How to Build a Component

by Steve Cousins

This paper gives a short overview of how to build a DLITE component. The major sections are organized around important basic concepts (like the coordinate systems used) and things programmers need to do like moving the component around. This paper does not try to explain the details of all of the parameters to the methods. Rather, it is intended to be an index into the code base, giving a high-level explanation that enables the code to be understood.

The important thing to remember is that the component works in concert with a component view that is typically running on a different computer. A component programmer does not write any code for the component view, but can only call methods on it. There is a fixed set of methods that the component view will call on the component. All of these methods are defined in ITCN.isl (see Appendix A).

In general, to make a component behave in a certain way, borrow code freely from an existing component that is similar to the one being designed. A sample component is provided in Appendix B.

1 SUPERCLASSES

All components inherit from the `component` class (defined in component.py). Services also inherit from the `service` class (defined in service.py) or one of its subclasses, and collections also inherit from the `collection` class (defined in collection.py) or one of its subclasses.

New classes inherit behavior from their superclass, and can override it as necessary. Overriding arbitrary methods from the superclasses provided should be done with care. This appendix points out methods that are ok to redefine.

2 COORDINATE SYSTEMS

The coordinate systems of DLITE are designed for portability and simplicity, so absolute pixels are used. The zero point is in the upper left. Points are passed to component views in Point or PointList structures. There are Point and PointList methods defined by component that handle the encoding. See the `addViewElements` method of any component for an example.

3 DRAWING THE COMPONENT

Components are drawn by calling add*Foo* methods on the component view, where *Foo* is a geometric shape like “Rectangle.” Since these methods need to be called for each viewer, the convention is to put all the calls in the `addViewElements` method, which is called each time a new viewer is added.

The basic drawing primitives are:

- addImage - add a GIF image to the rendering
- addLine - add a line segment
- addOval - add an oval
- addPolygon - add a polygon
- addRectangle - add a rectangle
- addText - add a text label
- addButton - add a GUI button
- addTextBox - add a GUI text type-in widget
- addTable - add a table (not implemented in latest version)

Almost every component has a reference element.* The reference element is defined in the `addReferenceElement` method. If this method is not defined, a default reference element graphic (currently a small circle) is added.

The painter’s algorithm is used to determine the order elements are drawn on the canvas: methods called first are drawn first, and possibly occluded by later calls. Unfortunately, the underlying communication system in the current implementation does not guarantee

* The exception is the label component, which is just text on the canvas.

the order of asynchronous method calls, so one parameter of each drawing call is a serial number. This call is entirely idiomatic and should be copied exactly.

Canvas elements may be named in order to be referred to later, by passing a string as the “cookie” parameter. The name can be used to look up the reference element in order to determine if the element was involved in a user operation like drop or activate, or to change one of its properties like color or text.

4 MOVING COMPONENTS AROUND

Components can move themselves or other components. In general, components should not move other components except in response to an external event like a process finishing or a user action. The two methods for moving components are `moveTo`, `animateMoveTo`, and `animateMoveToPt`.

- The `moveTo` method moves the object immediately, without animating it to its destination. In order not to disorient the user, its use is not recommended.
- The `animateMoveTo` method takes a list of 2-tuples, each of which is an x-y pair. It is used to move the component along a path. Historically, this method was used to implement the “reject drop” behavior, in which the component “waggled” back and forth before moving back to its original position. There is now a `rejectDrop` method on components that is used instead.
- The `animateMoveToPt` method is the most common move method. It animates an object straight to a specified point.

Note that both the component and component view objects have `moveTo` and `animateMoveTo` methods. The component versions cause movement in all registered component views. Use of the component view versions is discouraged.

5 CHANGING THE VISUALIZATION

A number of methods allow components to change the way their graphic elements appear. In general, the graphic attributes of a canvas element can be modified, but methods that are easier to use are defined to change labels and set colors. In addition, it is possible to add and remove canvas elements to a component at any time.

Only canvas elements that were named at creation can be manipulated. Canvas elements are named by passing a “cookie” parameter, which is simply a string. The canvas element can be retrieved using the `getViewPart` method on a viewer.

With the `modifyGraphicAttributes` call, it is possible to change some graphic attributes on any canvas element of any component view. Programmers may need to remember to update all component views if the change is to be seen by all people viewing the workcenter.

