods

`Integer` getNextTaggedToken;

This method retrieves and returns the next tag type and stores the curr tag value, so it can be retrieved by the currTag method.

class Methods

`tgio.EXMLTaggedInputStream.`make[(pTextInputStream)];

This creates an instance of a `EXMLTaggedInputStream` object with the specified text input stream.

class INITaggedInputStream<ETaggedInputStream

The `INITaggedInputStream` class is derived from the base class and has all its data and methods.This class parses input files that follow the basic Microsoft Windows ini tagging scheme.

The ini file structure is simple. An ini file looks something like the following:

```
[SectionName]
KeyName1=KeyValue1
KeyName2=Keyvalue2
...
KeyNameN=KeyValueN

[SectionName2]
...
```

When receiving data from an IniInputStream there are two ways to make this format appear to mimic Xml style input. The default way is to treat each section name as an element name (and therefore use the TGTnewElement and TGTendElement messages for each) and to treat each set of KeyName/KeyValue pairs as attribute name and value pairs (TGTattrName and TGTattrValue messages).

There is also an alternative way to mimic the Xml style. In this method, the section names are still sent using TGTnewElement and TGTendElement messages. However, the KeyName/KeyValue pairs are also sent as an element. The KeyName is returned as the TGTnewElement/TGTendElement combination and the KeyValue is returned between them as a TGTtext message.

The advange of using the default method is that if you load these into an EtdDocument, you can save it back to an ini file (using the INIOutputStream). If you use the second method and you try to save it back to an ini file data will be lost because it will not be able to handle the nested element. Of course, you could always write your own save

method that reads the EtdDocument objects and sends an attribute/value pair instead of an element for the second level nested element and the text value.

This class is not built-in, so it has be imported, as follows:

```
import('tgio\ETaggedInputStream');
```

Instance Methods

`Integer` getNextTaggedToken;

This method retrieves and returns the next tag type and stores the curr tag value, so it can be retrieved by the currTag method.

class Methods

`tgio.INITaggedInputStream.`make[(pTextInputStream)];

This creates an instance of a `INITaggedInputStream` object with the specified text input stream.

class ECSSTaggedInputStream<ETaggedInputStream

The `CSSTaggedInputStream` class is derived from the base class and has all its data and methods.This class parses css input files that follow the standard Css file structure.

A css file has the following items:

The main type of structure in a css file is the selector/property list structure as follows:

```
Selector[,Selector2]...[,SelectorN]  {
       PropertyName : PropertyValue;
       ...
       }
```


This generates the following set of tagged codes and data.

```
TGTnewElement       stylerule
TGTnewElement       selector
TGTtext             SelectorData1
TGTendElement       selector
TGTnewElement       propertylist
TGTnewElement       property
TGTattrName         name
TGTattrValue        PropertyName
TGTtext             PropertyValue
TGTendElement       property
TGTendElement       propertylist
TGTendElement       stylerule
```

And creates the following tagged document structure

```
<stylerule>
      <selector>Selector1</selector>
      [<selector>SelectorN</selector>]
      <propertylist>
         <property name="PropertyName">
            PropertyValue or <propertylist> ... </propertylist>
         </property>
         <property>
            <propname name="PropertyName"/>
            <propname name="PropertyName2"/>
            PropertyValue or <propertylist> ... </propertylist>
         </property>
         . . .
      </propertylist>
</stylerule>
```

There are also set of @ rules (AtRules) that handle special processing.

```
@import url [media queries];
```

This rule imports the specified file for the specified media types. It generates the following tagged data.

```
TGTnewElement       atrule
TGTattrName         type
TGTattrValue        import
TGTnewElement       ident
TGTtext             media queries
TGTendElement       ident
TGTendElement       atrule
```

And creates the following tagged document structure

```
<AtRule type="import">
      <ident>media queries</ident>
</AtRule>
```

```
@media mediaType[,MediaType2]...[,MediaTypeN] {
       PropertyName : PropertyValue;
       ...
       }
```

This rule specifies properties for the specified media types . It generates the following tagged data.

```
       TGTnewElement       atrule
       TGTattrName         type
       TGTattrValue        media
       TGTnewElement       ident
       TGTtext             mediaType
       TGTendElement       ident
       TGTnewElement       PropertyList
       TGTnewElement       Property
       TGTattrName         name
       TGTattrValue        PropertyName
       TGTtext             PropertyValue
       TGTendElement       Property
       TGTendElement       PropertyList
       TGTendElement       atrule
```

And creates the following tagged document structure

```
<AtRule type="media">
       <ident>mediaType</ident>
       [<ident>mediaType1</ident>]
       [<ident>mediaTypeN</ident>]
       <PropertyList>
          <Property name="PropertyName">
             PropertyValue
          </Property>
          . . .
       </PropertyList>
</AtRule>
```

```
@page pseudoType[,pseudoType2]...[,pseudoTypeN] {
       PropertyName : PropertyValue;
       ...
       }
```


