helix_c99 0.0.2.0

Generated by Doxygen 1.8.5

Fri Oct 2 2020 15:33:42

Contents

1	Sun	nmary o	f Helix SD	K	1
2	File	Index			5
	2.1	File Lis	st		5
3	File	Docum	entation		7
	3.1	include	e/helix_cry	oto.h File Reference	7
		3.1.1	Typedef I	Documentation	10
			3.1.1.1	blakfx_helix_event_handler_t	10
		3.1.2	Function	Documentation	10
			3.1.2.1	blakfx_helix_accountCreate	10
			3.1.2.2	blakfx_helix_accountDelete	10
			3.1.2.3	blakfx_helix_accountLogin	10
			3.1.2.4	blakfx_helix_apiShutdown	10
			3.1.2.5	blakfx_helix_apiStartup	11
			3.1.2.6	blakfx_helix_apiStartup_Advanced	12
			3.1.2.7	blakfx_helix_cPromiseManager_getStatus	12
			3.1.2.8	blakfx_helix_decryptGetOutputData	13
			3.1.2.9	blakfx_helix_decryptlsValid	13
			3.1.2.10	blakfx_helix_decryptPayloadSerializedRelease	13
			3.1.2.11	blakfx_helix_decryptStart	13
			3.1.2.12	blakfx_helix_encryptConclude	14
			3.1.2.13	blakfx_helix_encryptGetOutputData	14
			3.1.2.14	blakfx_helix_encryptOutputExists	14
			3.1.2.15	blakfx_helix_encryptPayloadGetSerialized	14
			3.1.2.16	blakfx_helix_encryptPayloadSerializedRelease	15
			3.1.2.17	blakfx_helix_encryptStart	15
			3.1.2.18	blakfx_helix_getUserData	15
			3.1.2.19	blakfx_helix_serverConnect	16
			3.1.2.20	blakfx_helix_serverDisconnect	16
			3.1.2.21	blakfx_helix_serverIsConnected	16
			3.1.2.22	blakfx_helix_simpleSearchForRecipientByEmail	16

iv CONTENTS

		3.1.2.23	blakfx_helix_simpleSearchForRecipientByName	16
		3.1.2.24	blakfx_helix_userFindByEmail	17
		3.1.2.25	blakfx_helix_userFindByEmailAsPromise	17
		3.1.2.26	blakfx_helix_userFindByName	17
		3.1.2.27	blakfx_helix_userFindByNameAsPromise	18
		3.1.2.28	blakfx_helix_userRelease	18
		3.1.2.29	blakfx_helix_userValidate	18
		3.1.2.30	blakfx_helix_waitEvent	18
		3.1.2.31	blakfx_helix_waitEventStatus	19
3.2	include	e/helix_type	es.h File Reference	19
	3.2.1	Typedef I	Documentation	21
		3.2.1.1	DECRYPT_ID	21
		3.2.1.2	ENCRYPT_ID	21
		3.2.1.3	invokeStatus_t	21
		3.2.1.4	KEY_ID	21
		3.2.1.5	logLevel_t	21
		3.2.1.6	option_t	21
		3.2.1.7	PROMISE_ID	21
		3.2.1.8	promiseStatusAndFlags_t	21
		3.2.1.9	serverResponseCode_t	21
		3.2.1.10	USER_ID	21
	3.2.2	Enumera	tion Type Documentation	21
		3.2.2.1	invokeStatus_t	21
		3.2.2.2	logLevel_t	22
		3.2.2.3	option_t	22
		3.2.2.4	promiseStatusAndFlags_t	22
		3.2.2.5	serverResponseCode_t	23
3.3	mainpa	age.md File	e Reference	23
3.4	src/hel	ix_crypto.c	File Reference	23
	3.4.1	Function	Documentation	25
		3.4.1.1	blakfx_helix_accountCreate	25
		3.4.1.2	blakfx_helix_accountDelete	25
		3.4.1.3	blakfx_helix_accountLogin	25
		3.4.1.4	blakfx_helix_apiCreateUID	26
		3.4.1.5	blakfx_helix_apiShutdown	26
		3.4.1.6	blakfx_helix_apiStartup	26
		3.4.1.7	blakfx_helix_apiStartup_Advanced	26
		3.4.1.8	blakfx_helix_cPromiseManager_getStatus	27
		3.4.1.9	blakfx_helix_decryptGetOutputData	27
		3.4.1.10	blakfx_helix_decryptStart	27

CONTENTS

		3.4.1.11	blakfx_helix_encryptConclude	28
		3.4.1.12	blakfx_helix_encryptGetOutputData	28
		3.4.1.13	blakfx_helix_encryptOutputExists	28
		3.4.1.14	blakfx_helix_encryptPayloadGetSerialized	28
		3.4.1.15	blakfx_helix_encryptPayloadSerializedRelease	29
		3.4.1.16	blakfx_helix_encryptStart	29
		3.4.1.17	blakfx_helix_getUserData	29
		3.4.1.18	blakfx_helix_serverConnect	30
		3.4.1.19	blakfx_helix_serverDisconnect	30
		3.4.1.20	blakfx_helix_serverIsConnected	30
		3.4.1.21	blakfx_helix_simpleSearchForRecipientByEmail	30
		3.4.1.22	blakfx_helix_simpleSearchForRecipientByName	30
		3.4.1.23	blakfx_helix_userFindByEmail	31
		3.4.1.24	blakfx_helix_userFindByEmailAsPromise	31
		3.4.1.25	blakfx_helix_userFindByName	31
		3.4.1.26	blakfx_helix_userFindByNameAsPromise	32
		3.4.1.27	blakfx_helix_userRelease	32
		3.4.1.28	blakfx_helix_userValidate	32
		3.4.1.29	blakfx_helix_waitEvent	32
		3.4.1.30	blakfx_helix_waitEventStatus	33
3.5	test/heli	ix_crypto_	test.c File Reference	33
	3.5.1	Function	Documentation	33
		3.5.1.1	main	33

34

Index

Chapter 1

Summary of Helix SDK

Helix is a library designed to provide cryptographic services to other software packages. This implementation of Helix is written in C language, compliant with C99 standard.

Helix offers multiple layers of protection of data, such as:

- single-use keys (similar to taking a private cab, instead of sharing a public bus or train)
- each user-defined data payload is encrypted with 5 symmetric algorithms, and wrapped with 2 layers using different asymmetric algorithms
- licensed end-user keys are continuously generated, with public parts published to Helix key-server (this does not compromise cryptographic strength those are shareable parts by design)
- · etc.

Helix library abstracts many cryptographic operations away from external developers in order to facilitate simple and easy to use API for a generalist developer, at any skill level. Helix exposes only necessary API for licensing, recipient identification, data input and output. Internally, Helix uses sophisticated processes to handle management of multitude of cryptographic keys.

Helix design allows to simplify integration and use of cryptographic module with superior strength without compromising the security of cryptographic functions (through mistakes or bad confutation), or on-boarding specialized developers and resources to complete the specialized work.

Helix is a universal solution, agnostic to:

- data transfer modes (we operate above network and protocol layers in the network stack)
- nature and type of data (text, binary, document, picture, video, etc. all formats are supported, including streaming)
- operating platform configuration (register size, endian mode, CPU type, or vendor type)

Integration Pattern

Below are examples of minimal Helix API calls, along with most-likely integration places, needed to complete logical operations of interest. Provided API calls are minimalistic non-production examples (no error code checking, etc.) to demonstrate the ease of core API use.

Initialization and Shutdown

Helix gives ability to choose when its module should be loaded/activated to the integration team. This allows integration teams to keep start-up unaffected, differentiate use of Helix by licensing or other considerations. These

actions are performed once per life-cycle of Helix module. They are frequently integrated into startup and shutdown sequences of applications.

To load Helix module, execute function:

blakfx_helix_apiStartup

To shutdown Helix module, execute function:

blakfx_helix_apiShutdown

Communication and Authentication with Helix Key-Server

Helix uses numerous number of keys to secure data at various stages of data life-cycle. Parts of many keys must be published to Helix key-server in order to facilitate discovery of communication partners (directory of known users). Helix key-server is a logical design concept that could be materialized in a number of configurations, from on-prem to managed solution, to cloud-first subscription model. There are also choices for federation in order to facilitate communication across disjoint organizations and/or groups. These actions are performed once per connection to key-server life-cycle. These actions are frequently integrated into login and logout sequences of applications.