The most common uses of updateGraphicAttributes are to update text labels and the color of objects. Two special-purpose methods on components handle these cases. The `updateLable` method changes a label. If the name of the canvas element given is `None`, then the yellow pop-up label for the component is changed. The `setPartColor` method is used to change the color of a canvas element. Both of these methods accept the view part name that would be given to `getViewPart` (set with the cookie parameter), and apply to all viewers.

New canvas elements can be added by calling the add*Foo* methods on all viewers. However, the programmer must be sure to put a hook into the `addViewElements` method so that these new canvas elements will appear when new viewers are registered. Canvas elements can be removed from all viewers with the `removeCanvasElement` method, or from only a particular viewer with `removeCanvasElementFromViewer`.

6 SETTING THE HTML REPRESENTATION

Every component in DLITE has a URL associated with it, because a DLITE workcenter has a web server that will translate URLs with object IDs in them to method calls on components. Components can define `GET` and `POST` methods in order to respond to HTTP requests directly. For less sophisticated components, a `showHTML` method can be defined: the default behavior of `GET` for the component base class is to call the `showHTML` method if it is available.

Components can actually have many URLs, since the entire URL string past the object ID will be passed to them. A common technique for making components editable via HTML

is to take their default URL (by calling `tcnutils.GetURLforObj(self)`) and append a parameter string like "?method=edit" and using this URL as the action for an HTML form. The default URL returns the form, and the extended URL processes the result.

7 RESPONDING TO ACTIVATE

When users double-click on component views, the component view sends an activate message to the component. The component is passed the particular canvas element that was double-clicked on as a parameter, so it can behave differently depending on exactly where the user clicked.

This is most commonly used to distinguish the reference element from others. The expected behavior is that when the user activates the reference element the view state (expanded or contracted) of the component will change. The proper thing to do in the first step of overriding the activate method is to pass the message on to the parent class if the reference element was activated.

8 HANDLING DROPPED OBJECTS

When users drop components onto other components, the destination component will receive a drop event. It is possible that multiple components will be dropped in a single action if the user selected multiple items before dragging. The destination component should do something to indicate to the user that the drop was received. A common operation is to check the input type, and reject the input if the check fails or accept it by moving the dropped object otherwise.

Three support methods exist to help with this behavior. `acceptDrop` takes an x and y coordinate and moves a dropped component to that location. It also attaches the object as a sub-component of the accepting component. The accepting component must release the dropped component when processing is completed, using `releaseComponent`. The `rejectDrop` method implements the "waggle and return" behavior, to indicate that the dropped object cannot be processed. The `returnToPreDrop` method moves the dropped object back where it came from without the waggle. It is used to set an attribute of the accepting component without taking control of the dropped component.

9 HANDLING BUTTONPRESS EVENTS

If a component has defined a button as one of its view elements, and a user presses the button, the component's `buttonPress` method gets called. The method call includes the viewer from which the button was pressed.

10 HANDLING KEYPRESS EVENTS

If a component has a textBox as one of its view elements, it may receive calls to its `keyPress` method. The frequency of these calls can be controlled with the setKeyEvent method on the textBox: the viewer will call the method each time the user hits any key, only when the enter key is pressed, or never.

11 HANDLING DRAG EVENTS

The component programmer cannot control what happens while an object is being dragged. However, it is possible to detect that a component has been dragged, and this is occasionally useful in order to handle the case when components are dragged off of other components (since the previous parent component will not see the drop method). In general, the method call to override here is the `setParent` method, but the programmer must be sure to call the inherited version in addition to performing the override action.

12 MAKING ATTRIBUTE VALUES PERSISTENT

Components are supposed to persist. If new attributes are added to a component subclass, those attributes may need to persist as well. The way to cause this to happen is to override the `__getstate__` and `__setstate__` methods. The `__getstate__` method is called to ask the object what elements are in its state. The state is represented as a dictionary, and the convention is for a subclass to add any fields it needs to the dictionary, and then call the inherited version. Similarly, the `__setstate__` method is used to restore an object's state, so the inherited version should be called first, and then any saved attributes should be restored from the dictionary. Note that if new attributes are added to an object, the `__setstate__` method should handle the case when an object is restored that doesn't have the desired key in the dictionary.

13 ATTACHING SUBCOMPONENTS

Components can be attached as subcomponents to other components, meaning that when the parent component is dragged, the subcomponent will move along with it. A programmer can make one object a subcomponent of another with the `subsumeComponent` method.

