DTS 4: Editor Module Guide

APELON

BECAUSE TERMINOLOGY MATTERS

75 Sgt William B Terry Dr, Suite 2005, Hingham, MA 02043

+1 (203) 431-2530

www.APELON.com

www.ApelonDTS.org

Table of Contents

A. Introduction 4
B. Framework Overview 5
C. Editor Configuration 6
D. DTS Editor Layouts 7
D.1 Layout XML File – Layout Tag 7
D.2 MenuBar Tag 8
D.3 ToolBar Tag 8
D.4 PanelLayout Tag 9
D.5 Panel Tag 9
D.6 TabPanel Tag 10
D.7 MultiPanel Tag 10
E. Layout Editor 13
E.1 Editor Overview 13
E.2 The Options Tab 14
E.3 The Menu Bar Tab 15
E.4 The Tool Bar Tab 17
E.5 The Layout Tab 19
E.5.1 The Panel Element 20
E.5.2 The TabPanel Element 22
E.5.3 The MultiPanel Element 24
E.6 Layout Editor Menus 25
F. Module Events 27
F.1 IIDs 27
F.2 Event Types 27
F.3 Event Processing 28
F.4 Programmatic Setting of IIDs 29
G. Internationalization 31
H. Example Module 32
H.1 Module Registration 32
H.2 Imports 33

H.3 Class Declaration 34
H.4 Initializing the Module 35
H.5 Getting Module Menu and Toolbar Items 37
H.6 Getting Plug-in Menus, Menu Items and Toolbar Items 39
H.6.1 Setting Plug-in Components 40
H.7 Handling Connection Events 41
H.8 Initialization Summary 41
H.9 Creating a Panel for the Module 41
H.10 DTSMonitor Functionality 42
H.10.1 Connection Event Functionality 45
H.10.2 Module Event Functionality 46
H.10.3 Data Change Event Functionality 47
H.11 Drop Functionality 49
H.12 Showing Drop Details 50
H.13 Error Handling 51
H.14 Configuration Management 52
I. Converting Plug-in Modules 54
I.1 Converting a Pre-V4 Plug-in 54
I.2 Converting to V4.3 54
J. Appendix A – DTS Editor Modules 55
K. Appendix B – Standard DTS Editor Layout File 60

A. Introduction

The **DTS Editor Plug-in Framework** was introduced in DTS Version 3.4 to provide a mechanism for extending the capabilities of the DTS Editor. Using the Plug-in Framework, developers were able to create new **plug-in modules** designed to interface and work with existing DTS Editor features. This enabled the DTS developer to create targeted, custom functionality without having to spend time and effort developing a GUI architecture and re-creating standard DTS Editor features such as connection handling and drag-and-drop (DnD).

With DTS Version 4, the Plug-in Framework has been extended to a **DTS Editor Module Framework** that provides additional capabilities and permits complete customization of the DTS Editor layout. In essence, the DTS Editor is no longer an *extensible application*, but a *customizable platform* for the delivery of DTS user functionality. The "classic" DTS Editor is still available as one example of an Editor implementation.

The DTS Editor Module Framework is backwards-compatible with the Plug-in Framework: pre-V4 Editor plug-ins run unmodified in the default DTS Editor configuration, and can be easily "upgraded" to full Module status with the addition of few new methods in their base class.

This Guide provides a comprehensive explanation of the DTS Editor Module Framework, including a description of DTS Editor Layouts and the interactive Layout Editor. An example of custom Module development is given and the catalog of available standard DTS Modules is provided. The Guide is written for developers and architects who would like to implement custom functionality in their DTS environments. A basic understanding of Java programming is assumed.

B. Framework Overview

In the DTS Editor Module Framework, functionality is delivered via discrete components called **Modules**. Modules encapsulate a set of features (menu items, toolbar items, GUI panels and executable classes) that perform related tasks. Examples of Modules from the classic DTS Editor implementation are the Search Panel and the Property Editor Panel. Using aspects of the Module Architecture such as the DTS Editor Configuration File and DTS Editor Layout File (described in succeeding sections), Modules can be selected, functionality tailored, and GUIs designed to meet specific user requirements. *Appendix A – DTS Editor Modules* is a catalog of all standard DTS Editor Modules annotated with details on Module menu and toolbar items, event processing, DnD support, and configuration options.

One of the strengths of the Module Architecture is its ability to support the creation and integration of custom, user-developed, DTS Editor Modules. Thus the default Editor structure can be augmented, or even replaced, with unique features and functions. A new DTS Editor Module (or plug-in) is created by writing a base Module class that extends the `DTSEditorModule` class and placing this class (and any related Module classes) into a "jar" package. On DTS Editor start-up, designated folders are scanned for these Module packages and identified Modules are registered for potential use. Once registered, the Module can be used as part of an Editor layout (see *DTS Editor Layouts* below) or "plugged-in" to the default layout.

When the Editor determines that a Module is to be used, it invokes specific methods (defined in the `DTSEditorModule` class) to initialize the module, create custom menu and toolbar items, etc. The `DTSEditorModule` class (and its associated `DTSEditorModuleMgr` class) provide methods for defining Modules, exposing functionality (like displaying panels) and accessing DTS Editor services including drag and drop support for DTS data objects such as Concepts, Properties, Terms, etc., connection and event detection, and error (exception) processing. Further details on Module Framework support classes are provided in the *Example Module* section below.

The following sections provide details on specific aspects of the Module Architecture and an example of Module development.

C. Editor Configuration

When the DTS Editor is invoked, it reads a configuration (Java `Properties`) file that defines local Editor attributes such as default connection parameters and preferred behaviors, such as whether clicking on a Namespace or Concept in the *Tree Panel* automatically loads the *Details Panel* with the Concept (a.k.a., "Click to Edit").

Since most Editor behaviors are set by user selections in the various panels, it is unlikely that you will need to manually modify configuration values, but there are two configuration parameters that should be understood:

Key	Value
`autoConn`	If "true", the DTS Editor performs an automatic Connect action when opening. Default value is "false". This parameter can be also set by the `Connect Parameters` panel.
`title`	If non-empty, the string placed in the Title Bar of the application. This value overrides the normal value of the `DTSEditor.Title` resource file entry (see the **Internationalization** section below for further information on the internationalization resource file). The default value for `title` is "".

The default configuration file for the DTS Editor is:

```
bin\editor\dtseditor.xml
```

in the DTS installation directory. A different configuration file can be specified, however, by providing another argument to the `runApp` command in the invoking command line. Thus:

```
C:\Program Files\Apelon\DTS 4.3\bin\runApp.bat
com.apelon.apps.dts.editor.DTSEditorApp –config:MyConfig.xml
```

will start the Editor using the `MyConfig.xml` file (in the `bin\editor` folder). It is unlikely that a different configuration file will be needed, but if it is, the custom file should be created by first opening the default file and then saving the file under a different name.

D. DTS Editor Layouts

The visual structure of the DTS Editor application, i.e., the items in the Menu Bar, the icons in the Tool Bar and the panels visible in the main Editor window, is called the Editor's *Layout*. Prior to DTS Version 4.3, Layout definitions were stored in XML files on the client and linked from the DTS Editor configuration file. Since 4.3, Layout definitions are maintained on the DTS server and users can interactively select desired Layouts, and design their own Layouts using a GUI-based designer, all directly from the DTS Editor. Management of Layouts is similar to that of Namespace and Subset Profiles (see the **DTS Editor Users Guide** for information on Profiles). Layout definitions can also be exchanged via Layout XML files, compatible with those used prior to DTS 4.3.

The remainder of this section describes the elements of a Layout definition using the XML exchange file format. The following section will explain how to design and maintain Layout definitions interactively using the Layout Editor GUI.

D.1 Layout XML File – Layout Tag

A Layout XML file specifies the contents and structure of a DTS Editor instance: its menu bar, tool bar and panels. Elements in the Layout also control whether plug-in Modules are allowed and how Modules interact. Layout files can be created using a text or xml editor, or can be written from an existing Layout using the `Export` option in the Layout Editor.

The format of the Layout file is defined by `dtslayout.xsd` which can be found in the `bin\editor` folder of the DTS installation directory. The default DTS Editor Layout file, `dtslayout.xml`, is shown in *Appendix B – Standard DTS Editor Layout File*. Note that the provided `dtslayout.xml` is for descriptive purposes only; it is not used by the DTS Editor. A read-only, internal representation of the default Layout is kept. The file can be imported into the Layout Editor, however, to "seed" new Layouts.

The top level element of a Layout is the `Layout` tag. This tag wraps the rest of the Layout elements and can contain four attributes. Here is an example of the `Layout` tag from the default Layout:

```
<Layout Description=" Apelon Standard Layout" EnablePlugins="true" >
```

The first attribute is `Description`. This optional attribute is a text description of the Layout.

The next attribute is the optional `EnablePlugins` attribute. When the value of this attribute is "true", any registered Modules that are not explicitly named in the Layout are considered plug-ins and are automatically added to the Layout at layout build-time according to the values returned by the plug-in methods from the base Module class. If this attribute is absent, or its value is "false", only those Modules explicitly identified in the Layout will be made a part of the application. This attribute is set to "true" in the default DTS Editor Layout for backwards compatibility, but when a custom Layout has been defined, it is usually appropriate to not include the `EnablePlugins` attribute so that the Layout components can be completely specified.

Two other optional attributes are available for the `Layout` tag: `Width` and `Height`. These elements declare the width and height (in pixels) of the Editor frame. If the width and height are not explicitly provided, the frame is built using a default width of 1024 and default height of 768 respectively. In either case, the realized width and height are subject to available screen size.

The three element tags under the `Layout` tag are `MenuBar`, `ToolBar` and `PanelLayout`.

D.2 MenuBar Tag

The `MenuBar` tag specifies the contents of the application's menu bar. As seen in the fragment below:

```
<MenuBar>
      <Menu Command="file" Mnemonic="f" Name="File">
            <SelectItem ModuleName="DTSConnect"/>
            <Separator/>
            <SelectItem ModuleName="DTSExit"/>
      </Menu>
      …
</MenuBar>
```

a `MenuBar` is made up of a sequence of `Menu` tags, each corresponding to a Java `JMenu`. Attributes in the `Menu` tag specify the internal action command (`Command`), menu mnemonic (`Mnemonic`) and visible menu name (`Name`). Within each `Menu` tag is a list of `SelectItem` elements corresponding to the specific selectable items in the menu. In addition to `SelectItem`, the `Separator` element is available to draw an item separator in the menu.

The only required attribute in the `SelectItem` tag is the name of the Module invoked when the item is selected. The `SelectItem` tag does support two other, optional, attributes, `Options` and `Tip`. The `Options` attribute value is a formatted string that will be passed to invoked Modules to modify Module layout and/or behavior. `Options` is described in more detail in the *Panel Tag* section below. The value of the `Tip` attribute is a ToolTip text string that can be associated with the item. The value overrides any default ToolTip provided by the Module.

A Module can expose multiple menu items when it is loaded (see *Appendix A – DTS Editor Modules* for examples). Note that the display name of the menu item(s) is accessed at build-time from the Module's base class.

D.3 ToolBar Tag

The `ToolBar` element has similar functionality for creation of the application's toolbar. Using the example below:

```
<ToolBar>
      <SelectItem ModuleName="DTSConnect"/>
      <Separator/>
      <SelectItem ModuleName="DTSTree"/>
      <SelectItem ModuleName="DTSSearch"/>
      …
<Toolbar>
```

the `ToolBar` tag defines the set of Modules that are shown (via icons accessed from the Module's base class) in the toolbar. The `ModuleName` attribute is required in each `SelectItem` tag, while the `Options` and `Tip` attributes are optional. As with the `Menu` tag, multiple icons can be exposed by the Module and the `Separator` element is available.

D.4 PanelLayout Tag

The most interesting portion of the `Layout` definition is the `PanelLayout` tag. `PanelLayout` implements a very flexible "language" for describing the structure of Module panels in the main window. A `PanelLayout` element contains one of the following elements:

- `Panel` – a single Module panel
- `TabPanel` – a tabbed panel, i.e., Java `JTabbedPane`, consisting of multiple panels, selectable by clicking on a tab label
- `MultiPanel` – a compound panel consisting of a set of panels organized sequentially either horizontally or vertically

D.5 Panel Tag

The `Panel` element typically consists of simply the name of its associated Module:

```
<Panel ModuleName="DTSStatus"/>
```

The `Panel` tag also supports the optional `Options`, `IID` and `TargetIID` attributes. The `IID` and `TargetIID` attributes are described later in the *Module Events* section of this document. The `Options` attribute is a formatted string that can be passed to Modules to modify Module layout and/or behavior. The format of the `Options` string is a semicolon-delimited list of key/value string pairs separated by a colon. The value string (and its colon delimiter) are optional. Spaces are ignored and quotation marks are not permitted in the attribute value. Here is an example of an `Options` attribute:

```
<Panel ModuleName="MyModule" Options="size:big;color:red;full" >
```

When a Module is instantiated from the Layout, the `Options` element is parsed by the Module Loader and converted to a Java HashMap before passing to the Module as described in *Programmatic Setting of IIDs*. Both keys and values are converted to lowercase. The `Options` parameter (with associated map) is currently only supported by the DTSDetail Module (see

Appendix A – DTS Editor Modules), but is available for use by user-developed Modules and Plug-ins.

D.6 TabPanel Tag

The `TabPanel` element consists of an optional `Placement` attribute, and a sequence of `Tab` elements, each element specifying the panel contents of the associated tab. The `Tab` element can contain an optional `Label` attribute which provides an explicit value for the tab's label.

The `Placement` attribute specifies where the panel's tabs are to be placed. A value of `"T"`, or `"t"` species the top of the panel, `"B"` or `"b"` specifies the bottom, `"R"` or `"r"` specifies the right side, and `"L"` or `"l"` specifies the left side. The default is `"T"`.

The `Label` attribute is usually not necessary when the tab content is a `Panel`. When it is missing, the value returned by the `getComponentShortName()` method from the Module's base class is used as the tab's label string.

