XEROX SDD ARCHIVES
I have read and understood
Pages_____ To_____
Reviewer_____ Date_____
# of Pages_____ Ref. 78SDD-151

XEROX
BUSINESS SYSTEMS
Systems Development Department

To:	Distribution	Date:	June 26, 1978
From:	P. Jarvis	Org:	SDD/SD System Architecture
Subject:	IUFTP Functional Specification	Filed:	[Iris] < Jarvis > iutfp-cover.bravo

The attached document is a draft section of **Pilot: The OIS Control Program Functional Specification** describing the facilities which support the IUTFP. The software interface which supports the UTVFC will almost be identical.

I would much appreciate receiving your comments before incorporating this section into the functional specification. Your comments can have a large impact; I circulate this draft more for purposes of discussion hoping to reach a final consensus on the form of the display interface. The salient points of this design include no hardware assist for implementing a cusor, low level support for a very primitive display controller, and no software for encoding keyboard data into a series of key strokes.

c: J. Cameron, L. Clark, J. Frandeen, C. Hankins, P. Heinrich, C. Irby, P. Jarvis, W. Kennedy, G. Kusnick, H. Lauer, M. Louros, W. Lynch, T. Malloy, R. Metcalfe, L. Padgett, G. Shaw, C. Thacker, T. Townsend

This document can be found on [Iris] < Jarvis > iutfp-specs.bravo

1.1 Client Control of the IUTFP

The Interim User Terminal Full Page, IUTFP consists of a mouse, a keyboard, and a high resolution video display. The display is refreshed from a bit map in main storage. The 128 key keyboard and the three mouse buttons are unencoded; the controller maintains their state as a bit array in main storage along with two words for the x and y mouse coordinates.

1.1.1 Creation of a IUTFP Channel

To create an IUTFP channel the client calls

```
IUTFP.Create: PROCEDURE[] RETURNS[IUTFP.ChannelHandle];
```

To delete the IUTFP channel, the client calls

```
IUTFP.Delete: PROCEDURE[IUTFP.ChannelHandle];

IUTFP.ChannelHandle: TYPE = PRIVATE . . . ;
```

Subsequent calls to the procedures described below allow the client to manipulate the display, keyboard, and mouse.

1.1.2 Display

The controller scans the display line by line from top to bottom starting with line zero. The client can partition the display into a series of horizontal stripes mapping a block of storage into a stripe with a record of the form

```
IUTFP.PhysicalRecord: TYPE = RECORD [
      background: {black, white},
      y: CARDINAL,
      leftMargin: CARDINAL,
      pixelsPerLine: CARDINAL,
      lines: CARDINAL,
      buffer: LONG POINTER];

IUTFP.PhysicalRecordHandle: POINTER TO IUTFP.PhysicalRecord;
```

The client sets the background to either black or white with **background** field. The background at the bottom of the display matches the background of the last stripe in the display list.

The **y** field specifies the location of the stripe relative to the other stripes in the display list. The value of **y** does not fix the absolute location of the stripe on the display, instead, this value serves as a hint to the channel about where to insert the stripe in the display list. This scheme makes overlapping stripes impossible; furthermore, if the client attempts to insert two **PhysicalRecords** with the same **y**, he cannot predict which stripe will appear first in the display list.

The **leftMargin** contains a count of the number of pixels to skip before displaying data from the buffer on the scan line. This number should be evenly divisible by 16, if not, the low four bits are ignored.

The **pixelsPerLine** field holds the number of pixels on a line which come from the scan line buffer. The other pixels on the line are displayed as background. This number should be evenly divisible by 64; the low six bits of this word are ignored. If the client wishes to display only background in the stripe, he should set this field to zero.

The number of lines scanned in the stripe comes from the **lines** field.

Finally, **buffer** points to the scan line buffer of (pixelsPerLine*lines)/16 words. The controller scans each line from left to right as it reads from the buffer word by word from low addresses to higher ones. The high order bits of a word are displayed to the left of the low order bits. A zero represents a pixel which matches the background, while a one represents a bit which complements the background.