This rule declares properties for the specified page types. It generates the following tagged data.

```
TGTnewElement       atrule
TGTattrName         type
TGTattrValue        page
TGTnewElement       ident
TGTtext             pseudoType
TGTendElement       ident
TGTnewElement       propertylist
TGTnewElement       property
TGTattrName         name
TGTattrValue        PropertyName
TGTtext             PropertyValue
TGTendElement       property
TGTendElement       propertylist
TGTendElement       atrule
```

And creates the following tagged document structure

```
<atrule type="page">
       <ident>pseudoType</ident>
       [<ident>pseudoType1</ident>]
       [<ident>pseudoTypeN</ident>]
       <propertylist>
          <property name="PropertyName">
             PropertyValue
          </property>
          . . .
       </propertylist>
</atrule>
```

The other AtRules follow a similar pattern to the ones above.

```
@document functions {
       PropertyName : PropertyValue;
       ...
       }
```

```
@font-face {
       PropertyName : PropertyValue;
       ...
       }

...
```

This class is not built-in, so it has be imported, as follows:

```
import('tgio\ETaggedInputStream');
```

Instance Methods

`Integer` getNextTaggedToken;

This method retrieves and returns the next tag type and stores the curr tag value, so it can be retrieved by the currTag method.

class Methods

`tgio.INITaggedInputStream.`make[(pTextInputStream)];

This creates an instance of a `INITaggedInputStream` object with the specified text input stream.

class ETaggedOutputStream

The `ETaggedOutputStream` class represents the base class of all tagged output streams. It is a class that provides the basic functionality for a stream of tagged types and tagged values that correspond to the basic tagged architecture. It requires a ETextOutputStream object (or one with the same methods) to provide a desstination for a stream of text characters, which are sent as tagged items. This base class provides the empty methods in case a derived method wants to ignore some types of output.

There are also some class methods

This class is not built-in, so it has be imported, as follows:

```
import('tgio\ETaggedOutputStream');
```

Instance Methods

`ETextOutputStream` outputStream[(pOutputStream)];

This method gets and optionally sets the text output stream object. This is usually set by a derived method.

`null` initDocument;

This method initializes the tagged output stream, by writing out any header markup to the output text file. This calls two methods to perform this initialization, putStartDocument and putDocumentHeader.

This method should be called before sending any other tagged information to the output stream.

`null` termDocument;

This method writes the footer information(if any) to the tagged output stream, by writing out any footer markup to the output text file.

This method should be called after sending all the other tagged information to the output stream.

`String` docTitle[(pTitle);]

This method gets and optionally sets the document title, if needed. This is usually written out during the initDocument call.

`null` putStartElement(pElementTag);

This method writes out the markup (if any) for start of an element.

`null` putCloseElement;

This method writes out the markup (if any) for the closing of a start element tag.

`null` putEndElement(pElementTag,pPutEOL=true);

This method writes out the markup (if any) for end of an element. If the pPutEOL is set to true it writes out an end of line character afterwards.

`null` putStartAttributeList;

This method writes out the markup (if any) for start of an attribute list.

`null` putAttribute(pAttrName, pAttrValue);

This method writes out the attribute name/value pair.

`null` putEndAttributeList;

This method writes out the markup (if any) for end of an attribute list.

`null` putElement(pElementTag, pAttrNames, pAttrValues, pText, pTermElement=true);

This method writes out the markup (if any) for an entire element. Use this method when you have all the information (attributes) necessary. You can optionally include some child text as well as terminating the element. This one call replaces the above putStartElement, putCloseElement, putStartAttributeList, putAttribute, and putEndAttributeList.

The pAttrNames/pAttrValues can both be string values. In this case only one attribute can be written out. They can also both be string arrays. In this case, a set of attribute name/values will be written out.

If the pTermElement is set to true, then the element will be closed and you cannot have any child elements.

The pText can be null.

