EMTPWorks - Overview of New Customization Features - *Draft*

Last updated: 2018-05-26 by Chris Dewhurst

1. Overview

This document provides an overview and reference for the new customization features in EMTPWorks 3.0. These features provide a great deal of control over the appearance and functionality of the package without having to modify any internal program code.

The features described here include:

1. Customizable “start” page, including:
    - Document templates
    - Specialized actions that the user can take
    - Example files with description
    - Custom controls
2. Customizable symbol library listing
3. Custom controls on the application “ribbon” and in popup menus
4. Adding custom responses to user actions such as control clicks and value changes
5. Displaying output in a rich text console
6. Displaying output in an HTML (Internet Explorer) panel
7. Customizable colours for user interface elements

In additional to the new features mentioned above, some features are included in this release on an experimental basis. These are not intended for end users but for test and evaluation and possible inclusion in a future release.

8. Customizable properties palette, configured using an XML schema
9. Customizable properties list showing properties of multiple objects in list format

2. Start Page Customization

The Start page is intended to give to user direct access to his recently-used designs and files and to examples, help and common program functions.

Open recent file

The maximum number of recent files display in the MRU (Most Recently Used) list can be specified in the INI file's System section as follows:

```
[System]
MRUMaxDocs = 20
```

Example Files

The Examples listing is a hierarchical (tree) control giving direct access to any document files in the specified directories. Note the following characteristics of this listing.

Listing Format

The Examples file listing is hierarchical, i.e. it will start from the top level directory provided and display another level in the tree for each sub-directory found. Any number of top level Examples folders can be specified in the INI file (or in the INI file associated with any toolbox), as follows:

```
[System]
Examples = %application%\Examples
```

Example File Extensions

The Examples tree will by default display all files in the specified folders. This can be overridden by specifying extensions of the desired files in the INI file as follows:

```
[System]
ExamplesExts = ecf,dwj
```

Descriptions

The application attempts to display a description of the file in the adjacent box when an item is selected in the Examples tree. This description is derived as follows:

- For design files, the file is opened and read until a design attribute field Description.Cct is located
- For text files, the first few lines are read, excess white space is removed

New Default Design

This control allows the user to create an empty document (usually a design file, but can be any file type) with a single button click. The new document is created by opening a specified template file and setting it to be a new document, that is, the association with the source file is removed. The template file can be any supported document file type, so a design with any initial settings and even initial design elements can be a template.

The default template file can be specified by either of the following:

- The path to the document template can be specified in the INI file as in:

```
[System]
DefaultTemplate = Templates\Simple Design.ecf
```

- The user can select a default template from the list of available templates by clicking the Set As Default Template button on the Start page. If this has been done, this setting overrides the DefaultTemplate keyword in the INI file. This information is stored in the application state file.

New Document from Template

This set of controls allows the user to select from any number of installed template files and create a new document based on the template. The new document is created by simply reading the template file and then removing any association to the original file, so that a Save command will prompt for a new location. Therefore, the template can contain any desired settings or content to create a starting point for a new document.

The displayed list shows all files located in all Template folders specified in the INI file, and in any toolbox INI files. The Template folder is specified as follows:

```
[System]
Templates = Design Templates
Templates = Other Templates
```

Other Tasks (NOT IMPLEMENTED IN THIS VERSION)

The Other Tasks section of the Start page gives the user direct access to program capabilities not associated with opening or creating a document. These may be maintenance tasks like backing up or access to non-document functions like creating a new device symbol.

The Other Tasks list is actually an alternate form of custom menu and is created by specifying an action item in the INI file. An example is given here and see Adding Custom Controls and Menu Items below for more details.

```
menu= "Start.Tasks:Subcircuit Wizard", "Design:SubcctWizard", "", "Open the
   subcircuit wizard, allowing you to quickly create a new subcircuit by
   specifying its connections and/or internal circuit"
```

In this case, the “start.tasks:” portion of the menu specifier indicates that this item will be located in the Other Tasks list. The second item, “Design:SubcctWizard”, specifies and action to take. In this case, it is associating with an internal action, but it could also execute a script or send a COM message to another application.

3. Customizable symbol library listing

The symbol palette displays the contents of parts libraries allowing the user to select items for placement in the design. The format of each element on the list is determined by a schema file. This is an XML text file that specifies which elements of part information, e.g. symbol name, library name, number of pins, symbol, attribute values, etc.) should be displayed.