The content of a `Tab` is the same as the content of the `PanelLayout` element: a `Panel`, `TabPanel` or `MultiPanel`:

```
<TabPanel Placement="t" >
      <Tab >
            <Panel ModuleName="DTSTree" TargetIID="detail" />
      </Tab>
      <Tab >
            <Panel ModuleName="DTSWalker" TargetIID="detail" />
      </Tab>
      <Tab >
            <Panel ModuleName="DTSSearch" TargetIID="detail" />
      </Tab>
      <Tab >
</TabPanel>
```

See the individual `Panel`, `TabPanel` and `MultiPanel` sections for details on these elements.

D.7 MultiPanel Tag

The `MultiPanel` element builds a horizontal or vertical sequence of panels, each constituent panel sharing the available space of the combined panel. The power of the `MultiPanel` is that the constituent panels can themselves be compound panels. Like the `TabPanel` element, the panel elements in a `MultiPanel` can be a `Panel`, a `TabPanel`, or another `MultiPanel`. A `Divider` element is also available to specify that a movable divider be placed between two adjacent panels. Here is a simplification of the familiar default DTS Editor Layout:

```
<PanelLayout continuousLayout="true">
      <MultiPanel Type="V">
            <MultiPanel Type="H">
                  <TabPanel>
                        <Tab Name="tree" ModuleName="DTSTree"/>
                        <Tab Name="walker" ModuleName="DTSWalker"/>
                        <Tab Name="search" ModuleName="DTSSearch"/>
                  </TabPanel>
                  <Divider/>
                  <TabPanel>
                        <Tab Name=”detail” ModuleName="DTSDetail"/>
                  </TabPanel>
            </MultiPanel>
            <Panel ModuleName="DTSStatus"/>
      </MultiPanel>
</PanelLayout>
```

The top level layout element is a vertical (attribute `Type=”V”`) `MultiPanel` consisting of another `MultiPanel` and the `DTSStatus` Module `Panel`. The inner, horizontal, `MultiPanel` consists of a `TabPanel` containing the `DTSTree, DTSWalker` and `DTSSearch` Module panels, a movable `Divider`, and a second `TabPanel` containing the `DTSDetail` Module panel.

The `continuousLayout` attribute on the initial `PanelLayout` tag is an optional attribute that specifies whether panels are continuously repainted when any `MultiPanel Dividers` are being moved. The default is “true”.

A `Weight` attribute is also available for the `MultiPanel`, `TabPanel` and `Panel` tags. `Weight` provides “guidance” to the Layout builder as to how to apportion panel space among constituent panels. The value of `Weight` for a panel is a decimal number that is compared to the sum of the other `Weight` values in the encompassing panel to determine the proportion of space allocated to the panel. For example, in the `MultiPanel` below:

```
<MultiPanel Type="H">
      <Panel ModuleName="FirstPanel” Weight=”.3"/>
      <Panel ModuleName="SecondPanel” Weight=”.6"/>
</MultiPanel>
```

the first `Panel` is allocated one third (.3/.9) of the space. Constituent panels not having a weight are given space according to their preferred sizes. While the Layout builder tries to honor weight requests and specified preferred, minimum and maximum sizes, any such request is always subject to constraints placed on the Layout from other Layout elements. Thus in the first (vertical) `MultiPanel` of the default DTS Editor Layout, all extra space is allocated to the upper panel because the `DTSStatus` Module panel asserts a single line maximum height. Finally, note the Layout Editor does not consider weight values in its display algorithm.

The actual layout directions used by the `MultiPanel` layout manager (top-to-bottom/bottom-to-top, left-to-right/right-to-left) is determined by Locale settings. See the *Internationalization* section below for details.

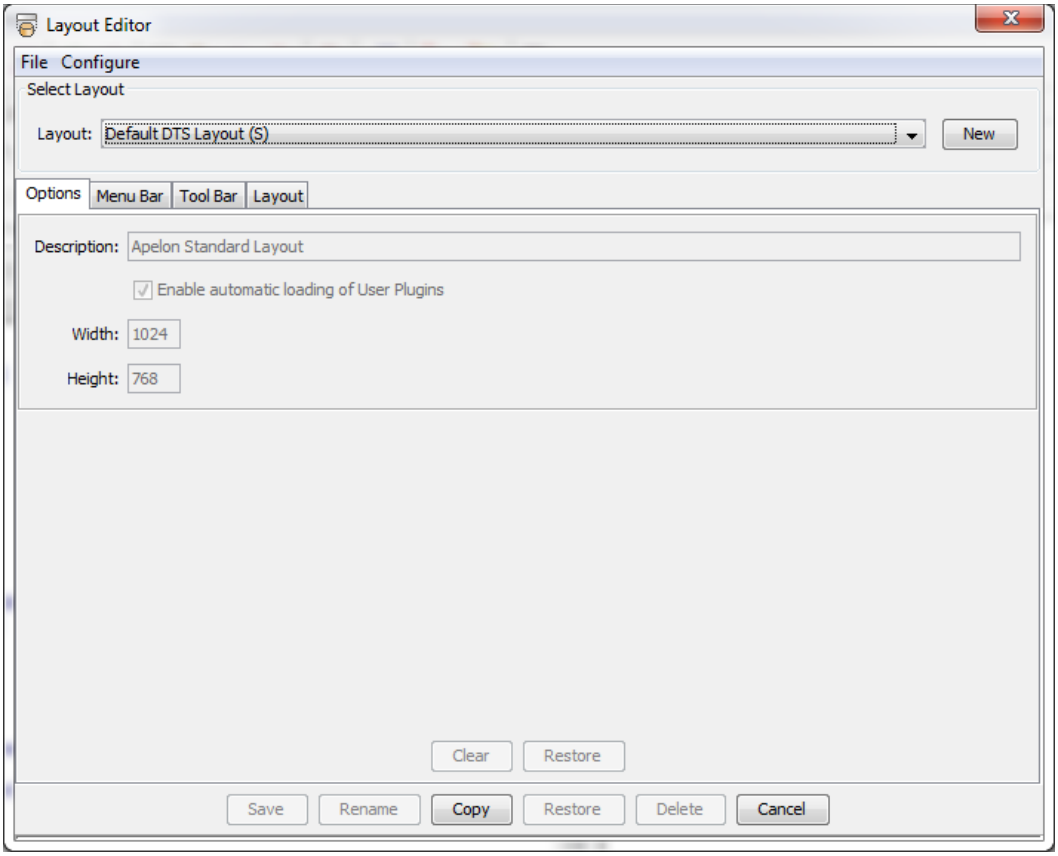
Panel attributes can also include the `IID` and `TargetIID` Instance Identifier (IID) attributes used to facilitate inter-Module communication. IIDs are described in the *Module Events* section below.

E. Layout Editor

The Layout Editor is a GUI panel for creating and maintaining DTS Editor Layouts. The Editor supports all of the Layout elements described in the preceding section. In the standard DTS Editor Layout, the panel as accessed via the `Layout Editor` item in the `Options` menu.

E.1 Editor Overview

Upon opening, the Layout Editor appears as below:



The Editor Menu bar holds the `File` and `Configure` menus. These menus will be described at the end of this section.

The main panel consists of an upper `Select Layout` area with a Layout combo and a `New` button, a middle definition area with four tabs which display the specifications of the Layout, and a lower area containing six buttons for performing Layout Editor actions.

DTS supports both User and System Layouts. User Layouts, shown with a "`(U)`" following the Layout name, are available only to the current user; users can see their own User Layouts, but not those of other users. System Layouts, shown with an " (S) " after the name, are available to (can be read by) all users but only users having the DTS Administrator privilege can create or edit System Layouts.

To load an existing Layout into the Editor, select the Layout from the dropdown in the upper selection area. The middle definition area will be loaded with the definition of the Layout. The screen shot above shows the System Default DTS Layout. This Layout is always available for selection, although in read-only form (all fields are disabled). See the description of the `Copy` button below for instructions on how to create a writable version for modification.

To create a new Layout, click the `New` button in the selection area, and enter the Layout's name in the dialog. The definition area will be loaded with an empty User Layout.

The six buttons at the bottom of the Editor panel perform actions on the selected Layout. Some buttons are only enabled when the Layout is writable, or has been modified.

- The `Copy` button copies the selected Layout to a new Layout with a different name. Any Layout, including a read-only Layout, can be copied. Non-Administrator users can copy System Layouts, but only to a new User Layout. This can provide "starting points" for Layout customization. If the user is a DTS Administrator, a User Layout can be copied to a new System Layout. This is called Layout *promotion*.
- `Delete` deletes the selected Layout. System Layouts can only be deleted by DTS Administrator users.
- `Rename` renames the current Layout.
- `Save` stores any modifications to the selected Layout.
- `Restore` discards current modifications to the Layout and loads the original Layout definition.
- `Cancel` closes the Layout Editor. If any modifications are pending, a confirmation dialog is shown.

The following sections describe the four tab panels that make up the definition area. These panels provide for viewing and updating of Layout specifications.

Note: The descriptions that follow assume that the selected Layout is writable. If the selected Layout is read-only, i.e. the `Default DTS Layout (S)` or another System Layout when the user does not have the DTS Administrator privilege, all element context menus will consist of a single `View` item which will open the associated parameter dialog in read-only mode (all fields will be edit-disabled).

E.2 The Options Tab

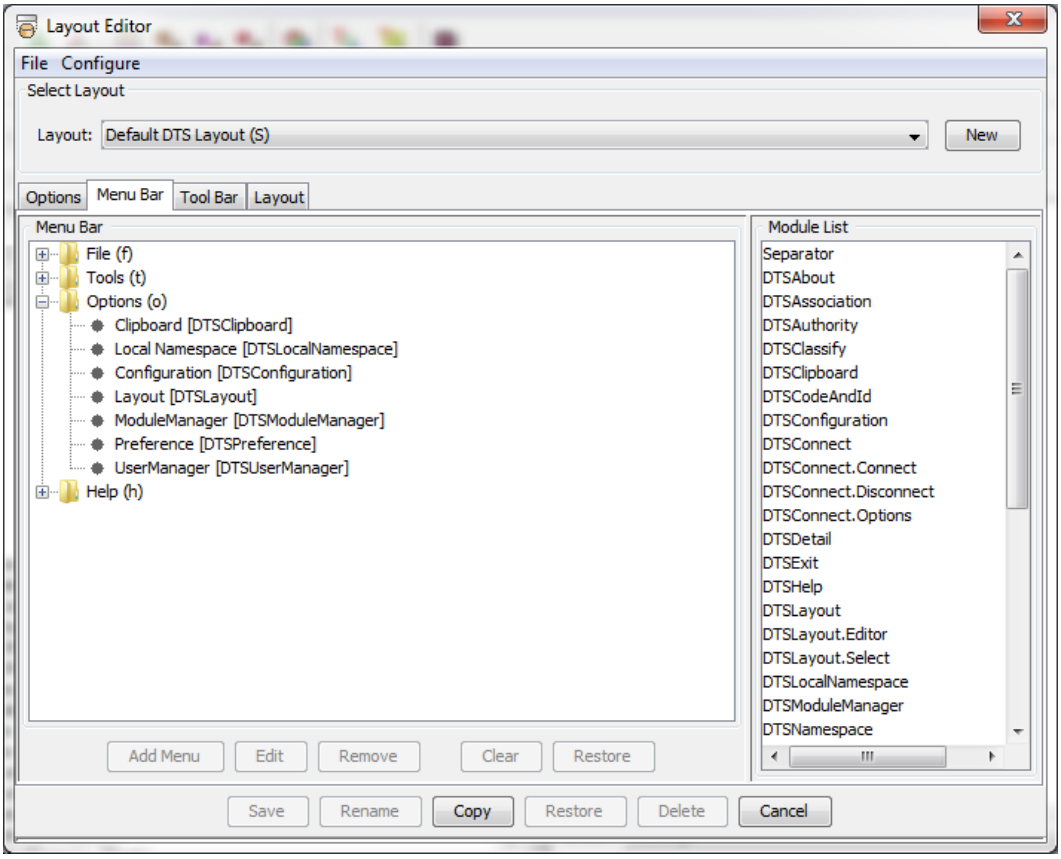
The `Options` tab (see screen shot above) shows the attributes associated with the `Layout` element. There is a text description of the Layout (the `Description` attribute), a checkbox for setting whether automatic loading of user plugins at build time should be performed (the `EnablePlugins` attribute), and fields for the optional `Width` and `Height` attributes. As described in the previous section, plugins are enabled for the default layout, but custom layouts will usually leave this box unchecked. The default width and height are shown in the fields. You can override the defaults by entering the desired integer value.

At the bottom of this, and the other tab panels, are two action buttons: `Clear` and `Restore`.

- `Clear` clears the contents of the tab panel, restoring the field defaults, if any.
- `Restore`, which is only enabled once modifications to the panel have been made, discards any modifications and restores the elements' previously saved values.

E.3 The Menu Bar Tab

The `Menu Bar` tab shows the menus (`Menu` elements) and associated menu items (`SelectItem` elements) for the Layout:



The menu bar menu and menu item entries are visualized in a tree format in the left portion of the tab panel. The right portion of the tab panel is the `Module List` panel. This latter panel lists all the Modules, and Module components, that can be used in menus. Note that some Modules, like the DTSStatus Module, are not present in the list because they do not have an associated menu item. As of DTS Version 4.3, new methods have been added to the Module API to support passing feature availability information to Layout Editors. See the *Example Module* section below for further information.

Menus are displayed as top level nodes with menu items as subordinate nodes. To see the menu items below any menu, just expand the menu node. (In the screen shot above, the `Options`

menu has been expanded). Menu items are displayed with their menu item name followed by their Module Name.

To add a new menu to the Menu Bar, click on the `Add Menu` button and enter the menu name in the resulting dialog. If a menu or menu item is selected, the menu is added before the selected menu. If there is no selection, the menu is added as the last menu.

