Loops Image Objects

Unknown IMAGEOBJ type
GETFN: LoopsImageObjectGetFn
Unknown IMAGEOBJ type
GETFN: LoopsImageObjectGetFn

Interlisp `IMAGEOBJ`s are "objects" that know how to display themselves on `IMAGESTREAM`s. `IMAGEOBJ`s are most often used to insert non-character items into a TEdit document. Standard Interlisp `IMAGEOBJ`s are available for displaying bitmaps, graphs, and horizontal bars.

Interlisp `IMAGEOBJ`s are not objects in the sense of Loops: there is no provision for specialization of existing `IMAGEOBJ`s and no default behavior is provided. Creation of a new type of `IMAGEOBJ`s requires some effort. Functions must be specified for printing, reading, displaying, and handling button events for the new `IMAGEOBJ` type.

Loops image objects are Loops objects that can be used as Interlisp `IMAGEOBJ`s. They let the programer use the full power of the Loops environment in the creation of new `IMAGEOBJ`s.

The LoopsImageObjects file defines a number of classes that can be used to create Interlisp `IMAGEOBJ`s, and an interface that makes it easy to insert these Loops image objects into a TEdit document. Section Unknown IMAGEOBJ type GETFN: LoopsImageObjectGetFn tells how to insert Loops image objects into a TEdit document; section Unknown IMAGEOBJ type GETFN: LoopsImageObjectGetFn describes some predefined Loops image object classes; sections Unknown IMAGEOBJ type GETFN: LoopsImageObjectGetFn Unknown IMAGEOBJ type GETFN: LoopsImageObjectGetFn describe the Loops image object protocol, for people wishing to define their own Loops image objects.

Unknown IMAGEOBJ type
GETFN: LoopsImageObjectGetFn

Unknown IMAGEOBJ type
GETFN: LoopsImageObjectGetFn Inserting Loops image objects into a TEdit document is easy.

`(LIO)` [Function]

The standard way to insert IMAGEOBJs into a TEdit file is via the CONTROL-O character. Hitting CONTROL-O lets you type in a LISP form, and the value of this form is an IMAGEOBJ that will be inserted into the document at the current location. (LIO) will present you with a menu of all known Loops image object classes. If you select a class from this menu, (LIO) will return the IMAGEOBJ that points to a new Loops image object of the specified class. If the instance has IVs that can be edited, you will be given a chance to edit the instance first.

LIOInsertCharCodes [Variable]

When the Loops image object package is loaded, it redefines the interpretation in TEdit of the characters specified by the variable LIOInsertCharCodes. Hitting one of these characters is equivilent to hitting CONTROL-O and then typing in the form (LIO). The default value of LIOInsertCharCodes is (the value of) (LIST (CHARCODE #W) (CHARCODE #w)). This is an INITVAR, so you override the default before you load the package.

Unknown IMAGEOBJ type
GETFN: LoopsImageObjectGetFn

Unknown IMAGEOBJ type
GETFN: LoopsImageObjectGetFn Many of these objects display themselves as text. This can be confusing for the user, who can easily mistake the object for a sequence of characters. To avoid this possibility for confusion, these objects appear boxed when displayed in a TEdit window. The box will not appear in a printed document.

The character looks of these objects can be set from TEdit in the same way that normal characters can have their looks set: select the object and use TEdit's Character Looks Menu, or the Looks item on the title bar pop-up menu. Note that the object can have only a single character looks--all the characters in the object will be displayed with the same looks. There is no way to change the looks of single characters displayed by a Loops image object.

These objects often violate the principle of WYSIWYG (What You See Is What You Get) in that they display more text in a TEdit window then they do when printed. They let you know what text will appear in print by displaying all non-printing text inverted. Thus, the object that displays in a TEdit window as Index: IndexEntry will show up as the string "IndexEntry" when printed. (Note the interaction of this inversion with the boxing mentioned in the previous paragraph: the inversion is performed after the boxing, so that the entire object will still appear boxed if displayed inverted (as in TEdit's highlighting to indicate pending-delete).)