Schema location

The location of the schema file is determined by an INI file entry like this:

```
[Libraries]
SymPanelSchema = Property Schemas\emtp_lib_sym_overview.xml
```

The schema will normally be located in a subfolder inside the program folder but the usual rules apply for locating files given a partial path.

Schema format

A typical symbol library format schema looks like this:

```
<schema Name="PartTypeOverview">
<unitsschema Name="emtp_units"/>
<element Name="Symbol" DisplayName="Symbol" Type="bitmap" DefaultValue=""
   Description="Name of the source library file" AltText="Device Symbol"
   Category="Main" Source="$SYMBOL" />'
<element Name="Name" DisplayName="Name" Type="string" DefaultValue=""
   Description="Name of this part type" Category="Main" Source="$TYPENAME" />'
<element Name="LibName" DisplayName="Library Name" Type="string" DefaultValue=""
   Description="Name of the source library file" Category="Main" Source="LibName"
   />'
</schema>
```

Available keywords are:

$SYMBOL	The graphic of the symbol itself

$TYPENAME	The type name, as used to store the symbol in the library
$NUMPINS	The number of pins defined on the symbol
&attrName or simply AttrName	The contents of the given attribute field

4. Adding Custom Controls and Menu Items

This section describes how you can use INI file entries to add custom controls to the application menu bar (ribbon) or to various context menus used in the application. Controls added to the ribbon can be simple buttons, popup menus, checkboxes or text entry boxes and can be added in a large or small format.

INI file format for custom controls

Here is an example of adding a toolbar item from the INI file:

```
menu= "Simulate\Advanced\Simulation Options...",
      "info scripts\options\sim_options.dwj",
      "Set EMTP simulation options",
      "Toolbar Icons\pencil16.png",
      button_small, $DESIGNPATH
```

NOTES:

- The entire specification must be on one line in the INI file
- Fields in this specification are separated by commas
- Fields should be surrounded by double quote marks if they may contain blanks or commas

Format specification:

```
menu = menuDesc, action, eventName, tipMsg, iconPath, formatOptions, arguments
```

menuDesc

This provides a hierarchical description of the menu item. The possible formats are:

tabName\groupName\item	specifies a standard button on the toolbar
tabName\groupName\title\subItem	specifies a menu item within a popup menu on the toolbar
siteName:item	specifies a menu item in a popup context menu
siteName:item\subItem	specifies a sub-item in a popup context menu

In the latter two examples, siteName can be one of

Device.Popup	
Signal.Popup	
Pin.Popup	
Circuit.Popup	
Start.Tasks	

action

To be added

eventName

The event name is used to associate a name with the control and the connected action. This is required only if it is desired to set the state of the control or respond to the event using scripting. In most cases it can be omitted, or specified as empty quotes “”.

tipMsg

This text is displayed in “tool tips” when the user hovers the mouse over the associated control and is used to provide additional information on the function of the control

iconPath

This specifies the location of an icon file used to create the control. Most graphic file types are accepted but PNG format is preferred.

formatOptions

Format options indicate the type and size of the control. They can be specified either as a numerical flag value or any one of a number of keywords.

Keyword	**Numerical Equivalent**	**Description**
button_small	1	A small toolbar button, displayed in a stacking panel with (typically) 3 buttons stacked vertically
button_large	0x101	A large toolbar button, displayed individually on the toolbar
checkbox_small	2	A small checkbox button, displayed in a

		stacking panel with (typically) 3 buttons stacked vertically
checkbox_large	0x102	A large checkbox button, displayed individually on the toolbar
popup_small	3	A small popup button (button displaying a child menu), displayed in a stacking panel with (typically) 3 buttons stacked vertically
popup_large	0x103	A large popup button (button displaying a child menu), displayed individually on the toolbar
textbox_small	5	A small text edit box
textbox_large	0x105	A large text edit box
separator_small	4	A separator line between small items
separator_large	0x104	A separator line between large items
label_small	6	A static text label in small (stacked) format. NOTE: This control type has a number of display and format options that make it useful for a variety of information and status display purposes. See below for more details.
label_large	0x106	A static text label in large format. See note below about special capabilities of the type of control
group	8	Used to specify the icon for a group. This is only required in specific instances. See below for more details.

arguments

The arguments are optional and are passed to the script when the control is activated. These can be used, for example, to indicate what type of action the script should take so that a single script can be shared by multiple controls.