An important parameter of the method is how the subcomponent is to be attached. Three options are possible: moveOff, copyOff, or permanent. With moveOff attachments, when the user drags on the subcomponent it becomes disconnected. With copyOff, when the user drags on the subcomponent a copy of it is moved with the mouse. With permanent attachments, when the user drags the subcomponent it is as if the super-component is being dragged.

14 HANDLING VIEWERS DIFFERENTLY

DLITE is designed so that multiple viewers can be attached to a workcenter at once. This feature has increased the complexity of the interface between components and component views immensely. It is possible for components to handle each viewer differently, but this must be done very carefully, keeping in mind what will happen when viewers are added or removed later.

Most methods have a viewer parameter, so it is possible to know which of many possible viewers initiated an action, or to draw only to a particular viewer. Component views are added and removed from a component with the `registerViewer` and `unregisterViewer` methods, so these methods can be overridden to track the current viewers, but the inherited versions must always be called.

A ILU Interface Specification

This appendix contains a simplified version of the ILU specification for the interface between a UI-client and a UI-server. The specification is in the ILU Interface Specification Language (ISL), which is a slight variant of the CORBA standard Interface Definition Language (IDL).

```
INTERFACE ITCN IMPORTS EpersRef END;

TYPE TString = ilu.CString;

TYPE TCookie = OPTIONAL TString;
TYPE TEpersRef = EpersRef.TEpersRef;
Type TStringList = SEQUENCE OF TString;

TYPE Filename = TString;

TYPE ComponentList = SEQUENCE OF Component;

TYPE Component = OBJECT OPTIONAL
  METHODS
         ASYNCHRONOUS registerCanvasElement(cookie : TCookie, ce :
CanvasElement,
                                        v : Viewer)
   “Callback from CanvasView, so addFoo calls could be asynchronous”,

    ASYNCHRONOUS registerViewer(v : Viewer),
    ASYNCHRONOUS unregisterViewer(v : Viewer),
    ASYNCHRONOUS keyPress(key : TString, ce : CanvasElement, v : Viewer,
          text : TString, cursorIndex : CARDINAL)
   “User typed one key in a textbox (or component).”,
    ASYNCHRONOUS activate(ce : CanvasElement, v : Viewer)
   “Corresponds to a double-click in the view”,
    dragTo(x : INTEGER, y : INTEGER, nowOver : Component,
                  SourceView : Viewer)
   “User moved the corresponding view to (x,y) and the view is
    now on top of nowOver”,

    ASYNCHRONOUS drop(compList : ComponentList, ce : CanvasElement,
       SourceView : Viewer)
   “User dragged the elements in compList onto the element ‘ce’
    in this object”,

    copy(SourceView : Viewer, x : INTEGER, y : INTEGER) : Component
   “Create a new copy of this component, located at (x,y)
    [this is used with ‘copy off’ operations]”,

    getPickleState() : TString
   “Encode the state of this component in a pickle string, and return
it”,

    ASYNCHRONOUS buttonPress(SourceView : Viewer, str : TString)
```

```
   "User pressed the button associated with str",

    ASYNCHRONOUS identify()
   "Print out debugging information about yourself on the console"
  END;

TYPE Point = RECORD
    x : INTEGER,
    y : INTEGER
  END;

TYPE TSize = RECORD
    width : INTEGER,
    height : INTEGER
  END;


TYPE PointList = SEQUENCE OF Point;

TYPE TFont = RECORD
  family : TString,
  style : TString,
  size : CARDINAL
END;

TYPE MoreAttribute = RECORD
  key : TString,
  value : TString
END;

TYPE EAttribute = ENUMERATION kFill, kOutline, kFont, kStipple, kWidth,
                    kArrow, kAnchor, kBackground, kMoreAttr
                  END;

TYPE EAlignment = ENUMERATION kNorthWest, kNorth, kNorthEast,
             kWest, kCenter, kEast,
                          kSouthWest, kSouth, kSouthEast
                  END;

TYPE Attribute = EAttribute UNION
   aFill : Color = kFill END,
   aOutline : Color = kOutline END,
   aFont : TFont = kFont END,
   aStipple : TString = kStipple END,
   aBackground : TString = kBackground END,
   aWidth : INTEGER = kWidth END,
   aArrow : TString = kArrow END,