These objects respond to a button event by presenting a menu of options to the user. The items in the menu depend on the class of the object. All objects include a menu item for storing the Loops instance in `(SavedValue)`, and for inspecting the instance. If the instance has IVs that can be set by the user, the menu will include an "Edit" item. The "Edit" item will bring up the standard Loops instance editor on the object. If the instance has a textual IV, the menu will include an option to edit the value of this IV in a new TEdit window.

Unknown IMAGEOBJ type
GETFN: LoopsImageObjectGetFn

Many people include in a TEdit file the time the file was last edited, or the name of the file that holds the document. The following classes can be used to automate this process.

`WhenLastSaved` [Class]

A `WhenLastSaved` image object will display a time stamp indidating the time the TEdit document was last saved. For example, a `WhenLastSaved` instance in this document might produce the image `3-Jan-86 18:21:15 by aml` when displayed in a TEdit window (but would appear without the box when hardcopied).

If the document has not been saved since the `WhenLastSaved` object was inserted, the `WhenLastSaved` object will display the string "NotYetSaved" instead of the time stamp.

`WhereLastSaved` [Class]

A `WhereLastSaved` image object is much like a `WhenLastSaved` object, except that it displays the name of the file that the TEdit document was last saved in, instead of the time last saved.

`WhenLastSaved` and `WhereLastSaved` objects are used in the running footers in this document.

Unknown IMAGEOBJ type
GETFN: LoopsImageObjectGetFn

Using image objects, it is easy to build an index table for a TEdit document.

`IndexEntry` [Class]

An `IndexEntry` object associates a string with a page number for inclusion in an index. The IV `text` of an `IndexEntry` will be added to the accumulated index, together with the number of the page containing the object. If the IV `displayText?` is non-`NIL` (the default), then the text string will also appear in the document, otherwise the object will be invisible in the final hardcopy. For example, an `IndexEntry` that adds a reference to the string "IndexEntry" appears in a TEdit window as

Index: IndexEntry . If the displayText? IV were NIL, it would appear as Index: IndexEntry it window, but would be invisible when printed.

`InitIndex` [Class]

An `InitIndex` object must appear in a TEdit document before any `IndexEntry` object. The `InitIndex` object initializes the collection of the index table. It will be invisible when printed.

`CollectIndex` [Class]

A `CollectIndex` object should appear after all the `IndexEntry` objects that appear in a TEdit document. It will sort the index and print the index information into a new TEdit window (you will be prompted for the window). You can then format the text, save it away, print it, or discard it. The IVs `looks`, `paraLooks`, `firstPageFormat`, `rectoPageFormat`, and `versoPageFormat` determine the initial formatting of the index. See the TEdit documentation for a discussion of the definition of these fields. The `CollectIndex` object will not be visible in the printed document.

Unknown IMAGEOBJ type
GETFN: LoopsImageObjectGetFn

Automatic chapter and section numbering, and generation of a table of contents, is provided by the following classes.

`SectionHeading` [Class]

`SectionHeading` objects have three IVs: `level`, `text`, and `displayText?`. The `level` IV determines the subsection nesting of the object. Each `SectionHeading` object in the document increments the current section number at the given level. Smaller level numbers will not be affected. Larger level numbers are reset to zero. For example, a sequence of `SectionHeading` objects with `level` IVs of `1`, `2`, `1`, `2`, `2`, `3`, `2`, and `1` will produce final section numbers of 1., 1.1., 2., 2.1., 2.2., 2.2.1., 2.3., and 3.

The `text` IV determines the section title for inclusion in the table of contents. If the `displayText?` is non-`NIL` (the default), the `text` will also be displayed in the document.

As an example, when first displayed in a TEdit window the `SectionHeading` for this subsection looked like n.n. Building a Table of Contents . If the `displayText?` IV were `NIL`, the text would have been inverted to indicate that it will not be present in the final hardcopy: n.n. Building a Table of Contents . Once the containing document had been hardcopied, the object displayed the section number used at hardcopy time instead of the placeholder "n.n.". (Due to a "bug", you have to redisplay the window to see the updated section numbers.)