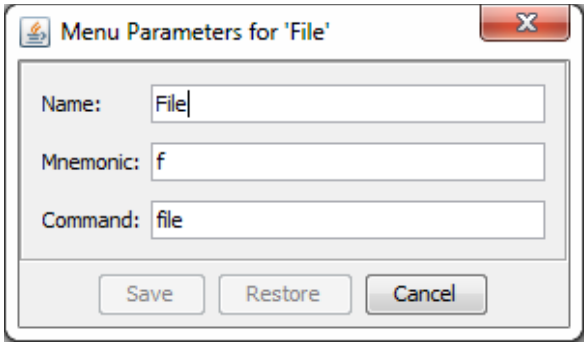
To add a menu item to a menu, drag a Module, or Module component, entry from the Module List panel to the menu tree. Tree nodes will be highlighted as the Module moves over the tree. . Entries dropped on a menu item will insert a menu item at the drop location. If the drop location is a menu node, the entry is inserted as the last menu item in the menu. The Separator pseudo-Module is available in the Module List panel to place a separator item in the menu. Remember that some Modules will expose multiple menu items when the menu is realized. Module components, on the other hand, will expose only one menu item.

Menu items, and whole menus, can be moved within the Menu Bar panel using drag and drop. Available drop locations (tree nodes) are highlighted as an item is dragged over the tree. Menu items can be dropped on any node. An item dropped on another menu item inserts the item at the drop location. If the drop location is a menu node, the item is inserted as the last item in the menu. Menus can only be dropped on other menu positions. The dropped menu, with all of its subordinate items, is inserted as a new menu at the location. Note that internal drops are always inserts; they do not replace the previous entry.

The normal internal drop mode is MOVE: after the drop, the moved item is removed from its original location. If the `Ctrl` key is pressed during the drag, on the other hand, the drop mode is COPY: the menu or menu item remains in its original location.

Selecting a menu or menu item enables the `Edit` and `Remove` buttons at the bottom of the tab panel. These same options are available via a right-click context menu on the entry.

Pressing the `Edit` button (or selecting the `Edit` context item) for a menu node opens the `Menu Parameters` dialog:



This dialog displays the `Menu` attributes:

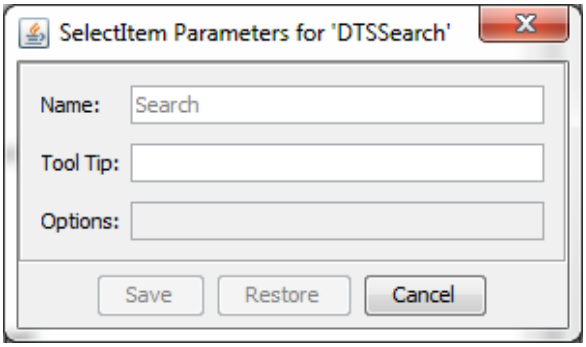
- `Name` is the name of the menu.

- `Mnemonic` is the optional one character mnemonic for the menu.
- `Command` is an optional string value that is set as the command attribute on the menu item. The command string is only used by Java programs and is not required.

Click the `Save` button to save any modifications, `Restore` to discard modifications and return to the original parameter values, and `Cancel` to exit the dialog.

Pressing the `Remove` button (or selecting the `Remove` context item) for a menu node opens a confirmation dialog to remove this menu and all of its associated items from the menu bar.

Pressing the `Edit` button or (or selecting the `Edit` context item) for a menu item node opens the `SelectItem Parameters` dialog:



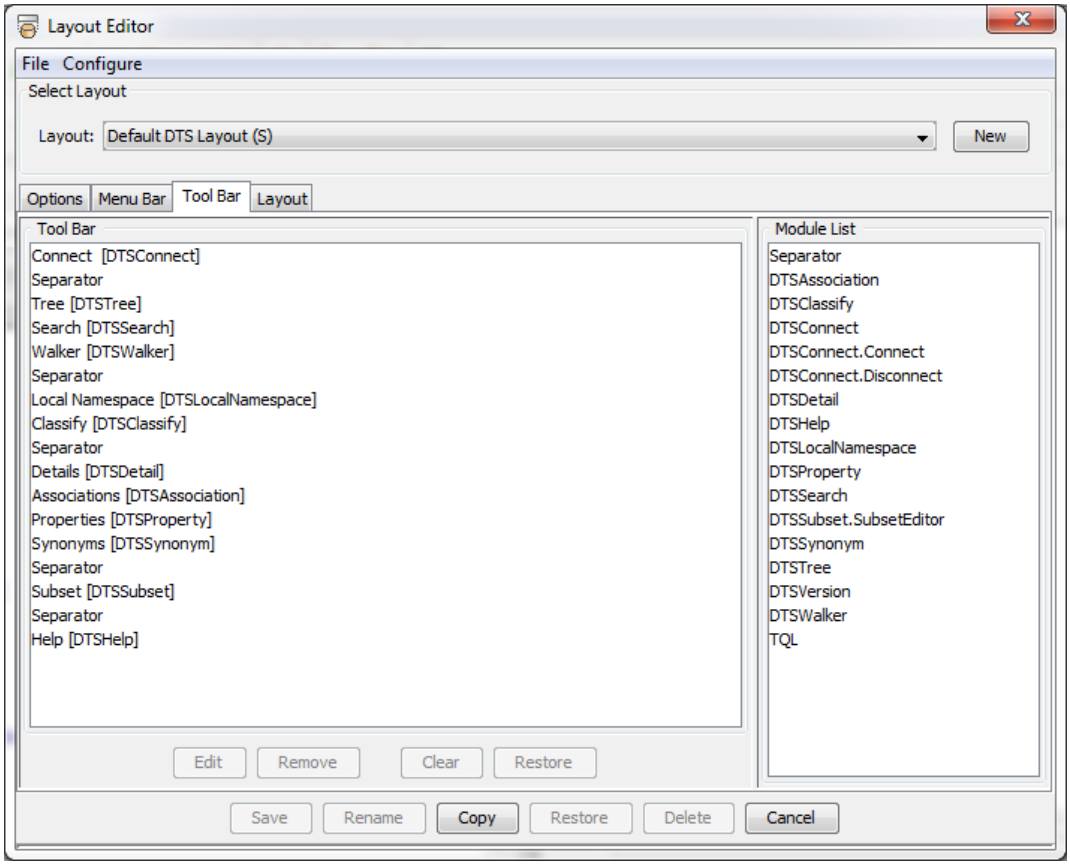
This dialog displays the `SelectItem` attributes associated with a menu item:

- `Name` is the name (`Name` attribute) of the menu item. This value defaults to the Module component's short name, as specified in the Module class file, but it can be overridden by typing an alternate value into the field.
- `Tool Tip` is the tool tip string (`Tip` attribute) displayed when the mouse hovers over the item. Any entered value overrides the value from the Module class file.
- `Options` is an optional parameter string that is passed to the Module when it is invoked. This field is only enabled if the associated Module class file has designated that the Module accepts an option string. See the *Panel Tag* section above for further information.

Pressing the `Remove` button (or selecting the `Remove` context item) for a menu item node opens a confirmation dialog to remove this menu item from its parent menu.

E.4 The Tool Bar Tab

The `Tool Bar` tab shows the tool bar items (`SelectItem` elements) for the Layout:



Tool bar entries are visualized in a list format. Entries are displayed with their Short Name and Module Name. Again, note that only Modules that expose tool bar icons are present in the `Module List` panel.

To add an item to the tool bar, drag a Module, or Module component, entry from the `Module List` panel to the list panel. Tool bar nodes will be highlighted as the Module moves over the list. A drop inserts a new tool item at the drop location. The `Separator` pseudo-Module is available in the `Module List` panel to place a separator item in the tool bar. Remember that some Modules will expose multiple tool bar items when the menu is realized. Module components, on the other hand, will expose only one tool bar item.

Like the Menu Bar panel, tool bar items can be moved within the Tool Bar panel using drag and drop. Dropped entries are always inserted and use a MOVE drop mode unless the `Ctrl` key is pressed.

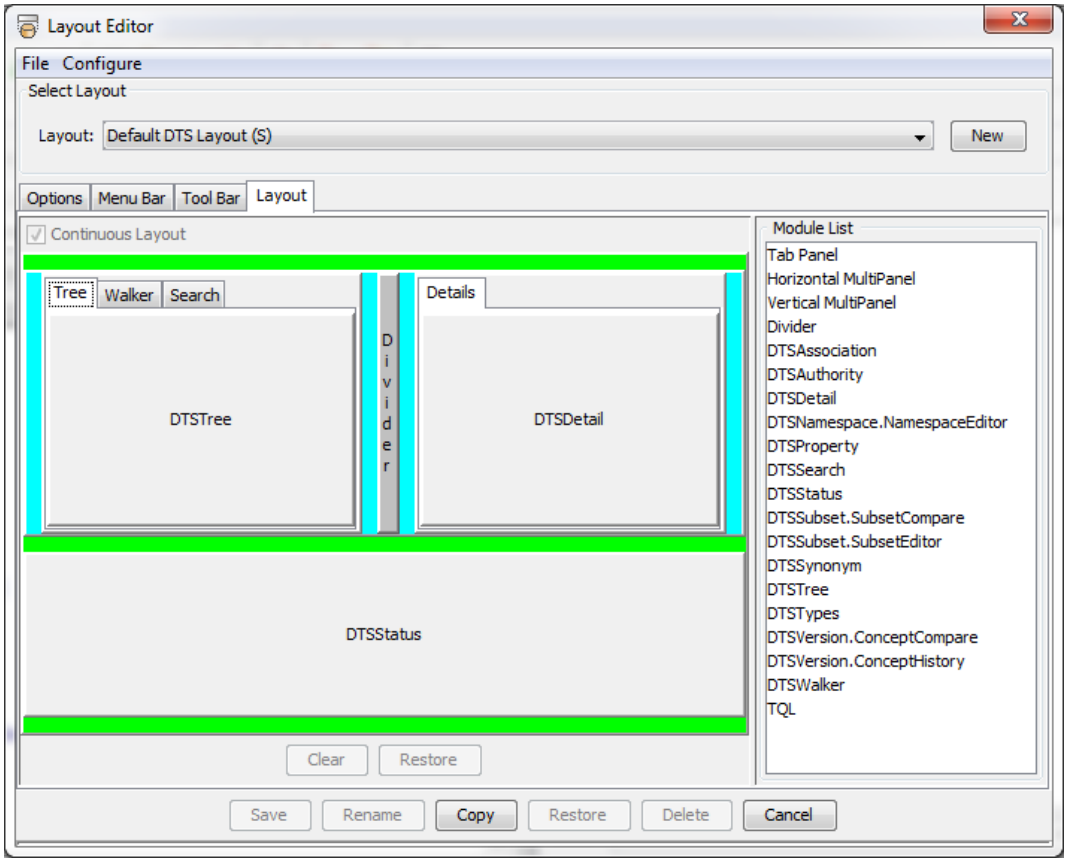
Selecting a tool bar item enables the `Edit` and `Remove` buttons at the bottom of the tab panel. These same options are available via a right-click context menu on the tool bar entry.

Pressing the `Edit` button or (or selecting the `Edit` context item) for a tool bar entry opens the `SelectItem Parameters` dialog. This dialog is the same as that of the menu item dialog described above; it contains the `Name`, `Tool Tip` and `Options` fields.

Pressing the `Remove` button (or selecting the `Remove` context item) for a tool bar node opens a confirmation dialog to remove this item from the tool bar.

E.5 The Layout Tab

The Layout Tab displays the `PanelLayout` element. This panel contains the `Continuous Layout` checkbox, the "drawing panel" for the Layout and another `Module List` panel. Here is the Layout tab for the default DTS Editor Layout:



The checkbox is normally selected so that the Layout will be continuously redrawn when panel dividers are moved. As with the Menu Bar and Tool Bar panels, the `Module List` panel only contains Modules, or Module components, that allow placement in the Layout. The `TabPanel`, `Horizontal MultiPanel`, `Vertical MultiPanel` and `Divider` pseudo-Modules are also available. The effects of entries will be described later.

As described in the *DTS Editor Layouts* section above, a Layout consists of one of the following elements:

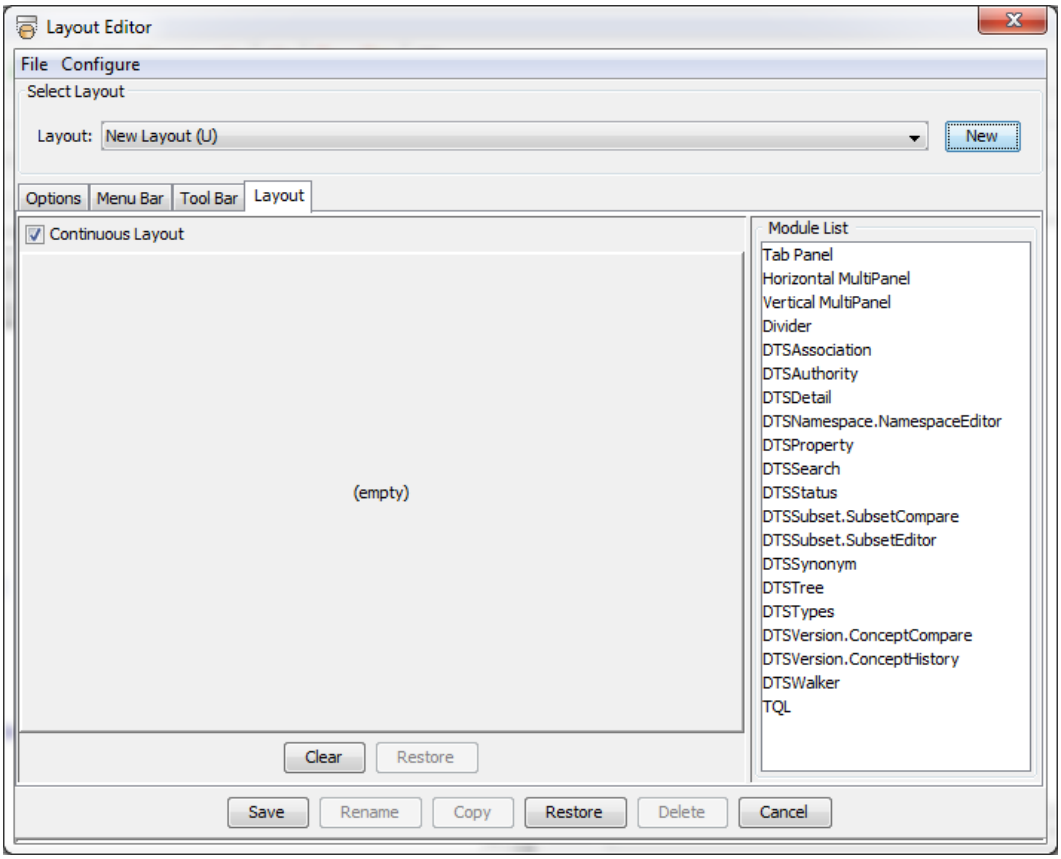
- `Panel` – a single Module panel
- `TabPanel` – a tabbed panel, i.e., Java `JTabbedPane`, consisting of multiple panels, selectable by clicking on a tab label

- `MultiPanel` – a compound panel consisting of a set of panels organized sequentially either horizontally or vertically

The following sections describe these three elements and their associated operations.