Arguments can also contain keywords that will be replaced by design information when the command is executed. For these keywords, see Argument Variables below.

Associating Ribbon Menu Items with Document Types

Each document editor has a document type associated with it. This type is denoted by a string, usually the same as the file extension of the associated document file type. E.g. JavaScript files have document type “dwj” and design files have type “ecf”. Device symbol editor windows have document type “sym” even though there is no associated file type.

Custom ribbon tabs added to the main application ribbon can have a document type associated with them and will appear only when that type of document is open. In addition, two special document types can be associated with a ribbon:

- “any” – means the ribbon tab will be visible for any open document, but not when no document is open
- “none” – means the ribbon tab will be visible when no document is open

If adding custom menu items causes a new ribbon tab to be created and no document type is specified, both “any” and “none” will be assumed, in other words, the tab will be always visible.

To change this behaviour, one or more document types can be associated with a ribbon tab by using an INI entry such as:

```
menuDocType= Design, ecf
```

Note these points:

- The first argument is the ribbon tab title, i.e. the top-level menu name used in creating the custom menu item
- You can specify any number of document types, separated by commas, e.g.

  ```
  menuDocType= Design, ecf, none
  ```

  will cause the menu to appear for any design file or if no file is open
- The ***menuDocType*** item must appear BEFORE any ***menu*** lines that cause the ribbon tab to be created.

Responding to Text Edit Box Entry

To Be Added

Special Capabilities of Label (Text Block) Controls

The label (static text) control type, created using the label_small or label_large format types, has a number of special capabilities which make it useful for a variety of status and information display purposes. These features include:

- the text label can be changed dynamically using the uiSetControlText script method
- the control generates a “clicked” event which can be used to run a script or other action in response to user input, like a button
- the size and alignment of the control can be specified to give it a specific look on the screen
- the background can be specified as any solid colour or any bitmap image
- any number of alternate backgrounds can be specified, each of which can be a bitmap or a colour. Each background is considered to be one “state” and is assigned a number in any order convenient. The state can be selected at any time using the script method uiSetControlState

Specifying Label Format and Images

The INI file format for a simple label is similar to other controls and looks like this:

```
menu= "TabName\GroupName\Label Text", "action", "label_name", "description text",
   "Icon path or format options", label_large
```

The difference in this case is in the field normally used for the icon path. You can still specify a simple icon path if desired, or you can provide a number of data fields as in this example (note that all this must be on one line in the INI file):

```
menu= "Design\Test\Lightbulb Test!", "", "lightbulb", "This is a test of a static
   text item", "font='aharoni 14 red' alignment=center width=100 padding=4
   background='Toolbar Icons\lightbulb_state0.png' foreground=red state0='Toolbar
   Icons\lightbulb_state0.png' state1='Toolbar Icons\lightbulb_state2.png'
   state2='Toolbar Icons\lightbulb_state1.png'", label_large
```

In other words, in place of a single path, you can have a number of fields of the form Name=Value or Name=’Value’. The possible fields are:

Field Name	Usage
`width`	The fixed width in pixels of the label control. If this is not specified, the control will adjust automatically to text and image changes, which may not be the desired effect
`font`	The font face, size. colour and options (bold/italic/undereline)
`background`	The background colour or bitmap image for the label
`foreground`	The foreground colour for the label
`margin`	The margin, i.e. the space around the label relative to other controls. Can be specified as a single integer, which will apply to all sides, or as left,top,right,bottom
`padding`	The padding area, that is the amount by which the control is expanded beyond its text content. This is specified the same way as margin above
`alignment`	Can be left, center, right or justify
`stateN`	Any number of state items can be specified, each one specifies the background of the

	control that will be applied when the uiSetControlState is called on this control. N can be any non-negative integer and does not have to be sequential

Responding to Clicks on a Label

action= control.clicked:lightbulb, "test items\handle lightbulb.dwj", DOWN

Specifying an Icon for a Control Group

To be added

5. Responding to Events with Actions

EMTPWorks provides an Event/Action mechanism that allows you to customize the way the program responds to user events (such as pressing a button on the toolbar) or program-generated events (like a message being received from another application).

The INI file “action” specification provides a way of causing an action to be taken (e.g. running a script) in response to one of these events.