Note: The controller does not support a cursor. The client can simulate the effect of a cursor by modifying the bit map from which the controller refreshes the display.

The client inserts a stripe into the display list by calling

```
IUTFP.Put: PROCEDURE[channel: IUTFP.ChannelHandle, rec:
    IUTFP.PhysicalRecordHandle] RETURNS[IUTFP.CompletionHandle];
```

The stripe is inserted into the display list before the procedure returns. Inserting a stripe into the middle of a display list scrolls down any stripes below.

The client can modify any or all of an existing stripe's parameters by calling

```
IUTFP.ModifyStripe: PROCEDURE[channel: IUTFP.ChannelHandle, rec:
    IUTFP.PhysicalRecordHandle, stripe: IUTFP.CompletionHandle]
    RETURNS[error: BOOLEAN];
```

If the client passes a **CompletionHandle** that does not match a stripe in the display list, the procedure returns true. If the location of the stripe relative to other stripes in the display list changes, the specified stripe moves on the display.

A stripe is removed from the display list by calling

```
IUTFP.Remove: PROCEDURE[channel: IUTFP.ChannelHandle, stripe:
    IUTFP.CompletionHandle];
```

The stripe is removed from the display before the procedure returns. Any stripes below the one removed are scrolled up.

The IUTFP controller supports more than one display configuration. To obtain information about the attached display the client calls:

```
IUTFP.DisplayParameters: PROCEDURE[channel: IUTFP.ChannelHandle]
     RETURNS[XPixels: CARDINAL, YPixels: CARDINAL, XLength: CARDINAL,
     YLength: CARDINAL];
```

The number of scan lines and pixels on a scan line are given by YPixels and XPixels respectively. The dimensions, in millimeters, of the displayed area are given by XLength and YLength.

1.1.3 Keyboard and Mouse

The controller maintains the state of mouse and keyboard in the KeyRecord.

```
IUTFP.KeyRecord: TYPE = MACHINE DEPENDENT RECORD [
     x--(2*w:w)--: CARDINAL,
     y--(3*w:w)--: CARDINAL,
     keys--(4*w: 8*w)--: ARRAY [0..7] OF UNSPECIFIED];
```

The pixel coordinates of the mouse are given by x and y. The mouse axes correspond exactly with the display axes.

Each of the 128 bits of keys corresponds to a keyboard key or a mouse button. If the bit is a one the correspoinding key is depressed.

Note: At present, the mapping between the keys on the keyboard and its corresponding location in the bit array is unknown.

The client discovers the location of the KeyRecord by calling

```
IUTFP.LocateKeyRecord: PROCEDURE[channel: IUTFP.ChannelHandle]
     RETURNS[LONG POINTER TO KeyRecord];
```

A process which wants to wait for a key stroke can call

```
IUTFP.WaitForKeystroke[channel: IUTFP.ChannelHandle];
```

This procedure does not return until a key is released, depressed, or the mouse location changes.

1.1.4 Programming the Controller

[*This section really belongs in the design specification. I include it to keep information from fragmenting.*]

The format of the IUTFP controller status block is

```
IUTFP.CSB: TYPE = MACHINE DEPENDENT RECORD [
     chain--(0:w)--: IUTFP.IOCBIndex,
     pad48--(w:3*w)--: UNSPECIFIED,
     pad8--(0:8)--: [0..377B],
     change--(8:24)--: LONG POINTER TO CONDITION,
          keys--(4*w, 12*w)--: IUTFP.KeyRecord];
```

The format for the IUTFP input/output control block is given by

```
     IUTFP.IOCB: TYPE = MACHINE DEPENDENT RECORD [
          next--(0:w)--: IUTFP.IOCBIndex, -- quad word aligned
          bow--(2*w:1)--: BOOLEAN,  -- black on white bit
          pad9--(2*w+1:9): [0..777B],
          left--(2*w+10:6)--: [0..77B], -- in words
          lineSize--(3*w:w)--: CARDINAL, -- in words
          pad8--(0:8)--: [0..377B],
          buffer--(8:24)--: LONG POINTER TO UNSPECIFIED];
```