

## XML Database Editor

### User's Guide – V3.2

© DiZTEK Software Company -1999-2003

[www.diztek.com](http://www.diztek.com)

# TABLE OF CONTENTS

[INTRODUCTION](#)

[XDE DISPLAY](#)

[THE LEFT-PANE](#)

[THE CENTER-PANE](#)

[THE RIGHT-PANE](#)

[GETTING STARTED WITH XML](#)

[CREATING TREES](#)

[XML FILTER](#)

[HOW TO](#)

[HOW TO ASSIGN DATA-TYPES](#)

[HOW TO ADD DATA-TYPES](#)

[HOW TO ASSIGN IMAGES](#)

[HOW TO ADD IMAGES](#)

[HOW TO ASSIGN STYLES](#)

[HOW TO ADD STYLES](#)

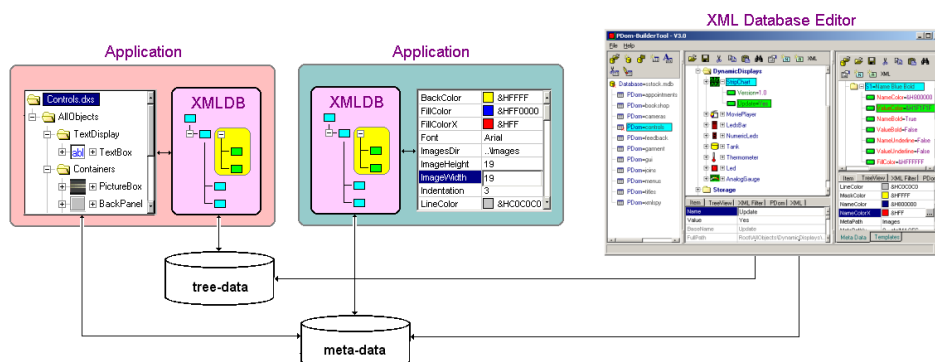
# Introduction

The **XML Database Editor (XDE)** is a **Rapid Application Development (RAD)** tool specifically designed for the editing, storage, and management of large amounts of XML under a safe, non-cryptic, and graphical environment. The tool is very simple to use and provides the following services:

- Create XML Databases.
- Create, delete, and rename Persistent Trees (Tree-tables).
- Create/Manage XML with 2 billion elements and 2 billion attributes.
- Import XML data from files or from the Clipboard.
- Export XML data to files or to the Clipboard.
- Add and remove data.
- Edit data and its properties through secure data-entry.
- Customizable data-entry by assignment of data-types with no coding.
- Expandable. Add new data-types, styles, and images with no coding.
- Browse/Navigate content.
- Seek and navigate data that complies with a query.
- Generate filtered XML with WYSIWYG.
- Easily process your data with the XML Database Engine.
- Easily publish your data with our family of Persistent TreeViews.
- Multi-user data access.

Note: All the services offered by this tool can be made programmatically with the [.NET, ASP.NET, COM] XML Database Engine (XMLDB). The XMLDB provides an object oriented and smart interface to the tree-data. Refer to the XML Database Engine Reference Manual for further details on this subject.

Persistent trees can be accessed, viewed, and modified safely, concurrently, and in real-time, without having to terminate the consumer applications.

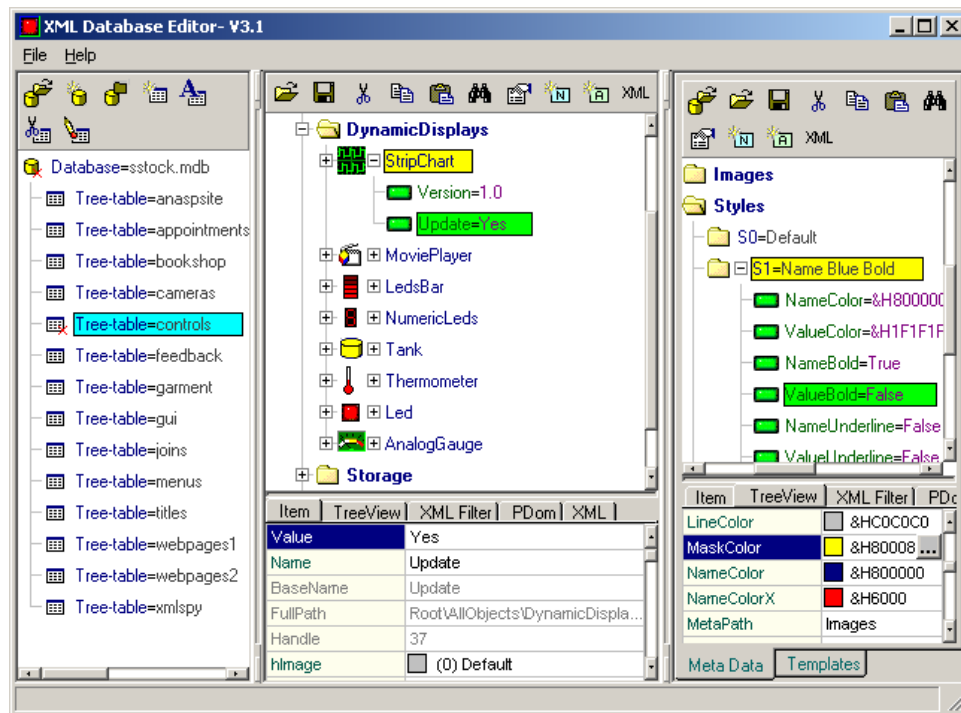


Note: The persistent trees can be published in the Internet with our ASP.NET Persistent TreeView server component with just one line of code. Also, they can be published on .NET Windows or VB6 Form-based applications with our Enhanced Persistent TreeView .NET-ready

component with just one line of code. Programmatically, perform VB-like secure data-entry in your persistent trees with our Persistent Property Grid. NET-ready component.

## XDE Display

The following figure shows the typical display of the XDE tool.



The XDE is divided functionally into three panes:

1. The Left-Pane: which is used for all database management services.
2. The Center-Pane: which is used to edit the currently selected tree-table.
3. The Right-Pane: which is used to edit meta-data and templates.

### The Left-Pane






The **Left-Pane** displays all the tree-tables contained in the currently open database. When a Tree-table is selected, the table is opened and displayed for editing in the Center-Pane. The following services are provided:



Open Database



Create Database



-  Save Database
-  Create Tree-table
-  Rename Tree-table
-  Remove Tree-table
-  Clear Tree-table









Note: The same services are provided under the File Menu

- When you open a database, the first Tree-table found is selected and its contents displayed in the Center-Pane
- When you left-click any one of the Tree-table icons in the Left-Pane the following actions take place:
  1. The previously selected Tree-table is closed.
  2. The selected Tree-table is opened and displayed in the Center-Pane.
- When you remove the currently selected Tree-table from the database, the XDE selects the root icon in the Left-Pane.
- The icon of the currently selected Tree-table is marked with a red X whenever you modify it.
- The root icon is marked with a red X whenever the user modifies any one of the Tree-tables in the database.
- When the user saves the database, any red Xs are removed.

## The Center-Pane

The **Center-Pane** displays the content of the currently selected tree-table and provides the tools necessary for adding, removing and editing the content of the tree-table. The Toolbar and the Popup menu provide the following services:

-  Import a branch from XML Script into the currently selected node.
-  Export the currently selected branch as XML.