Events

To be added

NOTE: Event names are not case sensitive

Application Events

`Application.Suspend`

`Application.Resume`

`Application.ExitRequest`

`Application.Exit`

`GlobalVar.Changed:`*`varName`*

`Control.Clicked:`*`ctrlName`*

Design Events

`Design.New`

`Design.Open`

`Design.Close`

`Design.Pack`

`Design.Save`

`Design.Saveas`

`Design.Revert`

`Design.Structchanged`

`Design.Selectchanged`

`Design.Objectselectchanged`

`Device.FormOpen`

`Circuit.Instanceopened`

`Attribute.Changed:`*`attrName`*

Symbol Library Events

Event Name	**Arguments…**			
`SymbolLib.Library.Created`	Library path			
`SymbolLib.Part.Copied`	Library path	Part name	Source library path	Source part name
`SymbolLib.Part.Duplicated`	Library path	Part name	Source part name	
`SymbolLib.Part.Deleted`	Library path	Part name		
`SymbolLib.Part.Edited`	Library path	Part name		
`SymbolLib.Part.New`	Library path	Part name		
`SymbolLib.Part.Renamed`	Library path	Part name	Old part name	
`SymbolLib.Part.Replaced`	Library path	Part name		
`SymbolLib.Part.SaveFromDesign`	Library path	Part name	Source design path	Source locator

Actions

The following types of actions are available:

- Run a JScript script
- Run an Export script
- Fire an event to an external COM client
- Open or run an external file using the Windows shell

Information can be passed to the action handler in the form of text arguments. The arguments can contain literal strings specified in the INI file or variables that evaluate to the name of the current design, the selected objects, etc. These are described in more detail below.

Specifying Actions

To be added

INI file format for actions

The association between an event and the desired action is made using the “action” item in the INI file.

For example:

```
action= Control.Clicked:Button.Sim.Run, "test items\handle sim buttons.dwj", RUN
```

In this case, the triggering event is a control click on the button named “Button.Sim.Run”. The action to take is running the Jscript specified. A single argument is passed to the Jscript which is the fixed string “RUN”.

Argument Variables

A number of argument variables are available which can be used to pass information about the event to the action handler.

NOTE: Not all variables are valid in all contexts. See comments on each item.

$OBJNAME	
$OBJHIERNAME	
$OBJLOCATOR	
$INSTLOCATOR	The instance locator of the associated object
$ATTRNAME	
$ATTRVALUE	
$DESIGNPATH	
$DESIGNNAME	
&*attribueName*	

Accessing Arguments from the Script

To be added

Script methods for events and actions

The following script methods are available to assist in setting up and responding to events and actions. More information on these methods is provided in the script documentation.

Method Name	Description
uiSetControlText	To be added
uiGetControlText	To be added
uiSetControlTitle	To be added
uiGetControlTitle	To be added
uiGetControlID	To be added
uiAddControl	To be added
uiAddAction	To be added
uiSetControlEnabled	To be added
uiGetControlEnabled	To be added
uiSetControlChecked	To be added
uiGetControlChecked	To be added

Tracing events and actions

A number of keywords can be placed after an action definition in the INI file to help with verifying or debugging action definitions. The available trace commands are:

>trace	Writes the action arguments to the console before the action is performed and the action return result after the action is completed.
>console	Writes the action result to the console.
>alert	Displays the arguments and result from the action in an alert box after the action is completed.

The keyword is appended to the action description as in the following example:

```
action= Design.Open, "event>trace", Design.Open, $DESIGNPATH
```

Tracing events using the Events Panel

To assist in debugging issues with application events, an Events Panel can be displayed by selecting the corresponding button in the View tab. This panel displays all registered events and highlights the most recent one, displaying the arguments provided and the result of the action.

6. Rich Text Console Display

To be added

Console display script methods

The following script methods are available to work with the rich text console. These methods are specified in more detail in the script documentation.

Method Name	**Description**
`Write`	
`Writeln`	
`setTitle`	
`setFont`	
`saveFile`	
`clear`	
`newline`	
`newPara`	
`newTable`	
`newRow`	
`newCell`	
`closeTable`	
`newListItem`	
`closeList`	
`newHyperlink`	
`closeHyperlink`	
`setHyperlinkNormalFont`	
`setHyperlinkActiveFont`	
`savePrompt`	
`setBold`	
`setItalic`	
`setUnderline`	
`setTextSize`	
`setParaSpacing`	
`setParaMargin`	
`setReadOnly`	
`setAutoScroll`	
`setTextColour`	
`setBackColour`	

7. Using the HTML (IE Control) Panel

To be added

8. Setting Colours of User Interface Elements

Some user interface elements have configurable background colours that can be set in the INI file, as in this example:

```
[System]
DocAreaBackground=Green,Orange,Cyan,Magenta,Blue
ToolBackground=Azure,LightGray
StatusBarBackground=Red,Purple,Aqua,Gray,Yellow
```

If a single colour is given, the background is set to that solid colour. If multiple colours are given, the background will be a gradient that transitions through the given colours from the top left corner to the bottom right.

