Apelon, Inc.
Suite 202, 100 Danbury Road
Ridgefield, CT 06877

Phone: (203) 431-2530
Fax: (203) 431-2523
www.apelon.com

Apelon Distributed Terminology System (DTS)

DTS Editor Plug-ins Guide

Table of Contents

Overview

The **DTS Editor Plug-in Framework**, introduced with DTS Version 3.4, provides a framework for customizing or extending the capabilities of the DTS Editor. Using the Plug-in Framework, developers may create new **plug-in** ‘modules’ designed to interface and work with existing DTS Editor features. This provides the DTS developer the ability to provide targeted, custom functionality without having to spend time and effort developing a GUI and the supporting DTS connection framework.

A plug-in is recognized by the DTS Editor by extending the DTSEditorModule class in a recognized package. Plug-ins import the classes in the apelon.apps.dts.editor.modules package to gain access to Editor services. Each plug-in overrides the DTSEditorModule classes initModule() method. This method is called by the DTS Editor at start-up and provides the plug-in with a copy of the DTSEditorModuleMgr class. Custom initialization codes may also be placed in this method. Additional methods are available for the plug-in to implement which are called by the Editor to initialize custom toolbar and menu items.

Beyond the basic requirements for recognizing and initializing a plug-in, a GUI can also be created. The DTSEditorModuleMgr class provides functionality for generating stand-alone panels and panes that may be displayed within the Editor. Drag and drop support is available for DTS data objects such as Concept, Property, Term, etc. A plug-in can also listen for and handle Editor connection and editing events. Methods for handling exceptions and showing error messages are also available.

Tutorial

This tutorial will walk through the process of creating a DTS Editor plug-in. The example is called “SimplePlugIn”.

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

```
samples\editorplugin\src\com\apelon\modules\dts\editor\simpleplugin
```

Module Registration

Plug-in modules must fulfill the following two conditions to be recognized as plug-ins by the DTS Editor.

1. The plug-in extends the DTSEditorModule class.
2. The plug-ins java class files must reside in a package recognized by the DTS Editor.

The DTS Editor must be aware that a custom plug-in module exists for it to load the plug-in and make it available through the Editor. To achieve this, the java class extending DTSEditorModule needs to be in a specific location. The DTS Editor will automatically recognize any valid plug-in created in com.apelon.modules.dts.editor or any subpackage (com.apelon.modules.dts.editor.*).

The DTS Editor will also search any packages specified as modulePackageName1…n in the dtseditor.xml file.

Example

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

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

The DTS Editor will search packages in the following order:

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

Imports

A DTS plug-in needs to import the com.apelon.apps.dts.editor.modules package. This package contains the following classes which provide 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.
- DTSModuleConfig – Used to access and save module specific properties.
- DTSEditorModuleMgr – Allows the plug-in access to DTS Editor functions and DTS Services.

The SimplePlugIn example also imports classes in com.apelon.beans.dts.plugin. connection, which manages events related to DTS connection objects.

Import Example

```
// Apelon Imports
import com.apelon.apps.dts.editor.modules.*;
import com.apelon.beans.dts.plugin.connection.DtsConnectionListener;
import com.apelon.beans.dts.plugin.connection.DtsConnectionEvent;
import com.apelon.beans.dts.plugin.connection.ConnectionCloseVeto;
```

Class Declaration

DTSEditorModule must be extended to create a custom plug-in for the Apelon DTS Editor. **The extending class must have a no-parameter constructor or the class won't be recognized as a plug-in.**

Any DTSEditorModule extension classes in com.apelon.modules.dts.editor or a subpackage (com.apelon.modules.dts.editor.*) are loaded by the DTS Editor. Classes residing in other packages must be specified in the DTS Editor property file (bin/editor/dtseditor.xml) as follows:

- Specify the number of module packages in modulePackageCount property (ex. there is one module:
  property name="modulePackageCount" value="1"
- Specify the names of module packages in modulePackageName property (ex. the module package name is 'myModulePackage':
  property name="modulePackageName1" value="myModulePackage"

A module is loaded during DTSEditor startup. The sequence of actions is:

- Custom menus returned by getModuleMenus() are added to menu bar
- Custom menu items returned by getModuleMenuItems(int) are added to DTS menus

- Custom toolbar items returned by getModuleToolbarItems() are added to toolbar
- If config file is specified by getDTSModuleConfigFile(), a DTSModuleConfig is set in the module using setDTSModuleConfig()
- The initModule() method is called

Additionally, DtsConnectionListener needs to be implemented in order to handle events related to DTS Server and database connections.

Class Declaration Code

```
public class SimplePlugIn extends DTSEditorModule implements
DtsConnectionListener
```

Initializing the Module

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

1. Provides the plug-in with a copy of the DTSEditorModuleMgr class.
2. Allows for any plug-in specific initialization steps to occur.

The copy of the DTSEditorModuleMgr class is the custom plug-ins interface to the exposed functionality of the DTS Editor. The following steps occur in the initModule() implementation in the SimplePlugIn example:

- Initialize an instance of the DTS Editor Module Manager
- Register connection listener
- Initialize swing panel settings

Module Initialization Code

```
DTSEditorModuleMgr moduleMgr;

public void initModule(DTSEditorModuleMgr mgr) {

  // Initialize our local instance of the Module Manager
  this.moduleMgr = mgr;

  // Register ourselves as a connection listener
  this.moduleMgr.registerConnectionListener(this);

  // Initialize panel settings
  initModulePanels();
}
```

Getting a Menu, Menu Items and Toolbar

Prior to calling the initModule method the DTS Editor checks to see if a module is to be accessible using custom menu and/or toolbar entries.

A module may provide for direct user access from the DTS Editor in three ways.

1. Custom DTS Editor menu (getModuleMenus)
2. Custom menu item incorporated into existing DTS Editor menus (getModuleMenuItems(int group))
3. Custom DTS Editor toolbar button (getModuleToolbarItems ())

Action listeners need to be added to each menu and toolbar item in this code to provide the functionality you want your plug-in to offer.

getModuleMenus() returns a new main menu to be displayed on the DTS Editor menu bar. This menu will be placed to the left of the Help menu.

Menu Code

```
public JMenu[] getModuleMenus (){

  JMenu editMenu = new JMenu("Simple Plugin Edit");
  editMenu.add(getModuleMenuItem1());
  editMenu.add(getModuleMenuItem2());
  JMenu[] menus = new JMenu[] {editMenu};
  return menus;
}
```

To add menu items to an existing DTSEditor menu you must override getModuleMenuItems(int group).

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[] {getSimplePlugInMenuItem(),
getSimplePlugInCfgMenuItem()};
      break;
    case HELP_MENU_ITEMS:
      menuItems = new JMenuItem[] {getSimplePluginHelpMenuItem()};
      break;
    default:
      break;
    }
  return menuItems;
}
```

The DTS Editor then calls getModuleToolbarItems() to check the returned array of JButtons for any custom icons to be displayed in its toolbar.

Toolbar Code

```
public JComponent[] getModuleToolbarItems() {

  Class c = SimplePlugIn.class;
  ImageIcon ic;
  URL url = c.getResource("apelicon16.gif");
  toolbarImage = new ImageIcon(url, "GIF");

  if (toolbarButton == null) {
    toolbarButton = new JButton();
    toolbarButton.setIcon(toolbarImage);
    toolbarButton.setMargin(new Insets(4, 2, 4, 4));
    toolbarButton.setToolTipText("DTS Editor Simple Plug-in");
    toolbarButton.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        actionPerformed_SimplePluginMenuItem(e);
      }
    });
    toolbarButton.setEnabled(false);
  }
  return toolbarButton;
}
```

Handling Connection Events

A plug-in may need to respond to the DTS Editor's server connection events. If so, it must register as a DtsConnectionListener. This provides the plug-in with the ability to take specific actions if the DTS Editor connects or disconnects from the DTS Server. These actions can include enabling or disabling menu items or ensuring proper handling of connection dependent operations. The SimplePlugIn code shows an example of the former.

Connection Event Handling Code

