krsnake

COLLABORATORS			
	TITLE : krsnake		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		July 20, 2024	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

krsnake

1.1 krsnake.doc

```
---overview---()
KsDeleteSoundObject()
KsGetClientSig()
KsPlaySoundObject()
KsReadEvent()
KsReadKRSNAkePrefs()
KsReadSoundObject()
KsRegisterClient()
KsRemoveClient()
KsWaitForEvent()
KsWriteKRSNAkePrefs()
```

1.2 krsnake.library/---overview---

```
WRITING KRSNAKE CLIENTS

     KRSNAke clients need to communicate with the main program. This is
     done through krsnake.library. A client would call
     KsRegisterClient() in order to receive messages from KRSNAke, wait
     around for the client's signal to arrive, act upon the received
     message, repeat that until a SNAKE_QUIT message arrives, then call
     KsRemoveClient(). There's nothing more to it.

     A KRSNAke client might look like this:

         client = KsRegisterClient();
             ...
             KsWaitForEvent(client);
             KsReadEvent(client,&event,&data);
             ...
         KsRemoveClient(client);

     Although more commonly you'd want to listen to other ports as
     well, for instance that of a window:
```

```
client = KsRegisterClient();
    ...
    mask = (1<<KsGetClientSig(client))|(1<<windowsigbit);
    Wait(mask);
    KsReadEvent(client,&event,&data);
    ...
KsRemoveClient(client);
```

If you experience difficulties following my explanation (which I suppose is entirely possible), try looking at the sources of the clients I've written. Sources speak more eloquently than words. If you don't read E - tough luck, you're stuck with my explanation.

NOTE ABOUT THE INTERFACE

Clients should never open any interface window, or generally give off any clue whatsoever that they exist, until told to do so by KRSNAke. Clients should do general initialisation, attach themselves to KRSNAke, then wait around for a SNAKE_SHOWINTERFACE event to be broadcast. At THIS point, they should render. Of course, when dealing with particularly weird users, this could never happen. Be prepared for everything.

EVENT CODES

KRSNAke broadcasts the following event codes (with data codes in parentheses):

SNAKE_QUIT (no data)

If your client receives SNAKE_QUIT, remove it as soon as possible. This means KRSNAke is trying to quit - and it can't until you've removed your client. Generally, this message doesn't just mean "call KsRemoveClient()", it means "quit your program".

SNAKE_NEWSCORE (current score)

This message is broadcast whenever the score (the length of the snake) changes.

SNAKE_GAMEOVER (final score)

When the snake crashes, SNAKE_GAMEOVER is broadcast, along with the snake's final length.

SNAKE_NEWGAME (no data)

This message is broadcast when the player begins a new game.

SNAKE_PAUSED (no data)

This message is broadcast when the game is paused.

SNAKE_RESTARTED (no data)

This message is broadcast when the game is resumed after a pause.

SNAKE_EATEN (fruit's value)

When the snake eats a fruit, this code is broadcast along with the fruit's value.

SNAKE_MOVES (head's coordinates)

Each time the snake's position changes (its head moves), this message is broadcast along with the new coordinates of the head. The coordinates are broadcast as a word, where the most significant byte is the Y coordinate (between 0 and 31), and the least significant byte is the X coordinate (also between 0 and 31).

SNAKE_NEWCHUNK (fruit data; see below)

Whenever a new fruit arrives (either at the start of the game, or when the snake eats a fruit), this message is broadcast along with all info about the new fruit. The data is coded as a longword, each value represented as a byte as follows, most significant byte first: fruit value (between 1 and 9), fruit colour (between 0 and 3; RGB values can be obtained from KRSNAke's prefs file), fruit Y coordinate (between 0 and 31), and fruit X coordinate (between 0 and 31).

SNAKE_SHOWINTERFACE (no data)

When this message is broadcast, the client should open its window. If the window is already open, do nothing - but make sure you handle the message, because these messages are sent occasionally just to be sure the client really did get the message.

SNAKE_HIDEINTERFACE (no data)

When this message is broadcast, the client should close its window and anything similar. The same applies here as to SNAKE_SHOWINTERFACE - duplicate messages do occur! It's generally unwanted that your client creates an AppIcon or something similar for itself when it's hidden - all show/hide info should come from KRSNAke.