E.5.1 The Panel Element

A `Panel` element can exist as the lone element in a Layout or as a component of a `TabPanel` or `MultiPanel`. The Layout panel for a new Layout is a single, empty, `Panel`, as shown below:

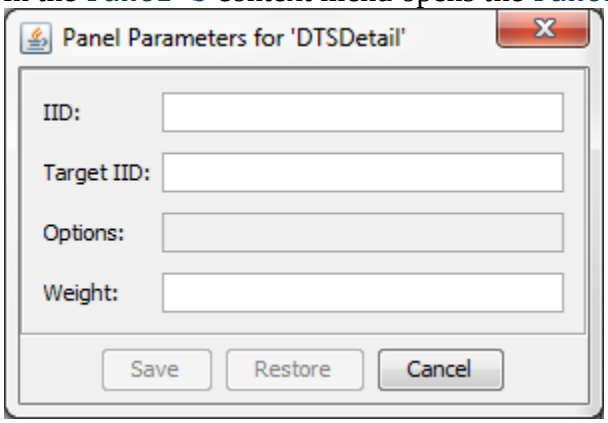


To associate a Module with an empty `Panel`, or change the Module currently associated with a `Panel`, drag a Module entry from the `Module List` panel and drop it on the `Panel` area. (The `TabPanel`, `Horizontal MultiPanel`, `Vertical MultiPanel` and Divider pseudo-Modules cannot be dropped in a `Panel` element.) The name of the Module will be displayed in the middle of the `Panel` area (see the `Default DTS Layout` example at the beginning of this section).

A number of `Panel` actions are available from the `Panel's` right-click context menu. To open the menu, right-click anywhere in the panel. (The context menu is not available for an empty `Panel`.) As described below, some of these actions may not be present in all `Panel` contexts.

- `Edit` – opens the `Panel Parameters` dialog. See details below.
- `Remove` – opens a confirmation dialog to remove the `Panel` from its parent element.
- `Wrap with Tab Panel` – replaces this `Panel` element with a `TabPanel` element and adds the original `Panel` to the `TabPanel`.
- `Wrap With Horizontal MultiPanel` – replaces this `Panel` element with a horizontally-oriented `MultiPanel` element and adds the original `Panel` to the `MultiPanel`.
- `Wrap with Vertical MultiPanel` – replaces this `Panel` element with a vertically-oriented `MultiPanel` element and adds the original `Panel` to the `MultiPanel`.
- `Unwrap TabPanel` – replaces this `Panel's` parent TabPanel element with the `Panel`. This action is only present if the `Panel` is the only element in the parent `TabPanel`.
- `Unwrap MultiPanel` – replaces this `Panel's` parent MultiPanel element with the `Panel`. This action is only present if the `Panel` is the only element in the parent `MultiPanel`.

Selecting the `Edit` item in the `Panel's` context menu opens the `Panel Parameters` dialog:



This dialog shows parameters common to all deployed Modules or Module components:

- `IID` is an optional Instance Identifier for this `Panel`. Use of this element is described later in the *Module Events* section of this document.
- `Target IID` is an optional Target Instance Identifier for the `Panel`. Use of this element is described later in the *Module Events* section of this document.
- `Options` is an optional parameter string passed to the Module or Module component when it is realized. This field is only enabled if the associated Module class file has designated that the Module accepts an option string. This string is described in the *Panel Tag* section above.
- `Weight` is a numeric parameter only enabled when the `Panel` is part of a `MultiPanel`. The Weight parameter is discussed in the *MultiPanel Tag* section above.

E.5.2 The TabPanel Element

A TabPanel element can exist as the lone element in a Layout or as a component of another TabPanel or a MultiPanel. The screen shot at the top of this section shows two TabPanels in the default DTS Editor Layout.

A TabPanel is displayed in normal tab panel style with a row of tabs at the top. To see the panel associated with any tab, just click on the desired tab in the tab area. The contents of a tab can be a Panel, another TabPanel or a MultiPanel.

TabPanels can be created in two ways:

1. By dropping the TabPanel pseudo-Module from the Module List panel into a TabPanel or MultiPanel, or
2. By selecting the Wrap with TabPanel context menu item from a Panel, TabPanel or MultiPanel.

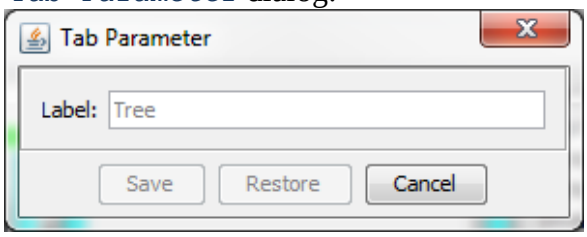
To add a Module Panel to an existing TabPanel, drag a Module or Module component from the Module List panel into the tab area of the TabPanel. If the drop position is over an existing tab, a new tab will be inserted at that position and the remaining tabs moved to the right. The new tab will contain a Panel holding the Module. If the drop position is in the "open" area past the last tab, the new tab will be added at end of the tab list. The label for the new tab will be the short name of the Module or Module component as given in the class file. This label can be changed by editing the Label field in the tab's edit dialog (see below).

TabPanels and MultiPanels can be added to the TabPanel in a similar manner by dropping these pseudo-Modules from the Module List panel. The default label of the new tab will be Tabn where n is the number of the tab position.

Tab can be reordered within the TabPanel using drag and drop. Drag any tab to another tab position, including the "open" position at the end of the tab area. When dropped, the tab will be moved to (inserted at) that position. Tab COPY is not available.

TabPanels have two context menus, both available by right-clicking in the tab area. If the click is on an existing tab, the context menu contains the Edit and Remove items.

Selecting Edit opens the Tab Parameter dialog:



This dialog shows the attribute associated with the Tab element:

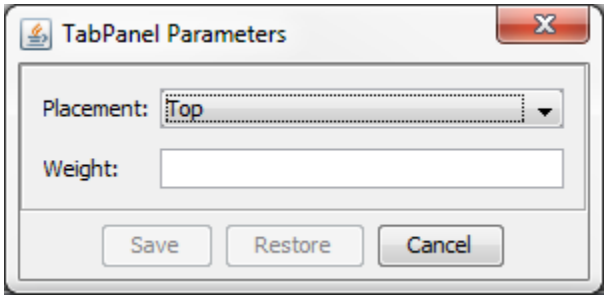
- `Label` specifies the label on the tab. The value defaults to the Module component's short name for Module `Panels`, or `Tabn` for `TabPanels` and `MultiPanels`, but the default can be overridden by entering an alternate value into the field.

Selecting `Remove` from the tab context menu opens a confirmation dialog to remove the tab, and its contents, from the `TabPanel`.

A right click on the right-most (open) part of the tab area opens the `TabPanel's` context menu. As described below, some of these actions may not be present in all `TabPanel` contexts.

- `Edit` – opens the `TabPanel Parameters` dialog. See details below.
- `Remove` – opens a confirmation dialog to remove the `TabPanel` from its parent element.
- `Wrap with Tab Panel` – replaces this `TabPanel` element with a new `TabPanel` element and adds the original `TabPanel` to the `TabPanel`.
- `Wrap With Horizontal MultiPanel` – replaces this `TabPanel` element with a horizontally-oriented `MultiPanel` element and adds the `TabPanel` to the `MultiPanel`.
- `Wrap with Vertical MultiPanel` – replaces this `TabPanel` element with a vertically-oriented `MultiPanel` element and adds the `TabPanel` to the `MultiPanel`.
- `Unwrap TabPanel` – replaces this `TabPanel's` parent TabPanel element with the `TabPanel`. This action is only present if the `TabPanel` is the only element in the parent `TabPanel`.
- `Unwrap MultiPanel` – replaces this `TabPanel's` parent MultiPanel element with the `TabPanel`. This action is only present if the `TabPanel` is the only element in the parent `MultiPanel`.

Selecting the `Edit` item in the `TabPanel's` context menu opens the `TabPanel Parameters` dialog:



This dialog displays the `TabPanel` attributes:

- `Placement` specifies the location of the tabs in the `TabPanel's JTabbedPane`. You can select `Top`, `Right`, `Bottom`, or `Left` from dropdown. The default is the top of the pane. The Layout Editor display reflects this selection.
- `Weight` is a numeric parameter only enabled when the `Panel` is part of a `MultiPanel`. The Weight parameter is discussed in the *MultiPanel Tag* section above.

E.5.3 The MultiPanel Element

A MultiPanel element can exist as the lone element in a Layout or as a component of a TabPanel or another MultiPanel. The screen shot at the beginning of this section shows two MultiPanels in the default DTS Editor Layout.

A MultiPanel is displayed as a horizontal or vertical sequence of constituent panels with each panel surrounded by a colored bar to the left and right (if a horizontal MultiPanel) or above and below (if a vertical MultiPanel). The colored bars are called Expanders. Expanders are not part of the realized MultiPanel, but are used by the Layout Editor to delimit the MultiPanel and act as droppable areas for new Modules (see below). When nesting MultiPanels, the panels use different colored Expanders to help distinguish each panel. The constituent panels of a MultiPanel can be any mix of Panels, TabPanels and MultiPanels.

MultiPanels can be created in two ways:

1. By dropping the Horizontal MultiPanel or Vertical MultiPanel pseudo-Module from the Module List panel into a TabPanel or MultiPanel, or
2. By selecting the Wrap with Horizontal MultiPanel or Wrap with Vertical MultiPanel context menu item from a Panel, TabPanel or MultiPanel.

To add a Module Panel to a MultiPanel, drag a Module or Module component from the Module List panel onto one of the MultiPanel's Expanders. The Expander will be replaced with a Panel containing the Module, and two new Expanders.

TabPanels and MultiPanels can be added to the MultiPanel in a similar manner by dropping these pseudo-Modules from the Module List panel.

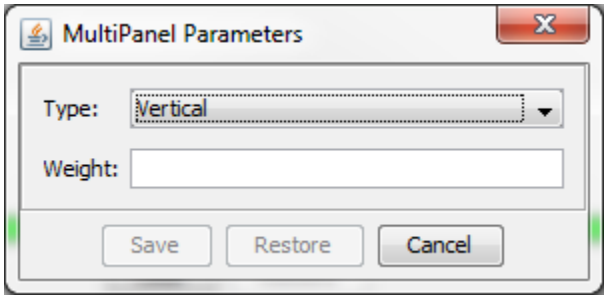
The Divider pseudo-Module in the Module List panel represents a movable border element that can be placed between any two Panels in a MultiPanel. When the Layout is realized, the Divider can be moved with the mouse to apportion the sizes of the adjacent Panels. To add a Divider to the MultiPanel, drop a Divider onto any Expander. Note that a Divider cannot be added at the beginning (above/left) or end (below/right) location in a MultiPanel. This condition will also be checked when the Layout is saved. A Divider has a single context menu item: Remove. Select this item to remove the Divider from the MultiPanel.

The MultiPanel's context menu is available by right clicking on any of the panel's Expanders. As described below, some of these actions may not be present in all MultiPanel contexts.

- Edit – opens the MultiPanel Parameters dialog. See details below.
- Remove – opens a confirmation dialog to remove the MultiPanel from its parent element.
- Wrap with Tab Panel – replaces this MultiPanel element with a new TabPanel element and adds the MultiPanel to the TabPanel.

- `Wrap With Horizontal MultiPanel` – replaces this `MultiPanel` element with a new horizontally-oriented `MultiPanel` element and adds the original `MultiPanel` to the new `MultiPanel`.
- `Wrap with Vertical MultiPanel` – replaces this `MultiPanel` element with a new vertically-oriented `MultiPanel` element and adds the original `MultiPanel` to the new `MultiPanel`.
- `Unwrap TabPanel` – replaces this `MultiPanel's` parent TabPanel element with the `MultiPanel`. This action is only present if the `MultiPanel` is the only element in the parent `TabPanel`.
- `Unwrap MultiPanel` – replaces this `MultiPanel's` parent MultiPanel element with the `MultiPanel`. This action is only present if the `MultiPanel` is the only element in the parent `MultiPanel`.

Selecting the `Edit` item in the `MultiPanel's` context menu opens the `MultiPanel Parameters` dialog:



This dialog displays the `MultiPanel` attributes:

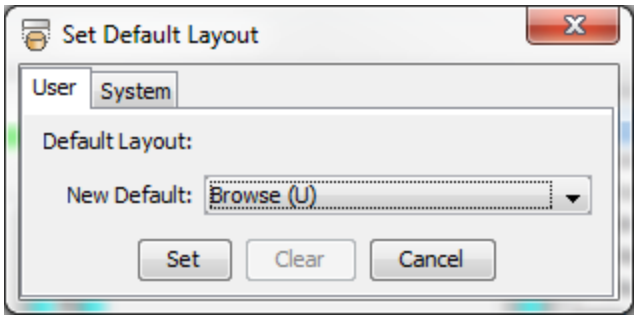
- `Type` specifies the orientation of the MultiPanel. You can select `Vertical` or `Horizontal` from the dropdown. The Layout Editor display reflects this selection.
- `Weight` is a numeric parameter only enabled when the `Panel` is part of a `MultiPanel`. The Weight parameter is discussed in the *MultiPanel Tag* section above.

E.6 Layout Editor Menus

The Layout Editor has two menus: `File` and `Configure`.

The `File` menu contains two items: `Import` and `Export`. These items convert between the internal DTS Layout formats and external, XML-based file formats as described in the *DTS Editor Layouts* section above. Layout files can be imported from, or exported to, the local client file system. These actions are most commonly used for the exchange of Layouts between users or between a user and a DTS Administrator for the purposes of promoting a User Layout to System status.

