Contents

1 class **Acs**

Access certificates server

Public Members

Private Members

Access certificates server.

Stores and distributes access certificates of other services like Eso Bank Mix. It handles requests for publishing and retreival of certificates.

See Also: Mix, Acs, Eso, Client
Author: Pechy, Venca

1.1 attributes

Names

MessageQueue*

	receiverToMajordomo	*Queue connecting Receiver and Majordomo*
MessageQueue*	**majordomoToSender**	*Queue connecting Majordomo and Sender*
CertificateRepository*	**certificateRepository**	*Repository containing certificates*
AddressManager*	**addressManager**	*Manager of certificates*
Killer*	**killer**	*Class responsible for shutdown of all objects*
ConfigFile*	**configFile**	*Configuration file*
LogFile*	**logFile**	*Log file*
Receiver*	**receiver**	*Message receiver*
Majordomo*	**majordomo**	*Requests handler*
Sender*	**sender**	*Message sender*
RunningObject*	**runningObject**	*Object that is used to wait for the threads to finish*

1.2 methods

Names

	Acs (string configName = ACS_CONFIG_FILE)	*Constructor*
virtual	**~Acs** ()	*Destructor*
virtual void	**Run** ()	*This method runs as a thread and processes all the requests*

1.4

Acs (string configName)

Acs constructor

Acs constructor.

Creates all the objects necessary for Acs to run.

Parameters: `configName` — Name of configuration file that should be opened and from where we should read all the configuration information.
See Also: Majordomo, Sender, Receiver
Author: Venca

1.5

RunningObjectID **StartObject** (Runable *object)

Starts object in aseparate thread

Starts object in aseparate thread.

Creates a thread in which the specified object will run its method Run().

Return Value: Id of the running object so that we can wait for the newly created thread to finish
Parameters: `object` — Runable object that implements virtual method Run that will be executed in the new thread
See Also: Runable Acs Run()
Author: Venca

1.6

~Acs ()

Acs destructor

Acs destructor.

Deletes all the objects receiver, sender, majordomo ... <p> Writes info about exiting into log file and closes it.

Author: Venca

1.3

methods

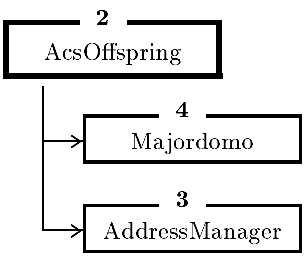
Names

RunningObjectID
StartObject (Runable *runableObject)
Used to dispatch an object in a thread

2 class **AcsOffspring**

Offspring of Acs class

Inheritance



Public Members

Protected Members

Offspring of Acs class.

Objects that inherit from this class are offsprings of a different object. The purpose of it is to maintain the relation between father and his sons.

See Also: Acs, Majordomo
Author: Venca

2.2 methods

Names

AcsOffspring (Acs * acs)
Constructor

2.3 AcsOffspring (Acs * acs)

Acs offspring constructor

Acs offspring constructor.

It just remembers pointer to its father so that it is possible to access its members like other objects.

See Also: Acs (→1, *page 11*)
Author: Venca

2.1 attributes

Names

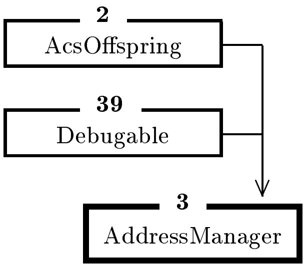
Acs*	**acs**	*Pointer to its parent (Acs)*

3

class **AddressManager** : public Debugable, public AcsOffspring

Access certificates manager

Inheritance



Public Members

Access certificates manager.

Takes care of expiring expunging and creating certificates that are stored in a container class called CertificateRepository.

See Also: CertificateRepository (→27, *page 136*)
Author: Venca

3.3

methods

Names

AddressManager (Acs * parent, LogFile * addressManagerLogFile)
Constructor

~AddressManager ()
Destructor

GMessage*
GetMixCertificate ()
Returns one Mix certificate from CertificateRepository

GMessage*
CreateAcsCertificate ()
Returns one Acs certificate from CertificateRepository

virtual void
Shutdown () *Prepare for clean shutdown*

GMessage*
CreateAcsCertificate (u_short port, GMessage **certificate)
Generates a new Acs certificate

Err **StoreAsymetricKeyPairOntoDisk** (GMessage *asymetricKeyPair)
Writes a pair of asymetric keys in a file anto the disk

3.4

AddressManager (Acs * parent, LogFile * addressManagerLogFile)

Address manager constructor

Address manager constructor.