`InitTOC` [Class]

An `InitTOC` object must appear in a TEdit document before any `SectionHeading` object. The `InitTOC` object initializes the collection of the table of contents. The IV `initialSectionNumbers` can be used to initialize the first section numbers to values other than `1`. For example, if the `initialSectionNumbers` IV is `(2 1 3)`, and the first `SectionHeading` object has a `level` IV of `3`, it will be given the subsection number "2.1.4.". The `InitTOC` object will be invisible when printed.

`CollectTOC` [Class]

A `CollectTOC` object should be placed at the end of a document, after all `SectionHeading` objects, to collect the table of contents information into a new TEdit stream. When the document is hardcopied, a new TEdit stream will be opened containing the accumulated table of contents. You can then format this TEdit document as you wish. `CollectTOC` uses the same IVs as does the `CollectIndex` to determine the initial formatting of the table of contents. The `CollectTOC` object not be visible in the final hardcopy.

Unknown IMAGEOBJ type
GETFN: LoopsImageObjectGetFn

The following classes provide a somewhat clumsy way of having page and section references included an a document. Because there is no lookahead in the TEdit page formatting, documents containing these objects must be hardcopied twice. The first time will compute the correct page and section numbers, the second time will incorporate them into the document.

Unknown IMAGEOBJ type
GETFN: LoopsImageObjectGetFn

`PageNote` [Class]

At hardcopy time, a `PageNote` object remembers the current page number, associating it with the value of its `tag` IV. See the `PageReference` class, below.

`PageReference` [Class]

At hardcopy time, a `PageReference` object looks up the value of its `tag` IV to see if a `PageNote` object has noted a page number for that tag. If there is page number stored, the `PageReference` object will display the number, otherwise it will display the string "nn". If the `PageNote` occurs before the `PageReference` object in the document, this reference will be found the first time the document is printed. If the `PageNote` occurs after the `PageReference` object, a second hardcopy must be generated.

Unknown IMAGEOBJ type
GETFN: LoopsImageObjectGetFn

`SectionNote` [Class]

At hardcopy time, a `SectionNote` object remembers the current section number, associating it with the value of its `tag` IV. See the `SectionReference` class, below.

`SectionReference` [Class]

At hardcopy time, a `SectionReference` object looks up the value of its `tag` IV to see if a `SectionNote` object has noted a section number for that tag. If there is section number stored, the `SectionReference` object will display the number, otherwise it will display the string "n.". If the `SectionNote` occurs before the `SectionReference` object in the document, this reference will be found the first time the document is printed. If the `SectionNote` occurs after the `SectionReference` object, a second hardcopy must be generated.

`SectionReference`s are used in this document in the last paragraph of the first section.

Unknown IMAGEOBJ type
GETFN: LoopsImageObjectGetFn

The following class illustrates the posiblity of nesting image objects inside of other image objects.

`BoxedImageObject` [Class]

IVs: boxShade (the shade of the box: either a shade or a form that will be evaluated to a shade; e.g. GRAYSHADE), boxWhiteSpace (blank space between the boxed object and the box), boxWidth (the width of the box), boxedObject (a LoopsImageObject).

Unknown IMAGEOBJ type
GETFN: LoopsImageObjectGetFn

A common document form consists of a table of text entries. These are easy to build in TEdit when each entry fits within its row without having to occupy multiple lines.

Unknown IMAGEOBJ type
GETFN: LoopsImageObjectGetFn

Unknown IMAGEOBJ type
GETFN: LoopsImageObjectGetFn

Unknown IMAGEOBJ type
GETFN: LoopsImageObjectGetFn

Example: BoxedImageObjects and TableTextObjects

The following class takes care of the case where the table entries are too large to fit on a single line.

`TableTextObject` [Class]

describes a (possibly multiple line) table entry. The class `TableTextObject` defines five important IVs.

`text` [IV of TableTextObject]

contains the text that will be displayed in the table entry.

