Working with Image List Controls

This document covers the following topics:

Introduction

An image list control is a container of ordered images that can be associated with particular control types, such as list view and tree view controls. It allows images to be efficiently re-used by the control’s items without the image being re-loaded from the disk each time. It also ensures that all images are compatible (e.g., are of the same size and color organization).

Creating the Image List Control

Image list controls are created, as usual, via the `ADD` action:

```
PROCESS GUI ACTION ADD WITH
PARAMETERS
   HANDLE-VARIABLE = #IMGLST-1
   TYPE = IMAGELIST
   PARENT = #DLG$WINDOW
   STYLE = ’LS’
END-PARAMETERS GIVING *ERROR
```

An image list control may consist of up to two sets of images internally, one consisting of large images (typically 32 by 32 pixels) and one consisting of small images (typically 16 by 16 pixels). Which of these (if any) is created internally depends on the image list control’s "Large Images (L)" and "Small Images (S)" `STYLE` flags. If neither of these flags are specified, a single set of images is created, with an explicit image size as determined by the image list control’s `ITEM-W` and `ITEM-H` attribute values. If both of these are zero, small images are assumed.

Adding Images

Images are added to an image list by creating an image control, based on the required image (bitmap or icon) file, as a child of the image list control:

```
PROCESS GUI ACTION ADD WITH
PARAMETERS
   HANDLE-VARIABLE = #IMG-1
   TYPE = IMAGE
   PARENT = #IMGLST-1
   BITMAP-FILE-NAME = ’example.bmp’
END-PARAMETERS GIVING *ERROR
```

Images are appended to the list by default, unless the `SUCCESSOR` attribute is used to insert them at a specific position.

Composite Images

Image controls can be categorized into two types: single-image image controls and multi-image image controls.

Single-image image controls contribute a single image to each set of images stored by the parent image list control. That is, if the image list contains both large and small images, one of each is provided by the image control. Single-image image controls may be bitmaps or icons.

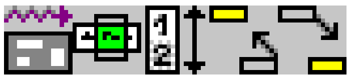
Multi-image image controls, as the name suggests, may contribute more than one image (in each required size) to the parent image list control. Multi-image image controls must be based on bitmap files, rather than icons, and are distinguishable from single-image image controls in that their "Composite image (C)" `STYLE` flag is set:

```
PROCESS GUI ACTION ADD WITH
PARAMETERS
   HANDLE-VARIABLE = #IMG-1
   TYPE = IMAGE
   PARENT = #IMGLST-1
   STYLE = ’CsT’
   BITMAP-FILE-NAME = ’composite.bmp’
END-PARAMETERS GIVING *ERROR
```

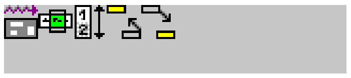
The number of images in the composite bitmap is automatically calculated from the size of the bitmap and the width and height of the images in the (smallest) set of images stored by the parent image list control. Thus, in the case where both large and small images are stored, the bitmap would typically be 16 pixels high, and (16 * N) pixels wide, where N is the number of images to be stored in the image control. Here is an example of a composite bitmap containing five images:

Scaling and Transparency

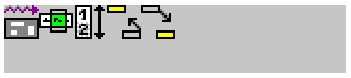
In the example provided in the preceding section, two other style flags were specified in addition to the "Composite image (C)" style: namely, the "Scaled (s)" and the "Transparent (T)" style flags. The first of these is absolutely necessary if the parent image list control contains multiple sets of images in different sizes. For example, if large images are also being used, the flag causes the composite image to be scaled internally first before being chopped up into its constituent images, as follows:



Note that if the "Scaled (s)" style flag were not specified, the composite bitmap would be extended in the background color, rather than being scaled, before being chopped up, as shown below:



This would result in the following five large images:



Needless to say, this is not normally what you want!

The "Transparent (T)" style flag indicates that the images should be rendered transparently, such that all pixels in the bitmap's background color are not drawn. The background color can be explicitly specified by setting the `BACKGROUND-COLOUR-VALUE` and/or `BACKGROUND-COLOUR-NAME` attributes for the image control to the required value. Otherwise, if no color is specified (as in the previous example), the color of the first (i.e., top-left) pixel in the bitmap is taken as being the background color.

Of course, both the "Scaled (s)" and the "Transparent (T)" style flags can also be applied to non-composite images.