Loads Acs certificate from file and if it is not present there then it generates a new one which is written to disk for fututre use.

See Also: CFG_CERTIFICATES_DIRECTORY, CFG_CERTIFICATE
Author: Venca

3.5

AddressManager (Mix * parent, LogFile * addressManagerLogFile)

AddressManager constructor

AddressManager constructor.

Responsible for loading Mix's certificate into memory. Creates the certificate if it is not found.

Parameters: `parent` — Pointer to the Mix object.
`addressManagerLogFile` — Log File.
See Also: certificate, Cipherer
Author: Venca, Pechy

3.6

void **CertificatesArrived** (GMessage *data)

Stores incoming certificates

Stores incoming certificates.

Stores certificates from ACS to its internal structure. Asumes that it gets a GMessage of certificates, that are GMessages too.

Return Value: void
Parameters: `data` — Certificates. <pre> GM_CERTIFICATE_1 GM_CERTIFICATE_2 ... GM_CERTIFICATE_"n" </pre>
See Also: certificates, Acs, GMessage
Author: Venca

3.7

GMessage* **CreateAcsCertificate** ()

Create Acs certifcate

Create Acs certifcate.

Generates an Acs certificate when there is missing. <p> Structure of the certificate is: <pre> Certificate-Type: Acs Name: acs-johny Mix-Certificate: Address: ... Port </pre>

See Also: AddressManager, Acs
Author: Venca

3.8

Err **CreateMixCertificate** (u_short port, GMessage **certificate)

Creates a new Mix's certificate

Creates a new Mix's certificate.

Finds out the hostname of Mix and creates the new certificate and stores it onto the disk.

Return Value: <pre> GM_CERTIFICATE_TYPE - the name of a server GM_ADDRESS - DNS of IP address GM_PORT - port where the server is running GM_PUBLIC_KEY_INFO - public key from the server's key pair GM_NOT_VALID_BEFORE - validity GM_NOT_VALID_AFTER - validity </pre>

Parameters: `port` — Port which we include in the new certificate.
`certificate` — Where we return the certificate.

See Also: Cipherer, certificates

3.9

void **DownloadCertificates** (Size count, string type)

Sends a request to ACS for certificates

Sends a request to ACS for certificates.

When some Six asks for certificates and Mix doesn't have enough of them of this type, we have to download them from an Acs.

Return Value: void

Parameters: `count` — How many certificates to download.
`type` — What certificate to download.

See Also: certificates, Acs

Author: Venca

3.10

GMessage* **GetMixCertificate** ()

Get Mix certificate

Get Mix certificate.

Returns a Mix certificate stored on disk.

See Also: GetAcsCertificate
Author: Venca

3.11

void **HereYouAre** (GMessage * data)

Store given certificates

Store given certificates.

When Address Manager asks for certificates, this method is called when certificates are returned, ready to be stored.

Return Value: void
Parameters: `data` — Certificates.
See Also: certificates
Author: Venca

3.12

Err **LoadCertificates** ()

Load certificates from disk

Load certificates from disk.

Forces certificate repository to load all certificates from disk.

See Also: SaveCertificates()
Author: Venca

3.13

Err ReturnAcsCertificate (GMessage **acsCertificate)

Returns a certificate of any Acs

Returns a certificate of any Acs.

Returns a certificate of any Acs.

Return Value: KO if not found, OK otherwise.
Parameters: `acsCertificate` — Where to return the certificate.
See Also: certificates
Author: Venca

3.14

Err ReturnLocalMixCertificate (GMessage **mixCertificate)

Returns Mix's own certificate

Returns Mix's own certificate.

Returns Mix's own certificate, that is stored somewhere in the disk. We consider the Mix machine as a safe one.