-  Cut the currently selected branch and keep it in the Clipboard.
-  Copy the currently selected branch into the Clipboard.
-  Paste a branch from the Clipboard into the currently selected node.
-  Find a node or attribute in the Tree-table.
-  Edit the properties of the currently selected node or attribute.
-  Add a child or sibling to the currently selected node.
-  Add an attribute to the currently selected node.
-  Generate an XML representation of the currently selected branch.

The following figure shows a detail of the lower part of the Center-Pane.

Item	TreeView	XML Filter	PDom	XML
Name	Appearance			
Value	Flat			
BaseName	Appearance			
ChildCount	0			
FullPath	Root\AllObjects\Containers\B...			
Generation	4			
Handle	1311			

The lower part of the Center-Pane has the following five tabs:

Tab	Function
<b>Item</b>	Data-entry for the currently selected node or attribute
<b>TreeView</b>	Data-entry for the properties of the view
<b>XML Filter</b>	Controls the filtering when generating XML
<b>PDom</b>	Displays some properties of the PDom
<b>XML</b>	Displays the XML representation of the currently selected node

## The Right-Pane

The **Right-Pane** contains two tabs; one displays the content of the meta table from the metas.mdb database, and the other displays templates that may be useful for the user.

The **Right-Pane** provides the same editing capabilities of the Center-Pane.

## Getting Started with XML

Trees are one of the most powerful data structures for the storage and exchange of heterogeneous information; they allow us to store data in an intrinsically structured manner, and they allow the exchange of their information preserving the structure and data relationship. One of the problems in developing applications based on tree technology has been the lack of a formal script representation of the tree information, so that we can load trees, save them, and exchange their data using a standardized convention.

Representing trees with text scripts is not a complicated subject, the technique is simple: for every node in the tree you place a pair of container braces; then inside the braces you place the node's children applying the same rule to them. Take a look at the following figure of a tree as we apply this concept:



The script representing the above tree could be written as:

```
Bookstore {  
    Book {  
        Title {}  
        Author {}  
    }  
}
```

As we can see, this type of scripting preserves the relationship among the nodes of the tree, and with it, we can duplicate the state of the tree.

When trees get larger, the scripts representing the trees become very cryptic and we lose sense of which closing brace is associated with which node. Also, when editing the script, if we miss a brace by mistake, finding the mistake is not a simple task. The above representation must be improved. It would be better if we could identify the closing braces, associating them with their opening braces by their name. It is also convenient to have some means of including node attributes within the node without affecting the node's relationship with the other nodes in the tree. To do this we could replace the name of the node and its opening brace by its name enclosed in angle brackets:

Replace	With
<b>Bookstore</b> {	< <b>Bookstore</b> >
<b>Book</b> {	< <b>Book</b> >
<b>Title</b> {	< <b>Title</b> >
<b>Author</b> {	< <b>Author</b> >

With this improvement, any attributes belonging to the node can be placed inside the bracket pair, to the right of the node's name, without affecting the tree structure. For example if we had to include two attributes in the **Book** node we could write:

**<Book Genre="Fiction" InStock="Yes">**

Now we must replace the closing brace by the name of the node enclosed in angle brackets but with a marker character indicating that it is a closing brace. For example, to replace the closing brace of the **Bookstore** node we use: **</Bookstore>**, using the "/" character as the marker to indicate that it is the closing brace of the **Bookstore** node.

Applying these changes to the scripting rules, our script expression becomes:

```

<Bookstore>
  <Book Genre="Fiction" InStock="Yes">
    <Title>< /Title>
    <Author></Author>
  </Book>
</Bookstore>

```

For nodes that don't have any content, like the **Title** and **Author** nodes in the above expression, their representation can be simplified if we combine the opening and closing components into one, using the name enclosed in brackets but placing the marker character at the end.

Replace	With
< <b>Title</b> >< / <b>Title</b> >	< <b>Title</b> / >
< <b>Author</b> ></ <b>Author</b> >	< <b>Author</b> / >

With this last change, our script expression representing the tree becomes:

```

<Bookstore>
  <Book Genre="Fiction" InStock="Yes">
    <Title/ >
    <Author/ >
  </Book>
</Bookstore>

```



What we have been scripting in blue is called an Element under XML. Every Element of the script corresponds to a node in the tree. Elements may contain attributes and more content in the form of more elements (nodes) and character data.

What we have been scripting in green is called an Attribute under XML. Attributes consist of zero or more expressions that have the name of the attribute, the equal sign, and the value of the attribute enclosed in double quotes. You may enclose attributes in single quotes also, but you cannot enclose attributes in a mix of double and single quotes.


What we have been scripting in purple is called Character Data under XML. Character Data is used to store the value of the Elements.

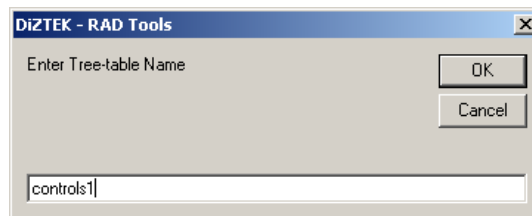
Now that we have a better understanding of how XML is used to represent the data stored in a tree, we can use XML to load and save our trees, or use it to exchange data across other mediums. As you can see, XML is just a type of script used to describe the contents and structure of hierarchical data.

## Creating Trees

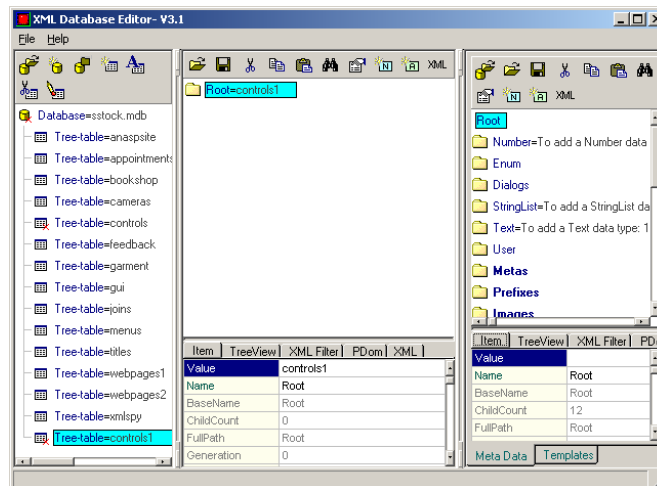
XML scripts are easy to digest if the number of elements is small. When the number of elements is large, a tool like the XDE is extremely valuable because it hides the XML and shows the true graphical representation of the data under a persistent environment. From our experience with the XDE, you only need to handle very small chunks of XML, at the editing level, in order to create large and complex tree structures and its equivalent XML representations. The following section guides you through the process of creating a typical tree by using the Cut & Paste capabilities of the XDE combined with the editing of small chunks of XML.


□ Run the XDE (XMLEDIT.EXE) and open the supplied **stock.mdb** database. The **stock.mdb** database is found in the **%XmlDatabaseInstall%** directory.