   aAnchor : EAlignment = kAnchor END,
   aMoreAttribute : MoreAttribute = kMoreAttr END
END;

TYPE GraphicAttributes = SEQUENCE OF Attribute;

TYPE CanvasElement = OBJECT OPTIONAL
  METHODS
    ASYNCHRONOUS modifyGraphicAttributes( newGA : GraphicAttributes )
   "change the appearance of the canvas element after it exists"
  END;
```

```
TYPE EKeyEvent = ENUMERATION kNone, kEnter, kAll END;

TYPE TextBox = OBJECT  OPTIONAL
  SUPERCLASS CanvasElement
  METHODS
    ASYNCHRONOUS setKeyEvent(pType : EKeyEvent)
   “request only certain types of event to callback.”,
    getText() : TString
   “Return the current text in the text box”,
    ASYNCHRONOUS setText(newText : TString)
   “Replace the current text in the text box with newText”,
    ASYNCHRONOUS setCursor(pIndex : CARDINAL)
   “Move the text cursor to this position in the text.”,
    ASYNCHRONOUS insertColoredText(pWhereIndex : CARDINAL,
                                   pColor : Color, newText : TString)
   “Insert some new text to this position with this color.”
  END;

TYPE TableElement = OBJECT  OPTIONAL
  SUPERCLASS CanvasElement
  METHODS
    ASYNCHRONOUS SetTitle(col : INTEGER, title : TString,
                      ga : GraphicAttributes)
   “Set the title for this column to title”,
    ASYNCHRONOUS SetText(row : INTEGER, col : INTEGER,
                         text : TString, ga : GraphicAttributes)
   “Set the text at location (row,col) to text”,
    ASYNCHRONOUS SetContent(row : INTEGER, col : INTEGER,
                            dcv : ComponentView, al : EAlignment)
   “Replace the contents of (row,col) with dcv”,
    ASYNCHRONOUS SetColumnWidth(col : INTEGER, width : INTEGER)
   “Set the width of col to width [could also be done with SetTitle”

  END;

TYPE Color = RECORD
  red : INTEGER,
  green : INTEGER,
  blue : INTEGER
END;

TYPE ECompViewAttachmentType = ENUMERATION
  kPermanent,  (* user-action does not detach sub-compview from compview.
*)
  kCopyOff,    (* dragging sub-compview causes copy to be made. *)
  kMoveOff     (* dragging sub-compview causes detachment. *)
END;

TYPE EDropBehaviorType = ENUMERATION
   kPassive,      (* nothing will happen if you drop something on this
element *)
  kRejectAll,  (* if you drop something on me, it will slide off *)
   kAcceptDrop  (* if you drop something on me, a drop message will get
sent *)
END;

(* This is an optimization for attaching lots of things at once. *)
(* eltype == 0; just attach cv, don’t draw anything. *)
(* eltype == 1; attach cv drawn as a document. *)
```

```
(* eltype == 2; attach cv drawn as a query. *)
TYPE Attachee = RECORD
  eltype : INTEGER,
  c : Component,
  label : TString,
  cv : ComponentView,
  attachmenttype : ECompViewAttachmentType,
  pt : Point
END;

TYPE AttacheeList = SEQUENCE OF Attachee;

TYPE ComponentView = OBJECT OPTIONAL
  DOCUMENTATION
     "Cookie Note: cookies are used to reference canvas elements
      when registerCanvasElement is called on Components.

     In most of these addFoo routines, a cookie parameter may be used to
      let the caller get a handle on the CanvasElement created.  If the
      cookie is not Null, a registerCanvasElement callback is made.

      All points specified in these addFoo routines are relative to the
      anchor of the ComponentView, which is specified when it is added
      to the viewer (see below)."

  METHODS
    ASYNCHRONOUS reduceToReferenceElement(serial : Cardinal)
  "remove all elements from the group except for the reference element",
    ASYNCHRONOUS setReferenceElement(ce : CanvasElement, label : TString,
   serial : Cardinal)
   "use ce as the reference element for this group",

    ASYNCHRONOUS addImage(p : Point, imageName : TString,
                          attrs : GraphicAttributes, cookie : TCookie,
                    onDrop : EDropBehaviorType,
          serial : Cardinal)
   "add the image pointed to by imageName to this view.
    imageName is treated as a relative URL to a GIF image",
    ASYNCHRONOUS addLine(pPoints : PointList, attrs : GraphicAttributes,
                         cookie : TCookie,
                   onDrop : EDropBehaviorType,
         serial : Cardinal)
   "add a polyline to the view",
    ASYNCHRONOUS addOval(p1 : Point, p2 : Point,
