Table Of Contents

Introduction to TreeWorld Scripts

What is a Treeworld script ?

A TreeWorld script is a BeanShell script that can be executed on any item that is displayed in TreeWorld browser.

TreeWorld Browser is a tree-based GUI for browsing arbitrary hierarchical information. It is built in to the Topologi Professional Edition. More information on TreeWorld and Topologi can be found on: http://www.topologi.com

BeanShell scripts are like lightweight java scripts. Because they are scripts, they do not need to be compiled: they are dynamically interpreted by TreeWorld. Also their structure is a bit more loose than usual Java classes, however, they give the developer full access to the Java API.

For more information on BeanShell script: http://www.beanshell.org/

What can I do with it ?

TreeWorld scripts are task-based. In other words, they can be used to do anything on the node(s) the user has selected in TreeWorld. They should but do not have to return something, typically it would be a tree of nodes that would be displayed by TreeWorld, the scripts allows the developer to decide what needs to be displayed by TreeWorld and how.

Because the scripts can use the complete java API as well as any java library that the developer would like to include, any functionality can be plugged into TreeWorld. Scripts could be used to resize images, validate XML files, generate an index, search documents, produce reports, etc…

For example, when TreeWorld displays the content of a directory, a set of tasks can be applied on any of the files, those tasks are TreeWorld scripts.

How does it work ?

When the user select one or several nodes in the TreeWorld browser, the list of tasks can be executed appears. Once the user has chosen which task to execute, and possibly filled in any required parameters, the corresponding script is interpreted the Universal Wire framework.

Universal Wire uses a client-server type of architecture, where TreeWorld acts as the client and the script is the service:

1. TreeWorld send request trough the Universal Wire.
2. The script receives the XML request.
3. The script produces XML response(s) that it sends via Universal Wire.
4. TreeWorld receives the XML response.

However, scripts do not need to wait until they have finished to send XML responses. In fact, TreeWorld can receive XML responses at any time during the execution of the script allowing more interactive et informative processing.

[diagram showing architecture]

What do I need to know ?

Writing TreeWorld scripts does not require a lot of experience in either BeanShell, Java or XML, however some basic knowledge of java and XML is essential. Having some understanding of XML namespaces and DOM (Document Object Model) is helpful.

Terminology

Topologi Terminology

Topologi

The Collaborative Markup Editor.

Universal Wire

A framework allowing peer to peer collaboration in a service-oriented environment using SOAP as its main communication protocol.

TreeWorld

A tree-based GUI for browsing hierarchical information: the browser for Universal Wire.

The XML interface

The request and response are using XML 1.0.

SOAP

Topologi's Universal Wire wraps all the requests and responses using SOAP 1.2 (Messaging Framework). This means that all requests and responses that will be received by the script will have the following structure:

```
<?xml version="1.0"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
<env:Header encodingStyle="http://www.topologi.com/ns/treeworld/header" />
<env:Body encodingStyle="http://www.topologi.com/ns/treeworld/body">

        [treeworld_request | treeworld_response]

</env:Body>
</env:Envelope>
```

For more information on SOAP, refer to the W3C website:
http://www.w3.org/TR/soap12-part1/

XML input: TreeWorld Request

The XML input corresponds to what TreeWorld sends through Universal Wire to the script, and that is wrapped by a SOAP envelope, it encapsulates the user input in an XML format.

The selected nodes are wrapped around a TreeWorld Request element. It belongs to the Topologi TreeWorld namespace. It includes the namespace declaration , the unique identifier of the TreeWorld browser (guarantees unicity amongst peers) and the behaviour of the script as attributes using the same namespace.

The parameters for the scripts are provided by TreeWorld as attributes name/value pairs, where each parameter name appears as the attribute name, and its value as the attribute value. Those attributes belong to the default namespace.

The children of the Request element correspond to the tree of node selected by the user in TreeWorld if any.

A TreeWorld request has the following structure:

```
<tw:Request xmlns:tw="http://www.topologi.com/ns/treeworld/Body"
            tw:treeworldId="[auto_generated_unique_id]"
            tw:behaviour="[behaviour_defined_by_the_script]"
            [param_name1]="[param_value_1]"
            [param_name2]="[param_value_2]"
            [param_name3]="[param_value_3]"
             ...
 >
              [input_nodes_selected]

</tw:Request>
```

XML output: TreeWorld Response

The XML output corresponds to what TreeWorld receives from the script, and that is wrapped by a SOAP envelope.