The `Configure` menu has only one item: `Set Default Layout`. Selecting the item opens the `Set Default Layout` dialog:



Layout defaults control what Layout is shown when a user first connects to a DTS server in a session. After connecting the first time, DTS looks for the user's default Layout. If one exists, this Layout is loaded. If the user has not designated a default Layout, DTS looks next for the default system Layout. (The default system Layout must be a System Layout). This Layout is set by a DTS Administrator and permits a common Layout to be loaded for all users. Finally, if a default system Layout has not been specified, DTS loads the standard, classic, Layout. As discussed earlier, this layout is maintained internally by DTS and cannot be modified.

To set a default user layout, select the desired Layout from the dropdown and click on the `Set` button. The selected Layout will be shown in the `Default Layout` line. To remove any default Layout, click on the `Clear` button.

The `System` tab in this dialog is used to set the default system Layout and is only available to DTS Administrators.

F. Module Events

In addition to standard DTS events such as connection events (`com.apelon.beans.dts.plugin.connection.*`) or knowledgebase events (subclasses of `com.apelon.dts.client.events.DataChangeEvent`), the DTS Editor Module Framework implements its own event class:

```
DTSEditorModuleEvent(String sourceIID, String targetIID,
      DTSEditorModuleEventType type, Object value)
```

The following sections provide further information on `DTSEditorModuleEvents`.

F.1 IIDs

Unlike standard Java events, the event source (initiating Module instance) and event target (destination Module instance) are denoted by Module Instance Identifier strings, or IIDs. IIDs are needed, rather than Module names, since it is common to have multiple "instances" of a Module (such as a popup, or floating, `Concept/Term Detail` panel in addition to the panel in the right tab pane), and a unique identifier is needed to refer to a specific Module instance.

IIDs are created by the DTS Editor whenever a Module must be instantiated, either during initialization as directed by the supplied Layout, or via a programmatic call. Editor-created IIDs are always unique integer Strings.

In addition to a Module instance's own IID, the instance maintains a single target IID which represents the destination IID for any events fired by the Module. A Module's target IID is assigned the null value by default, designating a "broadcast" event to all Modules. Module events fired by the DTS Editor itself have an "anonymous" source IID of null, and most Modules fire events with a target IID of null. TargetIIDs can, however, be set by the developer as described later in this section.

F.2 Event Types

Four types of events are defined in `com.apelon.apps.dts.editor.modules.DTSEditorModuleEventType`:

```
DTSEditorModuleEventType CURRENT_LOCAL_NAMESPACE_EVENT
DTSEditorModuleEventType STATUS_EVENT
DTSEditorModuleEventType TRANSFER_EVENT
DTSEditorModuleEventType EXIT_EVENT
```

The `CURRENT_LOCAL_NAMESPACE_EVENT` is fired by the DTS Editor on start-up, or when the user selects a new Current Local Namespace. Both the `sourceIID` and `targetIIDs` are always

null. The `value` is the name of the selected Namespace. The DTSStatus Module, for example, listens for this event and updates its display accordingly.

The `STATUS_EVENT` is a general information event. The `value` can be any object, although typically it is simply a message string. This string is shown in the DTSStatus Module message area.

The `TRANSFER_EVENT` is fired by a Module when it wishes to transfer a selected object to another panel, commonly due to an enabled "Click to Edit" option. The `sourceIID` is the initiating Module IID, the `targetIID` is usually null, and the value is a DTS Transferable, e.g. a `ConceptTransferable`, `TermTransferable`, `DTSPropertyTransferable`, etc. DTS transferables are all subclasses of the `com.apelon.beans.dts.plugin.transferable` package. Use of the `Transferable` interface can often facilitate processing of a `TRANSFER_EVENT` by mimicking existing Drag and Drop actions. See *Drop Functionality* in the Example section for more information.

The `EXIT_EVENT` is fired by the DTS Editor when the Editor is about to be closed. This event can be used to perform any required Module "clean-up".

F.3 Event Processing

All events are delivered to all Modules via the `DTSEditorModuleEventListener` interface. The default behavior for a Module listener is to process (recognize) any broadcast events (target is null) or any event for which the event's `TargetIID` is the same as the Module's IID. In order to provide fine-grained control of events, and to implement a general mechanism for inter-Module communication, there are ways of getting and setting both the source and target IIDs for a Module instance.

As described above, the DTS Editor assigns an integer string IID when a Module is instantiated. Developers can override this default assignment in the Layout file by using the `IID` and `TargetIID` elements in the `Tab` and `Panel` tags.

An example of IID assignment is the default DTS Editor `MultiPanel` repeated below:

```
<MultiPanel Type="V">
      <MultiPanel Type="H">
            <TabPanel>
                  <Tab Name="tree" ModuleName="DTSTree"
                  TargetIID=”detail”/>
                  <Tab Name="walker" ModuleName="DTSWalker"
                  TargetIID=”detail”/>
                  <Tab Name="search" ModuleName="DTSSearch"
                  TargetIID=”detail”/>
            </TabPanel>
            <Divider/>
            <TabPanel>
                  <Tab Name=”detail” ModuleName="DTSDetail"
                  IID=”detail”/>
            </TabPanel>
      </MultiPanel>
</MultiPanel>
```

Without the `TargetIID` assignments, `TRANSFER_EVENTs` generated by the `DTSTree`, `DTSWalker`, and `DTSSearch` panels would have their default target IID set to null and be “broadcast” to all panels. The `TargetIID` assignments, coupled with the explicit `IID` assignment on the `DTSDetail` panel, ensures that these events will only be accepted by the `DTSDetail` instance created at layout time.

See *Appendix A – DTS Editor Modules* for descriptions of event processing for all of the standard DTS Editor Modules.

F.4 Programmatic Setting of IIDs

IID values can be read programmatically using methods from the `DTSEditorModuleMgr` class instance passed to base Module classes during initialization (see the *Example Module* section below for details). The getter methods are:

```
public String getComponentIID(JComponent comp)
  public String getTargetIID(JComponent comp)
```

`JComponent comp` is the desired Module instance’s component, typically “`this`”.

There is no programmatic way to set a Module instance’s IID, but the target IID can be set when creating a new Module instance. Consider the case of a Module that wants to open a floating DTSDetail panel and then send Concepts for display (via a “Click to Edit” function); or a Module that wants to open a floating DTS Search panel and have search results sent back to itself. These Modules can invoke the floating panel, and set its target IID, via the following method in `DTSEditorModuleMgr`:

```
public String showModuleComponent(String compName,
HashMap<String,String> options, boolean modal, String title, String
targetIID)
```

This method displays a Module component as a floating dialog. It creates a new component instance (named by `compName`) passing an `options` Java HashMap of String key/value pairs, wraps it in a `JDialog`, and sets the Dialog's modality, title and the instance's target IID. The method returns the internally-generated IID of the new instance. Thus the Module can now send an event to the returned IID, or receive events from the Module. See the `DTSEditorModuleMgr` Javadoc for additional methods that deal with IIDs.

One clarification is needed for the `compName` argument. A Module usually, but not always, consists of a single menu option, optionally a single Toolbar item, and a single visible panel. There are, however, common exceptions:

- The DTSStatus Module includes only a panel. No menu or toolbar items are exposed.
- The DTSConnect Module builds three menu items: `Connect`, `Disconnect`, and `Connect Parameters`. Only one panel is exposed: that for `Connect Parameters`.
- The DTSSubset Module has two independent components, the `SubsetEditor` and `SubsetCompare`, with two associated menu items but only one toolbar item.

To accommodate Modules with multiple panels such as DTSSubset, the Module Framework defines **Module Components**. A Module Component is the logical name of a panel. When a Module has only one component, the Module name can be used as the Module Component name, e.g., `DTSDetail`. When there are multiple components, the Module Component name is the concatenation of the Module name, a period, and the panel name, e.g., `DTSSubset.SubsetEditor`.

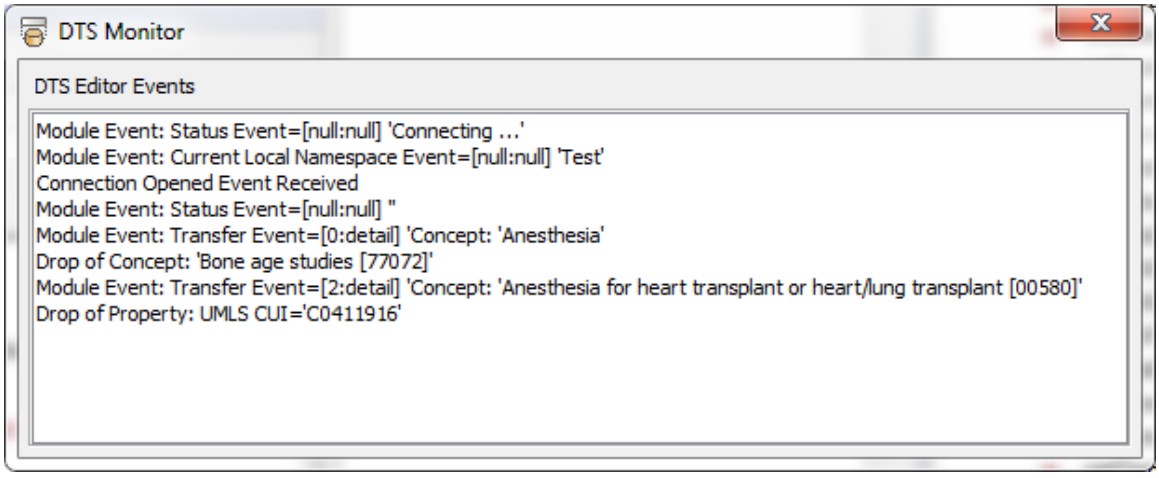
G. Internationalization

While not formally a part of the Module Framework, the DTS Editor has been extended to support "translation" (internationalization) of the Editor for non-English environments. Internationalization support includes resource files for translations of prompts and messages and layout considerations (top-to-bottom/bottom-to-top, left-to-right/right-to-left). Details on the procedures for internationalizing the DTS Editor, and on building internationalized Editor Modules, can be found in the ***Internationalizing DTS Guide***.

H. Example Module

This section will describe the process of creating a DTS Editor Module. The example is a module named "DTSMonitor" that displays descriptions of events fired during a DTS Editor session. (For ease of presentation, DTSMonitor does not follow internationalization policies.)

Here is a screen shot of the main DTSMonitor panel:



The complete DTSMonitor code can be found in your DTS installation at the following path:

`samples\editormodule\src\com\apelon\modules\dts\editor\monitor`

The `samples\editormodule` folder also contains readme and install.bat files. The latter file compiles the source, builds a jar and places the jar in your lib\modules folder where it will be recognized by the DTS Editor Module loader. As described below, the default Plug-in implementation adds an item in the Toolbar, an item in the Tools menu, an item in the Help menu, and an instance in the Editor right tab panel.

H.1 Module Registration

Like its Editor Plug-in predecessor, a Module base class must fulfill the following two conditions to be recognized by the DTS Editor.

1. The base class extends the `DTSEditorModule` class.
2. The base class (and all the associated java class files) must reside in a package recognized by the DTS Editor.

The simplest way for a prospective Module to be recognized is to create the Module in the `com.apelon.modules.dts.editor` package or in any of its subpackages (`com.apelon.modules.dts.editor.*`). This is the approach taken by the DTSMonitor Module:

Package

```
package com.apelon.modules.dts.editor.monitor;
```

The DTS Editor will also search in other packages that are specified in the DTS Editor configuration file (usually `bin/editor/dtseditor.xml`) as follows:

- Place the number of module packages to be searched in the `modulePackageCount` property. For example, if there are two modules:

```
<property name="modulePackageCount" value="2">
```

- Place the names of the module packages in `modulePackageName1..n` properties:

```
<property name="modulePackageName1" value="com.mycompany.dts.plugin"/>
<property name="modulePackageName2"
value="com.mycompany.dts.plugin2"/>
```

The DTS Editor will search packages in the following order:

```
modulePackageName1…n
com.apelon.modules.dts.editor
com.apelon.modules.dts.editor.*
```

H.2 Imports

A Module needs to import the `com.apelon.apps.dts.editor.modules` package. This package contains the following classes which contain fields and methods that provide access to DTS Editor functionality:

- `DTSEditorConfig` - Provides access to DTS Editor configuration settings.
- `DTSEditorModule` – Must be extended by your module in order to be recognized as a plug-in.
- `DTSEditorModuleEvent` – Defines DTSEditor Module events.
- `DTSEditorModuleEventListener` – Defines the interface for receiving Module events.
- `DTSEditorModuleEventType` – Defines the types of Module events.
- `DTSEditorModuleMgr` – Allows the Module to access DTS Editor functions and DTS Services.
- `DTSModuleConfig` – Used to access and save module specific properties.

The sections below describe how these classes are used in a typical custom Module. See the DTS Javadoc for full information on the classes.

```
// Apelon Imports
import com.apelon.apps.dts.editor.modules.*;
```

H.3 Class Declaration

`DTSEditorModule` class must be extended to create a custom Module for the Editor. **The extending class must have a no parameter constructor or the class will not be recognized as a Module.**

Modules must override the `getModuleName()`, `getModuleVersion()`, and `getComponentShortName(String name)` methods. The `getModuleName` method is called at Module registration. The other methods are called as described later in this section. As mentioned earlier, the Module short name is typically used as a tab name for `TabPanels` (see the *Editor Layout* section).