□ In the left-pane, click the  Create Tree-table button and in the displayed dialog type the name of the new tree-table. i.e.: “controls1”

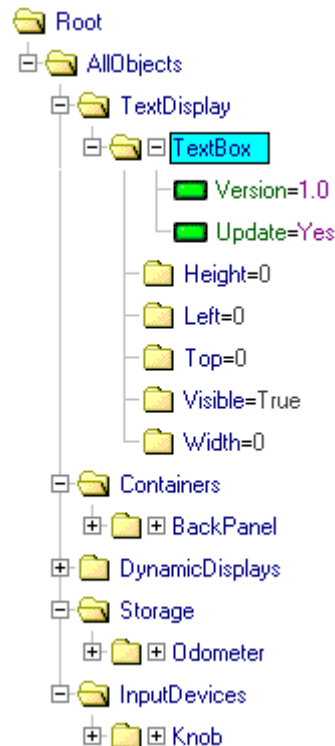


The XDE creates the new tree-table under the name specified, adds to it a node and displays the new tree-table in the center-pane, as shown in the following figure:



□ Because the following procedure requires the tree-table to be empty, we need to remove the root node from our table. Use the center-pane  Cut button to remove it.

We are going to use the XDE to create the following tree:



The tree consists of a collection of objects (like the TextBox shown above), grouped into five main categories: TextDisplay, Containers, DynamicDisplays, Storage, and InputDevices

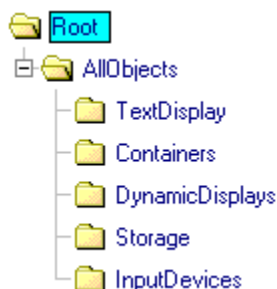
Every object has a set properties represented in the tree by a set of attributes and a set of children. The Attributes are: Version and Update. The object's properties are: Height, Left, Top, Width, and Visible.

□ To make this tree in the most efficient manner lets open Notepad and type the following basic XML representation of the root and the five main categories:

```
<Root>D:\Diztek\Controls
  <AllObjects>
    <TextDisplay/>
    <Containers/>
    <DynamicDisplays/>
    <Storage/>
    <InputDevices/>
  </AllObjects>
</Root>
```

□ Copy the above text to the clipboard.

□ In the center-pane, click the  Paste button. The XDE converts the clipboard's XML into the following persistent tree:




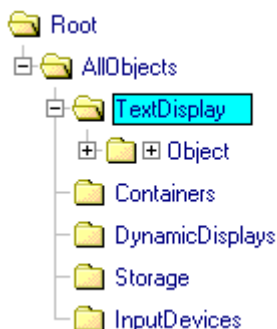
You can convert the persistent tree back into XML if you right-click the root node and select Copy.

□ Open Notepad and type the following XML representation of the basic object:

```
<Object Version="1.0" Update="Yes">
  <Height>0</Height>
  <Left>0</Left>
  <Top>0</Top>
  <Visible>True</Visible>
  <Width>0</Width>
</Object>
```

□ Copy the above text to the clipboard.

□ In the center-pane of the XDE, select the **TextDisplay** branch and click the  Paste button. The XDE adds the clipboard's XML as a branch of the selected node:

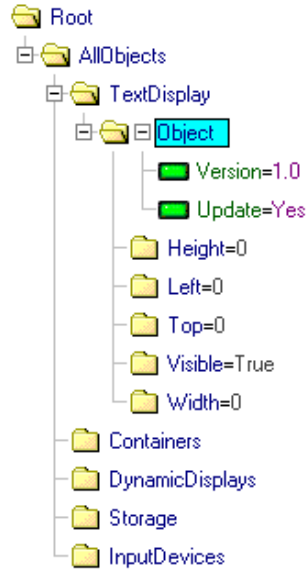


When we paste XML onto the selected node, it is expanded to show the pasted data.

Notice that the pasted object has two branches. The left branch is for its children, while the right branch is for its attributes.

This XDE is the only editor in the market that can display tree views with two branches per node.

□ Select the **Object** node and click the right [+] to expand its children and left [+] to expand its attributes, obtaining the following display:



□ Select the **Update** attribute. Notice that when you select an attribute, the BackColor of the owner node changes from blue to yellow, and the BackColor of the selected attribute changes to green.



Before we proceed to paste more copies of this basic object into our tree, we want to set the **hMeta** property for these nodes and attributes. The **hMeta** is an inherent property that determines the data-type and ensures secured data-entry on the data assigned to the value of every node or attribute. As you will see later, you can add new data-types to the XDE, without writing any code.

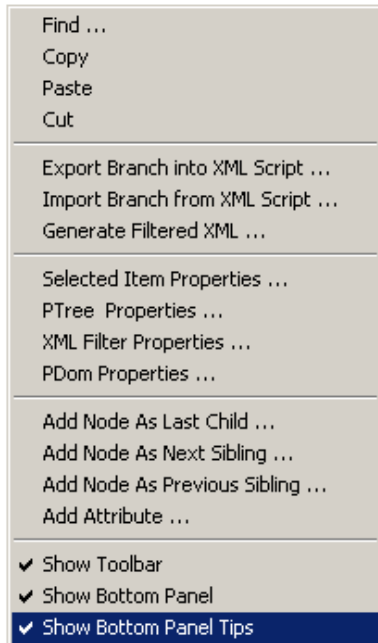
□ In the bottom part of the center-pane, select the **hMeta** property.

Item	TreeView	XML Filter	PDom	XML
Value	Yes			
Name	Update			
BaseName	Update			
FullPath	Root\AllObjects\TextDisplay\Object@Upd...			
Handle	10			
hImage	<input type="checkbox"/> (0) Default			
<b>hMeta</b>	(0) Text\Any			
hStyle	(0) Default			
NamespaceURI				
Mask	&H0			
NodeType	ATTRIBUTE_NODE			
Prefix				
Tag				

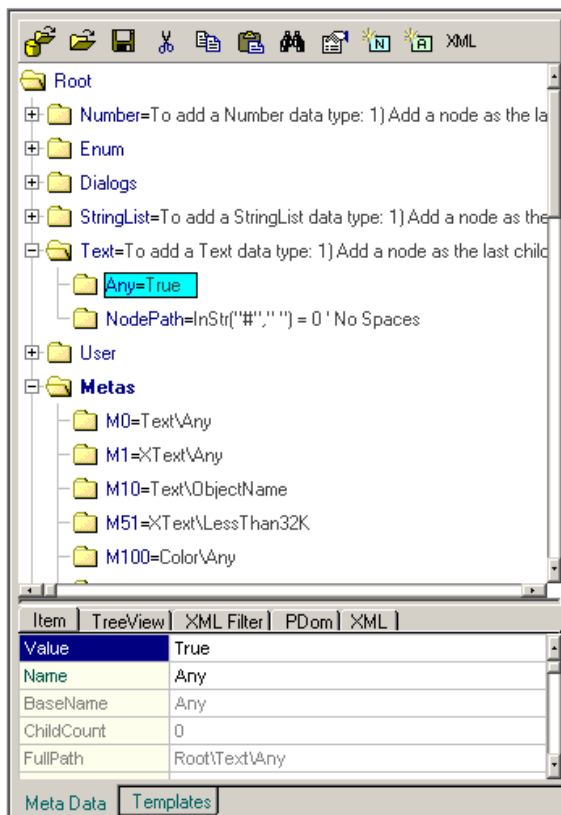
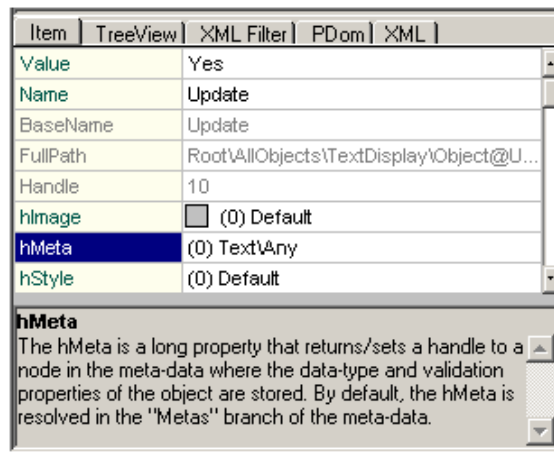
Notice that the value of this property is: **(0) Text\Any**.