The generated nodes are wrapped around a TreeWorld Response element which belongs to the Topologi TreeWorld namespace. It includes the namespace declaration and the behaviour of the script as attributes using the same namespace.

A TreeWorld response has the following structure:

```
<tw:Response xmlns:tw="http://www.topologi.com/ns/treeworld/Body"
tw:behaviour="replace">
               [message_or_nodes_from_script]
</tw:Response>
```

All children elements are used to create nodes for the tree. How these nodes are handled depends on the type of the response, specified in the attributes of the element Response.

Response Tag Attributes:

tw:error

Attribute used to specify that the response is an error response. The value should be **true** or **1** for the response to be considered an error response.
If a response is an error response, the tree is not modified and the error message is displayed in the status bar.

tw:behaviour

This attribute is required, if not present, the value is **replace**.
TreeWorld defines 5 different types of behaviour:

- **replace**: creates a new tree (in a new tab).
- **overwrite**: overwrites nodes in the current tree.
- **leave**: does not change the current tree.
- **append**: adds nodes to the current tree as children of the node specified in the attribute **tw:select_node.**
- **delete**: deletes nodes in the current tree (all nodes to delete are specified with their **tw:id** attribute in the response message).

tw:title

This attribute is used only in case of the creation of a new tree (tw:behaviour value is **replace**). The value is used as the title of the new tab.
This attribute is strongly recommended for a **replace** response.

tw:select_node

If specified, the value of this attribute is used to select a node in the tree after reception of the response. Its value is the **id** (attribute tw:id) of the node to select. This attribute is required if the type is **append**, as it specifies the node to append the response to.

tw:message

If specified, the value of this attribute is used as a status message and displayed in the status bar.

tw:warning
If specified, the value of this attribute is used as a warning message and displayed in the status bar.

tw:move
This attribute is used only for an **overwrite** response. Its value can be either **up** or **down**. The result is the movement of the replaced node(s) in the direction specified and if possible.

Children elements attributes
All these elements are used to create nodes for the tree. Each element represents a node in the tree and its node representation is defined by the attributes of the element.

tw:id
This attribute is not required as it is created if not specified (and needed). It's used to specify a node in the tree, as its value is unique in this tree.

tw:name
Used as the label of the node in the tree. If it's not specified, the element tag name will be used, it is therefore highly recommended to specify this attribute.

tw:open
Optional, this attribute is used to specify a default open Treeworld URL. Its value is a valid Treeworld URL (starting with treeworld://…). If a node has a default open action, a menu item called "Open" will appear as the first action available in the popup menu generated on a right-click. If not specified, the default

tw:flag
Optional, the value of this attribute is used to load an icon from the icon directory. This icon is displayed **after** the label of the node.

tw:decoration
Optional, the value of this attribute is used to load an icon from the icon directory. This icon is displayed **in front of** the label of the node.

tw:link
Optional, this attribute is used to point to another node in the same tree. Its value is the id of the node pointed to, preceded by the # character (based on html anchor). If this attribute is present, a link icon will appear after the label of the node and when this node is double-clicked on, the pointed node will be selected (if found).

The java interface

A topologi script should implement all of the methods defined in this section.

BeanShell script being a dialect of java, here are the important differences:

- no need to declare the class: methods are declared directly,
- no fields are allowed as a script is not a class, just an ensemble of methods,
- no need to define the methods as public or private, they all behave as if they were public,
- methods cannot throw exceptions,
- the following return statement must be included at the end:
  `return (ServicePlugInScript) this;`

GetParameters

This method defines the parameters that the user can send to the script. These parameters are passed in the request as attributes so their value is represented as a `String` object. The list of all parameters is a `Map` object, its keys are the names of all parameters and the values are their types or default values.

The parameters names are used to identify each parameter and will be displayed in the parameter window. To present the parameters in a specific order, they should be preceded by the order number followed by the dash character (see example). The values can be of two types: the type of the parameter or a set of values (or only one) to choose from.