```
public void connectionOpened(DtsConnectionEvent event) {
  getSimplePlugInMenuItem().setEnabled(true);
  getSimplePluginToolbarItem().setEnabled(true);
  getModuleMenuItem1().setEnabled(true);
  getModuleMenuItem2().setEnabled(true);
  getSimplePlugInPanel().addListeners();
}

public void connectionClosed(DtsConnectionEvent event) {
  getSimplePlugInMenuItem().setEnabled(false);
  getSimplePluginToolbarItem().setEnabled(false);
  getModuleMenuItem1().setEnabled(false);
  getModuleMenuItem2().setEnabled(false);
}
```

Set Up Summary

A plug-in module needs to implement initModule(). If there is a GUI, the module should provide access by implementing getModuleMenus() and/or getModuleToolbarItems(). To allow for handling any conditions arising from connection events the module must register as a DtsConnectionListener.

The next step is to design and implement functionality that the plug-in is going to provide!

Creating a GUI for the Plug-in Module

Although not required, a custom plug-in is likely to be GUI based. This will require the definition of one or more panels that will be the user access point to the plug-in's functionality. To define the module's functionality these panels may incorporate calls to the DTS API. The placement and access to these panels is handled via the copy of the DTSEditorModuleMgr class which was passed to the plug-in when the DTS Editor made the call to the initModule() method. In the SimplePlugIn example, a floating panel is displayed when the SimplePlugin menu item is selected.

Panel Handling Code

```
private void actionPerformed_SimplePluginMenuItem(ActionEvent e) {

  fSimplePluginPanel = new SimplePlugInPanel(this.moduleMgr,
getDTSModuleConfig());

  fSimplePluginPanel.configure();

  JDialog dialog = this.moduleMgr.createDialog(fSimplePluginPanel,
"Simple Plug-in");

  WindowListener wndCloser = new WindowAdapter() {

    public void windowClosing(WindowEvent e) {

      fSimplePluginPanel.removeListeners();
    }
  };

  dialog.addWindowListener(wndCloser);
  dialog.setModal(false);
  dialog.show();
}
```

Plug-in panels can be tabbed panes, popup panels and popup dialogs. The following code sample shows how SimplePlugInPanel is added to the DTS Editor's left pane when the 'Left Tabbed Pane' radio button is clicked.

Left Tabbed Pane Handling Code

```
void actionPerformed_rbnTabLeft(ActionEvent e) {

  JTabbedPane leftPane = fModuleMgr.getLeftTabbedPane();
  JTabbedPane rightPane = fModuleMgr.getRightTabbedPane();

  if (!findPluginPanel(leftPane)) {
    leftPane.add("Simple Plugin Panel", new
SimplePlugInPanel(fModuleMgr));
  }
  removePluginPanel(rightPane);
}
```

Establishing Drag and Drop Interoperability with other DTS Editor features

Much of the editing done in the DTS Editor relies on or is facilitated by drag and drop functionality. Support for drag and drop capability between existing DTS panels and a custom plug-in or between different custom plug-in panels can greatly enhance the capabilities of each.

Various DTS objects can be dragged and dropped between the DTS Editor and a plug-in. 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.

The developer needs to use all the standard java.awt.dnd objects and handling:

1. Create a default DragSource and a new DragGestureRecognizer.

Drag Source Code

```
DragSource dragSource = DragSource.getDefaultDragSource();
dragSource.createDefaultDragGestureRecognizer(subsetTable,
DnDConstants.ACTION_COPY_OR_MOVE, this);
```

2. Implement the dragGestureRecognized() method of DragGestureListener. The example creates a DTS specific Transferable.

Drag Gesture Recognized Code