The number inside the parenthesis is the value of the property and it represents a persistent handle. The XDE uses this handle to locate in the meta-data the rules that govern the data-entry. The string to the right of the handle indicates the branch in the meta-data where the rules are located.

As shown in the previous figure every attribute and every node in the tree has a set of properties that can be accessed through the [Item] tab of the bottom pane.



When the [Show Bottom Panel Tips] menu is checked, the bottom panel shows a brief description of the currently selected property.

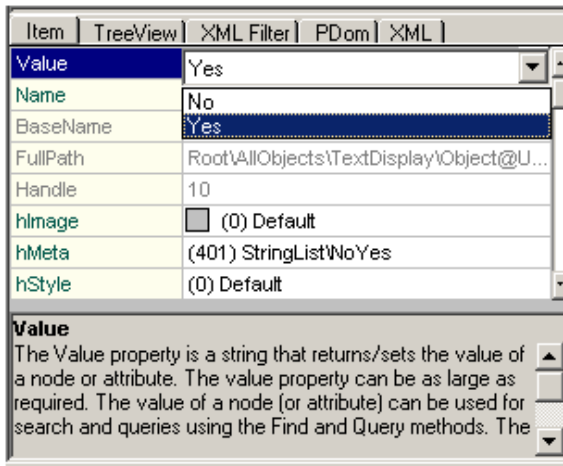


The Meta-Data Tab of the Right-Pane holds the tree containing all the information used by the XDE to control the data-entry and other image and style information.

When the XDE sees a hMeta = 0, it goes to the **Metas** branch of this pane and locates the node with the name equal to "M0" ("M" & CStr(hMeta)) and extracts the path of the tree where the rules are located: "Text\Any". The value of the node at the specified path is used as the rule governing the data-entry.

The rule contains a VBScript Boolean expression. In the case of Text/Any, the rule simply returns: True, indicating that any value entered is valid. Refer to the selected node in the left figure.

- In the bottom part of the center-pane, select the **hMeta** property and from the selection list select: **(401) StringList\NoYes**



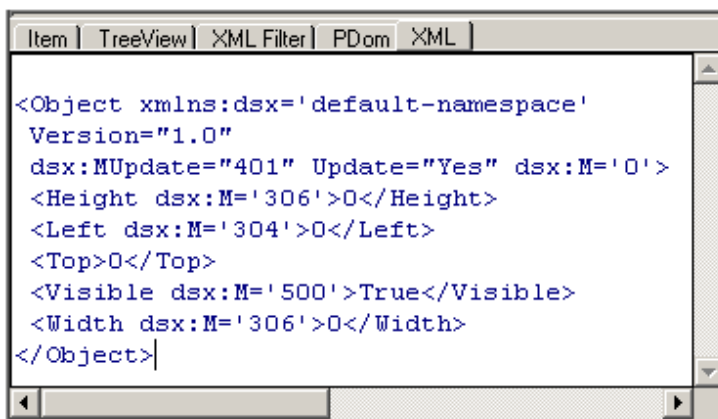
After assigning the hMeta to: **(401) StringList\NoYes**, when you try to change the Value property of this attribute it displays a list of selections, as shown in the left figure, restricting the data-entry to just the displayed list.

The hMeta only applies to the data-entry of the Value property of a node or attribute.

- Now that we have a better understanding of the **hMeta** property and its role in the data-entry process, lets set the **hMeta** property of the following nodes as shown in the next table:

Item Name	hMeta
Version	(0) Text\Any
Update	(401) StringList\NoYes
Height	(306) Number\PosInt16
Left	(304) Number\Int16
Top	(304) Number\Int16
Visible	(500) Enum\Boolean
Width	(306) Number\PosInt16

- Select the **Object** node and in the bottom pane select the XML tab:



When you click the XML tab, the XDE displays the XML representing the currently selected node. The XML generated is controlled by a filter, which allows you to remove specific nodes and attributes.

You will learn later how to use the XML Filter.

Refer to the previous figure. Notice that the XML includes now the following namespace declaration:

```
xmlns:dsx='default-namespace'
```

The XDE uses the **dsx** namespace to prefix the names of all the attributes that represent the inherent properties of a node or attribute.

Also, notice the inclusion of the **dsx:M** attribute in every one of the nodes. i.e.:

```
dsx:M='306'
```


The XDE uses the **dsx:M** attribute to include in the XML representation the value of the **hMeta** property. The XDE uses the same technique to include the values of the **hStyle** (dsx:S), **hImage** (dsx:I), and the **Mask** (dsx:M).

Notice, that the **dsx:M** attribute for the **Top** node is not present in the XML generated. The XDE excludes this **dsx:M** attribute because it uses delta compression in the XML generation to exclude **dsx** attributes, with equal name and values, detected in the document order. The XDE also excludes **dsx** attributes with the values equal to zero.

You don't have to worry about the **dsx** attributes; the XDE does not display them in the view, and you can filter them out from your XML by using the XML filter. At the persistent level, these attributes only take the space required to store a long data type. When doing XML creation and editing it is advisable to include the **dsx** attributes in your Copy & Paste operations, so you can transfer all the inherent properties and benefit from the data-entry protection and features that these attributes bring to your XML management. Also, future versions of the XDE will use these properties to automatically generate governing schemas.

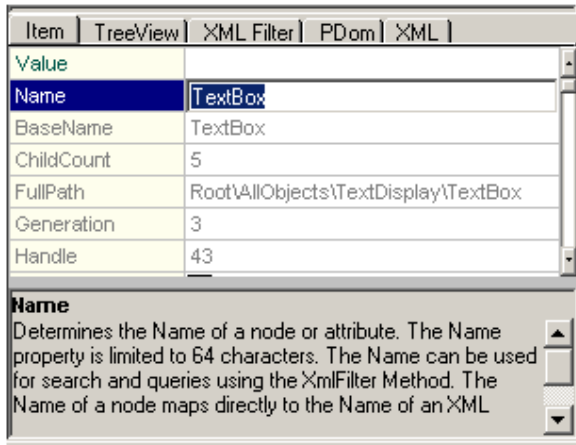
Now that our basic object has all the properties set, we can use it to create copies of it, minimizing the amount of work required in the creation of the tree. Lets get back to our tree creation.

Select the **Object** node and click the  Copy button. This action copies the XML representation of the Object branch onto the clipboard.

Now select the **Containers** node and click the  Paste button. Repeat this operation for the **DynamicDisplays**, **Storage** and **InputDevices** nodes. At this point every one of the object categories has an object.

We can now proceed to customize every one of these objects by changing its name.