To establish connection to Helix key-server, execute function:

blakfx_helix_serverConnect

To authenticate into Helix key-server, execute function:

blakfx_helix_accountLogin

To sever connection to Helix key-server, execute function:

blakfx_helix_serverDisconnect

Discovery of Communication Partners (Recipeints)

Through the use of key-server, Helix provides directory services where customers may discover necessary information (not necessarily identify, only public-keys) for intended recipients. This actions are performed once per recipient identification. This actions are frequently integrated into contact search sequence of applications.

To discover keys of intended recipient, execute function:

 $\verb|blakfx_helix_simpleSearchForRecipientByName|\\$

Cryptographic Operations

Helix takes on itself a lot of sophisticated processing, security-driven steps and key management. An integration team has a high-level steps it should execute, in order to secure any data.

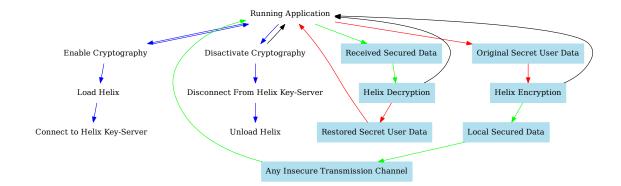
To encrypt unsecured data (before transmission), you may execute functions:

blakfx_helix_encryptStart
blakfx_helix_encryptPayloadGetSerialized

To decrypt secured data (upon receipt), execute functions:

blakfx_helix_decryptStart

 $\verb|blakfx_helix_decryptPayloadSerializedRelease|$



Chapter 2

File Index

2.1 File List

Here	is a	list	of all	files	with	hrief	descri	ntions
Here	is a	1151	ui aii	11162	VVILII	Dilei	UESCII	סווטווס

include/helix_crypto.h	7
include/helix_types.h	19
src/helix_crypto.c	23
test/helix_crypto_test.c	33

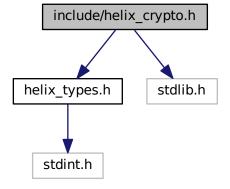
6 File Index

Chapter 3

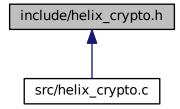
File Documentation

3.1 include/helix_crypto.h File Reference

#include "helix_types.h"
#include <stdlib.h>
Include dependency graph for helix_crypto.h:



This graph shows which files directly or indirectly include this file:



Typedefs

typedef int64_t(* blakfx_helix_event_handler_t)(PROMISE_ID promise_ID, promiseStatusAndFlags_t status)

Functions

invokeStatus t blakfx helix accountCreate (const char *userName)

Creates a new account with a given username. All previously generated keys for the account in local key storage will be deleted.

invokeStatus_t blakfx_helix_accountDelete (const char *userName)

Deletes a local account with a given username.

invokeStatus_t blakfx_helix_accountLogin (const char *userName)

Login to an existing account with a given username.

void blakfx_helix_apiShutdown (void)

Shut down/cleanup of Helix API module Helix module does a great deal of background work: key-generation, communication with key-server, in addition to asynchronous encryption and decryption processes. It is imperative to allow Helix module to step through orderly shutdown process in order to prevent unlikely but possible local-data corruption.

invokeStatus_t blakfx_helix_apiStartup (const char *serverIP, uint16_t port, int64_t flags)

Starts the Helix API module Helix module is composed of various components that will be loaded/activated on demand by the caller. This allows clients of Helix to minimize the start-time, optimize resource utilization, and segment Helix usage by various differentiating factors.

• invokeStatus_t blakfx_helix_apiStartup_Advanced (const char *serverIP, uint16_t port, const char *custom-DUID, int64_t flags, void *reserved)

Starts the Helix API module, with more options available.

• promiseStatusAndFlags_t blakfx_helix_cPromiseManager_getStatus (PROMISE_ID promise_ID)

Get the status of a specific promise.

invokeStatus_t blakfx_helix_decryptGetOutputData (DECRYPT_ID decrypt_id, uint8_t **data, size_t *length)

Get the result of a given decryption task.

invokeStatus t blakfx helix decryptlsValid (DECRYPT ID decrypt id)

Check whether a given decryption task is valid.

• invokeStatus_t blakfx_helix_decryptPayloadSerializedRelease (DECRYPT_ID decrypt_id)

Release the serialized payload of a given decryption task.

 DECRYPT_ID blakfx_helix_decryptStart (uint8_t *cipherData, size_t cipherMessageSize, const char *password, option_t anInvocationOptions)

Start decrypting some encrypted blob.

invokeStatus_t blakfx_helix_encryptConclude (ENCRYPT_ID encrypt_id)

Conclude/wrap up a given encryption task.

invokeStatus_t blakfx_helix_encryptGetOutputData (ENCRYPT_ID encrypt_id, uint8_t **serializedOut, size-t *length, option t anInvocationOptions)

Get the result of a given encryption task.

invokeStatus_t blakfx_helix_encryptOutputExists (ENCRYPT_ID encrypt_id)

Check whether encrypt output exists.

uint8 t * blakfx helix encryptPayloadGetSerialized (ENCRYPT ID encrypt id, size t *length)

Get the serialized payload of a given encryption task.

- invokeStatus_t blakfx_helix_encryptPayloadSerializedRelease (ENCRYPT_ID encrypt_id, uint8_t *serialized)

 Release the serialized payload of a given encryption task.
- ENCRYPT_ID blakfx_helix_encryptStart (USER_ID user_id, const void *data, size_t dataSize, const char *password, const char *fileName, option t anInvocationOptions)

Start encrypting some content intended for a given target user.

• void * blakfx helix getUserData (PROMISE ID promise ID, uint64 t user data id, size t *length)

Get the data for a given user.

invokeStatus t blakfx helix serverConnect (void)

Connect to previously specified (see blakfx_helix_apiStartup) Helix key-server. Helix module publishes and exchanges public cryptographic keys with its key-server, in order to facilitate cryptographically secure End-to-End communication.

invokeStatus t blakfx helix serverDisconnect (void)

Sever active connection to Helix key-server Helix operates numerous background tasks allowing clients to complete complex cryptographic operations seamlessly. To prevent unlikely but possible corruption, it is crucial for Helix users to allow Helix module to complete orderly disconnect from its key-server.

invokeStatus_t blakfx_helix_serverIsConnected (void)

Checks whether connection to previously defined key-server is alive (in valid state and responsive). This method is added for posterity, however avoid overusing it or placing it in your execution critical path - it's execution is network-bound. Under normal circumstances, it safe to assume the connection is active, and check for error conditions of Helix API calls in the critical-path. This method is appropriate for use as connectivity test after long period is inactivity. In all other situations - use of this API is superfluous and suboptimal.

- PROMISE_ID blakfx_helix_simpleSearchForRecipientByEmail (const char *lookup, int64_t waitInMillis)
 Search for an account with a given email on the current Helix Key Server.
- PROMISE_ID blakfx_helix_simpleSearchForRecipientByName (const char *lookup, int64_t waitInMillis)

Search for an account with a given username on the current Helix Key Server.

USER_ID blakfx_helix_userFindByEmail (const char *emailAddress, invokeStatus_t *result, blakfx_helix_event_handler_t crypto_notification_function)

Search for an user with a given email on the current Helix Key Server.

• PROMISE_ID blakfx_helix_userFindByEmailAsPromise (const char *emailAddress, invokeStatus_t *result, blakfx_helix_event_handler_t promise_notification_function)

Search for an account with a given email on the current Helix Key Server.

USER_ID blakfx_helix_userFindByName (const char *userName, invokeStatus_t *result, blakfx_helix_event_handler_t crypto_notification_function)

Search for an user with a given username on the current Helix Key Server.