`widthInWs` [IV of TableTextObject]

defines the width of the table entry. This width is measured in "W"s, in the current font. If the `text` is too long, it will be broken at word boundaries and displayed in multiple lines. The default value is `10`.

`nLines` [IV of TableTextObject]

defines the maximum number of lines to be used to print the text. If `nLines` is `NIL` (the default), the object will be as tall as it needs to be to display the entire `text` subject to the width constraint specified by the `widthInWs` IV.

`justify` [IV of TableTextObject]

determines the horizontal justification of the individual lines within the table entry. Possible values are `left` (for left–justify), `center` (for centered text), and `right` (for right–justify). The defualt is `left`.

`verticalJustify` [IV of TableTextObject]

determines the vertical justification of the lines within the table entry. Possible values are `top` (the first line will be flush with the top of the table entry), `center` (the text will be centered vertically), and `bottom` (the last line of text will be at the bottom of the entry). The defualt is `bottom`. Note that this IV only has an effect if the IV `nLines` is specified.

Unknown IMAGEOBJ type
GETFN: LoopsImageObjectGetFn

Unknown IMAGEOBJ type
GETFN: LoopsImageObjectGetFn

Image objects can be used to include prettyprinted forms in a document.

`PPImageObject` [Class]

prettyprints a form in the image stream. The relevant IVs of `PPImageObject` are:

`form` [IV of PPImageObject]

~mumble~

`minWidth` [IV of PPImageObject]

~mumble~

`maxWidth` [IV of PPImageObject]

~mumble~

Due to an unfortunate bug in Interlisp, `PPImageObject` work correctly on display devices .

Unknown IMAGEOBJ type
GETFN: LoopsImageObjectGetFn

Example: PPImageObject

Unknown IMAGEOBJ type
GETFN: LoopsImageObjectGetFn

Unknown IMAGEOBJ type
GETFN: LoopsImageObjectGetFn Loops image objects are Loops objects that are wrapped around an Interlisp `IMAGEOBJ`. The `IMAGEOBJ` in turn points back to the Lops image object. When Interlisp "sends a message" to the underlying `IMAGEOBJ` (more correctly, applies one of the `IMAGEOBJ`'s `IMAGEFNS` to the `IMAGEOBJ`), the `IMAGEOBJ` forwards this message on to the Loops image object. The translation from Interlisp `IMAGEOBJ` protocol to Loops image object protocol is accomplished by a special set of `IMAGEFNS` and the `LoopsImageObject` class.

`LoopsImageObject` [AbstractClass]

~mumble~

Unknown IMAGEOBJ type
GETFN: LoopsImageObjectGetFn

The following methods implement the IMAGEFNS for the Loops image object. These message are not intended to be sent by the user; they are sent automatically by TEdit and other packages.

Unknown IMAGEOBJ type
GETFN: LoopsImageObjectGetFn

(_ *self* ImageBox *imageStream currentX*
rightMargin) [Method of LoopsImageObject]

returns an instance of the record IMAGEBOX with fields XSIZE, YSIZE, XKERN, and YDESC. This is used by TEdit to determine the size of the object for formatting purposes. Specializations of LoopsImageObject should override this method.

(_ *self* Display *imageStream*) [Method of LoopsImageObject]

should actually display the object on the *imageStream*. Specializations of LoopsImageObject should override this method.

Unknown IMAGEOBJ type
GETFN: LoopsImageObjectGetFn

(_ *self* ButtonEventIn *windowStream selection*
relX relY window textStream button) [Method of LoopsImageObject]

~mumble~

(_ *self* CopyButtonEventIn *windowStream*) [Method of LoopsImageObject]

~mumble~

Unknown IMAGEOBJ type
GETFN: LoopsImageObjectGetFn

(_ *self* Copy) [Method of LoopsImageObject]

~mumble~

Unknown IMAGEOBJ type
GETFN: LoopsImageObjectGetFn

Due to restrictions imposed by IMAGEOBJects, LoopsImageObjects cannot duplicate the full generality of saving and retoring Loops objects. Specialized subclasses of LoopsImageObjects cannot change how they are stored or read back in. However, the following methods do give the user some control over what extra information is dumped out when a LoopsImageObjects is stored.