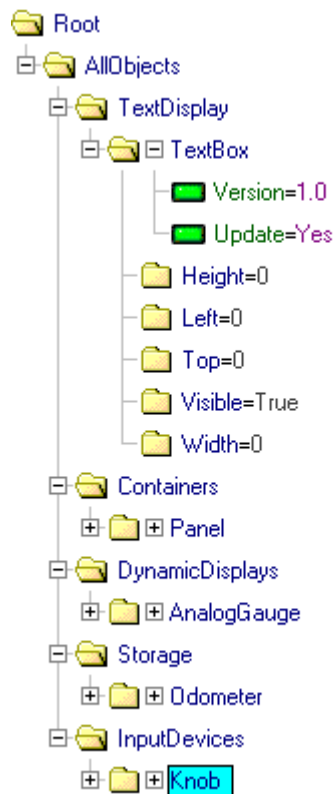
- Select the **Object** node that is under the **TextDisplay** category, and in the [Item] tab of the bottom pane click the **Name** property and change its name to: **TextBox**, as shown in the following figure:



Repeat the same operation and rename the other objects as follows:

Category	Child Object Name
Containers	Panel
DynamicDisplays	AnalogGauge
Storage	Odometer
InputDevices	Knob

The final tree should look like the following figure:



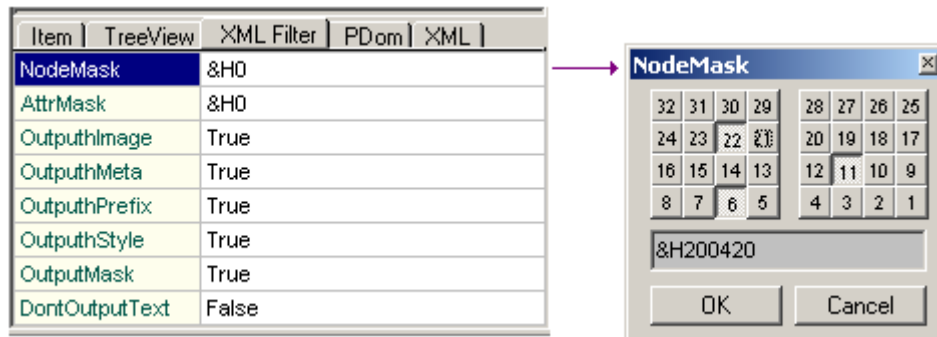
## XML Filter

The XML Filter is a tool that provides the capabilities to exclude nodes, attributes, and properties from the XML generated by the XDE without disturbing the stored data. The XML Filter controls the generation of XML in the following areas:

1. Cut & Paste
2. Export Branch into XML Script

When generating XML, it is important to have the capability to exclude specific nodes and attributes according to certain application requirements. For example, when doing Cut & Paste operations node properties like the **hMeta** or the **hImage** are included in the XML generated as namespaced attributes to prevent data-entry and view information loss, but in many circumstances you may want to exclude this information.

The XML Filter is available in the bottom section of the center and right panes:



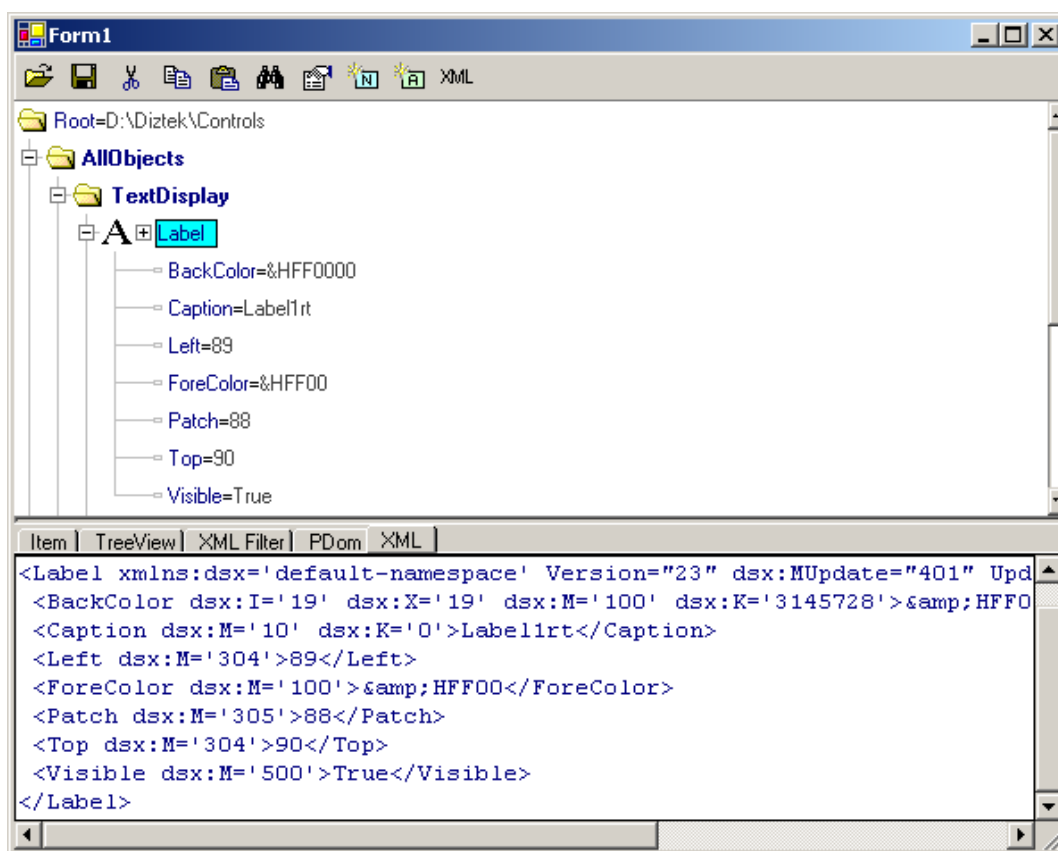
The **NodeMask** property works in concert with the **Mask** property of every node. The **Mask** property is a bit mask property with a length of 32 bits. If a bit (set to 1) in the **Mask** property of a node matches a bit in the **NodeMask** parameter, then the node is excluded from the XML generated. Excluding a node forces the exclusion of all its descendants.

Likewise, the **AttrMask** property works in concert with the **Mask** property of every attribute. If a bit (set to 1) in the **Mask** property of an attribute matches a bit in the **AttrMask** parameter, then the attribute is excluded from the XML generated.

The other XML Filter properties are used to force the output of node properties as attributes of a node in the XML generated.

Property	Comments	Attribute Name
OutputImage	If set, includes the <b>hImage</b> property as an attribute with the name: <b>dsx:I</b> ; and <b>hImageExp</b> with the name: <b>dsx:X</b> ; where <b>dsx</b> is the PDOM namespace prefix.	<b>dsx:I</b> <b>dsx:X</b>
OutputMeta	If set, includes the <b>hMeta</b> property as an attribute.	<b>dsx:M</b>
OutputPrefix	If set, includes the <b>hPrefix</b> property as an attribute.	<b>dsx:P</b>
OutputStyle	If set, includes the <b>hStyle</b> property as an attribute.	<b>dsx:S</b>
OutputMask	If set, includes the <b>Mask</b> property as an attribute.	<b>dsx:K</b>

□ To exercise the filter: open the **controls** table from the **sstock.mdb** (The **stock.mdb** database is found in the **%XmlDatabaseInstall%** directory) and select the **Label** node until you get a display similar to the one shown below. Then click the XML Tab in the bottom pane.



When you click the XML Tab, the XDE generates the XML for the currently selected node and displays it in the bottom pane. In the displayed XML, notice the attributes with the **dsx** namespace prefix; these attributes carry the **hImage**, **hStyle**, **hMeta**, and other properties of the node (or attribute). The XML generated provides no loss of information with respect to the properties of nodes, and attributes. If you paste this XML information into any branch of the tree, you will have an exact duplicate of the **Label** branch with all the color, image and all other properties included.