• PROMISE_ID blakfx_helix_userFindByNameAsPromise (const char *userName, invokeStatus_t *result, blakfx_helix_event_handler_t promise_notification_function)

Search for an account with a given username on the current Helix Key Server.

invokeStatus_t blakfx_helix_userRelease (USER_ID user_id)

Release an user.

invokeStatus t blakfx helix userValidate (USER ID user id)

Ensure that an user is valid.

• invokeStatus_t blakfx_helix_waitEvent (PROMISE_ID crypto_ID, int64_t time_in_ms)

Waits for an event for a specific time.

promiseStatusAndFlags t blakfx helix waitEventStatus (PROMISE ID aPromise id)

Retrieve status of the promise (referencing a promise to complete some operation) by its unique id.

3.1.1 Typedef Documentation

3.1.1.1 typedef int64_t(* blakfx_helix_event_handler_t)(PROMISE_ID promise_ID, promiseStatusAndFlags_t status)

Declaration of callback function Helix would accept to register it as event handler to execute at completion of select events

3.1.2 Function Documentation

3.1.2.1 invokeStatus t blakfx_helix_accountCreate (const char * userName)

Creates a new account with a given username. All previously generated keys for the account in local key storage will be deleted.

Parameters

in	userName	the username for the account

Returns

result code of the creation operation

3.1.2.2 invokeStatus t blakfx_helix_accountDelete (const char * userName)

Deletes a local account with a given username.

Parameters

in	userName	the username for the account

Returns

result code of the deletion operation

3.1.2.3 invokeStatus_t blakfx_helix_accountLogin (const char * userName)

Login to an existing account with a given username.

Parameters

in	userName	the username for the account

Returns

result code of the login operation

3.1.2.4 void blakfx_helix_apiShutdown (void)

Shut down/cleanup of Helix API module Helix module does a great deal of background work: key-generation, communication with key-server, in addition to asynchronous encryption and decryption processes. It is imperative to allow Helix module to step through orderly shutdown process in order to prevent unlikely but possible local-data corruption.

3.1.2.5 invokeStatus_t blakfx_helix_apiStartup (const char * serverIP, uint16_t port, int64_t flags)

Starts the Helix API module Helix module is composed of various components that will be loaded/activated on demand by the caller. This allows clients of Helix to minimize the start-time, optimize resource utilization, and segment Helix usage by various differentiating factors.

Parameters

in	serverIP	the IP/hostname of the Helix Key Server to connect to
in	port	the port of the Helix Key Server to connect to
in	flags	additional flags

Returns

result of startup operation

Here is the call graph for this function:



3.1.2.6 invokeStatus_t blakfx_helix_apiStartup_Advanced (const char * serverIP, uint16_t port, const char * customDUID, int64_t flags, void * reserved)

Starts the Helix API module, with more options available.

Parameters

in	serverIP	the IP/hostname of the Helix Key Server to connect to
in	port	the port of the Helix Key Server to connect to
in	customDUID	a fake custom device UID
in	flags	additional flags
in	reserved	unused

Returns

result of startup operation

Here is the caller graph for this function:



3.1.2.7 promiseStatusAndFlags_t blakfx_helix_cPromiseManager_getStatus (PROMISE_ID promise_ID)

Get the status of a specific promise.

Parameters

in	promise_ID	the ID of the promise to get the status of

Returns

status of the promise

3.1.2.8 invokeStatus_t blakfx_helix_decryptGetOutputData (DECRYPT_ID decrypt_id, uint8_t ** data, size_t * length)

Get the result of a given decryption task.

Parameters

in	decrypt_id	the ID of the decryption to get the result of
out	data	buffer to place the result of the decryption
out	length	the length of the decrypted data

Returns

result of get operation

3.1.2.9 invokeStatus_t blakfx_helix_decryptlsValid (DECRYPT_ID decrypt_id)

Check whether a given decryption task is valid.

Parameters

in	encrypt_id	the ID of the decryption to verify
----	------------	------------------------------------

Returns

whether the decryption is valid or not

 $3.1.2.10 \quad invoke Status_t \ blakfx_helix_decryptPayloadSerializedRelease (\ DECRYPT_ID \ decrypt_id \)$

Release the serialized payload of a given decryption task.

Parameters

in	decrypt_id	the ID of the encryption to release the serialized payload of

Returns

whether the data was released successfully

3.1.2.11 DECRYPT_ID blakfx_helix_decryptStart (uint8_t * cipherData, size_t cipherMessageSize, const char * password, option_t anInvocationOptions)

Start decrypting some encrypted blob.

Parameters

in	cipherData	the encrypted blob
in	cipherMessage-	the size of the encrypted blob
	Size	
in	password	optional password to decrypt with
in	anInvocation-	additional options to specify
	Options	

Returns

decryption ID for the ongoing decryption

3.1.2.12 invokeStatus_t blakfx_helix_encryptConclude (ENCRYPT_ID encrypt_id)

Conclude/wrap up a given encryption task.

Parameters

in	encrypt_id	the ID of the encryption to conclude
----	------------	--------------------------------------

Returns

whether the task was concluded successfully or not

3.1.2.13 invokeStatus_t blakfx_helix_encryptGetOutputData (ENCRYPT_ID encrypt_id, uint8_t ** serializedOut, size_t * length, option_t anInvocationOptions)

Get the result of a given encryption task.

Parameters

in	encrypt_id	the ID of the encryption to get the result of
out	serializedOut	buffer to place the result of the encryption
out	length	the length of the encrypted data
in	anInvocation-	additional options to specify
	Options	

Returns

result of get operation

3.1.2.14 invokeStatus_t blakfx_helix_encryptOutputExists (ENCRYPT_ID encrypt_id)

Check whether encrypt output exists.

Parameters

in	encrypt_id	the ID of the encryption to check
----	------------	-----------------------------------

Returns

whether the encryption has any output or not

3.1.2.15 uint8_t* blakfx_helix_encryptPayloadGetSerialized (ENCRYPT_ID encrypt_id, size_t * length)

Get the serialized payload of a given encryption task.

Parameters

in	encrypt_id	the ID of the encryption to get the result of
out	length	the length of the encrypted data

Returns

the serialized payload data

3.1.2.16 invokeStatus_t blakfx_helix_encryptPayloadSerializedRelease (ENCRYPT_ID encrypt_id, uint8_t * serialized)

Release the serialized payload of a given encryption task.

Parameters

in	encrypt_id	the ID of the encryption to release the serialized payload of
in	serialized	the serialized payload data

Returns

whether the data was released successfully

3.1.2.17 ENCRYPT_ID blakfx_helix_encryptStart (USER_ID user_id, const void * data, size_t dataSize, const char * password, const char * fileName, option_t anInvocationOptions)

Start encrypting some content intended for a given target user.

Parameters

in	user_id	the ID of the target user
in	data	the content to encrypt
in	dataSize	the size of the content to encrypt
in	password	optional password to encrypt with
in	fileName	optional name/path for file with content to encrypt
in	anInvocation-	additional options
	Options	

Returns

encryption ID for the ongoing encryption

 $3.1.2.18 \quad \text{void} * \ \text{blakfx_helix_getUserData} \ (\ \ \textbf{PROMISE_ID} \ \textit{promise_ID}, \ \ \text{uint64_t} \ \textit{user_data_id}, \ \ \text{size_t} * \ \textit{length} \)$

Get the data for a given user.

Parameters

in	promiseID	the promiseID to find the data for
in	user_data_id	the user's data ID
out	length	the length of the user's data

Returns

data as a void *

3.1.2.19 invokeStatus_t blakfx_helix_serverConnect (void)

Connect to previously specified (see blakfx_helix_apiStartup) Helix key-server. Helix module publishes and exchanges public cryptographic keys with its key-server, in order to facilitate cryptographically secure End-to-End communication.

Returns

Status of attempt to establish connection with key-server

3.1.2.20 invokeStatus_t blakfx_helix_serverDisconnect (void)

Sever active connection to Helix key-server Helix operates numerous background tasks allowing clients to complete complex cryptographic operations seamlessly. To prevent unlikely but possible corruption, it is crucial for Helix users to allow Helix module to complete orderly disconnect from its key-server.