If the value is a `Class` object, it will be used to select how to present the parameter in the window:

Class used	Visual Component used
`String`	A text box (single line)
`JTextArea`	A larger text box (multi-line)
`File`	A text box followed by a “Browse” button to select a file only
`FileDescriptor`	A text box followed by a “Browse” button to select a directory only
`Boolean`	A check box (not selected)
`JPasswordField`	A password text box (the text entered is hidden)
Anything else	A text box single line

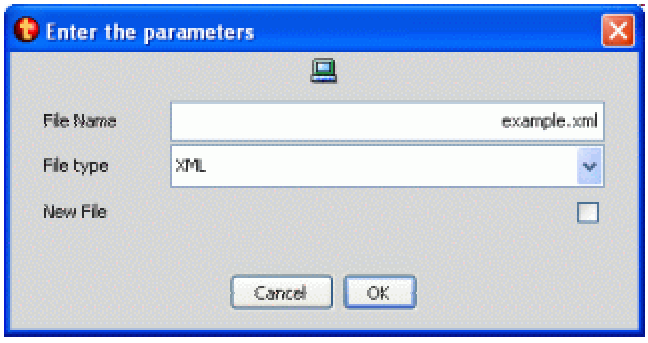
If the value is an object that is not a class, this object is used in the same fashion to display the parameter. There are currently two types of object supported: a `String` and a `Vector`, any other objects will be turned into a `String` and used as such. For a `String`, the value is used as an XPath relating to the node currently selected in the tree, if a simple default value is wanted, then enclosing it with single quotes is enough. For a Vector, the **first value** of the `Vector` is also considered as an XPath (can be used to set a default value as the first value of the list is the one selected by default).

Java Interface

```
Map getParameters()
```

Example (and its result):

```
Map params = new Hashtable();
params.put("1-File Name","@tw:name");
Vector v = new Vector();
v.add("'XML'");
v.add("PDF");
v.add("SGML");
params.put("2-File type",v);
params.put("3-New File",Boolean.class);
return params ;
```



ExecutePlugIn

The method executes the task it was meant to do. It does not return anything as it has access to the data and handler provided by TreeWorld. A utility class called `ScriptsUtil` provides various methods to help with the execution of the task.

Java Interface

```
void executePlugIn(PlugInData pd, PlugInResultHandler handler);
```

Getting the request

After retrieving the request object from the `PlugInData` object using the following line:

```
TreeWorldRequest twr = (TreeWorldRequest)
pd.getDocumentData().getProperty(DocumentData.TREEWORLD_REQUEST);
```

The best way to handle the request is to build a DOM object from it. The method `getRequestAsDOM(TreeWorldRequest twr);` returns the DOM object that can be used to get the request's details. Most of the relevant information is in the actual Request element that can be obtained with the method `getRequestElement(Document dom);`

Retrieving parameters

If any parameters were specified in the `getParameters()` method, they need to be retrieved to be used. Standard parameters are passed as attributes of the Request element and here again the `ScriptsUtil` class has the standard method `getAttribute(Node node, String attr);`, which returns the defined attribute from the given element. As these parameters are passed as attributes, the space character is not allowed and the underscore character replaces any space specified in the parameter name. Also, the ordering number sometimes used In the parameter name is stripped for the attribute name.

For specific attributes, more methods are defined in the utility class. `getAttributeAsBoolean(Node node, String attr, boolean defaultValue);` can be used to retrieve Boolean attributes, it returns a `boolean` and needs a `defaultValue` parameter in case the attribute is not found. `getFullPathAttribute(Node node);` can be used to obtain the full path attribute (`tw:full_path`) (usually nodes of type File or Directory) and `getNodeIDAttribute(Node node);` returns the id attribute (`tw:id`) of any node.

Parameters retrieval example:

Based on the previous example, here si the retrieval of the parameters:

```
String filename = ScriptsUtil.getAttribute(requestNode, "File_Name");
String fileType = ScriptsUtil.getAttribute(requestNode, "File_Type");
boolean isNewFile = ScriptsUtil.getAttributeAsBoolean(requestNode, "New_File",
true);
```

Sending the response

As said before, there are different types of request to send back to the client.
To send a response to the client, there are once again a couple of methods provided by the `ScriptsUtil` class. All these methods have common parameters, the request received (`twr`) used for its `uuid`, the `PlugInData` object (`pd`) and the `PlugInResultHandler` (`handler`) that will receive the response:

```
- sendNormalResponse(String xml, TreeWorldRequest twr, PlugInData pd,
PlugInResultHandler handler);
```

The `xml` parameter is the response that will be used to create the tree. This method is used to send a standard response to the caller. The response can be of any type as the entire XML response is passed as parameter (with soap headers).

```
- sendErrorResponse(String error, TreeWorldRequest twr, PlugInData pd,
PlugInResultHandler handler);
```

In this case, the response is only an error response. The `error` parameter is simply the error message and will appear as an error message in the status bar of the Treeworld application.