                         attrs : GraphicAttributes, cookie : TCookie,
                   onDrop : EDropBehaviorType,
       serial : Cardinal)
   "add an oval to the view",
    ASYNCHRONOUS addPolygon(pPoints : PointList,
                attrs : GraphicAttributes, cookie : TCookie,
                      onDrop : EDropBehaviorType,
       serial : Cardinal)
   "add a polygon to the view.",
    ASYNCHRONOUS addRectangle(p1 : Point, p2 : Point,
                            attrs : GraphicAttributes, cookie : TCookie,
                        onDrop : EDropBehaviorType,
       serial : Cardinal)
   "add a rectangle to the view",
    ASYNCHRONOUS addText(p : Point, txt : TString,
```

```
                        attrs : GraphicAttributes, cookie : TCookie,
                    onDrop : EDropBehaviorType,
        serial : Cardinal)
  “add text to the view.  The font is specified in attrs.”,

      ASYNCHRONOUS addButton(p : Point, title : TString, cbString :
TString,
                      attrs : GraphicAttributes, cookie : TCookie,
                      onDrop : EDropBehaviorType,
        serial : Cardinal)
  “add a button to the view.  When it is pressed, the component
   gets a buttonPress callback with cbString”,

    addTextBox(p : Point, textwidth : INTEGER, attrs : GraphicAttributes,
          onDrop : EDropBehaviorType, serial : Cardinal) : TextBox
  “add a text entry box to the view.  The TextBox object is returned
   (no cookie/callback is used in this case)”,

    addTable(p1 : Point, p2 : Point, attrs : GraphicAttributes,
        onDrop : EDropBehaviorType, serial : Cardinal) : TableElement
  “add a TableElement object to the view”,

    (* warning: optimization *)
     ASYNCHRONOUS attachCompViews(attachelist : AttacheeList, serial :
Cardinal)
  “make these cvs a part of me”,

    ASYNCHRONOUS attachCompView(cv : ComponentView,
                                attachmenttype : ECompViewAttachmentType,
          pt : Point,
          serial : Cardinal)
  “make cv a part of me”,

    ASYNCHRONOUS detachCompView(cv : ComponentView, serial : Cardinal)
  “disconnect cv from me”,

       ASYNCHRONOUS removeCanvasElement(ce : CanvasElement, serial :
Cardinal)
  “remove ce from the view”,
    ASYNCHRONOUS moveTo(p : Point, serial : Cardinal )
  “move the view to p”,
    ASYNCHRONOUS animateMoveTo( pl : PointList, serial : Cardinal )
  “move the view to each point in pl, animating the motion if possible”,

    ASYNCHRONOUS setLabelText( txt : TString, serial : Cardinal )
  “Change the text in the yellow pop-up label to txt.”,

    ASYNCHRONOUS initComponentView( c : Component, anchor : Point,
              label : TString,
              serial : Cardinal)
  “Initialize a previously unused componentView”
  END;

TYPE ComponentViewList = SEQUENCE OF ComponentView;

TYPE MenuAction = RECORD
   menu : TString,
   menuItem : TString
END;
```

```
TYPE MenuActionList = SEQUENCE OF MenuAction;

TYPE Viewer = OBJECT
  METHODS
    GetLocalResourceManager() : LocalResourceManager
  “Return the local resource manager object for this viewer”,

    addComponentView(c : Component, anchor : Point) : ComponentView
  “Add a new empty ComponentView, rooted at anchor, to this Viewer”,
    addManyComponentViews(n : INTEGER) : ComponentViewList
  “Return a list of n ComponentViews”,
    ASYNCHRONOUS removeComponentView(cv : ComponentView)
  “Remove cv from this Viewer”,
    ASYNCHRONOUS removeComponentViews(cv : ComponentViewList)
  “Remove cv from this Viewer”,

    ASYNCHRONOUS addMenuAction(menu : TString, menuItem : TString)
  “add menuItem to menu, creating menu if necessary.  When a menuItem
   is selected, a callback is given to the Task”,
    ASYNCHRONOUS removeMenuAction(menu : TString, menuItem : TString)
  “remove menuItem from menu”,

    ASYNCHRONOUS addMenuActions(actions : MenuActionList)
  “add menuItem to menu, creating menu if necessary.  When a menuItem
   is selected, a callback is given to the Task”,
    ASYNCHRONOUS removeMenuActions(actions : MenuActionList)
  “remove menuItem from menu”,

    ASYNCHRONOUS BringToFront()
  “Try to make the viewer the front window on the screen.”

  END;

TYPE ViewerList = SEQUENCE OF Viewer;


TYPE TaskRec = RECORD
  aTaskName : TString
(*  aTask : Task *) (* swh *)