Returns

Status of severing the connection to the key-server (normal or with errors)

3.1.2.21 invokeStatus_t blakfx_helix_serverlsConnected (void)

Checks whether connection to previously defined key-server is alive (in valid state and responsive). This method is added for posterity, however avoid overusing it or placing it in your execution critical path - it's execution is network-bound. Under normal circumstances, it safe to assume the connection is active, and check for error conditions of Helix API calls in the critical-path. This method is appropriate for use as connectivity test after long period is inactivity. In all other situations - use of this API is superfluous and suboptimal.

Returns

Status of connection to previously defined key-server

3.1.2.22 PROMISE_ID blakfx_helix_simpleSearchForRecipientByEmail (const char * lookup, int64_t waitInMillis)

Search for an account with a given email on the current Helix Key Server.

Parameters

	in	lookup	the email address of the target
İ	in	waitInMillis	the time in ms to attempt the search in

Returns

result of the search operation

3.1.2.23 PROMISE ID blakfx_helix_simpleSearchForRecipientByName (const char * lookup, int64_t waitInMillis)

Search for an account with a given username on the current Helix Key Server.

Param	eters
-------	-------

in	lookup	the username for the target account
in	waitInMillis	the time in ms to attempt the search in

Returns

result of the search operation

3.1.2.24 USER_ID blakfx_helix_userFindByEmail (const char * emailAddress, invokeStatus_t * result, blakfx_helix_event_handler_t crypto_notification_function)

Search for an user with a given email on the current Helix Key Server.

Parameters

in	emailAddress	the email address of the target
out	result	the result of the search operation
in	blakfx_helix	register custom function to be executed as registered event handler with
	event_handler_t	promise status notification

Returns

user ID of the found/not found user

3.1.2.25 PROMISE_ID blakfx_helix_userFindByEmailAsPromise (const char * emailAddress, invokeStatus_t * result, blakfx_helix_event_handler_t promise_notification_function)

Search for an account with a given email on the current Helix Key Server.

!

Parameters

in	emailAddress	the email address of the target
out	result	the result of the search operation
in	blakfx_helix	register custom function to be executed as registered event handler with
	event_handler_t	promise status notification

Returns

promise of found/not found user

3.1.2.26 USER_ID blakfx_helix_userFindByName (const char * userName, invokeStatus_t * result, blakfx_helix_event_handler_t crypto_notification_function)

Search for an user with a given username on the current Helix Key Server.

Parameters

in	userName	the name of the target
----	----------	------------------------

out	result	the result of the search operation
in	blakfx_helix	register custom function to be executed as registered event handler with
	event_handler_t	promise status notification

Returns

user ID of the found/not found user

3.1.2.27 PROMISE_ID blakfx_helix_userFindByNameAsPromise (const char * userName, invokeStatus_t * result, blakfx_helix_event_handler_t promise_notification_function)

Search for an account with a given username on the current Helix Key Server.

!

Parameters

in	userName	the username for the target account
out	result	the result of the search operation
in	blakfx_helix	register custom function to be executed as registered event handler with
	event_handler_t	promise status notification

Returns

promise of found/not found user

3.1.2.28 invokeStatus_t blakfx_helix_userRelease (USER_ID user_id)

Release an user.

Parameters

in	user_id	the ID of the user to release

Returns

result of release

3.1.2.29 invokeStatus_t blakfx_helix_userValidate (USER_ID user_id)

Ensure that an user is valid.

Parameters

in	user_id	the ID of the user to validate

Returns

result of validation

3.1.2.30 invokeStatus_t blakfx_helix_waitEvent (PROMISE_ID crypto_ID, int64_t time_in_ms)

Waits for an event for a specific time.

Parameters

in	crypto_ID	the ID of the event to wait for
in	time_in_ms	the time in ms to wait for

Returns

result of the wait operation

3.1.2.31 promiseStatusAndFlags_t blakfx_helix_waitEventStatus (PROMISE_ID aPromise_id)

Retrieve status of the promise (referencing a promise to complete some operation) by its unique id.

Parameters

in	aPromise_id	the unique id of the promise whose status to retrieve
----	-------------	---

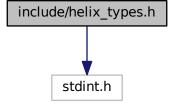
Returns

status code indicating the state of the promised work

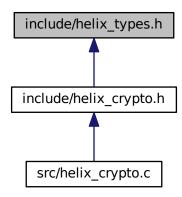
3.2 include/helix_types.h File Reference

#include <stdint.h>

Include dependency graph for helix_types.h:



This graph shows which files directly or indirectly include this file:



Typedefs

- typedef PROMISE_ID DECRYPT_ID
- typedef PROMISE ID ENCRYPT ID
- typedef enum __invokeStatus_t invokeStatus_t
- typedef PROMISE_ID KEY_ID
- typedef enum __logLevel_t logLevel_t
- typedef enum __option_t option_t
- typedef uint64_t PROMISE_ID
- · typedef enum
 - __promiseStatusAndFlags_t promiseStatusAndFlags_t
- typedef enum serverResponseCode t serverResponseCode t
- typedef PROMISE_ID USER_ID

Enumerations

```
enum invokeStatus t {
 INVOKE_STATUS_NOT_INITIALIZED = -255, INVOKE_IN_INVALID_STATE = -254, INVOKE_INVALID_I-
 NSIDE_CALLBACK = -253, INVOKE_STATUS_BAD_PROMISE_ID = -252,
 INVOKE_STATUS_TIMEOUT = -2, INVOKE_STATUS_FALSE = -1, INVOKE_STATUS_TRUE = 0 }
enum logLevel t {
 NO LOG = 0x00, ERROR LEVEL = 0x01, INFO LEVEL = 0x02, WARN LEVEL = 0x04,
 DEBUG LEVEL = 0x06, ALL_LEVEL = 0xffff }

    enum __option_t { USER_OWNS_MEMORY = 0x0000, HELIX_OWNS_MEMORY = 0x0001 }

    enum promiseStatusAndFlags t {

 PROMISE_INVALID = -254, PROMISE_INFINITE = -1, PROMISE_COMPLETE = 0x0001, PROMISE_DES-
 TROY = 0x0002,
 PROMISE_DATA_AVAILABLE = 0x0004, PROMISE_EVENT = 0x0008, PROMISE_USER_EVENT =
 0x0010, PROMISE_RESULT_ERROR = 0x0020,
 PROMISE_MEMORY_ALLOCATED = 0x0040, PROMISE_MEMORY_RELEASING = 0x0080, PROMISE_-
 MEMORY POST RELEASED ID = 0x0100, PROMISE ALLOW RECURSIVE EVENTS = 0x1000,
 PROMISE NO STATUS = 0x2000, PROMISE WAIT STATUS = 0x4000, PROMISE ERROR UNDEFIN-
 ED = 0x8000 }
enum __serverResponseCode_t { SERVER_SUCCESS = 0, SERVER_FAIL = -1 }
```

- 3.2.1 Typedef Documentation
- 3.2.1.1 typedef PROMISE_ID DECRYPT_ID
- 3.2.1.2 typedef PROMISE ID ENCRYPT_ID
- 3.2.1.3 typedef enum invokeStatus t invokeStatus t

Collection of codes indicating possible conditions as result of function invocation.

- 3.2.1.4 typedef PROMISE ID KEY ID
- 3.2.1.5 typedef enum __logLevel_t logLevel_t

Collection of log-level modes Helix module has.

3.2.1.6 typedef enum __option_t option_t

Collection of codes indicating memory ownership model Helix caller will be using.

- 3.2.1.7 typedef uint64_t PROMISE ID
- 3.2.1.8 typedef enum __promiseStatusAndFlags_t promiseStatusAndFlags_t

Collection of codes indicating state of a promise (result of computation to be completed in the future).

3.2.1.9 typedef enum serverResponseCode t serverResponseCode t

Collection of code values key-server could respond with after various requests.

- 3.2.1.10 typedef PROMISE_ID USER_ID
- 3.2.2 Enumeration Type Documentation
- 3.2.2.1 enum invokeStatus t

Collection of codes indicating possible conditions as result of function invocation.

Enumerator

INVOKE_STATUS_NOT_INITIALIZED Status code indicating invoked module is not initialized.