- `sendStatusMessageResponse(String message, TreeWorldRequest twr, PlugInData pd, PlugInResultHandler handler);`

This last method is used to send a response of type **leave** to the caller. A simple status message (passed as the parameter `message`) is displayed in the status bar and the tree currently displayed is not modified.

GetStatus

This method defines the names of service and action provided by the script.

Java interface

```
String[] getStatus();
```

In TreeWorld, the returned String array appears as a list of menus and submenu, the last item in the array being the highest hierarchical item, the first item being the name of the action.

Example:

```
new String[]{'Resize Image', 'Image Scripts', 'My Scripts'}
```

(will appear in TreeWorld as My Scripts > Image Scripts > Resize Image)

GetXpath

This method defines which nodes the script is attached to.

Java interface

```
String getXpath();
```

The returned Xpath corresponds to the nodes the script applies to. The Xpath context is the level of the task. Any Xpath 1.0 function can be used. Note that Xpath 1.0 is case sensitive.

Example:

```
File[translate(@extension,'XML', 'xml') = 'xml']
```

(applies to all files which extension is ‘xml’ regardless of the case)

getVisibility

This method defines the visibility of script, that is where in TreeWorld it can be triggered.

Java interface

```
String getVisibility();
```

TreeWorld defines 4 levels of visibility:

- **element**: for scripts that are triggered from the element node in the main TreeWorld window.
- **attribute**: for scripts that are triggered from the attribute node in the main TreeWorld window. In this case, the XPath needs to point to an XML attribute.
- **go**: for scripts that are triggered from the “go” menu in TreeWorld (those scripts generally don’t have parameters).
- **no**: for the scripts that the user does not trigger directly, but that could be used by other scripts.

The most common option here are “element” or “go”.

GetExecutability

This method defines the level of executability of the script.

Java interface

```
String getExecutability();
```

TreeWorld currently has 4 different levels:

- **local**: the script is stored and for use on the local machine only, local scripts are not shared,
- **peer**: the script can be shared and used by the Topologi peers on the network,
- **central**: same as peer but when “update script” is ticked, the script is downloaded from the central server into the folder,
- **topologi**: same as peer but when “update script” is ticked, the script is downloaded from the Topologi web server into the folder.

Most customised scripts would use ‘local’ by default.

The "Show Pretty" option

What is the "Show Pretty" option?

This option allows an HTML display of the node selected in the tree. The HTML view is visible on the right-hand side panel of the client frame. The "show pretty" option is the default view and this cannot be turned off (but another view can be selected instead: "show properties" or "view source").

How does it work?

The first step is the selection of the XSL Stylesheet used to preview the node. This selection is based on the type of the node: if there is a file in the stylesheet directory which name correspond to the current node then the "Show pretty" option is turned on. The file must follow a naming convention: **<Type>ToHTML.xsl** (i.e. FileToHTML.xsl, see Installation section).
Once the stylesheet is found, an XML representation of the selected node is created. The XML data is then transformed to HTML using the stylesheet and the result is displayed as HTML in the right-hand side panel. The XML representation of the node has a specific format, containing variables that can be used in the stylesheet:

```
<root>
        <imageDir>[Location of the images directory]</imageDir>
        <libDir>[Location of the lib directory]</libDir>
        <tmpDir>[Location of the temp directory]</tmpDir>
        <contents>
                [XML representation of the node (as seen in the "view source" option)]
        </contents>
</root>
```

What can I do?

Java HTML Pane supports HTML version 3.2 (migrating towards 4.0).

Images are allowed and must be located in a specific directory (treeworld/images/html, see Installation section). This directory should be retrieved in a varaiable in the stylesheet:

```
        <xsl:variable name="imageDir" select="/root/imageDir" />
```

The stylesheet should transform the XML representation of the node, this is available using the following template:

```
<xsl:template match="/">
        <xsl:apply-templates select="root/contents" />
</xsl:template>
<xsl:template match="contents">
        [Enter your code here…]
</xsl:template>
```

A very special feature added in the Treeworld HTML preview panel is the use of Treeworld URL. As a matter of fact, a Treeworld URL can be used as a normal HTML link in an "a" tag (i.e. `<a href="treeworld://localhost/File?doOpen">Open</a>`).
When the link is clicked on, a request is sent to the action pointed by the URL. The node selected in the tree is used in the request.