The elements that can accept this setting are:

Keyword	INI Section	Description
DocAreaBackground	[System]	Background of document area
ToolBackground	[System]	Background of tool panels (e.g. symbol library)
StatusBarBackground	[System]	Background of status bar area

The allowable colour names and samples are given on this web page

http://msdn.microsoft.com/en-us/library/system.windows.media.brushes.aspx

9. Miscellaneous User Interface Options

Blocking User Actions

These features allow a script or external client to temporarily block user interface operation to prevent data from being modified or other actions during external processes.

The following JavaScript/COM method allows general blocking of the user interface:

void **uiSetBlocked**(block, timeoutMillis);

Parameters		
Name	**Expected Type**	**Description**
block	Bool	true to disable all user interface controls and document editing actions, false to re-enable
timeoutMillis	int	A timeout until the user interface is unblocked automatically, in milliseconds. If < 0 or not specified, no automatic unblocking action is taken. Ignored if block is false.

The following INI file keywords allow specifying the initial blocked state at application startup:

```
UIBlockedInitially = true          // User controls disabled initially
UIBlockedTimeout = 10000           // Milliseconds until automatic unblock
```

10. Connecting to EMTPWorks Using ActiveX

Overview

To be added

Connecting Using ProgIDs and CLSIDs

An external client application uses a unique identifier to connect to the various resources in EMTPWorks. To cater to various application requirements, there are a number of different types of IDs with different operating characteristics.

Version-Specific IDs

All objects accessible by COM (e.g. the BrowserPanel and Application objects) have 3 CLSIDs associated with them

- a multi-use id (allows any number of clients to connect to the same instance)
- a single-use id (which guarantees you always get a new instance)
- a private id (which allows you to connect to a specific instance).

These are described below. These IDs are always available and allow EMTPWorks to be compatible with any existing callers as well as providing unique connections to a specific instance.

The Multiple-use CLSID

Using this ID, any number of callers can connect to the same instance of the program. If EMTPWorks is not running, using this CLSID will cause it to be started. If it is already running, this CLSID will connect to an existing instance. If multiple copies are running, there is no way of choosing any specific instance, the system will choose one at random.

- *See the ProgID and CLSID Table below for specific IDs.*

The Single-use CLSID

Using this CLSID will always cause a new instance of the program to start. If the user started EMTPWorks directly, you cannot connect to it with this id, you will get a new instance. These ids are registered with the system as follows:

- *See the ProgID and CLSID Table below for specific IDs.*

The Private CLSID

A unique, private CLSID is associated with each running instance of the program. Using this CLSID allows you to connect to a specific instance of the program. This CLSID is not stored in the registry as it is randomly generated each time the program it run. There is no human-readable progID associated with this CLSID. The only way of knowing the CLSID is to get it from the program, either using the report generator keyword $BROWSERPANELCLSID or by using the JScript/ActiveX method getPrivateCLSID(progID)

Version-less IDs

In addition to the version-specific types of CLSIDs and ProgIDs described above, there are two additional IDs available which are not specific to a single version of EMTPWorks but will connect to any available version. These were implemented starting in version 3.4 and will be available for all subsequent versions. These IDs guarantee that you will get some version of EMTPWorks, but make no guarantee which. In practice, you will normally get he most recently installed version since it will have updated the system registry last. The version-less IDs come in “single” and “multi” flavours as for the version-specific IDs described above.