INVOKE_IN_INVALID_STATE Status code indicating invoked module is not ready – not initialized or shutting down.

INVOKE_INVALID_INSIDE_CALLBACK Status code indicating provided callback is invalid.

INVOKE_STATUS_BAD_PROMISE_ID Status code indicating provided promise id is not valid.

INVOKE_STATUS_TIMEOUT Status code indicating invocation has timeout.

INVOKE_STATUS_FALSE Status code indicating invocation has failed.

INVOKE_STATUS_TRUE Status code indicating invocation completed successfully.

```
3.2.2.2 enum __logLevel_t
```

Collection of log-level modes Helix module has.

Enumerator

NO_LOG Disable all logging inside Helix module.

ERROR_LEVEL Enable logging of serious error conditions.

INFO_LEVEL Enable logging of information-level messages.

WARN_LEVEL Enable logging of warning messages.

DEBUG_LEVEL Enable logging of debug-level messages.

ALL_LEVEL Enable logging of all possible messages.

```
3.2.2.3 enum option t
```

Collection of codes indicating memory ownership model Helix caller will be using.

Enumerator

USER_OWNS_MEMORY Helix should not take copy of the supplied (decryption) buffer and use exclusively user supplied one. User takes on responsibility for ensuring the memory remains valid and accessible for the duration of all Helix operations to complete its work involving that memory location. In case of encryption, caller is responsible to deallocate returned memory buffer with encrypted data.

HELIX_OWNS_MEMORY Helix should take a copy of the supplied (decryption) buffer and will manage its life-cycle internally. Caller is free to destroy original (decryption) inputs at any time. In case of encryption, Helix will own memory used to return (encrypted) outputs - user is responsible to signal to Helix when the contents are eligible for destruction by calling ConcludeEncryption with promise-id corresponding to original encryption request.

3.2.2.4 enum __promiseStatusAndFlags_t

Collection of codes indicating state of a promise (result of computation to be completed in the future).

Enumerator

PROMISE_INVALID Promise state is invalid (either corrupted, or one of internal operations exited with code invokeStatus_t::INVOKE_IN_INVALID_STATE.

PROMISE_INFINITE Promised work has no timeout for its completion - promise will remain active until task signals completion (ex: daemon services)

PROMISE_COMPLETE Indicated promised computation has been completed.

PROMISE_DESTROY Request destruction of the specified promise (release of resources once promise is complete)

PROMISE_DATA_AVAILABLE There is data available for extraction as result of completion of promised work.

PROMISE EVENT N/A.

PROMISE_USER_EVENT N/A.

PROMISE_RESULT_ERROR Promised work completed with an error.

PROMISE_MEMORY_ALLOCATED N/A.

PROMISE_MEMORY_RELEASING N/A.

PROMISE_MEMORY_POST_RELEASED_ID N/A.

PROMISE ALLOW RECURSIVE EVENTS N/A.

PROMISE_NO_STATUS Status of promised work is unknown (most likely work is in progress)

PROMISE_WAIT_STATUS Promised work is in wait status (most likely task is waiting for completion is another sub-task)

PROMISE_ERROR_UNDEFINED Unknown error condition has been detected.

3.2.2.5 enum __serverResponseCode_t

Collection of code values key-server could respond with after various requests.

Enumerator

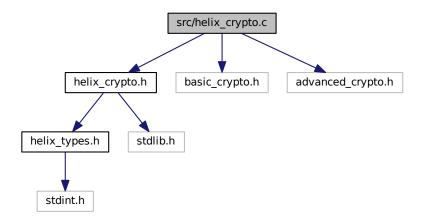
SERVER_SUCCESS Server successfully completed requested action by the client.

SERVER_FAIL Server failed to complete requested action by the client.

3.3 mainpage.md File Reference

3.4 src/helix_crypto.c File Reference

```
#include "helix_crypto.h"
#include "basic_crypto.h"
#include "advanced_crypto.h"
Include dependency graph for helix crypto.c:
```



Functions

invokeStatus_t blakfx_helix_accountCreate (const char *userName)

Creates a new account with a given username. All previously generated keys for the account in local key storage will be deleted.

• invokeStatus t blakfx helix accountDelete (const char *userName)

Deletes a local account with a given username.

invokeStatus_t blakfx_helix_accountLogin (const char *userName)

Login to an existing account with a given username.

- invokeStatus_t blakfx_helix_apiCreateUID (uint8_t *inputBuffer16Bytes, size_t inputBufferSize)
- void blakfx_helix_apiShutdown (void)

Shut down/cleanup of Helix API module Helix module does a great deal of background work: key-generation, communication with key-server, in addition to asynchronous encryption and decryption processes. It is imperative to allow Helix module to step through orderly shutdown process in order to prevent unlikely but possible local-data corruption.

invokeStatus_t blakfx_helix_apiStartup (const char *serverIP, uint16_t port, int64_t flags)

Starts the Helix API module Helix module is composed of various components that will be loaded/activated on demand by the caller. This allows clients of Helix to minimize the start-time, optimize resource utilization, and segment Helix usage by various differentiating factors.

 invokeStatus_t blakfx_helix_apiStartup_Advanced (const char *serverIP, uint16_t port, const char *custom-DUID, int64_t flags, void *reserved)

Starts the Helix API module, with more options available.

promiseStatusAndFlags_t blakfx_helix_cPromiseManager_getStatus (PROMISE_ID promise_ID)

Get the status of a specific promise.

invokeStatus_t blakfx_helix_decryptGetOutputData (DECRYPT_ID decrypt_id, uint8_t **data, size_t *length)

Get the result of a given decryption task.

 DECRYPT_ID blakfx_helix_decryptStart (uint8_t *cipherData, size_t cipherMessageSize, const char *password, option_t anInvocationOptions)

Start decrypting some encrypted blob.

• invokeStatus t blakfx helix encryptConclude (ENCRYPT ID encrypt id)

Conclude/wrap up a given encryption task.

 invokeStatus_t blakfx_helix_encryptGetOutputData (ENCRYPT_ID encrypt_id, uint8_t **serializedOut, size-_t *length, option_t anInvocationOptions)

Get the result of a given encryption task.

invokeStatus t blakfx helix encryptOutputExists (ENCRYPT ID encrypt id)

Check whether encrypt output exists.

uint8_t * blakfx_helix_encryptPayloadGetSerialized (ENCRYPT_ID encrypt_id, size_t *length)

Get the serialized payload of a given encryption task.

- invokeStatus_t blakfx_helix_encryptPayloadSerializedRelease (ENCRYPT_ID encrypt_id, uint8_t *serialized)

 Release the serialized payload of a given encryption task.
- ENCRYPT_ID blakfx_helix_encryptStart (USER_ID user_id, const void *data, size_t dataSize, const char *password, const char *fileName, option t anInvocationOptions)

Start encrypting some content intended for a given target user.

• void * blakfx_helix_getUserData (PROMISE_ID promise_ID, uint64_t user_data_id, size_t *length)

Get the data for a given user.

invokeStatus_t blakfx_helix_serverConnect (void)

Connect to previously specified (see blakfx_helix_apiStartup) Helix key-server. Helix module publishes and exchanges public cryptographic keys with its key-server, in order to facilitate cryptographically secure End-to-End communication.

invokeStatus_t blakfx_helix_serverDisconnect (void)

Sever active connection to Helix key-server Helix operates numerous background tasks allowing clients to complete complex cryptographic operations seamlessly. To prevent unlikely but possible corruption, it is crucial for Helix users to allow Helix module to complete orderly disconnect from its key-server.

invokeStatus_t blakfx_helix_serverIsConnected (void)

Checks whether connection to previously defined key-server is alive (in valid state and responsive). This method is added for posterity, however avoid overusing it or placing it in your execution critical path - it's execution is network-bound. Under normal circumstances, it safe to assume the connection is active, and check for error conditions of Helix API calls in the critical-path. This method is appropriate for use as connectivity test after long period is inactivity. In all other situations - use of this API is superfluous and suboptimal.