Installation and configuration

Directories and files

All scripts are located in the **treeworld** directory of the installation directory, under the **services** folder. Each service has its own directory, which name is used has a reminder of what the service is but not used anywhere (except in the URL of each script but that is transparent to the user). Each service directory must implement the Topologi hierarchy, that means that each Service directory must contain four subdirectories called **central**, **local**, **peer** and **topologi**. Any of these can contain scripts, the only difference is how the script is shared or where it is coming from. A script coming from Topologi (from installation or a web service update) is more likely to be in the **topologi** folder, the **peer** directory contains scripts to share with other peers (scripts can be downloaded from one peer to the other, feature not implemented yet). The **central** folder is likely to contain scripts coming from a central server (i.e. intranet) and finally **local** scripts are not shared and are usually customized scripts created locally.
Each script file name must follow the pattern: <script name>-<version>.bsh

Java libraries

TreeWorld being part of Topologi professional edition, uses the same java instance. At present, Topologi uses the Java 2 Standard Edition version 1.4.2, although this could change in the future, this will be the minimum supported by topologi.

Information regarding Java 1.4.2 can be found on the java website at:
http://java.sun.com/j2se/1.4.2/docs/

It is possible to include more java libraries for use by the scripts, they simply need to be placed under the /jar directory of the Topologi's installation directory.

Graphic files

There are different types of images used in Treeworld, they are gathered in the following table:

Description	**Image type**	**Location**
Node icon. Used to represent the node in the tree, for each type an expanded icon can be specified. Naming convention: <Type>.gif and <Type>_exp.gif	Gif file (16x16 recommended)	treeworld/images/types
Decoration icon. Used to decorate the node in the tree (see attributes **tw:flag** and **tw:decoration**). No naming convention.	Gif file (16x16 recommended)	treeworld/images/decoration
HTML images. Used by the XSL transformation to display the node in mode "Show pretty".	Any image file.	treeworld/images/html

XSL stylesheet for the “show pretty” option

Any XSL stylesheet used to show a node in the preview panel (see the “Show pretty” section) is supposed to be located in the **lib** directory, under the **nodeStyles** folder. The XSL files must follow the naming convention: “**<Node Type>ToHTML.xsl”**. <Node Type> is the type of the node to preview, in other words it’s the name of the element in the source of the tree (i.e. File or Directory).