Bitmaps vs. Icons

Apart from not being able to source mulitple images (as described above), icons differ from bitmaps in two important ways. Firstly, a single icon (*.ICO*) file can contain multiple versions of the icons in different sizes. Thus when Natural requires the large image, and the source is an icon file, the large icon defined in the icon file is used, if present, in preference to synthesizing it from one of the other icons in the file by scaling. Similarly, when Natural requires the small image, and the source is an icon file, the small icon defined in the icon file is used, if present. In contrast, bitmap files do not contain multiple images, so if both large and small images are required for an image list, one of the two images (usually the large image) must by synthesized from the other as described in the previous section.

Secondly, icons typically contain a monochrome bitmap (known as the image mask) that determines which pixels in the image are transparent (i.e., should not be drawn). Thus, when Natural loads an image from an icon file, and the image control's `BACKGROUND-COLOUR-NAME` attribute is set to `DEFAULT` (or is not specified), and the image control's "Transparent (T)" style flag is specified without the "Scaled (s)" style flag, Natural uses the icon's transparency mask, instead of making the above-mentioned assumption that all pixels in the same color as the first pixel are to be rendered transparently, as is the case for images loaded from a bitmap file. If an explicit (i.e., non-default) background color is specified, all pixels in this color are treated as transparent, regardless of whether an icon or bitmap is being used. The icon's transparency mask is ignored here, as is also the case if the icon is scaled.

Therefore, if both large and small images are needed, it may be preferable to use single-image image controls based on icon files containing both large and small representations of the image, rather than use a multi-image image control based on a single composite bitmap. The use of individual icon (*.ICO*) files has the advantage that the and large representations of the image (assuming that both are provided in the file) can have different levels of detail. The main disadvantage is that it normally takes longer to load the images from multiple icon files than it does to load them from a single composite bitmap file.

Using an Image List

Before any images from the image list can be used by a control (such as the tree view or list view control), the image list must be associated with the control. This association is achieved by assigning the handle of the image list control to the host control's `IMAGE-LIST` attribute. For example:

```
#LV-1.IMAGE-LIST := #IMGLST-1
```

Having set the image list, the image list control's images are now available for use by the control's items.

Referencing Images from the Image List

To use a particular image from the parent control's image list for a particular item (e.g. list view item or tree view item), the image to be used has to be specified in one of two ways:

1. By setting the item's `IMAGE` attribute to the handle of the image control and (if necessary) the item's `IMAGE-INDEX` attribute to the relative offset of the required image (starting from zero) within the image control. If the image control only contains one image, it is not necessary to specify an image index. The image specified must belong to the image list control assigned to the item's container.

2. By setting the item's `IMAGE-INDEX` attribute to the ordinal of the image within the image list (1=first image, 2=second image, and so on). The item's `IMAGE` attribute must be either not specified or set to the default value of `NULL-HANDLE` in this case.

In the first case (relative indexing), wrap-around is used on the index. Thus, if an image control has N images, an image index of 0 refers to the first image in the image control, an image index of (N - 1) refers to the last image, and an image index of N refers to the first image again, and so on. Thus, if the image control only contains one image, the relative image index (if specified at all) has no effect: due to wrap-around, the first (and only) image will always be taken.

In the second case (absolute indexing), no wrap-around is used on the image index, which must be in the range 1 through to the number of images in the image list (inclusive). If the specified value is not in this range, no image is displayed for the specified item.

Note that the `IMAGE-INDEX` attribute can also be applied to an image control. In this case, the attribute is read-only, and returns the offset (starting from zero) of the image control's first image within the parent image list control.

One advantage of using relative indexing is that Natural keeps track the references to the specified image (both in the dialog editor and at run-time) and automatically propagates changes to the image control or to its position in the image list. In practice, absolute indexing is probably most useful in situations where an image list control with a single composite (i.e., multi-image) image control is used, and where the images are not modified at run-time.

Overlay Images

There are situations where it is desirable to be able to offer several variations of an image. For example, the displayed image for an item representing a folder may need to be modified to indicated that the folder is active. Rather than providing an image of a folder and an image of an active folder, it may be more convenient to provide only the first of these images, and to indicate the active state via a second image containing only the "active" symbol, which is then superimposed on the first. Such an image is referred to as an *overlay* image, to distinguish it from the underlying *base* image.