- PROMISE_ID blakfx_helix_simpleSearchForRecipientByEmail (const char *lookup, int64_t waitInMillis)
 - Search for an account with a given email on the current Helix Key Server.
- PROMISE_ID blakfx_helix_simpleSearchForRecipientByName (const char *lookup, int64_t waitInMillis)

Search for an account with a given username on the current Helix Key Server.

USER_ID blakfx_helix_userFindByEmail (const char *emailAddress, invokeStatus_t *result, blakfx_helix_event_handler_t crypto_notification_function)

Search for an user with a given email on the current Helix Key Server.

• PROMISE_ID blakfx_helix_userFindByEmailAsPromise (const char *emailAddress, invokeStatus_t *result, blakfx_helix_event_handler_t promise_notification_function)

Search for an account with a given email on the current Helix Key Server.

USER_ID blakfx_helix_userFindByName (const char *userName, invokeStatus_t *result, blakfx_helix_event_handler_t crypto_notification_function)

Search for an user with a given username on the current Helix Key Server.

PROMISE_ID blakfx_helix_userFindByNameAsPromise (const char *userName, invokeStatus_t *result, blakfx_helix_event_handler_t promise_notification_function)

Search for an account with a given username on the current Helix Key Server.

• invokeStatus t blakfx helix userRelease (USER ID user id)

Release an user.

• invokeStatus_t blakfx_helix_userValidate (USER_ID user_id)

Ensure that an user is valid.

• invokeStatus_t blakfx_helix_waitEvent (PROMISE_ID crypto_ID, int64_t time_in_ms)

Waits for an event for a specific time.

promiseStatusAndFlags_t blakfx_helix_waitEventStatus (PROMISE_ID crypto_ID)

Retrieve status of the promise (referencing a promise to complete some operation) by its unique id.

3.4.1 Function Documentation

3.4.1.1 invokeStatus_t blakfx_helix_accountCreate (const char * userName)

Creates a new account with a given username. All previously generated keys for the account in local key storage will be deleted.

Parameters

in	userName	the username for the account
----	----------	------------------------------

Returns

result code of the creation operation

3.4.1.2 invokeStatus t blakfx_helix_accountDelete (const char * userName)

Deletes a local account with a given username.

Parameters

in	userName	the username for the account

Returns

result code of the deletion operation

3.4.1.3 invokeStatus_t blakfx_helix_accountLogin (const char * userName)

Login to an existing account with a given username.

Parameters

in	userName	the username for the account

Returns

result code of the login operation

3.4.1.4 invokeStatus_t blakfx_helix_apiCreateUID (uint8_t * inputBuffer16Bytes, size_t inputBufferSize)

3.4.1.5 void blakfx_helix_apiShutdown (void)

Shut down/cleanup of Helix API module Helix module does a great deal of background work: key-generation, communication with key-server, in addition to asynchronous encryption and decryption processes. It is imperative to allow Helix module to step through orderly shutdown process in order to prevent unlikely but possible local-data corruption.

3.4.1.6 invokeStatus_t blakfx_helix_apiStartup (const char * serverIP, uint16_t port, int64_t flags)

Starts the Helix API module Helix module is composed of various components that will be loaded/activated on demand by the caller. This allows clients of Helix to minimize the start-time, optimize resource utilization, and segment Helix usage by various differentiating factors.

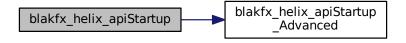
Parameters

in	serverIP	the IP/hostname of the Helix Key Server to connect to
in	port	the port of the Helix Key Server to connect to
in	flags	additional flags

Returns

result of startup operation

Here is the call graph for this function:



3.4.1.7 invokeStatus_t blakfx_helix_apiStartup_Advanced (const char * serverIP, uint16_t port, const char * customDUID, int64_t flags, void * reserved)

Starts the Helix API module, with more options available.

Parameters

in	serverIP	the IP/hostname of the Helix Key Server to connect to
in	port	the port of the Helix Key Server to connect to
in	customDUID	a fake custom device UID
in	flags	additional flags
in	reserved	unused

Returns

result of startup operation

Here is the caller graph for this function:



3.4.1.8 promiseStatusAndFlags_t blakfx_helix_cPromiseManager_getStatus (PROMISE_ID promise_ID)

Get the status of a specific promise.

Parameters

in	promise_ID	the ID of the promise to get the status of

Returns

status of the promise

3.4.1.9 invokeStatus_t blakfx_helix_decryptGetOutputData (DECRYPT_ID decrypt_id, uint8_t ** data, size_t * length)

Get the result of a given decryption task.

Parameters

in	decrypt_id	the ID of the decryption to get the result of
out	data	buffer to place the result of the decryption
out	length	the length of the decrypted data

Returns

result of get operation

3.4.1.10 DECRYPT_ID blakfx_helix_decryptStart (uint8_t * cipherData, size_t cipherMessageSize, const char * password, option_t anInvocationOptions)

Start decrypting some encrypted blob.

Parameters

in	cipherData	the encrypted blob
in	cipherMessage-	the size of the encrypted blob
	Size	

in	password	optional password to decrypt with
in	anInvocation-	additional options to specify
	Options	

Returns

decryption ID for the ongoing decryption

3.4.1.11 invokeStatus_t blakfx_helix_encryptConclude (ENCRYPT_ID encrypt_id)

Conclude/wrap up a given encryption task.

Parameters

in	encrypt_id	the ID of the encryption to conclude

Returns

whether the task was concluded successfully or not

3.4.1.12 invokeStatus_t blakfx_helix_encryptGetOutputData (ENCRYPT_ID encrypt_id, uint8_t ** serializedOut, size_t * length, option_t anInvocationOptions)

Get the result of a given encryption task.

Parameters

in	encrypt_id	the ID of the encryption to get the result of
out	serializedOut	buffer to place the result of the encryption
out	length	the length of the encrypted data
in	anInvocation-	additional options to specify
	Options	

Returns

result of get operation

3.4.1.13 invokeStatus_t blakfx_helix_encryptOutputExists (ENCRYPT_ID encrypt_id)

Check whether encrypt output exists.

Parameters

in	encrypt_id	the ID of the encryption to check

Returns

whether the encryption has any output or not

3.4.1.14 uint8_t* blakfx_helix_encryptPayloadGetSerialized (ENCRYPT_ID encrypt_id, size_t * length)

Get the serialized payload of a given encryption task.

Parameters

in	encrypt_id	the ID of the encryption to get the result of
out	length	the length of the encrypted data

Returns

the serialized payload data

3.4.1.15 invokeStatus_t blakfx_helix_encryptPayloadSerializedRelease (ENCRYPT_ID encrypt_id, uint8_t * serialized)

Release the serialized payload of a given encryption task.

Parameters

in	encrypt_id	the ID of the encryption to release the serialized payload of
in	serialized	the serialized payload data

Returns

whether the data was released successfully

3.4.1.16 ENCRYPT_ID blakfx_helix_encryptStart (USER_ID user_id, const void * data, size_t dataSize, const char * password, const char * fileName, option_t anInvocationOptions)

Start encrypting some content intended for a given target user.

Parameters

in	user_id	the ID of the target user
in	data	the content to encrypt
in	dataSize	the size of the content to encrypt
in	password	optional password to encrypt with
in	fileName	optional name/path for file with content to encrypt
in	anInvocation-	additional options
	Options	

Returns

encryption ID for the ongoing encryption

3.4.1.17 void* blakfx_helix_getUserData (PROMISE_ID promise_ID, uint64_t user_data_id, size_t * length)

Get the data for a given user.

Parameters

i	n	promiseID	the promiseID to find the data for
i	n	user_data_id	the user's data ID
Ol	ıt	length	the length of the user's data

Returns

data as a void *

3.4.1.18 invokeStatus_t blakfx_helix_serverConnect (void)

Connect to previously specified (see blakfx_helix_apiStartup) Helix key-server. Helix module publishes and exchanges public cryptographic keys with its key-server, in order to facilitate cryptographically secure End-to-End communication.

Returns

Status of attempt to establish connection with key-server

3.4.1.19 invokeStatus_t blakfx_helix_serverDisconnect (void)

Sever active connection to Helix key-server Helix operates numerous background tasks allowing clients to complete complex cryptographic operations seamlessly. To prevent unlikely but possible corruption, it is crucial for Helix users to allow Helix module to complete orderly disconnect from its key-server.