```
public void dragGestureRecognized(DragGestureEvent dge) {

  int selIndex = subsetTable.getSelectedRow();

  if (selIndex == -1) {

    return;
  }

  Subset si =
subsetTableModel.getSubsetAt(subsetTableSorter.modelIndex(selIndex));

  SubsetTransferable st = new SubsetTransferable(si);
  int colIndex = subsetTable.getSelectedColumn();
  Object obj = subsetTable.getValueAt(selIndex, colIndex);

  if ( obj != null) {

    st.setStringValue(obj.toString());
  }

    dge.startDrag(null, st);
}
```

3. Instantiate a DropTarget with GUI components that you want to accept drops.

Drop Target Code

```
private JTextArea getTextArea() {
  if (textArea == null) {

    textArea = new JTextArea("Simple PlugIn Panel Text Area \n");
    textArea.setEditable(false);
    textArea.setPreferredSize(new Dimension(380, 480));

    new DropTarget(textArea, new PanelDropTargetListener());
    }
  return textArea;
}
```

4. Handle the drop. SimplePlugInPanel defines a private class called PanelDropTargetListener that extends DropTargetAdapter. This class overrides the methods needed to support drop().

Drop Handling Code

```
public void drop(DropTargetDropEvent dtde) {

  if (!acceptable) {
    dtde.rejectDrop();
    return;
  }

  int dropAction = dtde.getDropAction();
  dtde.acceptDrop(dropAction);

  Transferable trans = dtde.getTransferable();

  if (trans == null){
    dtde.dropComplete(false);
    return;
  }

  try {
    if (trans.isDataFlavorSupported(DTSMultiTransferable.multiDataFlavor)) {
      // Process Multi-Data Flavor
    }
    else if
(trans.isDataFlavorSupported(ConceptTransferable.conceptDataFlavor)) {
      // Process Concept Data Flavor
    }
    else if (trans.isDataFlavorSupported(TermTransferable.termDataFlavor)) {
      // Process Term Data Flavor
    }
  }
  catch (Exception ex) { }

   dtde.dropComplete(true);
  }
}
```

See ‘Objects Supported for Drag and Drop’ in the appendix for reference as to which DTS Editor panels support DND for which DTS data objects.

Get API Query

A plug-in can use com.apelon.beans.dts.plugin.DTSAppManager to get the necessary API query.

Get query code

```
    DTSAppManager.getQuery().getDTSConceptQuery();
    DTSAppManager.getQuery().getNamespaceQuery();
    DTSAppManager.getQuery().getSearchQuery();
......
```

Event Handling

DTS fires event actions such as ConceptEvent, SubsetEvent, TermEvent, etc. A plug-in can register listeners such as ConceptListener to handle these events. See the com.apelon.dts.client.events package for details. A plug-in can use com.apelon.beans.dts.plugin.DTSAppManager to get queries and register listeners.

Register a Listener

```
public void addListeners() {

  this.dtsModuleMgr.registerConnectionListener(this);

DTSAppManager.getQuery().getThesaurusQuery().addConceptListener(this);
}
```

Handling a Concept Event

```
public void conceptActionOccurred(ConceptEvent event) {

  int eventNum = event.getEventType();

  if (eventNum == event.EVENT_TYPE_NEW) {

    getTextArea().append("Concept " + event.getConcept().getName() + "
is added!\n");
  } else if (eventNum == event.EVENT_TYPE_MODIFY) {
      getTextArea().append("Concept " + event.getConcept().getName() +
" is modified!\n");
  } else if (eventNum == event.EVENT_TYPE_DELETE) {
      getTextArea().append("Concept " + event.getConcept().getName() +
" is deleted!\n");
  } else {
      getTextArea().append("Invalid event occurred!\n");
  }
}
```

Error Handling

Plug-ins can write error messages to the log file initialized in DTS Editor. A plug-in panel can also display its own popup message dialog by using the DTSEditorModuleMgr showErrorMessage() or handleException() methods. SimplePlugIn contains examples of both.

Error Handling Code