A sample script

A simple example but functionally complete.
Comments are included as java comments.

```
// use the following standard java libraries
import java.io.*;
import java.util.*;
import org.w3c.dom.*;

// use the following libraries from topologi
import com.topologi.tme2.io.plugins.*;
import com.topologi.tme2.io.worklistmodel.*;
import com.topologi.tw.message.*;
import com.topologi.tw.scripts.*;
import com.topologi.tw.service.*;



void executePlugIn(PlugInData pd, PlugInResultHandler handler) {
  // get the TreeWorld request
  TreeWorldRequest twr = (TreeWorldRequest)pd.getDocumentData()
                         .getProperty(DocumentData.TREEWORLD_REQUEST);
  // proceed only if the request is not null
  if (twr != null) {
    Document request = null;
    try {
      // get the request as a DOM tree, which make it easier to manage
      request = ScriptsUtil.getRequestAsDOM(twr);
      // Retrieve the TreeWorld Request element
      Node body = ScriptsUtil.getRequestElement(request);
      // retrieve the parameter from the user
      String newName = ScriptsUtil.getAttribute(body, "New_file_name");
      // if the new name is null, it probably means that the user did not specify it
      if (newName == null) {
        // we return an error message to TreeWorld and end processing
        ScriptsUtil.sendErrorResponse("No new file name", twr, pd, handler);
        return;
      }
      // we can now retrieve all the file nodes and directory nodes from the request
      List files = ScriptsUtil.getChildrenNamed(body, "File");
      List dirs = ScriptsUtil.getChildrenNamed(body, "Directory");
      // if no files or directory were selected
      if (files.isEmpty() && dirs.isEmpty()) {
        // we return an error message to TreeWorld and end processing
        ScriptsUtil.sendErrorResponse(
                    "No file or directory selected", twr, pd, handler);
        return;
      }
      // apply our 'renameFiles' method on the files and directories
      renameFiles(files, newName, twr, pd, handler);
      renameFiles(dirs, newName, twr, pd, handler);
    // apply our 'renameFiles' method on the files and directories
    } catch (Exception e) {
      ScriptsUtil.sendErrorResponse("An error occurred while creating a "+
               "response: "+e.getMessage(), twr, pd, handler);
    }
  }
}

void renameFiles(List dirs, String newName, TreeWorldRequest twr, PlugInData pd,
PlugInResultHandler handler) {
  String type = "";
  for (int i = 0; i < dirs.size(); i++) {
    Node dir = (Node) dirs.get(i);
    String path = ScriptsUtil.getFullPathAttribute(dir);
    String fileId = ScriptsUtil.getNodeIDAttribute(dir);
    String error = null;
    File toFile = null;
    if (path != null) {
      File f = new File(path);
      if (!f.exists()) {
        error = "The file "+path+" is not valid";
      } else {
```

```
        if (f.isFile()) type = "File";
        else type = "Directory";
        toFile = new File(f.getParent(), newName);
        if (toFile.exists()) {
          error = "The file "+toFile.getAbsolutePath()+" already exists";
        } else {
          if (!f.renameTo(toFile)) {
            error = "Renaming of the file failed";
          }
        }
      }
    }
    if (error != null) {
      ScriptsUtil.sendErrorResponse(error, twr, pd, handler);
      return;
    }
    ScriptStringBuffer responseXML = new
        ScriptStringBuffer(TreeWorldMessageUtils.createStartResponse());
    responseXML.append(" tw:behaviour=\"overwrite\">");
    // Append the XML for the current file
    responseXML.openTag(type);
    // Append the name attribute
    responseXML.appendXMLAttribute("tw:name", newName);
    // add the id
    responseXML.appendXMLAttribute("tw:id", fileId);
    // Append the full_path attribute
    responseXML.appendXMLAttribute("tw:full_path", toFile.getAbsolutePath());
    responseXML.appendDateAttribute("date", toFile.lastModified());
    if (type.equals("Directory")) {
      responseXML.append(" size=\"-1\"");
      responseXML.append(" extension=\"___\"");
    } else {
      responseXML.append(" size=\"").append(toFile.length()).append("\"");
      if (toFile.getName().indexOf('.') != -1) {
        String name = toFile.getName();
        responseXML.append("extension=\"");
        responseXML.append(name.substring(name.lastIndexOf('.')+1)).append("\"");
      }
    }
    responseXML.append("/>");

    responseXML.append(TreeWorldMessageUtils.createEndResponse());
    ScriptsUtil.sendNormalResponse(responseXML.toString(), twr, pd, handler);
  }
}


/*
 * This script only require the new file name to be able to work.
 * Because we're nice we even give the user the original name of the file
 * by grabbing it from the File or Directory node.
 */
Map getParameters() {
  Map m = new Hashtable();
  m.put("New file name", "@tw:name");
  return m;
}

/*
 * This script will appear as 'Rename' under the 'File System' menu.
 */
String[] getStatus() {
  return new String[] {"Rename", "File"};
}

/*
 * The script should only apply to the File and Directory element nodes the
 * user has selected.
 */
String getXpath() {
  return "File | Directory";
}

/*
 * The script should only be visible for element nodes in the TreeWorld
 * browser. Its name will appear in the list of available tasks for
 * element nodes that match the given Xpath.
```

```
 *
 * @see getXpath
 */

String getVisibility() {
  return "element";
}

/*
 * We only want to allow local execution of the script.
 * (Other peers will not be able to remotely use this script)
 */
String getExecutability() {
  return "local";
}


// Always include this statement (required by the BeanShell specifications)
return (ServicePlugInScript) this;
```

References

Some references.

- BeanShell scripts:
  http://www.beanshell.org/

- XML Extensible Markup Language (XML) 1.0 (Second Edition) – a W3C Recommendation 6 October 2000:
  http://www.w3.org/TR/REC-xml

- XML Path Language (XPath) Version 1.0 – a W3C Recommendation 16 November 1999:
  http://www.w3.org/TR/xpath

- SOAP (Simple Object Access Protocol) Version 1.2 Part 1: Messaging Framework – a W3C Recommendation 24 June 2003:
  http://www.w3.org/TR/soap12-part1/

- Namespaces in XML – World Wide Web Consortium 14-January-1999:
  http://www.w3.org/TR/REC-xml-names/

- XSL Transformations (XSLT) Version 1.0 – a W3C Recommendation 16 November 1999:
  http://www.w3.org/TR/xslt

- Topologi, TreeWorld and Universal Wire – Topologi Collaborative Markup Editor:
  http://www.topologi.com

- Java 2 Standard Edition Documentation – version 1.4.2:
  http://java.sun.com/j2se/1.4.2/docs/

Appendices

A – Developing scripts in java using the ScriptCreator

[to do]

B – How to debug a script

[some typical errors, could be under FAQ as well, to do]

D – Other XML by TreeWorld

[put here, the XML for files and directories, to do]