Overlay images are contained within the same image list that is used to display the base images, as determined via the host control's `IMAGE-LIST` attribute. They are therefore the same size as the base images, but are always rendered transparently, to allow the underlying image to show through.

To use an overlay image for an item, a value must be specified for the item's `OVERLAY` and/or `OVERLAY-INDEX` attributes. These attributes are used analogously to the `IMAGE` and `IMAGE-INDEX` attributes (respectively) for base images (see above).

For technical reasons, images intended for use as overlay images must be "pre-registered". In Natural, this is done by setting the image list control's "O" (Overlay) `STYLE`. However, if the overlay controls are defined statically, this style is automatically set by the dialog editor. The presence or absence of this style distinguishes base images from overlay images. Consequently, the `OVERLAY` attribute (if specified) can only refer to an image control with this style, whereas the `IMAGE` attribute (if specified) can only refer to an image control without it. If absolute indexing (see above) is being used, the `IMAGE-INDEX` can refer to an overlay image (which is then "misused" as a base image). However, a corresponding attempt to use the `OVERLAY-INDEX` attribute to refer to a base image fails (no overlay image is drawn).

Windows sets a limit on the number of overlay images that may be defined for an image list. This limit is currently 15. Note that if any composite overlay image controls are used, each sub-image in the composite bitmap counts separately towards this quota.

As an example, suppose we create an image control based on a composite image containing the individual overlay images, as follows:

```
PROCESS GUI ACTION ADD WITH
PARAMETERS
   HANDLE-VARIABLE = #IMG-2
   TYPE = IMAGE
   PARENT = #IMGLST-1
   STYLE = 'COs'
   BITMAP-FILE-NAME = 'overlays.bmp'
END-PARAMETERS GIVING *ERROR
```

Then, we could create a list view item (say) using the second overlay image from the composite bitmap by executing the following code:

```
PROCESS GUI ACTION ADD WITH
PARAMETERS
   TYPE = LISTVIEWITEM
   PARENT = #LV-1
   STRING = 'Item with overlay'
   IMAGE = #IMG-1
   IMAGE-INDEX = 3
   OVERLAY = #IMG-2
   OVERLAY-INDEX = 1
END-PARAMETERS GIVING *ERROR
```

In the above example, the list view item will use the fourth image from *COMPOSITE.BMP* as its base image, and the second image from *OVERLAYS.BMP* as the overlay image (relative image indexes are, as already mentioned, zero-based). Note that the list view item is created anonymously (i.e., no explicit `HANDLE-VARIABLE` attribute value specified).

Modifying Images

Image controls may be modified even if they are currently in use. For example:

```
#IMG-1.BITMAP-FILE-NAME := 'new.bmp'
```

Natural keeps track of, and automatically updates and redraws, each item that explicitly (i.e., via relative indexing) references an image from the modified image control. However, if absolute indexing is used, the corresponding items are not updated, even if they are implicitly referring to an image within the modified image control.

Deleting Images

Images may be removed from the image list by deleting the complete image control via the `DELETE` action. For example:

```
PROCESS GUI ACTION DELETE WITH #IMG-1 GIVING *ERROR
```

All items that *explicitly* (i.e., via relative indexing) reference an image from the deleted image control are automatically updated and redrawn to show no image.

However, if absolute indexing is being used, no automatic updating occurs. For example, suppose an image list control contains three single-image image controls and that items exists that refer to all three images via absolute indexing. If the second image control is deleted, the items that used to refer to the second image would suddenly reference the third image and the items that used to refer to the third image would "fall off the end" and not reference anything. Furthermore, the controls containing the items would not automatically be redrawn to reflect the changes.

It is, of course, also possible to delete all images in the image list in one go, via the `DELETE-CHILDREN` action:

```
PROCESS GUI ACTION DELETE-CHILDREN WITH #IMGLST-1 GIVING *ERROR
```

This is equivalent to deleting each image in the image list individually.

Note that it is not possible to delete individual images within a composite (i.e., multi-image) image control.

Deleting the Image List Control

An image list control may be deleted when no longer required, even if it is in use. For example:

```
PROCESS GUI ACTION DELETE WITH #IMGLST-1 GIVING *ERROR
```

All controls using the image list control are updated accordingly, and their `IMAGE-LIST` attribute is automatically reset to `NULL-HANDLE`.