□ Lets exclude the properties namespaced with **dsx** from the XML generated. To do this, click the XML Filter tab and change the properties to match the following settings:

Item	TreeView	XML Filter	PDom	XML
NodeMask		&H0		
AttrMask		&H0		
OutputImage		False		
OutputMeta		False		
OutputPrefix		False		
OutputStyle		False		
OutputMask		False		
DontOutputText		False		

□ Now click the XML tab to generate the XML with the new filter settings.

```

<Label Version="23" Update="Yes">
  <BackColor>&HFF0000</BackColor>
  <Caption>Label1rt</Caption>
  <Left>89</Left>
  <ForeColor>&HFF00</ForeColor>
  <Patch>88</Patch>
  <Top>90</Top>
  <Visible>True</Visible>
</Label>

```

Notice that under this filter setting, the XML does not include any of the attributes with the **dsx** namespace prefix. If you paste this XML into any branch of the tree, you will have a duplicate of the **Label** branch, but without the image and other properties of the original.

□ Now lets exclude the **ForeColor** and **Patch** nodes from the XML generated. To do this we must set a bit in the **Mask** property of these objects

and then set the corresponding bit in the **NodeMask** in the filter. In this sample we used the first bit, but you are free to use any of the bits.

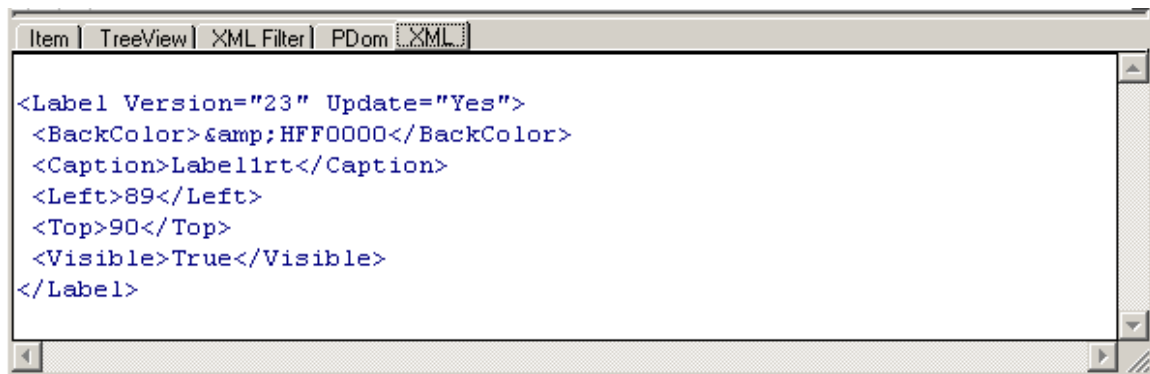
□ In the view, click the **ForeColor** node to select it and then click the Item tab in the bottom pane to select its properties. Change the **Mask** property of the node to &H1.

□ In the view, click the **Patch** node to select it and in the bottom properties change the **Mask** property of the node to &H1.

□ Now click the XML Filter Tab and change the properties to match the following settings:

Item	TreeView	XML Filter	PDom	XML
NodeMask		&H1		
AttrMask		&H0		
OutputImage		False		
OutputMeta		False		
OutputPrefix		False		
OutputStyle		False		
OutputMask		False		
DontOutputText		False		

□ In the view, click the **Label** node to select it and then click the XML tab to generate the XML with the new filter settings.



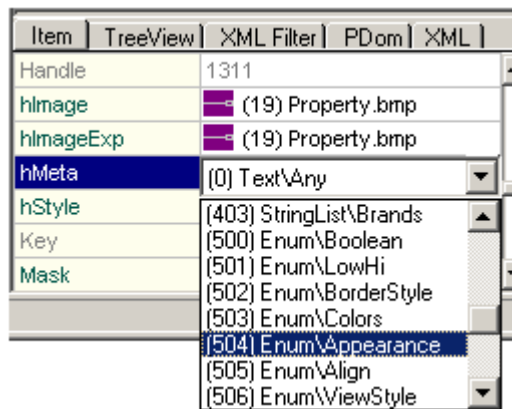
```
<Label Version="23" Update="Yes">
  <BackColor>&HFF0000</BackColor>
  <Caption>Label1rt</Caption>
  <Left>89</Left>
  <Top>90</Top>
  <Visible>True</Visible>
</Label>
```

Notice that the XML does not include the **ForeColor** or **Patch** nodes or any of the attributes with the **dsx** namespace prefix.

## How To ...

### How to Assign Data-Types

- Select a node or attribute.
- Select **[Item][hMeta]** and pick the required data-type from the selection list.



After making the selection the data-entry for the **Value** property of the selected item changes accordingly.

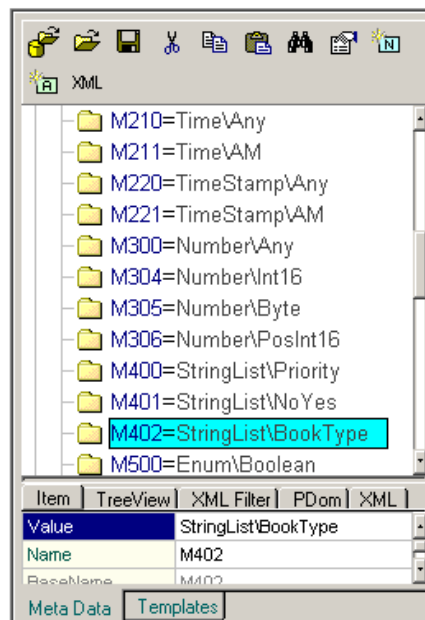
Note: The XDE builds the list of data-types from the **Metas** branch of the **[Right-Pane][Meta-data]**.

## How to Add Data-Types

In this example we will add a new StringList data-type called **Brands**. Nodes or attributes with the **hMeta** property set to this new data-type will accept one of the following selections:

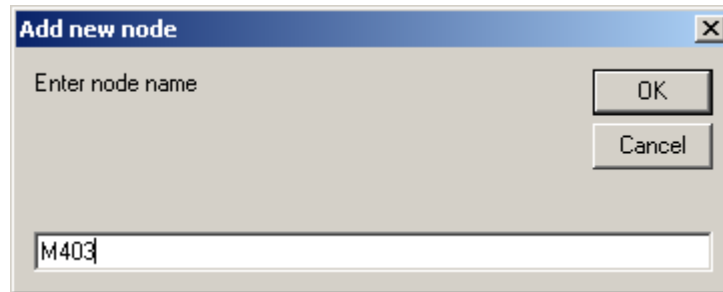
1. "AMD"
2. "Apple"
3. "Compaq"
4. "Dell"
5. "Gateway"
6. "Hewlett-Packard"
7. "IBM"
8. "Intel"
9. "Sony"

□ In the **[Right-pane][Meta-data]** select the **Metas** branch, expand it and select the last node of the StringList section: "StringList\BookType"; we want to add a new node as a sibling of this node. We do this so that all the StringList types are grouped together. Right-click the "StringList\BookType" node and in the displayed menu select **[Add Node As Next Sibling...]**