Return Value: KO if not found, OK otherwise.
Parameters: `mixCertificate` — Where to return the certificate.
See Also: certificates
Author: Venca

3.15

Err **ReturnMixCertificate** (GMessage **mixCertificate)

Returns a certificate of any Mix

Returns a certificate of any Mix.

Returns a certificate of any Mix. These certificates should be in repository. If there are not enough of certificates, a request for Acs is made and sent.

Return Value:	KO if not found, OK otherwise.
Parameters:	`mixCertificate` — Where to return the certificate.
See Also:	certificates
Author:	Venca

3.16

Err **SaveCertificates** ()

Save certificates to disk

Save certificates to disk.

Forces certificate repository to save all certificates to disk.

See Also:	LoadCertificates()
Author:	Venca

3.17

void **Shutdown** ()

Prepare for clean shutdown

Prepare for clean shutdown.

This method should implement all the tasks necessary for preparation for a clean shutdown like saving inconsistent data to disk etc. Now it is empty as AddressManger is completely stateless as it stores certificate to disk immediatelly after it is generated.

See Also: Shutdownable, Killer
Author: Venca

3.18

Err **StoreAsymetricKeyPairOntoDisk** (GMessage *asymetricKeyPair)

When creating a new certificate, it saves asymetric key pair onto the disk

When creating a new certificate, it saves asymetric key pair onto the disk.

Saves asymetric key pait onto the disk. This can happen when creating a new certificate (we have to create the new pair, too).

Return Value: KO if the pair can't be saved, OK otherwise.
Parameters: `asymetricKeyPair` — Key pair to store.
See Also: Cipherer (→34, *page 150*)
Author: Pechy

3.19

˜AddressManager ()

Acs address manger destructor

Acs address manger destructor.

It just deletes Acs certificate from memory.

See Also: AddressManager(), Acs
Author: Venca

3.1

attributes

Names

ID	**lastID**	*Last used certificate ID*
GMessage*	**acsCertificate**	*Certificate of this Acs*

3.2

methods

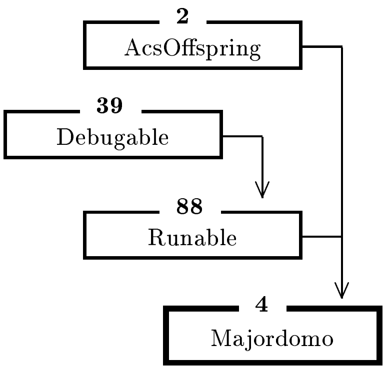
Names

Err	**SaveCertificates** ()	*Storing certificates to disk*
Err	**LoadCertificates** ()	*Loading certificates from disk*

4 class **Majordomo** : public Runable, public AcsOffspring

Acs majordomo

Inheritance



Public Members

Acs majordomo..

Processes various requests for certificates comming from network and takes apropriate action like sends back the requested certificates

See Also: Acs (→1, *page 11*)
Author: Venca

4.1 methods

Names

Majordomo (Acs * parent, LogFile * majordomoLogFile)
Constructor

virtual void*
Run (void *) *This method runs in thread and processes all the requests*

4.2 GMessage* **CreateSimpleReqForFile** (GMessage *pubKey)

Creates REQ_FOR_FILE

Creates REQ_FOR_FILE.

Creates REQ_FRO_FILE message, that is to be sent to Esos.

Return Value:	created GMessage
Parameters:	`pubKey` — PubKey for encrypting FFID.
See Also:	Cipherer (→34, *page 150*)
Author:	Marek

4.3 GMessage* **CreateSimpleReqForHeaders** (GMessage *pubKey)

Creates REQ_FOR_HEADERS

Creates REQ_FOR_HEADERS.

Creates REQ_FRO_HEADERS message, that is to be sent to Esos.

Return Value: created GMessage
Parameters: `pubKey` — PubKey for encrypting FAM.
See Also: Cipherer (→34, *page 150*)
Author: Marek

4.4

GMessage* **CreateSimpleReqForStor** (File *fileToStore)

Creates simple req for storage

Creates simple req for storage.

Creates simple request for storage, that is the message sent to Eso as a CE_REQUEST_FOIR_STOR.