END;

TYPE TaskList = SEQUENCE OF TaskRec;

TYPE TaskFactory = OBJECT
  METHODS
     NewTaskFromTemplate(templateName: Filename, newName : Filename) :
Task
  “Start a new task based on a template”,

    OpenExistingTask(fn : Filename) : Task
  RAISES NoSuchTask END
  “Attach to the running task in fn”,

    DeleteTask(fn : Filename)
  “Delete the running task associated with fn.  Does *not*
   remove the file fn”,

    Tasks() : TaskList
```

```
  "Return a list of the 'known' tasks",

(*    StartupURL() : TString
  "Each TaskFactory has an http server associated with it.  This
  method returns the url of that server's startup page",*)
    ASYNCHRONOUS RegisterLRM(lrm : LocalResourceManager,
  loginName : TString, passwd : TString)
        "Start a session with an lrm.  If loginName/passwd are
           missing or incorrect, user is set to nobody.",
    ASYNCHRONOUS UnregisterLRM(lrm : LocalResourceManager)
  "Proper way to finish a 'session'"
(*    RegisterLRM(lrmName : TString, cookieList : TStringList)
  "TEMPORARY METHOD. Used with the unix-startup stuff to set cookies
  from the task factory"*)
  END;

TYPE WebPage = RECORD
  title : TString,
  url : TString
END;

TYPE WebBrowser = RECORD
  target : TCookie
END;

TYPE WebBrowserList = SEQUENCE OF WebBrowser;

EXCEPTION NoSuchTask : TString;
EXCEPTION SecurityViolation : TString;

TYPE LocalResourceManager = OBJECT OPTIONAL
  DOCUMENTATION
    "local resources needs to be enhanced with some type of
interesting security."

  METHODS
(*    GetLRMName() : TString
  "Return a short name for this lrm", *)

   (* Methods for dealing with Viewer resources *)
    GetViewerList() : ViewerList
      RAISES SecurityViolation END
  "What tasks does the user have open?",
    OpenViewer(task : Task, taskname : TString)
      RAISES SecurityViolation END
  "Open a new window containing a viewer for task",
    CloseViewer(viewer : Viewer)
      RAISES SecurityViolation END
  "Close the window corresponding to viewer",

   (* Methods for the Web Browser resources *)
    GetWebBrowsers() : WebBrowserList
      RAISES SecurityViolation END,
    OpenURL(pURL : TString, pTarget : TCookie)
      RAISES SecurityViolation END
  "if pTarget == null, then just pick any web browser",
    GetCurrentWebPage(pTarget : TCookie) : WebPage
      RAISES SecurityViolation END
  "if pTarget == null, then just pick any web browser",
```

```
    (* Methods for dealing with voice communication resources *)
     MakeVoiceConnection( phoneNumber : TString )
       RAISES SecurityViolation END
    “Make a voice connection.  At the least, give the user a
     dialog box with the phone number so she can dial.”,

    (* Methods for printer resources *)
    (* Methods for scanner resources *)
    (* Methods for local disk/storage resources *)

    (* Methods for E-person interactions *)
     GetUser() : TEpersRef
    RAISES SecurityViolation END
    “Return the epers of the currently logged in user, or None.”,
     SetUser(epers : TEpersRef)
    RAISES SecurityViolation END
    “Set epers as the currently logged in user at this lrm.”,
     GetUserName() : TString
         “Return the user’s name as a string”

  END;

TYPE Task = OBJECT
  METHODS
    GetName() : TString
   “return the name of the task. “,
    GetSize() : TSize
   “return the size of the task’s canvas”,

    clearTask()
   “remove all of the components from task”,
    ASYNCHRONOUS saveTask()
   “save task on disk”,
    ASYNCHRONOUS saveTaskAs(fn : Filename)
   “save task on disk as fn”,
    loadTask()
   “load task from disk (fn given to a task at create time)”,
    ASYNCHRONOUS addComponent(c : Component)
   “add a new component to task.”,
    ASYNCHRONOUS copyComponent(c : Component, state : TString)
   “make a local copy of c, using the state (which is intended
    to be a Pickle encoding of the object for now.”,
    ASYNCHRONOUS deleteComponent(c : Component)
   “remove the component from task”,
    ASYNCHRONOUS registerViewer(v : Viewer)
   “add a new viewer client to task”,
    ASYNCHRONOUS unregisterViewer(v : Viewer)
   “remove v from the clients of task”,
    menuAction(menu : TString, menuItem : TString, v : Viewer)
   “callback - user on v chose menuItem from menu”
  END;
```