NOTE

Generally, it's a bad idea to call any of the server functions from your own code. Only KRSNAke should do this; at any one time, there can be only one server in existence, and this server should be owned by KRSNAke's main task. This library is not intended as a multipurpose client/server library - it's only for KRSNAke.

In fact, I've removed the docs for the server functions below. Just be a nice fellow and don't call them, OK?

```
     YOU HAVE BEEN WARNED!

         "We hear what you say,
          we see what you do;
          we know everything we need to know about you."
```

1.3 krsnake.library/KsDeleteSoundObject

```
NAME
     KsDeleteSoundObject -- delete a sound object

SYNOPSIS
     KsDeleteSoundObject(object)
                         D0

     void KsDeleteSoundObject(APTR);

FUNCTION
     Deletes a sound object created with KsReadSoundObject().

INPUTS
     object - pointer to an object returned from KsReadSoundObject()

SEE ALSO
     KsPlaySoundObject(), KsReadSoundObject()
```

1.4 krsnake.library/KsGetClientSig

```
NAME
     KsGetClientSig -- get the signal of a client port

SYNOPSIS
     signal = KsGetClientSig(client)
     D0                      D0

     UBYTE KsGetClientSig(APTR);

FUNCTION
     This function returns the signal that will be set when a message
     arrives at the given client's message port. Using a Wait() for
     this signal is somewhat more useful than using KsWaitForEvent(),
     as this will let you listen to other signals as well.

INPUTS
     client - a pointer to the client whose signal you wish to obtain.

RESULT
     signal - the number of the signal associated with the given
              client's port.

SEE ALSO
     KsReadEvent(), KsWaitForEvent()
```

1.5 krsnake.library/KsPlaySoundObject

```
NAME
     KsPlaySoundObject -- trigger a sound object

SYNOPSIS
     success = KsPlaySoundObject(object)
     D0                          D0

     ULONG KsPlaySoundObject(APTR);

FUNCTION
     Starts playing a sound object created with KsReadSoundObject(),
     either by triggering the sound.datatype or by starting the
     respective replayer. Note that only datatypes can be played more
     than one at a time. Also, beware of the ProTracker replayer, which
     is not working at present.

INPUTS
     object - pointer to a sound object returned from
              KsReadSoundObject()

RESULT
     success - TRUE if the object was successfully triggered, FALSE
               otherwise

SEE ALSO
     KsDeleteSoundObject(), KsReadSoundObject()
```

1.6 krsnake.library/KsReadEvent

```
NAME
     KsReadEvent -- receive a message from the server

SYNOPSIS
     success = KsReadEvent(client,event,data)
     D0                    A0     D1    D0

     ULONG KsReadEvent(APTR,ULONG *,ULONG *);

FUNCTION
     This function reads a message from the client's message port. If a
     message exists, it will copy the event and data codes into the
     addresses provided by the caller, and dispose of the message.

     You should keep calling this function until it returns FALSE, as
     several messages might arrive at the port simultaneously.

INPUTS
     client - the client whose port to check.
     event - pointer to a longword in which to store the event code of
             the message, if there is one.
     data - ditto for the data code.
```

```
RESULT
     success - TRUE if there was a message, in which case the longwords
               pointed to by the event and data fields will contain the
               info broadcast by the message, FALSE if there was no
               message.

SEE ALSO
     KsNotifyClients()
```

1.7 krsnake.library/KsReadKRSNAkePrefs

```
NAME
     KsReadKRSNAkePrefs -- read the user preferences

SYNOPSIS
     prefs = KsReadKRSNAkePrefs()
     D0

     struct KPrefs * KsReadKRSNAkePrefs(void);

FUNCTION
     Reads the file "ENVARC:KRSNAke/KRSNAke.prefs" and creates a
     KPrefs structure out of it. This structure is allocated with
     AllocVec(), so when you're done with it, call FreeVec(). If
     you've changed anything and want to make it permanent, call
     KsWriteKRSNAkePrefs() rather than FreeVec().

RESULT
     prefs - a pointer to an allocated KPrefs structure. Unless you
             are the KRSNAke prefs program, you should consider this
             structure read only! If no prefs could be found, or if
             there was no memory to read them, the function returns
             NULL.

SEE ALSO
     KsWriteKRSNAkePrefs()
```