Return Value: created messsage.
Parameters: `fileToStore` — File that is wanted to be stored.
See Also: GMessage
Author: Pavel

4.5

Err **DRCAnswerFile** (const MsgField *dataFld)

Serves CE_REPLY_FILE message

Serves CE_REPLY_FILE message.

Gets the file out of the GMessage and saves it into a file specified by user.

Return Value: error code
Parameters: `dataFld` — Data from the GMessage.
See Also:
Author: Marek

4.6

Err **FillReplyHeader** (GMessage *outMsg, TID inReplyToTID, GMessage *aHeader)

Sets the outMsg as ReplyHeader

Sets the outMsg as ReplyHeader .

Fills all needed fields in a message that is sent as a CMD_REPLY_HEADER command.

Return Value: error code
Parameters: `parametr` —
See Also: co_se_toho_tyka
Author: Marek

4.7

Err **FillRequestForFile** (GMessage *outMsg, MsgField * aFFID, GMessage *transportKey, Int depth, Int width)

Sets the outMsg as RequestForFile

Sets the outMsg as RequestForFile.

Fills all needed fields in a message that is sent as a reply to CMD_REQ_FOR_FILE command.

Return Value: error code
Parameters: `parametr` —
See Also: co_se_toho_tyka
Author: Marek

4.8

include . H majordomo.h Err **FillRequestForHeaders** (GMessage *outMsg, MsgField *aFAM, GMessage *transportKey, Int depth, Int width)

Sets the outMsg as RequestForHeaders

Sets the outMsg as RequestForHeaders .

Fills all needed fields in a message that is sent as a reply to CMD_REQ_FOR_HEADERS command.

Return Value: error code
Parameters: `parametr` —
See Also: co_se_toho_tyka
Author: Marek

4.9

void **FindFile** (ArrayGMessage *certificates)

Tries to find file on Esos

Tries to find file on Esos.

Creates request message, sends it to given number of Esos and waits for reply.

Return Value: void
Parameters: `certificates` — Certificates of Esos to ask.

See Also:
Author: Marek

4.10

Majordomo (Client * parent, LogFile * logFile)

Constructor of Majordomo in Client

Constructor of Majordomo in Client.

Initializes the specific attributes.

Parameters: `parent` — Parent of the object.
`logFile` — LogFile to write to.
See Also: SixCertificates
Author: Pavel

4.11

include H majordomo.h **Majordomo** (Eso * parent, LogFile * majordomoLogFile)

Eso Majordomo constructor

Eso Majordomo constructor.

Calls SixMajordomo constructor.

Parameters: `parent` — Eso this Majordomo belongs to
`majordomoLogFile` — log file for debugging messages
See Also: Six, SixMajordomo
Author: Pavel

4.12

include . .. .. Common H common.h **Majordomo** (Mix * parent, LogFile * majordomoLogFile)

Majordomo constructor

Majordomo constructor.

Saves pointers to object passed as parameters.

Parameters: `parent` — Pointer to the main object Mix.
`majordomoLogFile` — Log file.
See Also: ~Majordomo()
Author: Venca

4.13

MsgField* **ParseCipheredData** ()

Parses ciphered data

Parses ciphered data.

Parses encrypted data to get the stored file. Decrypts them with added publicKey. <pre> The incoming ciphered data looks like this: GM_PUBLIC_KEY: GM_DATA: (encrypted with Private Key) GM_RANDOM: (random string) GM_FILE: (decrypted file) </pre>

Return Value: msgField decrypted file
See Also: Cipherer (→34, *page 150*)
Author: Pavel

5

include . .. Common H common.h int **main** (int argc, char ** argv)

Acs main function

Acs main function.

Parses command line and sets global variables according to switches specified. It also creates Acs' main object and runs its method Run().