- *See the ProgID and CLSID Table below for specific IDs.*

ProgID and CLSID Table for Version 4.0

Object	ID Type	ProgID	CLSID
Application	Version-Specific Multi	`EMTPWorks400.Application`	`{AAE134E6-7852-4AA4-A233-3FD7644F8000}`
Application	Version-Specific Single	`EMTPWorks400.Application.Single`	`{AAE134E6-7852-4AA4-A233-3FD7644F8001}`
Application	Version-less Multi	`EMTPWorks.Application`	`{AAE134E6-7852-4AA4-A233-3FD7644F99C0}`
Application	Version-less Single	`EMTPWorks.Application.Single`	`{AAE134E6-7852-4AA4-A233-3FD7644F99C1}`
BrowserPanel	Version-Specific Multi	`EMTPWorks400.BrowserPanel`	`{AAE149D5-7852-4AA4-A233-3FD7644FE000}`
BrowserPanel	Version-Specific Single	`EMTPWorks400.BrowserPanel.Single`	`{AAE149D5-7852-4AA4-A233-3FD7644FE001}`
BrowserPanel	Version-less Multi	`EMTPWorks.BrowserPanel`	`{AAE149D5-7852-4AA4-A233-3FD7644F9803}`
BrowserPanel	Version-less Single	`EMTPWorks.BrowserPanel.Single`	`{AAE149D5-7852-4AA4-A233-3FD7644F9804}`

Top-Level Objects Available to COM

To be added

Calling Javascript Methods from COM

To be added

Connecting External COM Clients to Event Notifications

Added 2015-11-27

All events have the same parameter format:

```
result = eventMethod(actionKey, string1, string2, dispatchObject);
```

actionKey	string1	string2	dispatchObject	return value
DW.Design.Open	Full path of design	Not used	design	not used
DW.Design.Closing	Full path of design	Not used	design	Can be one of `eDNR_NotHandled`, `eDNR_Handled`, `eDNR_HandledDirty`, `eDNR_HandledVetoSilent`, `eDNR_HandledVetoAlert`

DW.Design.Closed	Full path of design	Not used	design	not used
DW.Design.Save	Full path of design	Not used	design	not used
DW.Design.SaveAs	Full path of design	Not used	design	not used
DW.Design.Revert	Full path of design	Not used	design	not used
DW.Design.NewFromTemplate	Full path of template (in/out)	Default Open directory	not used	Can be one of `eDNR_NotHandled`, `eDNR_Handled`, `eDNR_HandledVetoSilent`, `eDNR_HandledVetoAlert`, `eDNR_ProceedWithNewPath`
DW.Exit	Not used	Not used	Not used	not used
DW.Circuit.StructChanged	Full path of design	Change flags, see below	circuit	not used
DW.Device.Form.Open	Full path of design	Not used	device	TRUE=handled, FALSE=defer to default operation

Notify Response Values

For notifications that return a status value, the following enumeration is defined:

```
	eDNR_NotHandled = 0,		// No one has handled this notification
	eDNR_Handled = 1,		// One or more clients has handled the notification but is not requesting any action
	eDNR_HandledDirty = 2,		// A client has asked to treat the document as dirty (i.e. prompt for save)
	eDNR_HandledVetoSilent = 3,	// A client has vetoed closing the document and is handling user contact
	eDNR_HandledVetoAlert = 4,	// A client has vetoed closing the document and wants a message displayed to the user
	eDNR_ProceedWithNewPath = 5
```

DW.Circuit.StructChanged Change Flags

Extra information about the type of changes that occurred is passed using a comma-separated string list of keywords, which may be any combination of the following:

Keyword	**Meaning**
HIERARCHY	Any change in hierarchy structure, e.g. a sub-circuit being attached or detached to a parent symbol
SIGCONN	Any change in signal connection
TYPEDEF	Change in the type definition for a device. E.g. Update from Lib

DEVADD	One or more devices have been added
SIGADD	One or more signals have been added
DEVDEL	One or more devices have been deleted
SIGDEL	One or more signals have been deleted

NOTE: A typical operation will result in multiple keywords. For example:

- deleting a device will typically result in the signals associated with the pins to be also deleted.
- connecting two device pins together will cause the two signals to be merged into one, giving a signal connection change and a signal deletion

11. Tracking Physical Instances of Objects in a Hierarchical Design

Overview

In a hierarchical design, a device symbol can be defined as having a subcircuit, which itself may contain other devices and signals. If the parent symbol is then used (instantiated) multiple times in the design, then an object in its subcircuit actually represents multiple real physical objects. Even though the subcircuit is defined only once in the design, the user may wish to view different data, e.g. simulation results, associated with the subcircuit depending on which physical instance is of interest at that time.