B Translator Component

```
# $Id: translator.py,v 1.10 1997/02/16 19:54:14 cousins Exp $
#
# $RCSfile: translator.py,v $
#
# Classes for translating documents.
#
# Steve Cousins              created 7/27/95
#
# $Revision: 1.10 $
# $Author: cousins $
# $Date: 1997/02/16 19:54:14 $

from logging import log
module = __name__

import component
import services
import iluTranslations
import ITCN
import tcnfonts
import tcncolors
import tcnutils
#import netscape
import USMarc
import auxils
import epersUtils

import rand

import string

import documents
import dlcoslib

try:
  import prescriptClient
except ImportError, reason:
      log(module,   “WARNING:   prescript   Translator   Not   Available
(“+`reason`+”)”)
  prescriptClient = None

class translator(services.service):
  def addViewElements(self,viewer):
    myview = self.views[viewer]
    myview.addPolygon(self.PointList([(0, 25),
                (-10, 70),
                (50, 115),
                (110, 70),
                (100, 25),
                (0, 25)]),
            iluTranslations.OutputGraphicAttributes(
       {“fill” : “#96a7d4”}),
            cookie = None,
            onDrop = ITCN.EDropBehaviorType.kRejectAll,
            serial = self.nextSerial(viewer))
```

```
    myview.addImage(self.Point(50, 60),
         self.logo,
         iluTranslations.OutputGraphicAttributes(
           {“anchor” : “center”}),
         cookie = None,
         onDrop = ITCN.EDropBehaviorType.kRejectAll,
         serial = self.nextSerial(viewer))
    myview.addOval(self.Point(0, 0),
        self.Point(100, 40),
        iluTranslations.OutputGraphicAttributes(
          {“fill” : tcncolors.documentdropcolor}),
        cookie = ‘drop’,
        onDrop = ITCN.EDropBehaviorType.kAcceptDrop,
        serial = self.nextSerial(viewer))
    myview.addImage(self.Point(50, 20),
         “doc-emb.gif”,
         iluTranslations.OutputGraphicAttributes(
           {“anchor” : “center”}),
         cookie = ‘demb’,
         onDrop = ITCN.EDropBehaviorType.kAcceptDrop,
         serial = self.nextSerial(viewer))
    myview.addText(self.Point(50, 85),
        self.textLabel,
        iluTranslations.OutputGraphicAttributes(
          {“font” : tcnfonts.mediumfont,
           “anchor” : “center”}),
        cookie = None,
        onDrop = ITCN.EDropBehaviorType.kRejectAll,
        serial = self.nextSerial(viewer))
    myview.addText(self.Point(50, 95),
        self.statuslabel,
        iluTranslations.OutputGraphicAttributes(
          {“font” : tcnfonts.tinyfont,
           “anchor” : “center”}),
        cookie = “status”,
        onDrop = ITCN.EDropBehaviorType.kRejectAll,
        serial = self.nextSerial(viewer))


  def addGumballView(self, viewer):
    myview = self.views[viewer]
    myview.addPolygon(self.PointList([(0,0),(10,0),
                (12,5),(5,10),(-2,5),(0,0)]),
               iluTranslations.OutputGraphicAttributes(
                 {“fill”:self.color}),
               cookie=’gumball’,
               onDrop = ITCN.EDropBehaviorType.kRejectAll,
               serial = self.nextSerial(viewer))

  def drop(self, componentList, viewObj, viewer):
    idlist = []
    for c in componentList:
      idlist.append(c.id())
    log(module,”DROP”,idlist,”onto”,self.id())
    ddrop = self.getViewPart(viewer,’drop’)
    demb = self.getViewPart(viewer,’demb’)
    if viewObj == ddrop or viewObj == demb:
      log(module,”  ...into document drop slot”)
      for c in componentList:
```

```
   try:
     foo = c.isDocument()
     self.gotDocument(c, viewer)
   except AttributeError,key:
     log(module, “AttributeError in drop:”,key)
     c.rejectDrop()
     else:
       log(module,”  ... NOT into ANY drop slot”)
       for c in componentList:
   c.rejectDrop()

  def gotDocument(self, doc, viewer):

    x,y = self.xloc+45, self.yloc+55
    doc.tempSetColor(“green”)
    doc.animateMoveToPt(x,y)

    text = doc.getDocumentText()

    log(module, “Computing the translation”)
    self.DoIt(doc,text)

  def TranslateException(self, reason, document):
    log(module, “TranslateException”, reason, document)
    document.animateMoveToPt(document.xloc-65, document.yloc)
    document.revertColor()

  def TranslateCallback(self, translation, doc):
    log(module, “TranslateCallBack()”)
    obj = doc.getAttribs()

    translationattrs = iluTranslations.createCItem({})
    try:
              translationattrs[USMarc.kTitle]  =  “Translation  of  “  +
obj[USMarc.kTitle]
    except TypeError:
              translationattrs[USMarc.kTitle]  =  “Translation  of  “  +
`obj[USMarc.kTitle]`
    except:
         translationattrs[USMarc.kTitle] = “Translation of an unnamed
document”
    self.setTranslationAttrs(obj,translationattrs,translation)
    translationattrs[“original”] = doc

    translationdocument = documents.document(translationattrs,
                x=doc.xloc,
                y=doc.yloc,
                color=”#c1dde5”)
    doc.task.addComponent(translationdocument)

    doc.animateMoveToPt(doc.xloc+65, doc.yloc)
    doc.revertColor()

    translationdocument.animateMoveToPt(doc.xloc+5, doc.yloc+10)
    log(module, “exit TranslateCallback()”)


class prescriptTranslator(translator):
  def __init__(self, *args, **kw):
```

```
    self.sourceObjName = “prescript”
    self.sourceName = “prescriptTranslator”
    self.textLabel = “prescript Translator”
    self.logo = “ps2html.gif”

    if not kw.has_key(“color”):
      kw[‘color’] = tcncolors.gumballcolor
    if not kw.has_key(‘label’):
      kw[‘label’] = self.sourceName
    if not kw.has_key(‘servName’):
      kw[‘servName’] = self.sourceObjName
    self.fAlarmPeriod = 180 + rand.whrandom.randint(60*1, 1)

    # service proxy object and client property set
    self.owner = None
    self.description = “”
    self.obj = None
    self.objProps = None
    # try to find some of the values for these...
##    self.GetServiceProxy()

    apply(services.service.__init__,(self,)+args,kw)

  def GetServiceProxy(self):
    if not self.sourceObjName:
      return
    try:
      self.obj = epersUtils.get_epersHP(self.sourceObjName)
      self.objProps = dlcoslib.CosPropertySetClientImpl(self.obj)
      self.owner = self.objProps[“owner”]
    except:
      log(module, “Cannot locate this service right now.”)
      return

  def DescribeService(self):
    props = self.objProps
    if not props:  return “”

  ##  baseURL = self.url
    h1 = “” ##props[“organization”]
    title = h1
    if props.has_key(“description”):
      page = props[“description”]
    else:
      page = “No description available for this service.<P>\n”

    if self.owner:
      ownerLink = auxils.epershref(self.owner)
      page = page + “<P>Owner: “+ownerLink+”<P>”

##    page = auxils.finishPage(h1, page, baseURL)
    return page

  def DoIt(self,doc,text):
##    log(module, “DoIt: self=”,self,”doc=”,doc,”len(text)=”,len(text))
    try:
      inst = prescriptClient.prescriptClient(text, self, doc)
    except:
      self.TranslateException(tcnutils.Exception(), doc)
```

```
  def setTranslationAttrs(self,obj,translationattrs,translation):
    try:
        translationattrs[USMarc.kAuthor] = “[NZDL prescript Translator
(orignal author: “ + \
             string.joinfields(obj[USMarc.kAuthor],’ and ‘)+”)]”
    except KeyError:
      translationattrs[USMarc.kAuthor] = “[NZDL prescript Translator]”

    translationattrs[USMarc.kContent] = dlcoslib.OutputMime(‘text/html’,
translation)
```