1.8 krsnake.library/KsReadSoundObject

```
NAME
     KsReadSoundObject -- create a sound object

SYNOPSIS
     object = KsReadSoundObject(filename)
     D0                         D0

     APTR KsReadSoundObject(char *);

FUNCTION
     This function reads a file from a filesystem, and if the file is
     a recognised sound data file, a sound object, which can
     subsequently be played, is created.
```

```
INPUTS
     filename - a pointer to the name of the file to attempt to load

RESULT
     object - a sound object which can be played with KsPlaySoundObject,
              or NULL if something went wrong

SEE ALSO
     KsDeleteSoundObject(), KsReadSoundObject()
```

1.9 krsnake.library/KsRegisterClient

```
NAME
     KsRegisterClient -- create a KRSNAke message client

SYNOPSIS
     client = KsRegisterClient()
     D0

     APTR KsRegisterClient(void);

FUNCTION
     This function creates a client of the KRSNAke message server. A
     KRSNAke client should call this in order to receive messages from
     KRSNAke through the KsReadEvent() call.

     The message server must be active before any clients can be
     created. If it is not, this function will fail. Normally, clients
     will be launched by KRSNAke after the server has been created, so
     this should not be a problem.

RESULT
     client - a pointer to a KRSNAke message client structure, or FALSE
              if the client could not be created.

SEE ALSO
     KsRemoveClient(), KsRegisterServer(), KsReadEvent()
```

1.10 krsnake.library/KsRemoveClient

```
NAME
     KsRemoveClient -- destroy a KRSNAke message client

SYNOPSIS
     success = KsRemoveClient(client)
     D0                       D0

     ULONG KsRemoveClient(APTR);

FUNCTION
     This function detaches a client from the server and destroys it.
```

```
     All clients created by the KsRegisterClient() function must be
     disposed of with this function.

INPUTS
     client - a pointer to a client returned from the
              KsRegisterClient() function.

RESULT
     success - TRUE if the client was removed, FALSE otherwise.

SEE ALSO
     KsRegisterClient()
```

1.11 krsnake.library/KsWaitForEvent

```
NAME
     KsWaitForEvent -- wait for a message to arrive at a client's port

SYNOPSIS
     success = KsWaitForEvent(client)
     D0                       D0

     ULONG KsWaitForEvent(APTR);

FUNCTION
     This function calls WaitPort() for the given client's port, which
     causes the calling task to wait until the client receives a
     message from the server.

INPUTS
     client - the client whose port to wait on.

RESULT
     success - TRUE if the wait succeeded, FALSE otherwise. At present,
               the FALSE return value only happens if you pass a NULL
               client pointer.

SEE ALSO
     KsGetClientSig(), KsReadEvent()
```

1.12 krsnake.library/KsWriteKRSNAkePrefs

```
 NAME
      KsWriteKRSNAkePrefs -- write the user preferences

 SYNOPSIS
      success = KsWriteKRSNAkePrefs(prefs)
      D0                            D0

      ULONG KsWriteKRSNAkePrefs(struct KPrefs *);

 FUNCTION
```

```
     Writes the KPrefs structure passed as the file
     "ENVARC:KRSNAke/KRSNAke.prefs" and deallocates it.

     If you just want to deallocate it, call FreeVec(prefs),
     NOT KsWriteKRSNAkePrefs(prefs)! This function is only
     meant for changes you want to be permanent - ie. if you're
     the prefs program or something similar.

INPUTS
     prefs - pointer to a KPrefs structure.

RESULT
     success - TRUE if the operation succeeded, FALSE otherwise.
               Note that if this function fails, the structure is
               not deallocated. This is generally bad, but there's
               nothing the library will do about it, so you should
               FreeVec() it yourself.

SEE ALSO
     KsReadKRSNAkePrefs()
```