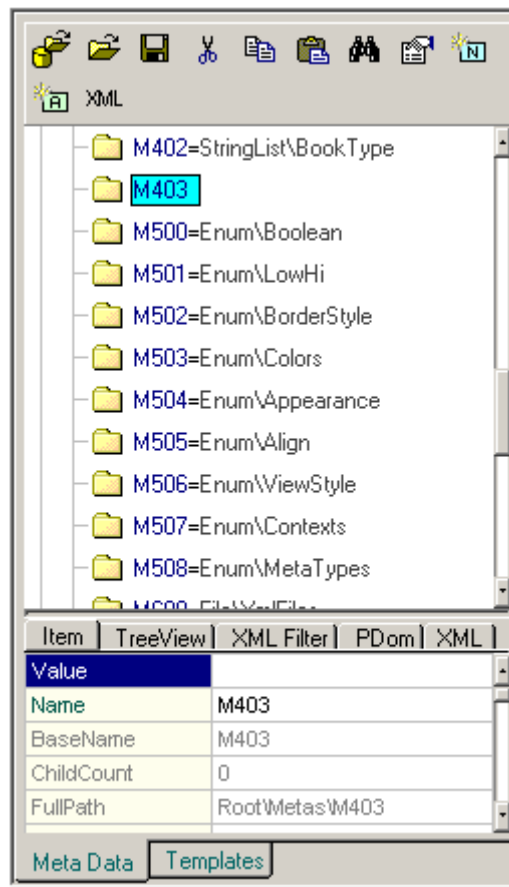


□ In the displayed dialog type the name of the new node. The name must start with the letter "M" followed by a number. This number is the handle that will be associated with the data-type being added. You can use any number different

from 0, and that has not been assigned to other data-type; this number must be unique among the children of the Metas branch. For this example enter: M403



The new node is created and selected.



□ In the **[Right-pane][Meta-data][Item][Value]** enter: “StringList\Brands”.


The string assigned to the value of the “M403” node, is a node path that determines the location in the meta-data where the new Brands data-type is stored. For this example, the node path is:

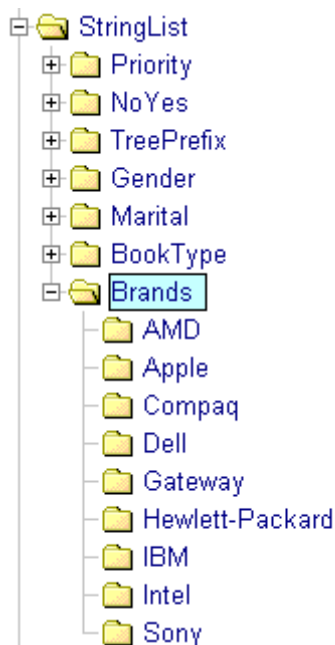
## “StringList\Brands”

This node path tells us that the **Brands** data-type is a child of the **StringList** branch and the validation processors will use this path to locate more information about this type.

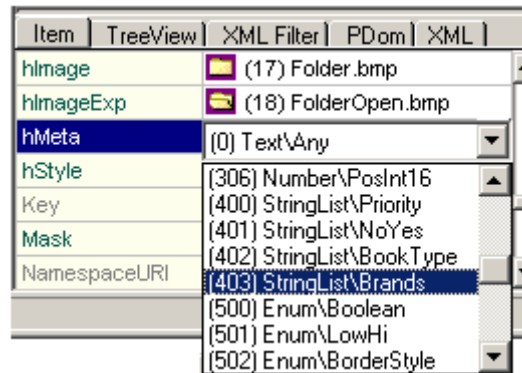
- Now we have to add the **Brands** node and its list to the **StringList** branch. In the **[Right-pane][Meta-data]** select the **StringList** branch.
- Open Notepad and type the following XML representation of the **Brands** list:

```
<Brands>
  <AMD/>
  <Apple/>
  <Compaq/>
  <Dell/>
  <Gateway/>
  <Hewlett-Packard/>
  <IBM/>
  <Intel/>
  <Sony/>
</Brands>
```

- Copy the above text to the clipboard.
- In the **[Right-pane][Meta-data][StringList]** click the  Paste button. The XDE adds the clipboard's XML as a branch of the selected node. Click the **Brands** node and expand it.

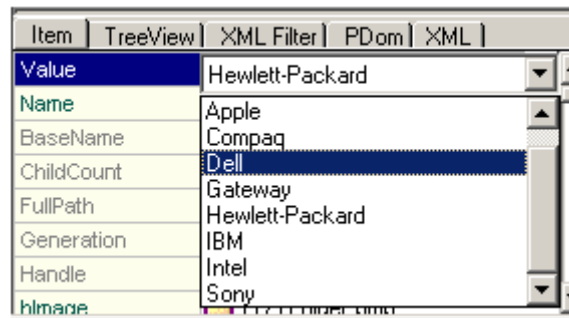


□ To test this, select a node in the Center-pane, and then select **[Center-pane][Item][hMeta]** to change it. Notice that the list of data-types now includes the data-type just added.



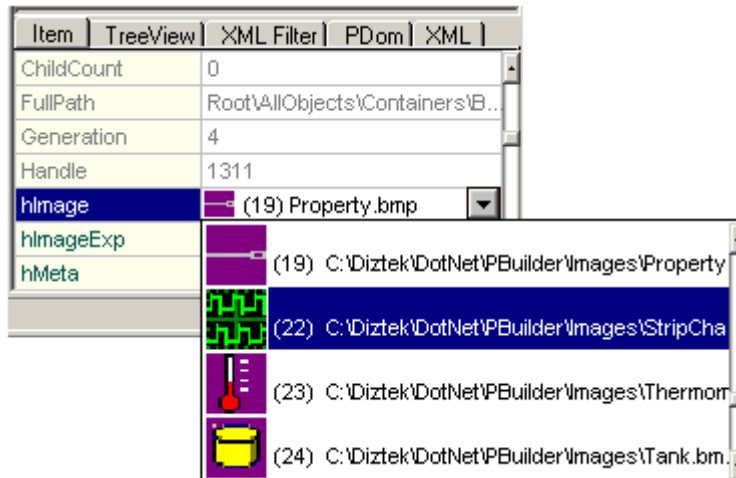
□ From now on, the data-entry performed on any node or attribute with the **hMeta** property set to: **(403) StringList\Brands**, will display a list containing the different types of brands.

Change the **hMeta** property of a node to this data-type, and then change the value of the node. Notice that the value displays for selection the **Brands** list, as shown in the following figure:



## How to Assign Images

- Select a node or attribute.
- Select **[Item][hImage]** and pick the required image from the selection list.



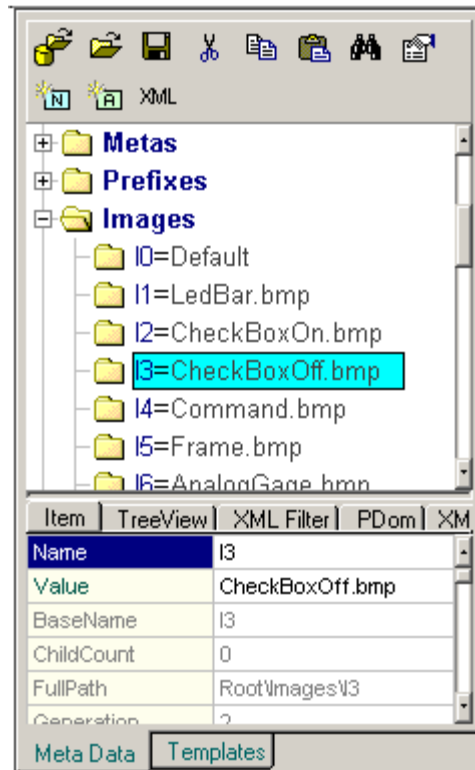
After making the selection, notice that the image of the selected item changes accordingly.