To allow this, EMTPWorks introduces the concept of an *instance locator*. An instance locator is a string that uniquely specifies a physical instance of an object, such as device, signal, or circuit, in a design. A number of script methods and properties and an event have been added to support this concept. Using instance locators allows the user to determine which physical instance of an object is being viewed and allows scripts and external packages such as simulators to display appropriate physical data associated with the selected physical objects.

Using Instance Locators

Note the following points related to instance locators:

- A subcircuit is defined only once, even if the parent device type is used in multiple places. For this reason only one physical instance of the circuit can be viewed at a time. Using script methods to display an object instance or changing the instance locator of the subcircuit or any of the devices or signals it contains, changes the physical instance of all objects in the circuit.

- The instance locator is a temporary setting that is used to select which physical instance of a circuit is being viewed. This setting made by user or scripted actions and is not stored with the design.
- The instance locator is set when the user navigates from the top down in a design, that is, if he opens a specific device in the top level design, the displayed subcircuit will now represent that specific physical instance, and the device name path to that instance will be displayed in the title bar. If the user were then to move back to a higher level window and push into another instance of the same device, the same subcircuit will be redisplayed but the name path will be updated to reflect the new physical instance.
- The instance locator for a subcircuit can be changed by script methods or external ActiveX calls. Such a change will change the displayed name path and will affect all objects in that subcircuit, but does not affect any higher level circuits that were part of the user's path in getting to that circuit.
- It is possible to directly open a subcircuit using script methods without navigating though the design hierarchy. In this case, there is no defined path to the subcircuit and its instance locator is not known and will be empty.
- Whenever a circuit is opened or its instance locator is changed (i.e. it now represents a different physical instance), the `Circuit.Instanceopened` event is fired. This event allows scripts or external modules to respond to the change and display different data on the circuit.
- The instance locator is in fact the same meaning (and has the same format) as the locator string already in use. The difference is that the locator is always defined and cannot be changed by script methods. The locator will always return a value that can be used to find an object, even if it is not known which specific instance is being referred to.

Using Hierarchical Names

Instance locators are guaranteed to be a unique way of locating an object in a design but they are not intended to be user-readable. To display a user-readable path to a physical object, methods are provided to get the hierarchical name of an object. This consists of the name of the object prefixed by the names of all parent devices up to the top level design, with separating "/" characters.

Since names are not guaranteed to be unique within a circuit, this is not a guaranteed way of specifying a single device. If names are not unique and a name path is used to select an object, the program returns the first matching item found.

Using the Circuit.Instanceopened Event

The `Circuit.Instanceopened` event is fired whenever a circuit or subcircuit is opened for which the instance locator is known and whenever the instance locator is changed on a subcircuit that is open. It will be fired for the top-level design when it is first opened.

The typical way of handling this event is to place an action line in the INI file, such as:

```
action = circuit.instanceopened, "instance_test.dwj", OPEN, $INSTLOCATOR
```

A simple script could look like:

```
a1 = getScriptArg(1);
a2 = getScriptArg(2);
des = currentDesign;
name = des.getNameByLocator(a2);
writeln("Instance opened with " + a1 + " - " + a2 + " - " + name);
```

Instance Locator Methods and Properties

The following script properties and methods are relevant to using the instance locator system:

`locator`	DWCircuit, DWDevice, DWSignal	Read only, always returns a path that can be used to locate the object. This may or may not be the same as the current instance locator.
`instanceLocator`	DWCircuit, DWDevice, DWSignal	A locator value that is specific to the path the user used to get to the object. Assigning to this can be used to change the physical instance displayed. If the physical instance is not known, this is null.
`instanceHierName`	DWCircuit, DWDevice, DWSignal	A hierarchical name that is specific to the path the user used to get to the object. Assigning to this can be used to change the physical instance displayed. If the physical instance is not known, this is null.
`allInstances`	DWCircuit, DWDevice, DWSignal	Returns an array of strings listing all possible physical instances of the given object in the design.
`showByLocator`	DWCircuit	This opens the circuit containing the object referred to by the locator, selects it, and sets its instance locator to the given value.
`getNameByLocator`	DWCircuit	This converts a given locator value to a hierarchical name.
`getCircuitLocatorByName`	DWCircuit	This converts the hierarchical name of a device to an instance locator.
`getSignalLocatorByName`	DWCircuit	This converts the hierarchical name of a signal to an instance locator.
`getDeviceLocatorByName`	DWCircuit	This converts the hierarchical name of a circuit to an instance locator.