```
public SimplePlugInCfgPanel(DTSEditorModuleMgr mgr) {

  super();

  fModuleMgr = mgr;

  try {
    initialize();
  }
  catch (Exception ex) {
    String s = "Exception in SimplePlugInCfgPanel Constructor";
    dtsModuleMgr.handleException(s, ex);
    fModuleMgr.showErrorMessage("Cannot load SimplePlugInCfgPanel.");
  }
}
```

Appendix A

Objects Supported for Drag and Drop

Depending on the panel, single (**s**) or multiple (**m**) DTS data object(s) can be dragged or dropped as outlined in the following tables:

Table 1 - Dragging Objects from Editor Components

Drag object	Ontylog Concept	DTS Concept	DTS Role	DTS Property	Concept Assoc	Synonym	Term	Term Assoc
Tree Panel	m	m	m		m			
Concept Walker	s	s						
Search Panel	m	m					m	
Detail Panel	m	m	m	m	m	m	s	m
Association Editor	s	s					s	
Property Editor	s	s					s	
Synonym Editor								

Table 2 - Dropping Objects on Editor Components Table

Drop object	Ontylog Concept	DTS Concept	DTS Role	DTS Property	Concept Assoc	Synonym	Term	Term Assoc
Tree Panel	s	s	s		s			
Concept Walker	s	s	s		s			
Search Panel	m	m	m	m	m	m	m	m
Detail Panel	s	s	s		s	s	s	s
Association Editor	s	s	s		s	s	s	s
Property Editor	s	s	s		s	s	s	s
Synonym Editor	s	s	s		s	s	s	s

s – Single drag or drop
m – Multiple drag or drop

Additionally, the various Editor components handle drag and drop in different ways as follows:

Tree Panel – Checks for a drop object. A popup warning message will be shown for an invalid drop object. The valid drop object is DTSConcept. For DTSRole and conceptAssociation the tree panel will get and display their value concepts which are DTSConcept.

Concept Walker Panel – Checks for a drop object. For an invalid drop object there is no warning message and the focus concept combo box remains unchanged. The valid drop object is DTSConcept. For DTSRole and conceptAssociation the concept walker will get and display their value concepts which are DTSConcept.

Search Panel – There is no check for a drop object. It just sets the toString() of the drop object as search text.

Detail Panel - Checks for a drop object. A popup warning message will be shown for an invalid drop object. The valid drop objects are DTSConcept and Term. For DTSRole and conceptAssociation the detail editor will get and display their value concepts which are DTSConcept. For Synonym and TermAssociation the detail editor will get and display their value terms.

Association Editor - Checks for a drop object. There is different handling for "From Concept/Term" and "To Concept/Term" for invalid drop objects.

- From Concept/Term combo box – there is no warning message for an invalid drop object. If the user drops a DTSProperty object, the combo box will go blank. This is different with concept walker. The walker will retain the original concept instead of blank out.
- To Concept/Term combo box – If the From Concept/Term combo box contains a concept, the To Concept/Term combo box must have a concept. If a Term is dropped into the To Concept/Term combo box, a warning message will pop up. If the From Concept/Term combo box contains a Term, the To Concept/Term combo box must have a Term. Otherwise a warning message will pop up. If the user drops a DTSProperty, the combo box will blank out.

Property Editor - There is no check for a drop object. If the user drops a DTSProperty to the Concept/Term combo box, the box will go blank.

Synonym Editor - Checks for a drop object. The user can only drop a concept into the Concept combo box and only drop a Term into a Term Combo box. Otherwise, a warning message will pop up. If the user drops a DTSProperty object into the Concept or Term combo box, the combo box will remain unchanged like the concept walker.

Back to Top