`null` putText(pText,pPutEOL=false);

This method writes out a text string to the output stream. If the stream needs to change some of the characters, due to markup conditions, (such as Xml/Html styles), it will perform that function here. If the pPutEOL is set to true it writes out an end of line character afterwards.

null putTextNX(pText);

This method writes out a text string without any translation. This outputs the raw text to the output stream.

null putEOL;

This method writes out an end of line character.

null putNonBreakingSpace;

This method writes out a non-breaking space, if the tagged output stream has special handling for it.

null putStartDocType(pName);

This method writes out the markup (if any) for start of a doc type list.

null putDocTypeData(pText);

This method writes out the doc type text.

null putEndDocType(pName);

This method writes out the markup (if any) for end of a doc type list.

null putStartNotation(pName);

This method writes out the markup (if any) for start of a notation.

null putNotationData(pText);

This method writes out the notation text.

null putEndNotation(pName);

This method writes out the markup (if any) for end of a notation.

null putStartElementDef(pName);

This method writes out the markup (if any) for start of an elementdef.

null putElementDefData(pText);

This method writes out the elementdef text.

null putEndElementDef(pName);

This method writes out the markup (if any) for end of an elementdef.

null putStartAttList(pName);

This method writes out the markup (if any) for start of an attlist.

null putAttListData(pText);

This method writes out the attlist text.

null putEndAttList(pName);

This method writes out the markup (if any) for end of an attlist.

null putStartProcessingInstruction(pName);

This method writes out the markup (if any) for start of a PI.

null putProcessingInstruction(pText);

This method writes out the PI text.

null putEndProcessingInstruction(pName);

This method writes out the markup (if any) for end of a PI.

null putStartCDataSection;

This method writes out the markup (if any) for start of a CData Section.

null putCDataSection(pText);

This method writes out the CData Section text.

null putEndCDataSection;

This method writes out the markup (if any) for end of a CData Section.

null putStartComment;

This method writes out the markup (if any) for start of a comment.

null putComment(pText);

This method writes out the comment text.

null putEndComment;

This method writes out the markup (if any) for end of a comment.

null putStartEntity(pName,pParmEntity=false);

This method writes out the markup (if any) for start of an entity.

null putEntity(pText);

This method writes out the entity text.

null putEndEntity;

This method writes out the markup (if any) for end of an entity.

Class Methods

The following are class methods. They can be called without creating an object.

`ETaggedOutputStream` makeTaggedOutputStream(pOutput,pType,pEncoding);

This method creates a tagged output stream based on the information provided. This is a convenience method to make it easier to create a standard tagged output stream. You can do these steps manually by creating a text output

stream and then the appropriate tagged output stream, then putting them together, but this does it all in one method. If you have a non-standard tagged output stream, then you will need to do these steps separately.

The pOutput parameter can be a file name, a byte output stream or a text output stream. The pType parameter is a string value that indicates the type of tagged output stream that you wish to create. The possible values are 'xml', 'html', 'xhtml' or 'ini'. The 'xml' type will create a XmlOutputStream, the 'html' will create an HtmlOutputStream and 'xhtml' type will create an XhtmlOutputStream. The 'ini' type creates an IniOutputStream. The pEncoding parameter indicates the type of encoding to be used with the text output stream.

If the pOutput parameter is a string value, then it is assumed to be a file name. A ETextOutputFile will be created using this file name and the specified encoding. If the pOutput is a byte output stream, then a ETextOutputStream will be created using this byte output stream.

Once we have an ETextOutputStream, an ETaggedOutputStream object will be created (based on the pType parameter). The output text stream will be set and the new tagged output stream will be returned.

class IniOutputStream<ETaggedOutputStream

The `IniOutputStream` class represents the tagged output stream in the style of an Microsoft windows ini file.

The ini file structure is simple. When sending data to an `IniOutputStream` only one level of elements and one set of attributes. An ini file looks something like the following:

```
[SectionName]
KeyName1=KeyValue1
KeyName2=Keyvalue2
...
KeyNameN=KeyValueN

[SectionName2]
...
```

The section names correspond to elements and the KeyNameX/KeyValueX pairs correspond to attribute Names/Values of a single level. Subsequent levels are ignore, so this can be thought of as a lossy tagged output stream.

This class is not built-in, so it has be imported, as follows:

```
import('tgio\ETaggedOutputStream');
```

Instance Methods

`ETextOutputStream` outputStream[(pOutputStream)];

This method gets and optionally sets the text output stream object. This is usually set by a derived method.

null initDocument;

This method initializes the tagged output stream, by writing out any header markup to the output text file. This calls two methods to perform this initialization, putStartDocument and putDocumentHeader.

This method should be called before sending any other tagged information to the output stream.

null termDocument;

This method writes the footer information(if any) to the tagged output stream, by writing out any footer markup to the output text file.

This method should be called after sending all the other tagged information to the output stream.

class Methods

`tgio.IniOutputStream.`make[(pTextOutputStream)];

This creates an instance of a `IniOutputStream` object with the specified text output stream.

class XmlOutputStream<ETaggedOutputStream

The `XmlOutputStream` class represents the tagged output stream in the style of an Microsoft windows ini file.

This class is not built-in, so it has be imported, as follows:

import('tgio\ETaggedOutputStream');

Instance Methods

null initDocument;

This method initializes the tagged output stream, by writing out any header markup to the output text file. This calls two methods to perform this initialization, putStartDocument and putDocumentHeader. The Xml output the putStartDocument method writes out the <?xml ... > line and the putDocumentHeader writes out the header, stylesheet, title and root element. If you wish these to be different, then you can override these methods to product the results you wish.