12. Protection and Licensing of Device Symbols and Subcircuits

Overview

###to be added

License Levels

Level	Description	<--------All conditions must be satisfied-------->		
		License	**Access Password**	**Modify Password**
Use	Part can be used in a design but there is no access to the subcircuit	No license, or license with no level specified or license with suffix >= 10	NA	NA
SubScript	Part can be used and scripts can access the subcircuit	No license, or license with no level specified or license with suffix >= 20	Not specified or entered	NA
SubView	Part can be used and subcircuit can be viewed but not copied or modified	No license, or license with no level specified or license with suffix >= 30	Not specified or entered	NA
SubCopy	Part and can be used and subcircuit can be viewed and copied, but not modified	No license, or license with no level specified or license with suffix >= 40	Not specified or entered	NA
SubEdit	No restrictions	No license, or license with no level specified or license with suffix >= 50	Not specified or entered	Not specified or entered

13. Creating Dynamic-Symbol Devices

Dynamic-Symbol Devices are device symbols that can be resized after being placed in a design and whose contents can be redrawn dynamically by a script or external client to display status or results and receive user input. These capabilities are made possible by a collection of features described here.

Specifying a Device Script File

To be added

Dynamic Drawing – the ondraw method

The dynamic drawing of the device symbol is implemented by specifying an "ondraw" method in the script file associated with the device.

Here is a simple example:

```
function ondraw(dev)
{
   win = dev.window;
   win.ellipse(20,20,80,80);
}
```

Note the following principles regarding drawing the symbol:

- Each device instance has a separate window represented by a DWWindow object. I.e. even if two devices of the same library type are placed on a diagram, the ondraw method will be called separately for each device instance, so they can display different information or even have different sizes. The parameter passed to ondraw is the DWDevice object associated with the device instance. It can be used to access any device attributes or other information.
- The ondraw method is called every time the device needs to be updated, which could be quite frequently, so it should be as compact as possible. The ondraw method should not perform any time-consuming operations and should not display any alerts or make any other visible changes that could disrupt the update process. There are some system-level restrictions on what operations can be performed during an update procedure. Any operation that modifies another window or accesses another Windows application or subsystem may have unpredictable results
- The default coordinate system for drawing operations is "percentage", with 0,0 at the top, left corner of the symbol and 100,100 at the bottom, right. Using this system, the scaling will automatically adapt to any changes in symbol size. See more information on coordinate systems below.

- The window provided is always a rectangle bounding all elements of the original symbol. If it is desired to maintain some other symbol shape, it is the responsibility of the script to only draw in the desired areas.
- The EMTPWorks application draws the original device symbol on the screen first, before calling ondraw. This symbol is stretched to the current size of the device instance. The symbol can provide a background fill, and any desired static elements, or can be completely empty and transparent.

The DWWindow object and drawing methods

Drawing methods such as line, rect, ellipse, etc. can be used to render the desired symbol or status information into the device window. See information on this object in the JavaScript documentation provided.

Coordinate Systems

In order to simplify scripting in an environment where the symbols may change in size, shape and zoom level, a number of methods are provided for specifying X and Y drawing coordinates. The available coordinate systems are:

Title	Code	Description
Percent	`pct`	X and Y coordinates are specified as a percentage of the width or height of the device, as currently displayed on the screen
Inch	`in`	Coordinates are specified in inches, assuming “normal” screen zoom factor. I.e. the size of items drawn in this coordinate system will not be affected by resizing the device symbol, but they will be scaled by the current zoom factor.
Cm	`cm`	Coordinates are specified in centimetres, assuming “normal” screen zoom factor. I.e. the size of items drawn in this coordinate system will not be affected by resizing the device symbol, but they will be scaled by the current zoom factor.

###reference point

```
win.setCoordSystem("cm", "tl");
```

Getting Pin Positions

To be added

```
p1 = dev.getPinWindowPos(1, win);
```

Receiving User Input – the onclick and ondblclick methods

To be added

Creating a Resizable Symbol

To be added

Specifying Pin Movement

To be added

Symbol Picture Stretching

To be added

14. Enhanced Line Graphic Styles

To be added

Specifying line styles in INI file

To be added

15. Associating Properties with Design Objects

To be added

Specifying Schema Search Paths

By default, the application searches for XML schema files in the following folders:

EMTPWorks Program Folder\Property Schemas

and

Documents\EMTP\Property Schemas

Additional search directories can be added by specifying them in the INI file using the SchemaFolder keyword.