Returns

Status of severing the connection to the key-server (normal or with errors)

3.4.1.20 invokeStatus_t blakfx_helix_serverlsConnected (void)

Checks whether connection to previously defined key-server is alive (in valid state and responsive). This method is added for posterity, however avoid overusing it or placing it in your execution critical path - it's execution is network-bound. Under normal circumstances, it safe to assume the connection is active, and check for error conditions of Helix API calls in the critical-path. This method is appropriate for use as connectivity test after long period is inactivity. In all other situations - use of this API is superfluous and suboptimal.

Returns

Status of connection to previously defined key-server

3.4.1.21 PROMISE_ID blakfx_helix_simpleSearchForRecipientByEmail (const char * lookup, int64_t waitInMillis)

Search for an account with a given email on the current Helix Key Server.

Parameters

in	lookup	the email address of the target
in	waitInMillis	the time in ms to attempt the search in

Returns

result of the search operation

3.4.1.22 PROMISE ID blakfx_helix_simpleSearchForRecipientByName (const char * lookup, int64_t waitInMillis)

Search for an account with a given username on the current Helix Key Server.

Parameters

in	lookup	the username for the target account
in	waitInMillis	the time in ms to attempt the search in

Returns

result of the search operation

3.4.1.23 USER_ID blakfx_helix_userFindByEmail (const char * emailAddress, invokeStatus_t * result, blakfx_helix_event_handler_t crypto_notification_function)

Search for an user with a given email on the current Helix Key Server.

Parameters

in	emailAddress	the email address of the target
out	result	the result of the search operation
in	blakfx_helix	register custom function to be executed as registered event handler with
	event_handler_t	promise status notification

Returns

user ID of the found/not found user

3.4.1.24 PROMISE_ID blakfx_helix_userFindByEmailAsPromise (const char * emailAddress, invokeStatus_t * result, blakfx_helix_event_handler_t promise_notification_function)

Search for an account with a given email on the current Helix Key Server.

!

Parameters

in	emailAddress	the email address of the target
out	result	the result of the search operation
in	blakfx_helix	register custom function to be executed as registered event handler with
	event_handler_t	promise status notification

Returns

promise of found/not found user

3.4.1.25 USER_ID blakfx_helix_userFindByName (const char * userName, invokeStatus_t * result, blakfx_helix_event_handler_t crypto_notification_function)

Search for an user with a given username on the current Helix Key Server.

Parameters

in	userName	the name of the target
		<u> </u>

	out	result	the result of the search operation
Ī	in	blakfx_helix	register custom function to be executed as registered event handler with
		event_handler_t	promise status notification

Returns

user ID of the found/not found user

3.4.1.26 PROMISE_ID blakfx_helix_userFindByNameAsPromise (const char * userName, invokeStatus_t * result, blakfx_helix_event_handler_t promise_notification_function)

Search for an account with a given username on the current Helix Key Server.

!

Parameters

in	userName	the username for the target account
out	result	the result of the search operation
in	blakfx_helix	register custom function to be executed as registered event handler with
	event_handler_t	promise status notification

Returns

promise of found/not found user

3.4.1.27 invokeStatus_t blakfx_helix_userRelease (USER_ID user_id)

Release an user.

Parameters

in	user_id	the ID of the user to release

Returns

result of release

3.4.1.28 invokeStatus_t blakfx_helix_userValidate (USER_ID user_id)

Ensure that an user is valid.

Parameters

in	user_id	the ID of the user to validate

Returns

result of validation

3.4.1.29 invokeStatus_t blakfx_helix_waitEvent (PROMISE_ID crypto_ID, int64_t time_in_ms)

Waits for an event for a specific time.

Parameters

in	crypto_ID	the ID of the event to wait for
in	time_in_ms	the time in ms to wait for

Returns

result of the wait operation

3.4.1.30 promiseStatusAndFlags_t blakfx_helix_waitEventStatus (PROMISE_ID aPromise_id)

Retrieve status of the promise (referencing a promise to complete some operation) by its unique id.

Parameters

in	aPromise_id	the unique id of the promise whose status to retrieve
----	-------------	---

Returns

status code indicating the state of the promised work

3.5 test/helix_crypto_test.c File Reference

Functions

- int main ()
- 3.5.1 Function Documentation
- 3.5.1.1 int main ()

Index

invokeStatus_t	blakfx helix encryptGetOutputData
helix_types.h, 21	helix_crypto.c, 28
logLevel t	helix_crypto.h, 14
helix_types.h, 21	blakfx_helix_encryptOutputExists
option_t	helix_crypto.c, 28
helix types.h, 22	helix crypto.h, 14
promiseStatusAndFlags_t	blakfx_helix_encryptPayloadGetSerialized
helix_types.h, 22	helix_crypto.c, 28
serverResponseCode_t	helix crypto.h, 14
helix_types.h, 22	blakfx_helix_encryptPayloadSerializedRelease
_71	helix_crypto.c, 29
ALL_LEVEL	helix_crypto.h, 15
helix_types.h, 22	blakfx_helix_encryptStart
	helix_crypto.c, 29
blakfx_helix_accountCreate	helix_crypto.h, 15
helix_crypto.c, 25	blakfx_helix_event_handler_t
helix_crypto.h, 10	helix_crypto.h, 10
blakfx_helix_accountDelete	blakfx_helix_getUserData
helix_crypto.c, 25	— —
helix_crypto.h, 10	helix_crypto.c, 29
blakfx_helix_accountLogin	helix_crypto.h, 15
helix_crypto.c, 25	blakfx_helix_serverConnect
helix_crypto.h, 10	helix_crypto.c, 29
blakfx_helix_apiCreateUID	helix_crypto.h, 15
helix_crypto.c, 25	blakfx_helix_serverDisconnect
blakfx_helix_apiShutdown	helix_crypto.c, 30
helix_crypto.c, 26	helix_crypto.h, 16
helix_crypto.h, 10	blakfx_helix_serverIsConnected
blakfx_helix_apiStartup	helix_crypto.c, 30
helix_crypto.c, 26	helix_crypto.h, 16
helix_crypto.h, 10	blakfx_helix_simpleSearchForRecipientByEmail
blakfx_helix_apiStartup_Advanced	helix_crypto.c, 30
helix_crypto.c, 26	helix_crypto.h, 16
helix_crypto.h, 12	blakfx_helix_simpleSearchForRecipientByName
blakfx_helix_cPromiseManager_getStatus	helix_crypto.c, 30
helix_crypto.c, 27	helix_crypto.h, 16
helix_crypto.h, 12	blakfx_helix_userFindByEmail
blakfx_helix_decryptGetOutputData	helix_crypto.c, 31
helix_crypto.c, 27	helix_crypto.h, 17
helix_crypto.h, 13	blakfx_helix_userFindByEmailAsPromise
blakfx_helix_decryptIsValid	helix_crypto.c, 31
helix_crypto.h, 13	helix_crypto.h, 17
blakfx_helix_decryptPayloadSerializedRelease	blakfx_helix_userFindByName
helix_crypto.h, 13	helix_crypto.c, 31
blakfx_helix_decryptStart	helix_crypto.h, 17
helix_crypto.c, 27	blakfx_helix_userFindByNameAsPromise
helix_crypto.h, 13	helix_crypto.c, 32
blakfx_helix_encryptConclude	helix_crypto.h, 18
helix_crypto.c, 28	blakfx_helix_userRelease
helix_crypto.h, 14	helix_crypto.c, 32