This method should be called before sending any other tagged information to the output stream.

null termDocument;

This method writes the footer information(if any) to the tagged output stream, by writing out any footer markup to the output text file. For Xml documents, it writes out the ending markup of the root element.

This method should be called after sending all the other tagged information to the output stream.

String styleSheet[(pStyleSheetFileName);]

This method gets and optionally sets the style sheet filename. This is usually written out during the initDocument call.

class Methods

tgio.XmlOutputStream.make[(pTextOutputStream)];

This creates an instance of a XmlOutputStream object with the specified text output stream.

class HtmlOutputStream<ETaggedOutputStream

The HtmlOutputStream class represents the tagged output stream in the style of an Html file. This html file must be well formed. All tags must have closing tags.

This class is not built-in, so it has be imported, as follows:

import('tgio\ETaggedOutputStream');

Instance Methods

null initDocument;

This method initializes the tagged output stream, by writing out any header markup to the output text file. This calls two methods to perform this initialization, putStartDocument and putDocumentHeader. For Html output the putStartDocument method writes out the DocType line and the putDocumentHeader writes out the header, stylesheet, title and starts the body element. If you wish these to be different, then you can override these methods to product the results you wish.

This method should be called before sending any other tagged information to the output stream.

null termDocument;

This method writes the footer information(if any) to the tagged output stream, by writing out any footer markup to the output text file. For Html documents, it writes out the ending markup of the body and html elements.

This method should be called after sending all the other tagged information to the output stream.

String styleSheet[(pStyleSheetFileName);]

This method gets and optionally sets the style sheet filename. This is usually written out during the initDocument call.

class Methods

`tgio.HtmlOutputStream.`make[(pTextOutputStream)];

This creates an instance of a `HtmlOutputStream` object with the specified text output stream.

class XhtmlOutputStream<ETaggedOutputStream

The `XhtmlOutputStream` class represents the tagged output stream in the style of an XHtmlfile.

This class is not built-in, so it has be imported, as follows:

```
import('tgio\ETaggedOutputStream');
```

Instance Methods

`null` initDocument;

This method initializes the tagged output stream, by writing out any header markup to the output text file. This calls two methods to perform this initialization, putStartDocument and putDocumentHeader. For Xhtml output the putStartDocument method is not called. The putDocumentHeader writes out the <?xml ... > line and the header, stylesheet, page template,title and starts the body element. If you wish these to be different, then you can override these methods to product the results you wish.

This method should be called before sending any other tagged information to the output stream.

`null` termDocument;

This method writes the footer information(if any) to the tagged output stream, by writing out any footer markup to the output text file. For Xhtml documents, it writes out the ending markup of the body and html elements.

This method should be called after sending all the other tagged information to the output stream.

`String` styleSheet[(pStyleSheetFileName)];

This method gets and optionally sets the style sheet filename. This is usually written out during the initDocument call.

`String` pageTemplate[(pPageTemplateFileName)];

This method gets and optionally sets the page template filename. This is usually written out during the initDocument call.

class Methods

`tgio.XhtmlOutputStream.`make[(pTextOutputStream)];

This creates an instance of a `XhtmlOutputStream` object with the specified text output stream.

class ECssOutputStream<ETaggedOutputStream

The `ECssOutputStream` class represents the tagged output stream in the style of a css file. Like all tagged output streams, this class mimics output css items in the form of xml-style output methods. This is usually the reverse of the ECssTaggedInputStream. For example, to write the following style rule to a css file:

```
SelectorData[,SelectorData2]...[,SelectorDataN]  {
       PropertyName : PropertyValue;

       ...
       }
```

Use the following method calls (assuming you have created a `ECssOutputStream` object instance into the `cssOut` variable):

```
cssOut.PutStartElement('stylerule');
cssOut.PutStartElement('selector');
cssOut.PutText('SelectorData');
cssOut.PutEndElement('selector');
// Repeat the above three lines for each selector
cssOut.PutStartElement('propertylist');
cssOut.PutStartElement('property');
cssOut.PutAttribute('name','PropertyName');
cssOut.PutText('PropertyValue');
cssOut.PutEndElement('property');
...// Repeat the above four lines for each property/value combination
cssOut.PutEndElement('propertylist');
cssOut.PutEndElement('stylerule');
```

This class is not built-in, so it has be imported, as follows:


```
import('tgio\ETaggedOutputStream');
```

Instance Methods

class Methods

`tgio.ECssOutputStream.`make[(pTextOutputStream)];

This creates an instance of a `ECssOutputStream` object with the specified text output stream.