Modules should also override, or at least confirm the default implementation, of the five informational Layout methods. These methods are provided to assist Layout editors, in particular the DTSLayout Module, to make Modules available in menus, toolbars, and layouts. (See the *Layout Editor* section above for further information.) The methods are only called by Layout editors. The five methods are:

- `getComponentNames()` This method returns an array of Strings representing the components exposed by the Module. The strings should include the Module name and the names of any Module components. The default implementation returns a single-entry array with the Module name.
- `isMenuComponent(String name)` This method returns a Boolean designating whether the named Module/Module component can server as a menu item. This method should support all names returned by `getComponentNames` and the return should be consistent with the values returned by `getModuleMenuItems`. The default implementation returns `true` for the Module name and `false` otherwise.
- `isToolbarComponent(String name)` This method returns a Boolean designating whether the named Module/Module component can server as a toolbar item. This method should support all names returned by `getComponentNames` and the return should be consistent with the values returned by `getModuleToolbarItems`. The default implementation returns `true` for the Module name and `false` otherwise.
- `isLayoutComponent(String name)` This method returns a Boolean designating whether the named Module/Module component can server as a layout item (has a callable component). This method should support all names returned by `getComponentNames` and the return should be consistent with the values returned by `getModuleComponent`. The default implementation returns `true` for all names.
- `hasOptions(String name)` This method returns a Boolean designating whether the named Module/Module component accept options. This method should support all names returned by `getComponentNames` and the return should be consistent with the option parameter passed in `getModuleComponent`. The default implementation returns `false` for all names.

```
Class Declaration
public class DTSMonitor extends DTSEditorModule
      implements DtsConnectionListener {

/** Component names */
private final static String MODULE_NAME = "DTSMonitor";
private final static String VERSION = "1.2";
private final static String SHORT_NAME = "Monitor";
private final static String HELP_NAME = "DTS MonitorHelp";
private final static String CONFIG_NAME = "DTSMonitorConfig";

public DTSMonitor() {
      super();
}

public String getModuleName() {
      return MODULE_NAME;
}

public String getModuleVersion() {
      return VERSION;
}

public String getComponentShortName(String name) {
      if (name.equals(getModuleName())) return SHORT_NAME;
            return "";
}

//default implementations are correct for layout information methods
```

H.4 Initializing the Module

As previously described, only `getModuleName` is called at Module registration. The `getModuleVersion` is called as required for informational purposes. Most other Module methods are invoked when the Module is "loaded", either because the Module is explicitly named in the Layout, or it is requested as a plug-in. The specific sequence of method calls is different for these two cases. The description immediately below is for the Layout use-case. A subsequent section will describe the plug-in case.

The sequence of actions taken by the Editor loader for a Layout use-case is:

- If a Module configuration file is specified by `getDTSModuleConfigFile()`, a `DTSModuleConfig` object is set up. This object is available via the `getDTSModuleConfig` method.
- The `initModule` method is called.
- If the Module name is specified in a `Menu` tag, menu items returned by `getModuleMenuItems` are added to the menu

- If the Module name is specified in a `ToolBar` tag, toolbar items returned by `getModuleToolbarItems` are added to the toolbar.
- The `getModuleComponent` method should be implemented if the Module name is specified in the `PanelLayout` tag, or if floating panels are displayed via the `DTSEditorModuleMgr.showModuleComponent` method.

Details of these steps are described below.

A Module can use a private configuration file to hold operational parameters/policies that can persist across DTS Editor instances. To use a configuration file, a Module must override `getDTSModuleConfigFile` to return the name of the configuration file. Typically, the file is located in the default Editor directory which is `bin/editor`, and `getDTSModuleConfigFile` returns just the file name (no path). The DTS Editor loader then opens the file, creating it if it does not exist, and creates a `DTSModuleConfig` object which is available to the Module through the `getDTSModuleConfig` method. We will describe the use of the configuration file later in this example.

All plug-ins must implement `initModule`. This method is called by the DTS Editor at start-up and serves three purposes:

1. Provides the Module with a copy of the `DTSEditorModuleMgr` class;
2. Informs the Module whether it is being initialized in the Layout or Plug-in mode; and
3. Allows for any Module-specific initialization steps to occur.

The reference to the `DTSEditorModuleMgr` class is the Module's interface to the exposed functionality of the DTS Editor. The `DTSEditorModule.UsageType` parameter is one of the `UsageType` values `LAYOUT` or `PLUGIN`. DTSMonitor is written to permit its use in either Layout or Plug-in modes. The primary difference is that in Plug-in mode, the Module must explicitly place any components in the layout.

The following steps occur in the `initModule` implementation in the DTSMonitor example:

- Invokes the super class method to perform common initialize steps
- If in Plug-in mode, place the DTSMonitor panel in the Editor layout (additional details will be provided in a later section)

The superclass `initModule` method performs two actions:

1. Initializes the `moduleManager` instance variable to hold a reference to the supplied `DTSEditorModuleMgr` parameter. This value is frequently required within the custom Module class.
2. Registers a `DtsConnectionListener` on the instance. This enables connection events to be handled by the base Module class such as enabling and disabling of menu items.

```
Initialization

DTSEditorModuleMgr moduleMgr;

public String getDTSModuleConfigFile() {
      return "DTSMonitor.xml";
}

public void initModule(DTSEditorModuleMgr mgr,
                  DTSEditorModule.UsageType usage) {

// let the base class do its thing
super.initModule(mgr, usage);

//if in plug-in mode, add to layout
if (usage==DTSEditorModule.UsageType.PLUGIN){
      addMonitorComponent();
}
}
```

H.5 Getting Module Menu and Toolbar Items

After calling the `initModule` method, the DTS Editor retrieves Module menu and/or toolbar entries if such entries are requested in the Layout. Action listeners need to be added to each item to provide the desired functionality, usually opening of a floating panel.

The `getModuleMenuItems` method passes a parameter which is the name of the associated menu. Since most Modules only place menu items in one menu, usually this parameter can usually be ignored, but DTSMonitor includes a "help" menu item so the menu name is tested.

The `buildMenuItem` and `buildToolbarItem` methods are "helper" methods provided by `DTSEditorModule` to simplify creation of select (menu and toolbar) items. See the `DTSEditorModule` Javadoc for details.

The `registerSelectItems(JComponent[] items)` method is provided by `DTSEditorModule` to support enabling and disabling of select items. This method simply returns the array of `JComponents` passed in so it can be used "in-line". The `buildMenuItem` and `buildToolbarItem` methods described above disable their returned items. This is typically appropriate since most select items are disabled before a server connection is established. `registerSelectItems` adds the argument items to a `selectItems` List. When a DTS `connectionOpened` event is received, all items on this list are enabled. When a DTS `connectionClosed` event is received, the items are disabled. Individual Modules can, of course, provide their own select item processing by overriding the `connectionOpened` and `connectionClosed` events and/or not using `registerSelectItems`. Remember that a given Layout can have multiple instances of a menu or toolbar item, so single instance variables with these items should not be used.

Menu Item Code

```
public JMenuItem[] getModuleMenuItems(String menuName) {
  if (menuName.equalsIgnoreCase("help")) {
    return (JMenuItem[])registerSelectItems(
             new JMenuItem[] { getDTSMonitorHelpMenuItem() });
  }
  else {
    return (JMenuItem[])registerSelectItems(
           new JMenuItem[] { getDTSMonitorMenuItem(),
                             getDTSMonitorConfigMenuItem() };
  }
}

private JMenuItem getDTSMonitorMenuItem() {
  return buildMenuItem("DTS Monitor …", -1,
                 "Open DTSMonitor",
                 null, null, MODULE_NAME,
                new ActionListener() {
                     public void actionPerformed(ActionEvent e) {
                       dtsMonitorAction(e);
                     }
                  }
                );
}

private JMenuItem getDTSMonitorConfigMenuItem() {
  return buildMenuItem("DTS Monitor Configuration …", -1,
                "Open DTSMonitor Configuration",
                null, null, CONFIG_NAME,
                new ActionListener() {
                  public void actionPerformed(ActionEvent e) {
                       dtsMonitorConfigAction(e);
                  }
                }
             );
}

private JMenuItem getDTSMonitorHelpMenuItem() {
   return buildMenuItem("DTS Monitor Help …", -1,
                "Open Help for DTS Monitor ",
                 null, null, HELP_NAME,
                 new ActionListener() {
                     public void actionPerformed(ActionEvent e) {
                       dtsMonitorHelpAction(e);
                     }
               }
             );
}
```

Toolbar Item Code

```
public JComponent[] getModuleToolbarItems() {
  return registerSelectItems(
new JComponent[] { getDTSMonitorToolbarItem() });
}

private JComponent getDTSMonitorToolbarItem() {
  URL url = getClass().getResource("apelicon16.gif");
  ImageIcon toolbarImage = new ImageIcon(url, "GIF");
  return buildToolbarItem(toolbarImage, "Open DTSMonitor",
           new ActionListener() {
             public void actionPerformed(ActionEvent e) {
               dtsMonitorAction(e);
             }
         }
      );
}
```

H.6 Getting Plug-in Menus, Menu Items and Toolbar Items

In Plug-in mode, the Module can specify whole new menus, menu items and toolbar items. For Plug-in use, the `getModuleToolbarItems` method is called to retrieve any plug-in toolbar items. The method is the same as that used in Layout mode. For menu items, however, the `getModuleMenuItems` method has an integer parameter, corresponding to a particular menu group in the standard DTS Editor layout. The menu group parameter value is defined in `DTSEditorModule`:

```
public static final int FILE_MENU_ITEMS = 1;
public static final int VIEW_MENU_ITEMS = 2;
public static final int TOOLS_MENU_ITEMS = 3;
public static final int OPTIONS_MENU_ITEMS = 4;
public static final int HELP_MENU_ITEMS = 5;
```

Here is the code for creating the DTSMonitor Plug-in menu items:

Menu Item Code

```
public JMenuItem[] getModuleMenuItems(int group) {
// Create an empty array of JMenuItems to hold our custom items
      JMenuItem[] menuItems = new JMenuItem[0];

// Create a list for the desired menu group
switch (group) {
       case TOOLS_MENU_ITEMS:
 menuItems = new JMenuItem[] {getDTSMonitorMenuItem(),
 getDTSMonitorConfigMenuItem()};
       break;
       case HELP_MENU_ITEMS:
            menuItems = new JMenuItem[]
{ getDTSMonitorHelpMenuItem()};
            break;
       default:
            break;
}
return (JMenuItem[])registerSelectItems(menuItems);
}
```

Again, `registerSelectItems` is used to automatically handle enabling and disabling these items.

Finally, while not used in the DTSMonitor Module, `getModuleMenus` can be used to place one or more complete menus in the DTS Editor Menu bar. Menus must already be populated with the desired menu items. Menu(s) will be placed to the left of the Help menu.

H.6.1 Setting Plug-in Components

As shown in the DTSMonitor `initModule` listing above, Plug-in modules must explicitly place their components in the standard DTS Editor layout. Two methods are provided in `DTSEditorModule` for this purpose:

```
public JTabbedPane getLeftTabbedPane()
public JTabbedPane getRightTabbedPane()
```

These methods return the left and right `JTabbedPane` instances for the standard DTS Editor layout. The following method shows how DTSMonitor places its main component for Plug-ins:

```
Component Placement Code
/** Add the component to the right tab pane */
private void addMonitorComponent() {
      JTabbedPane pane = moduleMgr.getRightTabbedPane();
      pane.addTab(getComponentShortName(getModuleName()),
            getModuleComponent(getModuleName(), null));
}
```

The second argument to the `getModuleComponent` method is a Java HashMap of String key/value pairs that is passed to the component (`null` in the above example). This `options` HashMap argument is often generated from the `Options` element in the `Panel` and `Tab` tags in the DTS Editor Layout file as described in the *Panel Tag* section above. The interpretation of the option values is entirely up to the component.

H.7 Handling Connection Events

A Module can respond to the DTS Editor's server connection events by overriding the default `connectionOpened`, `connectionWillClose`, `connectionClosing`, and `connectionClosed` methods provide in `DTSEditorModule`. This is typically not required unless special select item handling is necessary. DTSMonitor does not override the default methods.

H.8 Initialization Summary

A Module needs to implement `initModule` and optionally, `getDTSModuleConfigFile`. If there is a GUI, the Module should provide access by implementing `getModuleMenus` (if a Plug-in), `getModuleMenuItems` and/or `getModuleToolbarItems`. To support enablement of select items, the Module should use the `registerSelectItems` method or provide its own custom handling of items and connection events

The next step is to design and implement Module panel functionality.

H.9 Creating a Panel for the Module

Although not required, a custom Module is likely to be GUI-based. This will require the definition of one or more panels that will be the user interface to the Module's functionality. The panels access DTS and DTS Editor functionality (APIs) via the copy of the `DTSEditorModuleMgr` class which was passed to the Module in the `initModule` method.

Layout panels are accessed via the `getModuleComponent` method. Floating panels use the `DTSEditorModuleMgr.createDialog` or the `DTSEditorModuleMgr.showComponent` methods. All these methods require creation of a `JPanel` that contains the Module GUI.

The DTSMonitor Module implements a Layout and floating panel for displaying events, and a configuration panel for specifying the types of events to be displayed. A `DTSEditorModuleMgr` helper method is used to show the help URL. This is the code to invoke the panels:

Panel Handling Code

```
/** Open Monitor panel */
private void dtsMonitorAction(ActionEvent e) {
      final DTSMonitorPanel panel = getDTSMonitorPanel();
      JDialog dialog = moduleMgr.createDialog(panel,
            "DTS Monitor", 0,);
      dialog.setModal(false);
      dialog.setVisible(true);
}

/** Open Monitor Configuration Panel */
private void dtsMonitorConfigAction(ActionEvent e) {
      DTSMonitorCfgPanel panel = getDTSMonitorConfigPanel();
      JDialog dialog = moduleMgr.createDialog(panel,
            "Configure DTSMonitor", 0, 0);
      dialog.setModal(true);      //modal
      dialog.setVisible(true);
}

/** Show the DTSMonitor help file */
private void dtsMonitorHelpAction(ActionEvent e) {
      URL url = getHelpRelativePath();
      moduleMgr.showHelpPanel(url);
}

/** get a new DTSMonitorPanel */
private DTSMonitorPanel getDTSMonitorPanel() {
      return new DTSMonitorPanel(moduleMgr, getDTSModuleConfig());
}

/** get a new DTSMonitorConfigPanel */
private DTSMonitorCfgPanel getDTSMonitorConfigPanel() {
      return new DTSMonitorCfgPanel(moduleMgr,
      getDTSModuleConfig());
}
```

H.10 DTSMonitor Functionality

The objective of the DTSMonitor Module is to display/report the various events that are available in the DTS Editor environment. Four types of events are supported:

- DTS Connection Events
- DTS Editor Module Events
- DTS Data Change Events
- DTS Drop Events

A configuration panel is available to select/deselect event types for display. Standard Java Swing techniques and components are used in the implementation, and will not be reviewed here. Similarly, the reader is referred to the sample code for details on the specific output logic for the various event types.