INDEX 35

helix_crypto.h, 18	blakfx_helix_accountLogin, 25
blakfx_helix_userValidate	blakfx_helix_apiCreateUID, 25
helix_crypto.c, 32	blakfx_helix_apiShutdown, 26
helix_crypto.h, 18	blakfx_helix_apiStartup, 26
blakfx_helix_waitEvent	blakfx_helix_apiStartup_Advanced, 26
helix_crypto.c, 32	blakfx_helix_cPromiseManager_getStatus, 27
helix_crypto.h, 18	blakfx_helix_decryptGetOutputData, 27
blakfx_helix_waitEventStatus	blakfx_helix_decryptStart, 27
helix_crypto.c, 33	blakfx_helix_encryptConclude, 28
helix_crypto.h, 19	blakfx_helix_encryptGetOutputData, 28
	blakfx_helix_encryptOutputExists, 28
DEBUG_LEVEL	blakfx_helix_encryptPayloadGetSerialized, 28
helix_types.h, 22	blakfx_helix_encryptPayloadSerializedRelease, 29
DECRYPT_ID	blakfx_helix_encryptStart, 29
helix_types.h, 21	blakfx_helix_getUserData, 29
EDDOD 1 EVE	blakfx_helix_serverConnect, 29
ERROR_LEVEL	blakfx_helix_serverDisconnect, 30
helix_types.h, 22	blakfx_helix_serverIsConnected, 30
ENCRYPT_ID	blakfx helix simpleSearchForRecipientByEmail,
helix_types.h, 21	30
HELIX OWNS MEMORY	blakfx_helix_simpleSearchForRecipientByName,
helix_types.h, 22	30
helix_types.h	blakfx_helix_userFindByEmail, 31
ALL LEVEL, 22	blakfx_helix_userFindByEmailAsPromise, 31
DEBUG LEVEL, 22	blakfx_helix_userFindByName, 31
ERROR LEVEL, 22	blakfx_helix_userFindByNameAsPromise, 32
HELIX_OWNS_MEMORY, 22	blakfx_helix_userRelease, 32
INFO LEVEL, 22	blakfx_helix_userValidate, 32
INVOKE_IN_INVALID_STATE, 21	blakfx_helix_waitEvent, 32
INVOKE INVALID INSIDE CALLBACK, 21	blakfx_helix_waitEventStatus, 33
INVOKE_STATUS_BAD_PROMISE_ID, 21	helix_crypto.h
INVOKE_STATUS_FALSE, 21	blakfx_helix_accountCreate, 10
INVOKE STATUS NOT INITIALIZED, 21	blakfx_helix_accountDelete, 10
INVOKE_STATUS_TIMEOUT, 21	blakfx_helix_accountLogin, 10
INVOKE_STATUS_TRUE, 21	blakfx_helix_apiShutdown, 10
NO_LOG, 22	blakfx_helix_apiStartup, 10
PROMISE_ALLOW_RECURSIVE_EVENTS, 22	blakfx_helix_apiStartup_Advanced, 12
PROMISE_COMPLETE, 22	blakfx_helix_cPromiseManager_getStatus, 12
PROMISE_DATA_AVAILABLE, 22	blakfx_helix_decryptGetOutputData, 13
PROMISE DESTROY, 22	blakfx_helix_decryptIsValid, 13
PROMISE_ERROR_UNDEFINED, 22	blakfx_helix_decryptPayloadSerializedRelease, 13
PROMISE EVENT, 22	blakfx_helix_decryptStart, 13
PROMISE INFINITE, 22	blakfx_helix_encryptConclude, 14
PROMISE_INVALID, 22	blakfx_helix_encryptGetOutputData, 14
PROMISE MEMORY ALLOCATED, 22	blakfx_helix_encryptOutputExists, 14
PROMISE MEMORY POST RELEASED ID, 22	blakfx_helix_encryptPayloadGetSerialized, 14
PROMISE MEMORY RELEASING, 22	blakfx_helix_encryptPayloadSerializedRelease, 15
PROMISE_NO_STATUS, 22	blakfx_helix_encryptStart, 15
PROMISE_RESULT_ERROR, 22	blakfx_helix_event_handler_t, 10
PROMISE_USER_EVENT, 22	blakfx_helix_getUserData, 15
PROMISE_WAIT_STATUS, 22	blakfx_helix_serverConnect, 15
	blakfx_helix_serverDisconnect, 16
SERVER FAIL, 23	
SERVER_FAIL, 23 SERVER_SUCCESS, 23	blakfx_helix_serverIsConnected, 16
SERVER_SUCCESS, 23	blakfx_helix_serverIsConnected, 16
SERVER_SUCCESS, 23 USER_OWNS_MEMORY, 22	blakfx_helix_serverIsConnected, 16 blakfx_helix_simpleSearchForRecipientByEmail,
SERVER_SUCCESS, 23 USER_OWNS_MEMORY, 22 WARN_LEVEL, 22	blakfx_helix_serverIsConnected, 16 blakfx_helix_simpleSearchForRecipientByEmail, 16
SERVER_SUCCESS, 23 USER_OWNS_MEMORY, 22 WARN_LEVEL, 22 helix_crypto.c	blakfx_helix_serverIsConnected, 16 blakfx_helix_simpleSearchForRecipientByEmail, 16 blakfx_helix_simpleSearchForRecipientByName,

36 INDEX

blakfx_helix_userFindByEmailAsPromise, 17 blakfx_helix_userFindByName, 17	option_t helix_types.h, 21
blakfx_helix_userFindByNameAsPromise, 18	
blakfx_helix_userRelease, 18	PROMISE_ALLOW_RECURSIVE_EVENTS
blakfx_helix_userValidate, 18	helix_types.h, 22
blakfx helix waitEvent, 18	PROMISE COMPLETE
blakfx_helix_waitEventStatus, 19	helix_types.h, 22
	PROMISE DATA AVAILABLE
helix_crypto_test.c	helix_types.h, 22
main, 33	PROMISE_DESTROY
helix_types.h	helix_types.h, 22
invokeStatus_t, 21	— · ·
logLevel_t, 21	PROMISE_ERROR_UNDEFINED
option_t, 22	helix_types.h, 22
promiseStatusAndFlags_t, 22	PROMISE_EVENT
serverResponseCode_t, 22	helix_types.h, 22
DECRYPT_ID, 21	PROMISE_INFINITE
ENCRYPT_ID, 21	helix_types.h, 22
invokeStatus_t, 21	PROMISE_INVALID
KEY_ID, 21	helix_types.h, 22
logLevel_t, 21	PROMISE_MEMORY_ALLOCATED
- -	helix types.h, 22
option_t, 21	PROMISE_MEMORY_POST_RELEASED_ID
PROMISE_ID, 21	helix_types.h, 22
promiseStatusAndFlags_t, 21	PROMISE_MEMORY_RELEASING
serverResponseCode_t, 21	
USER_ID, 21	helix_types.h, 22
	PROMISE_NO_STATUS
INFO_LEVEL	helix_types.h, 22
helix_types.h, 22	PROMISE_RESULT_ERROR
INVOKE_IN_INVALID_STATE	helix_types.h, 22
helix_types.h, 21	PROMISE_USER_EVENT
INVOKE_INVALID_INSIDE_CALLBACK	helix_types.h, 22
helix_types.h, 21	PROMISE_WAIT_STATUS
INVOKE_STATUS_BAD_PROMISE_ID	helix_types.h, 22
helix types.h, 21	PROMISE ID
INVOKE_STATUS_FALSE	helix_types.h, 21
helix types.h, 21	promiseStatusAndFlags t
INVOKE_STATUS_NOT_INITIALIZED	helix_types.h, 21
	11011/typootif, = 1
helix_types.h, 21	SERVER FAIL
INVOKE_STATUS_TIMEOUT	helix_types.h, 23
helix_types.h, 21	SERVER_SUCCESS
INVOKE_STATUS_TRUE	helix types.h, 23
helix_types.h, 21	— • · · · ·
include/helix_crypto.h, 7	serverResponseCode_t
include/helix_types.h, 19	helix_types.h, 21
invokeStatus t	src/helix_crypto.c, 23
helix_types.h, 21	test/helix_crypto_test.c, 33
KEY ID	
helix_types.h, 21	USER_OWNS_MEMORY
Tielix_types.ti, 21	helix_types.h, 22
logLevel_t	USER_ID
-	helix_types.h, 21
helix_types.h, 21	
main	WARN_LEVEL
main	helix_types.h, 22
helix_crypto_test.c, 33	
mainpage.md, 23	
NO LOG	
NO_LOG	
helix_types.h, 22	