Note: The XDE builds the list of images from the **Images** branch of the **[Right-Pane][Meta-data]**.

Note: The BackColor of some of the images is purple because it is used as the transparent color when painting the image.

## How to Add Images

□ Select **[Right-pane][Meta-data]** and then select the **Images** branch, expand it and then select any one of the child nodes. For example, select the “I3” node. Right-click the node and select **[Copy]** in the displayed menu (you can also use Ctrl-C). The selected node is now copied to the clipboard.



□ Now select the **Images** branch, then right-click and select **[Paste]** from the displayed menu (also you can use Ctrl-V). The node is now added as a child of the **Images** branch.

Note: The reason we created the node through Copy & Paste was to retain all the properties of an image node in the new node. The image node we selected has the hMeta property already set to “Files\Bitmaps” and we need this data-entry type to select the location of the image file.

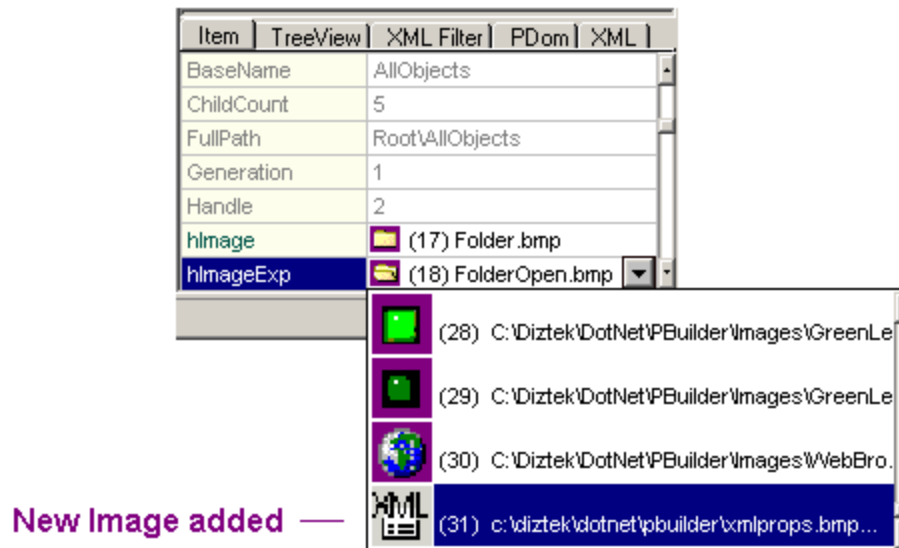
□ Now scroll down the view and select the node just added.

□ Select **[Right-pane][Meta-data][Item][Name]** and change it to the letter “I” followed by a number. This number is the handle that will be associated with the image being added. The name just given to this node must be unique among all the names of the children of the **Images** branch. For this example enter: I31

□ Select **[Right-pane][Meta-data][Item][Value]** and from the displayed dialog, find the location and select the image file.

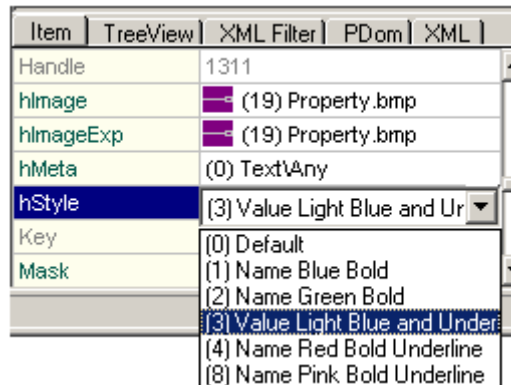
□ From now on, any node or attribute with a **hImage** property set to 31 will use this image for its display.

□ To test this, select a node in the Center-pane and then select **[Center-pane][Item][hImage]** to change it. Notice that the list of images includes now the image just added.



## How to Assign Styles

- Select a node or attribute.
- Select **[Item][hStyle]** and pick the required style from the selection list.

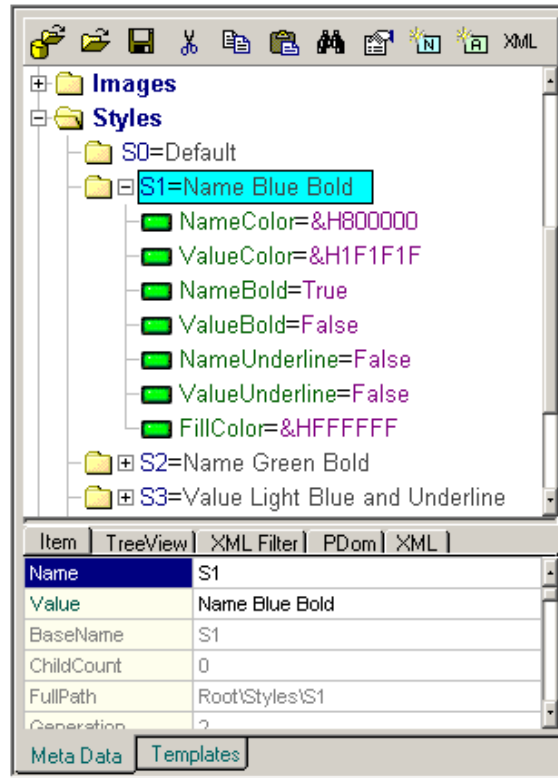


After making the selection, notice that the style of the selected item changes accordingly

Note: The XDE builds the list of styles from the **Styles** branch of the **[Right-Pane][Meta-data]**.

## How to Add Styles

□ Select **[Right-pane][Meta-data]** and then select the **Styles** branch; expand it and then select a style that resembles the new style to be added. For example, select the **S1** node. Right-click the node and select **[Copy]** from the displayed menu (you can also use Ctrl-C). The selected node branch is now copied to the clipboard.



□ Now select the **Styles** branch, right-click and select **[Paste]** from the displayed menu (you can also use Ctrl-V). A new node branch is now added as a child of the **Styles** branch.

Note: The reason we created the node through Copy & Paste was to retain in the new node all the properties and descendants of an existent style node.

□ Now scroll down the view and select the node just added.

□ Select **[Right-pane][Meta-data][Item][Name]** and change it to the letter “S” followed by a number. This number is the handle that will be associated with the style being added. The name just given to this node must be unique among all the names of the children of the **Styles** branch. For this example enter: S1000

□ Select **[Right-pane][Meta-data][Item][Value]** and enter the name you want to give to the new style. In this example we used: “Name Green Bold”. This value is used only to provide some idea of the changes inflicted by the style.

□ Now we must edit the values of the properties of the new style branch. Expand the attributes branch of the style node just added. Select the NameColor attribute. Change the Value of this attribute to green.

□ From now on, any node or attribute with the hStyle set to 1000 will be displayed with its name bold and green.

□ To test this, select a node in the Center-pane and then select **[Center-pane][Item][hStyle]** to change it. Notice that the list of styles now includes the style just added.

The following Style attributes are supported:

Attribute	Type	hMeta
NameColor	Long	(100) Color\Any
NameBold	Boolean	(500) Enum\Boolean
NameUnderline	Boolean	(500) Enum\Boolean
ValueColor	Long	(100) Color\Any
ValueBold	Boolean	(500) Enum\Boolean
ValueUnderline	Boolean	(500) Enum\Boolean
FillColor	Long	(100) Color\Any