Two points are worth noting. First, the constructor tests to see if a connection is already present. This is typically required when the Module can be used as a Layout panel, since it cannot be assumed that a connection is open when the Layout is built, and data listeners can only be added when a connection is present. Second, the Module supports an associated Details popup panel. If a connection is already present, and the popup is specified in the configuration, a DTSDetails Module panel is opened using the familiar `SwingUtilities.invokeLater` pattern.

Panel Constructor Code

```
public class DTSMonitorPanel extends JPanel
            implements DtsConnectionListener,
                  DTSEditorModuleEventListener,
                    ConceptListener, TermListener,
                    KBTypeListener, ClassifyListener,
                    SubsetListener {
  private DTSEditorModuleMgr dtsModuleMgr;
  private DTSModuleConfig config;
  private JTextArea textArea;
  private String detailIID;           //IID of DTSDetail popup

  public DTSMonitorPanel(DTSEditorModuleMgr mgr,
                       DTSModuleConfig config) {
   super();
   dtsModuleMgr = mgr;
   this.config = config;
   try {
     buildPanel();
     addListeners();

     setTransferHandler(new MonitorTransferHandler());

     //if this is a floating panel, enable the data listeners
     if (DTSAppManager.getQuery().isOpen()) {
       enableDataListeners(true);
      detailIID = null;              //to be used later
      showPopup();
     }
   }
   catch (Exception ex) {
    dtsModuleMgr.handleException("Error constructing
DTSMonitorPanel", ex);
    dtsModuleMgr.showErrorMessage("Cannot construct
DTSMonitorPanel.");
   }
 }
. . .
 public void addListeners() {
  dtsModuleMgr.registerConnectionListener(this);
  dtsModuleMgr.registerModuleEventListener(this);
 }
```

```
private void openPopup()    //if enabled, open a popup DTSDetail
panel
   //defer so it doesn't get hidden by main panel
   if (config.getBooleanProperty(DTSMonitor.DETAIL_POPUP, false)) {
     SwingUtilities.invokeLater(new Runnable() {
       public void run() {
        detailIID = dtsModuleMgr.showModuleComponent("DTSDetail",
                              null, false, "DTS Monitor Details",
                                   null,                  //no target
needed
                              20, 20,              //relative location
                             300, 300);       //DTSDetail size
       }
     });
   }
 }
```

H.10.1 Connection Event Functionality

As shown above, the listener for Connection events is set up in the `DTSMonitorPanel` constructor. This listener can be registered independent of the connection status. The events themselves are processed by the `DtsConnectionListener` interface methods:

Connection Processing Code

```
public void connectionOpened(DtsConnectionEvent event) {
  enableDataListeners(true);
  if (config.getBooleanProperty(DTSMonitor.CONNECT_EVENTS, true)) {
    addEvent("Connection Opened Event Received");
    enableEventArea(true);
  }
  openPopup();
}

public void connectionWillClose(DtsConnectionEvent event,
                ConnectionCloseVeto vetoClose) {
  if (config.getBooleanProperty(DTSMonitor.CONNECT_EVENTS, true)) {
    addEvent("Connection Will Close Event Received");
  }
}

public void connectionClosing(DtsConnectionEvent event){
  if (config.getBooleanProperty(DTSMonitor.CONNECT_EVENTS, true)) {
    addEvent("Connection Closing Event Received");
  }
}

public void connectionClosed(DtsConnectionEvent event) {
  enableDataListeners(false);
  if (config.getBooleanProperty(DTSMonitor.CONNECT_EVENTS, true)) {
    addEvent("Connection Closed Event Received");
    enableEventArea(false);
  }
}
```

Note how the `DTSModuleConfig` object is used to "filter" the events to be displayed.

H.10.2 Module Event Functionality

The DTS Module event listener is similarly set up in the `DTSMonitorPanel` constructor. The listener interface is straightforward: it checks if Module events are to be reported, then prints the event in a standard format. The code to construct a `Transferable` string is shared with that for DnD:

Module Event Processing Code

```
public void eventOccurred(DTSEditorModuleEvent me) {
  if (config.getBooleanProperty(DTSMonitor.MODULE_EVENTS, true)) {
      //build a common preamble
      String preamble = "Module Event: "+me.getEventType()+"=["+
          (me.getSourceIID()==null?"null":me.getSourceIID())+":"+
          (me.getTargetIID()==null?"null":me.getTargetIID())+"] ";
      if
(me.getEventType()==DTSEditorModuleEventType.TRANSFER_EVENT) {
     addEvent(preamble+
       getTransferableDescription((Transferable)me.getValue()));
      }
   else {
     addEvent(preamble+"'"+me.getValue().toString()+"'");
      }
  }
}
```

H.10.3 Data Change Event Functionality

The DTSMonitor Module recognizes all six DTS "data" events :

- ConceptEvent
- TermEvent
- KBTypeEvent
- ClassifyEvent
- NamespaceEvent
- SubsetEvent

Listeners for these events can only be registered once a connection has been made, so the registration is encapsulated by an `enableDataListener()` method called from the connection listeners. Refer to the individual event listeners for details on event object reporting.

Data Change Processing Code

```
/**
 * Add/remove data listeners
 * @param enable true to register listeners, false to remove
 */

private void enableDataListeners(boolean enable) {
  if (enable) {

DTSAppManager.getQuery().getAssociationQuery().addConceptListener(th
is);

DTSAppManager.getQuery().getTermSearchQuery().addTermListener(this);

DTSAppManager.getQuery().getAssociationQuery().addKBTypeListener(thi
s);

DTSAppManager.getQuery().getClassifyQuery().addClassifyListener(this
);

DTSAppManager.getQuery().getNamespaceQuery().addNamespaceListener(th
is);

DTSAppManager.getQuery().getSubsetQuery().addSubsetListener(this);
}
  else {

DTSAppManager.getQuery().getAssociationQuery().removeConceptListener
(this);

DTSAppManager.getQuery().getTermSearchQuery().removeTermListener(thi
s);

DTSAppManager.getQuery().getAssociationQuery().removeKBTypeListener(
this);

DTSAppManager.getQuery().getClassifyQuery().removeClassifyListener(t
his);

DTSAppManager.getQuery().getNamespaceQuery().removeNamespaceListener
(this);

DTSAppManager.getQuery().getSubsetQuery().removeSubsetListener(this)
;
  }
}
    else {
      addEvent("Invalid Concept Event”);
    }
  }
}
```

```
/** Representative data event handler */
public void conceptActionOccurred(ConceptEvent event) {
  if (config.getBooleanProperty(DTSMonitor.DATA_EVENTS, true)) {
    int eventNum = event.getEventType();
    if (eventNum == ConceptEvent.EVENT_TYPE_NEW) {
      addEvent("Concept '"+event.getConcept().getName()+"' added");
    }
    else if (eventNum == ConceptEvent.EVENT_TYPE_MODIFY) {
      addEvent("Concept '"+event.getConcept().getName()+"'
modified");
    }
    else if (eventNum == ConceptEvent.EVENT_TYPE_DELETE) {
      addEvent("Concept '"+event.getConcept().getName()+"'
deleted");
    }
```

H.11 Drop Functionality

If enabled in the configuration file, the DTSMonitor Module reports drop events on the monitor panel. Much of the editing done in the DTS Editor relies on or is facilitated by DnD functionality. Support for DnD between existing DTS panels and custom Modules or between different custom Modules panels can greatly enhance the capabilities of each.

Various DTS objects can be dragged and dropped between the DTS Editor and a Module. These include Concept Association, Concept, Property, Role, Subset, Synonym, Term Association and Term. Each of these has a corresponding Transferable object such as `ConceptTransferable`, `TermTransferable`, etc. In turn, each of these Transferable objects contain certain `DataFlavors` that can be retrieved once the object is dropped.

For instance, if a `ConceptAssociationTransferable` is dropped, a Concept Association, DTS Concept or String object can be obtained and used in the plug-in. DTS transferable objects are all contained in subclasses of the `com.apelon.beans.dts.plugin.transferable` package. DnD support for standard DTS Modules is described in *Appendix A – DTS Editor Modules*.

DTSMonitor drop recognition is provided via an implementation of the Java `TransferHandler` class called `MonitorTransferHandler`. An instance of this class is added to `DTSMonitorPanel` in the constructor (see the DTSMonitor Functionality section above). To accept drops only the `canImport()` and `importData()` methods need be implemented. In the code below, the `isDataFlavorSupported()` method checks for a valid DTS Transferable flavor. If this call is successful, `importData()` extracts the transfer data and decodes the DTS object type using the (shared) `getTransferableDescription()` method.

Drop Functionality Code

```
// Handle a Drop on the panel
private class MonitorTransferHandler extends TransferHandler {

  //do we support this drop
  boolean canImport(JComponent comp, DataFlavor[] transferFlavors) {
       for (DataFlavor flavor : transferFlavors)
            if (isDataFlavorSupported(flavor)) return true;
       false;
  }

  //process a drop
  public boolean importData(JComponent comp, Transferable trans) {
       if (!config.getBooleanProperty(DTSMonitor.DND_EVENTS, true))
      return false;

       if (trans == null) {
         textArea.append("Drop failed. No data found in the drop
object.");
         return false;
       }
       addEvent("Drop of "+getTransferableDescription(trans));
       return true;
  }

  //test for supported flavors
  private boolean isDataFlavorSupported(DataFlavor flavor) {
    if (flavor.equals(DTSDataFlavor.multiFlavor)) return true;
       if (flavor.equals(DTSDataFlavor.conceptFlavor)) return true;
       if (flavor.equals(DTSDataFlavor.synonymFlavor)) return true;
       if (flavor.equals(DTSDataFlavor.propertyFlavor)) return true;
       if (flavor.equals(DTSDataFlavor.roleFlavor)) return true;
       if (flavor.equals(DTSDataFlavor.conceptAssociationFlavor))
return true;
    if (flavor.equals(DTSDataFlavor.termFlavor)) return true;
       if (flavor.equals(DTSDataFlavor.termAssociationFlavor))
return true;
       if (flavor.equals(DTSDataFlavor.subsetFlavor)) return true;
       return false;
   }

} //end of transfer handler
```

H.12 Showing Drop Details

Floating Module panels are usually invoked through Menubar or Toolbar item selections, but the DTS Editor Module Framework also provides a method for programmatically opening panels within Module code. The DTSMonitor Module implements an option to show a popup

DTSDetail panel and fill this panel from DnD Concept/Term events. The code to invoke a floating DTSDetail panel is show below. Invocation is based on a configuration setting and is deferred to let the UI settle.

Panel Invocation Code

```
if (config.getBooleanProperty(DTSMonitor.DETAIL_POPUP, true)) {
  SwingUtilities.invokeLater(new Runnable() {
    public void run() {
      detailIID = dtsModuleMgr.showModuleComponent(
             "DTSDetail",
             "",
             false,
             "DTS Monitor Details",
                  null,                //no target needed
             20, 20,             //Monitor-relative location
             300, 300);                //DTSDetail size
    }
  });
}
```

To direct the DnD transferable to the created panel, a Module event is fired whose target IID is the IID of the DTSDetail panel. This is the reason the IID is saved in the `detailIID` variable above. Here is the associated code from the DnD handler:

Transfer Event Code

```
if (detailIID!=null) {
  dtsModuleMgr.fireModuleEvent(
    new DTSEditorModuleEvent(
          dtsModuleMgr.getComponentIID(DTSMonitorPanel.this),
            detailIID,
          DTSEditorModuleEventType.TRANSFER_EVENT,
          trans));
}
```

H.13 Error Handling

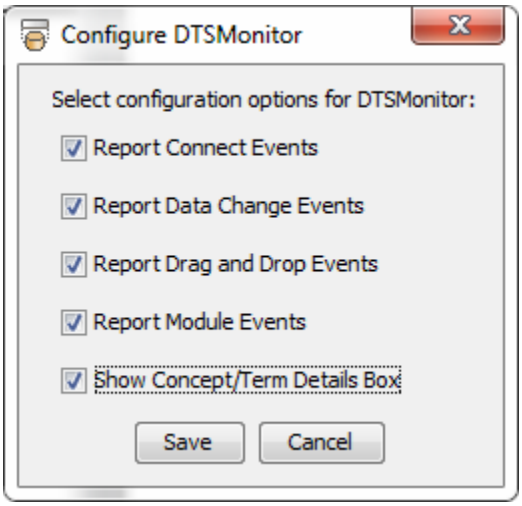
Modules can write error messages to the standard DTS Editor log file and also display popup message dialogs by using the `DTSEditorModuleMgr showErrorMessage()` and `handleException()` methods. The example below is taken from the `DTSMonitorCfgPanel` constructor.

Error Handling Code

```
public DTSMonitorCfgPanel(DTSEditorModuleMgr mgr,
             DTSModuleConfig config) {
  super(new BorderLayout());
  moduleMgr = mgr;
  this.config = config;
  try {
    buildPanel();
  }
  catch (Exception ex) {
    //log error and show error dialog
    moduleMgr.handleException(
            "Cannot construct DTSMonitorCfgPanel.", ex);
  }
}
```

H.14 Configuration Management

The `DTSMonitorCfgPanel` enables the selection of which of the four event types should be reported and whether a Details box should be opened for DnD events. These user selections are saved in the Module's configuration file, as specified by the `getDTSModuleConfigFile()` method. A screen shot of the panel is shown below:



This panel uses standard Java Swing components for the GUI and will not be described further here, but the code to load and unload the `JCheckBox` elements from and to the configuration file is shown below. Access to the configuration object is provided via the `DTSModuleConfig` object passed in the `DTSMonitorCfgPanel` constructor (see section above). Note that the defaults used in the boolean property getters should be the same as those in the `DTSMonitorPanel` event handling code to avoid inconsistent behavior.