(_ *self* BeforePutToFile *stream*) [Method of LoopsImageObject]

~mumble~

(_ *self* AfterPutToFile *fileStream*) [Method of LoopsImageObject]

~mumble~

(_ *self* AfterGetFromFile *textStream*) [Method of LoopsImageObject]

~mumble~

(_ *self* PrePrint) [Method of LoopsImageObject]

~mumble~

Unknown IMAGEOBJ type
GETFN: LoopsImageObjectGetFn

(_ *self* WhenCopied *targetWindowStream*
sourceTextStream targetTextStream) [Method of LoopsImageObject]

~mumble~

(_ *self* WhenDeleted *targetWindowStream*
sourceTextStream targetTextStream) [Method of LoopsImageObject]

~mumble~

(_ *self* WhenInserted *targetWindowStream*
sourceTextStream targetTextStream) [Method of LoopsImageObject]

~mumble~

(_ *self* WhenMoved *targetWindowStream*
sourceTextStream targetTextStream) [Method of LoopsImageObject]

~mumble~

`(_ `*`self`*` WhenOperatedOn `*`windowStream howOperatedOn`*
    `selection textStream``)` [Method of LoopsImageObject]

~mumble~

Unknown IMAGEOBJ type
GETFN: LoopsImageObjectGetFn

Unknown IMAGEOBJ type
GETFN: LoopsImageObjectGetFn

The class `LoopsImageObject` provides a number of other methods that can be used by specializations of the above methods.

`(_ `*`self`*` CachedImageBox `*`imageStream`*`)` [Method of LoopsImageObject]

returns the image box computed by the last `ImageBox` method. This is often used inside the `Display` method to avoid resending the `ImageBox` message.

`(_ `*`self`*` DisplayImageStream? `*`imageStream`*`)` [Method of LoopsImageObject]

returns `NIL` unless the image stream is a display image stream.

`(_ `*`self`*` PrintText `*`imageStream text font`*`)` [Method of LoopsImageObject]

prints the string *`text`* in the font *`font`*, centered in the object's `CachedImageBox`.

Unknown IMAGEOBJ type
GETFN: LoopsImageObjectGetFn

Certain subclasses of `LoopsImageObject` capture some commonly desired functionality. These may be useful to anyone interested in building their own Loops image objects.

`TEditImageObject` [AbstractClass]

The class `TEditImageObject` contains a few methods that are only applicable when the object is being displayed in a TEdit stream.

`(_ `*`self`*` CurrentFont `*`imageStream`*`)` [Method of TEditImageObject]

returns the font of the image object in the current TEdit stream.

`(_ `*`self`*` TextStream `*`imageStream`*`)` [Method of TEditImageObject]

returns the text stream that is being displayed in *`imageStream`*.

`(_ `*`self`*` TEditIV `*`ivName textStream`*`)` [Method of TEditImageObject]

edits the text stored in the IV *`ivName`* of *`self`* in a new TEdit window. The image object should be part of the TEdit stream *`textStream`*. The textStream argument is used to update the display when the IV has been edited.

`(_ `*`self`*` AllObjects `*`textStream`*`)` [Method of TEditImageObject]

returns a list of all imageobjects contained in *`textStream`*, in order.

`WhenSavedImageObject` [AbstractClass]

~mumble~

`HardcopySideEffectObject` [AbstractClass]

~mumble~

`LabelImageMixin` [AbstractClass]

~mumble~

`EditableImageObjectMixin` [AbstractClass]

~mumble~

`TEditableImageObjectMixin` [AbstractClass]

~mumble~

Unknown IMAGEOBJ type
GETFN: LoopsImageObjectGetFn

On occasion, it may be useful to wrap a Loops image object around an existing Interlisp `IMAGEOBJ`, say to wrap a BoxedImageObject around it.

`ImageObjectWrapper` [Class]

~mumble~

Unknown IMAGEOBJ type
GETFN: LoopsImageObjectGetFn