Configuration Management Code

```
/**
 * Load the JCheckBoxes from the configuration file
 * Default is all true
 */
private void loadFromConfig() {
  connectBox.setSelected(config.getBooleanProperty(
                  DTSMonitor.CONNECT_EVENTS, true));
  dataBox.setSelected(config.getBooleanProperty(
                  DTSMonitor.DATA_EVENTS, true));
  dndBox.setSelected(config.getBooleanProperty(
                  DTSMonitor.DND_EVENTS, true));
  moduleBox.setSelected(config.getBooleanProperty(
                  DTSMonitor.MODULE_EVENTS, true));
  detailBox.setSelected(config.getBooleanProperty(
                  DTSMonitor.DETAIL_POPUP, true));

}

/** Unload the JCheckBoxes into the configuration file
 * Return success flag
 */
private boolean saveToConfig() {
  config.storeBooleanProperty(
    DTSMonitor.CONNECT_EVENTS, connectBox.isSelected());
  config.storeBooleanProperty(
    DTSMonitor.DATA_EVENTS, dataBox.isSelected());
  config.storeBooleanProperty(
    DTSMonitor.DND_EVENTS, dndBox.isSelected());
  config.storeBooleanProperty(
    DTSMonitor.MODULE_EVENTS, moduleBox.isSelected());
  config.storeBooleanProperty(
    DTSMonitor.DETAIL_POPUP, detailBox.isSelected());

  try {
    config.saveProperties();
    return true;
  } catch (Exception e) {
    String err = "Unable to save DTSMonitor congiguration.";
    Categories.uiView().error(err, e);
    moduleMgr.showErrorMessage(err);
    return false;
  }
}
```

I. Converting Plug-in Modules

I.1 Converting a Pre-V4 Plug-in

The process to convert a pre-V4 plug-in module to a V4.3 Module is straightforward. Simply perform the following steps in the Module's base class:

1. Replace the previous `initModule` method with

   ```
   initModule(DTSEditorModuleMgr mgr, UsageType usage).
   ```

2. Add the `getModuleName()`, `getModuleVersion()`, and `getComponentShortName(String name)` methods.
3. Add the `getModuleComponent(String compName, HashMap<String,String> options)` method if a Layout component can be specified or the `DTSEditorModuleMgr.showModuleComponent()` method is used.
4. Add the `getModuleMenuItems(String menuName)` method.
5. See the following sections for additional, version-specific, steps.

I.2 Converting to V4.3

V4.3 added a number of new `DTSEditorModule` methods to support Layout editors and simplify management of menu and toolbar items. None of these additions are required, all operations are backwards compatible, but review of the features is recommended to provide more accurate representation of Module capabilities in Layout editors and elimination of potentially redundant code.

1. Review and add, if necessary, implementations of the five Layout information methods: `getComponentNames()`, `isMenuComponent()`, `isToolbarComponent()`, `isLayoutComponent()`, and `hasOptions()`.
2. Add `super.initModule(mgr, usage)` to the local `initModule` method.
3. Delete any local instance variable for the Module's `DTSEditorModuleMgr` and replace references with `moduleManager`.
4. Delete any local `DtsConnectionListener` registration in favor of that from the parent class.
5. Delete any menu or toolbar item instance variables and use `registerSelectItems` to maintain item references. Review use of custom `connectionOpened` and `connectionClosed` handlers for select items enablement versus that provided by the parent class. See the descriptions in Getting Module Menu and Toolbar Items above for further information.

J. Appendix A – DTS Editor Modules

This Appendix describes the Modules provided in the standard DTS Editor distribution. The descriptions include details on access options, events Drag and Drop (DnD) support, and Module options.

Module Name	Short Name	Access	Events	DnD Capability	Options
DTSAbout: A panel that displays information about the DTS Editor.	About	Menu item and floating panel	None	None	None
DTSAssociation: A panel for creating and editing DTS Associations.	Associations	Menu item, toolbar item, floating panel and layout panel	Fires DTS `ConceptEvents`	Drag and Drop is supported in `From Concept/Term` and `To Concept/Term` combos. Drop of Associations, Synonyms and Roles are also supported (dropped value is the Target).	None
DTSAuthority: A panel for creating and editing DTS Authorities.	Authorities	Menu item, floating panel and layout panel	None	Drag of Authorities from edit panel is supported.	None
DTSClassify: A panel for performing classification of OntylogExtension Namespaces.	Classify	Menu item, toolbar item and floating panel.	None	Drag of Concepts from error tabs is supported.	None
DTSClipboard: A panel for modifying the "Cut" string value for DTS objects.	Clipboard	Menu item and floating panel.	None	None	None
DTSConfiguration: A panel for viewing and setting DTS Editor configuration options	Configuration	Menu item and floating panel	None	None	None
DTSConnect: A set of operations (`Connect`, `Disconnect`, and `Connect Parameters`) for handling	Connect	Three menu items, two toolbar items (`Connect` and `Disconnect`). `Connect`	Fires `DTSConnectionEvents`	None	None

connection to and disconnection from a DTS Knowledgebase.		`Parameters` floating panels.			
DTSDetail: A panel for displaying the details for DTS Concepts and Terms.	Detail	Menu item, toolbar item, floating panel and layout panel	Responds to incoming Concept/Term `TRANSFER_EVENTs` and loads the panel with the Concept/Term if the event's Target IID is the IID of the DTSDetail instance.	Loads the associated tab panel on drop of Concepts, Terms, Namespaces, Authorities and Subsets. Drops of Synonyms, Roles and Associations load the associated target Term/Concept. Copy (`Ctrl`) drop of Synonyms, Properties, Roles, and Associations, adds a copy of the attribute to the focus as permissible. Drag supported from focus Concept, Term, Namespace, Authority, and Subset as well as Synonyms, Properties, Roles and Associations.	The following options are supported: do - display-only cn - Concept tab tm – Term tab ns – Namespace tab au – Authority tab sb – Subset tab pos:t – tabs on top (default) pos:r – tabs on right pos:b – tabs on bottom pos:l – tabs on left
DTSExit: Executes an exit operation from the DTS Editor application.	Exit	None	Fires an `EXIT_EVENT` on selection	None	None
DTSHelp: A panel that displays the DTS Editor help topics.	Help	A menu item and floating panel	None	None	None
DTSLayout: A panel for editing and selecting Layouts.	Layout	Two menu items, and a floating panel	None	None	None
DTSLocalNamespace: A panel that permits selection of the Current Local Namespace.	Local Namespace	Menu item, toolbar item and floating panel	Fires `CURRENT_LOCAL_NAMESPACE_EVENT`	None	None
DTSModuleManager: A panel for adding, updating and deleting Modules.	Module Manager	Menu item and floating panel	None	None	None
DTSMonitor: A demonstration panel that displays DTSEditor events.	Monitor	Menu item, toolbar item, floating panel and layout panel	On receipt of an object drop in the Module, fires a `TRANSFER_EVENT` to a Module-subordinate Detail Panel if so enabled	Drop of all objects is supported.	None

			in the DTSMonitor configuration file.		
DTSNamespace: A panel for creating and editing of Namespaces, including Namespace and Version Properties.	Namespaces	Two menu items, two floating panels and one layout panel (`NamespaceEditor`)	Fires DTS `NamespaceEvents`	Drag of Namespaces from edit panel is supported.	None
DTSNotificationsConfig: A panel for configuring server (Pub/Sub) notifications.	Options	One menu item and one floating panel	None	None	None
DTSPreference: A set of menu items for configuring/resetting DTS Editor panel preferences.	Preference	Menus Items	Preference	Menu Items	None
DTSProperty: A panel for creating and editing Properties.	Properties	Menu item, toolbar item, floating panel and layout panel	Fires DTS `ConceptEvents`	Drag and Drop supported by `Concept/Term` combo. Drop of Associations, Synonyms, and Roles are also supported (dropped value is the Target).	None
DTSSearch: A panel for searching the DTS Knowledgebase.	Search	Menu item, toolbar item, floating panel and layout panel	Fires `TRANSFER_EVENT` on concept selection if the *Click to Edit* option is enabled.	Drag and Drop supported by `Concept/Term` combo. Drop of Associations, Synonyms, and Roles are also supported (dropped value is the Target). Drag supported from any Concept/Term in the results window.	None
DTSStatus: The information panel typically displayed at the bottom of the DTS Editor.	Status	A layout panel.	Listens for the `TRANSFER_EVENT`, `CURRENT_LOCAL_NAMESPACE`, and `STATUS_EVENT` events and displays event content.	None	None
DTSSubset: Panels for creating, editing and comparing subsets.	Subset	Three menu items, one toolbar item, three floating panels and two layout panels (`SubsetEditor` and	Fires DTS `SubsetEvents`.	Drag of Subsets from edit panel is supported.	None

		SubsetCompare).			
DTSSynonym: A panel for creating and editing Synonyms.	Synonyms	Menu item, toolbar item, floating panel and layout panel	Fires DTS ConceptEvents.	Drag and Drop is supported by Concept and Term combos. Drop of Associations, Synonyms, and Roles are also supported (dropped value is the Target).	None
DTSTree: A panel that displays Namespaces and Namespace trees.	Tree	A menu item, toolbar item, floating panel and layout panel	Fires a Namespace/Concept TRANSFER_EVENT on selection if the *Click to Edit* option is enabled. Responds to incoming Concept TRANSFER_EVENTs and loads the tree with the Concept if the event's Target IID is the IID of the DTSTree instance.	Drag and Drop is supported by the Focus Concept combo. Drop of Associations and Roles are also supported (dropped value is the Target). Drop into the tree window loads tree for Concept. Drag supported from any Concept in tree window. Drag supported from any Namespace or Concept in tree window.	None
DTSTypes: A panel for creating and editing Attribute Types.	Types	Menu item, floating panel and layout panel	Fires DTS KBTypeEvents.	None	None
DTSUserManager: A panel that supports creation and editing of DTS Roles and assignment to Users.	UserManager	Menu item and floating panel.	None	None	None
DTSVersion: Panels for comparing Concept Versions and viewing Concept History.	Version	Two menu items, two floating and two layout panels (ConceptCompare and SubsetCompare).	None	Drop is supported by Concept combos in both panels.	None
DTSWalker: A panel that displays expandable tree views of a focus Concept's parents and children.	Walker	Menu item, toolbar item, floating panel and layout panel	Fires a Concept TRANSFER_EVENT on selection if the *Click to Edit* option is enabled. Responds to incoming Concept	Drop is supported by the Focus Concept combo and both tree views: effect is to set the Focus Concept. Drop of Associations and Roles are also supported (dropped value is the	None

			TRANSFER_EVENTs and loads the focus concept with the Concept if the event's Target IID is the IID of the DTSWalker instance.	Target). Drag supported from the Focus Concept combo and any Concept in the tree windows.	

K. Appendix B – Standard DTS Editor Layout File

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Layout V2 Default DTS Layout saved by dtsadmin on 12 Apr 2015
14:37:20 -
   ->
<Layout EnablePlugins="true" Description="Apelon Standard Layout"
Width="0"
   Height="0" >  <MenuBar>
      <Menu Command="file" Mnemonic="f" Name="File">
        <MenuItem ModuleName="DTSConnect"/>
        <Separator/>
        <SelectItem ModuleName="DTSExit"/>
      </Menu>
      <Menu Command="tools" Mnemonic="t" Name="Tools">
         <SelectItem ModuleName="DTSTree"/>
         <SelectItem ModuleName="DTSSearch"/>
         <SelectItem ModuleName="DTSWalker"/>
         <Separator/>
         <SelectItem ModuleName="DTSDetail"/>
         <SelectItem ModuleName="DTSAssociation"/>
         <SelectItem ModuleName="DTSProperty"/>
         <SelectItem ModuleName="DTSSynonym"/>
         <Separator/>
         <SelectItem ModuleName="DTSTypes"/>
         <Separator/>
         <SelectItem ModuleName="DTSNamespace"/>
         <SelectItem ModuleName="DTSSubset"/>
         <SelectItem ModuleName="DTSAuthority"/>
         <Separator/>
         <SelectItem ModuleName="DTSClassify"/>
         <SelectItem ModuleName="DTSVersion"/>
      </Menu>
      <Menu Command="options" Mnemonic="o" Name="Options">
         <SelectItem ModuleName="DTSClipboard" />
         <SelectItem ModuleName="DTSConfiguration"/>
         <SelectItem ModuleName="DTSLayout" />
         <SelectItem ModuleName="DTSLocalNamespace"/>
         <SelectItem ModuleName="DTSModuleManager"/>
         <SelectItem ModuleName="DTSPreference"/>
         <SelectItem ModuleName="DTSUserManager"/>
      </Menu>
        <Menu Command="help" Mnemonic="h" Name="Help">
         <SelectItem ModuleName="DTSHelp"/>
         <Separator/>
         <SelectItem ModuleName="DTSAbout"/>
      </Menu>
  </MenuBar>
  <ToolBar>
```



```
    <SelectItem ModuleName="DTSConnect"/>
    <Separator/>
    <SelectItem ModuleName="DTSTree"/>
    <SelectItem ModuleName="DTSSearch"/>
    <SelectItem ModuleName="DTSWalker"/>
    <Separator/>
    <SelectItem ModuleName="DTSLocalNamespace"/>
    <SelectItem ModuleName=”DTSClassify” />
    <Separator/>
    <SelectItem ModuleName="DTSDetail"/>
    <SelectItem ModuleName="DTSAssociation"/>
    <SelectItem ModuleName="DTSProperty"/>
    <SelectItem ModuleName="DTSSynonym"/>
    <Separator/>
    <SelectItem ModuleName="DTSSubset"/>
    <Separator/>
    <SelectItem ModuleName="DTSHelp"/>
  </ToolBar>
  <PanelLayout continuousLayout="true">
    <MultiPanel Type="v" >
      <MultiPanel Type="h" >
        <TabPanel Placement="t" >
          <Tab >
            <Panel ModuleName="DTSTree" TargetIID="detail" />
          </Tab>
          <Tab >
            <Panel ModuleName="DTSWalker" TargetIID="detail" />
          </Tab>
          <Tab >
            <Panel ModuleName="DTSSearch" TargetIID="detail" />
          </Tab>
        </TabPanel>
        <Divider />
        <TabPanel Placement="t" >
          <Tab Label="Details" >
            <Panel ModuleName="DTSDetail" IID="detail" />
          </Tab>
        </TabPanel>
      </MultiPanel>
      <Panel ModuleName="DTSStatus" />
    </MultiPanel>  </PanelLayout>
</Layout>
```