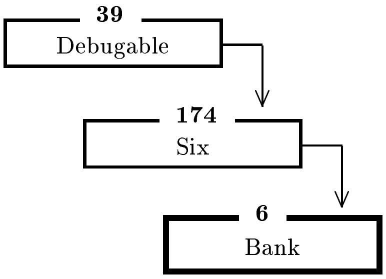
See Also: Acs (→1, *page 11*)
Author: Venca

6

class **Bank** : public Six

Main class managing Bank service

Inheritance



Public Members

		attributes		
6.1		**methods**		38
6.2		**Bank** (string configName)	*Constructor*	39
6.3	void	**CreateMajordomo** ()	*Creates Bank Majordomo*	39
6.4	Err	**Initialize** ()	*Initializes Bank*	40
6.5		**~Bank** ()	*Destructor*	40

Main class managing Bank service.

This class derives its behaviour from general Six extending its functionality with some features we would like the Bank to have.

See Also: Six (→174, *page 514*)
Author: Marek

6.1 methods

Names

6.1.1 virtual void **CreateMajordomo** ()

Sets Six::majordomo to new Eso's Majordomo

Sets Six::majordomo to new Eso's Majordomo. Overrides Sixes virtual method.

6.1.2 Err **Initialize** ()

Does some really cool initializations needed especially for debugging purposes

Does some really cool initializations needed especially for debugging purposes.

6.2

Bank (string configName)

Constructor

Constructor.

Creates all the objects necessary to run.

Parameters:	`configName` — name of configuration file for Bank
See Also:	ConfigFile (→37, *page 178*)
Author:	Marek

6.3

void **CreateMajordomo** ()

Creates Bank Majordomo

Creates Bank Majordomo.

Overrides Six::CreateMajordomo method to use proper SixMajordomo.

Return Value:	void
See Also:	Six, SixMajordomo, Majordomo
Author:	Marek

6.4

Err Initialize ()

Initializes Bank

Initializes Bank.

Used especially for debugging. These days it simply asks for onion to generate own access certificate.

Return Value:	error code
See Also:	AskForOnion()
Author:	Marek

6.5

~Bank ()

Destructor

Destructor.

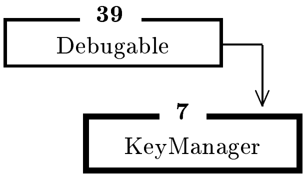
Deletes all the objects - frees memory.

See Also:	
Author:	Marek

7 class **KeyManager** : public Debugable

KeyManager

Inheritance



Public Members

7.3 **methods** 42

7.4 TableFieldQueue* **CreateDefinition** ()
Creates definition of acTable - used only in constructor 42

7.5 GMessage* **GetMixPrivateKey** (char * , char *)
Returns Mix private key 43

7.6 GMessage* **GetMixPublicKey** (char * , char *)
Returns Mix public key 43

7.7 GMessage* **GetPrivKeyByACID** (ACID acID)
Finds the key according to its keyID 44

Protected Members

7.1 **attributes** 44

7.2 **methods** 44

KeyManager.

manages the public and private keys in the table

Author: Marek

7.3 methods

Names

- **KeyManager** (LogFile *aLogFile, Char* aTableName) *Constructor*
- virtual **~KeyManager** () *Destructor*
- Err **SaveKeys** (ACID acid, GMessage *pubKey, GMessage *privKey) *inserts AC into the table*
- GMessage* **GetPrivKeyByACID** (ACID acID) *finds the AC according to its ACID*

7.4 TableFieldQueue* **CreateDefinition** ()

Creates definition of acTable - used only in constructor

Creates definition of acTable - used only in constructor.

The records from the table have 3 fields: ACID, PUB_KEY and PRIVATE_KEY.

Return Value: tableFieldQueue - model queue
See Also:
Author: Marek

7.5

GMessage* **GetMixPrivateKey** (char * , char *)

Returns Mix private key

Returns Mix private key.

Extracts private part from asymetric key and returns it.

Return Value: Pointer to GMessage containing asymetric private key.
See Also: Cipherer (→34, *page 150*)
Author: Pechy

7.6

GMessage* **GetMixPublicKey** (char * , char *)

Returns Mix public key

Returns Mix public key.

Extracts public part from asymetric key and returns it.

Return Value: Pointer to GMessage containing asymetric public key.
See Also: Cipherer (→34, *page 150*)
Author: Pechy

7.7

GMessage* **GetPrivKeyByACID** (ACID acID)

Finds the key according to its keyID

Finds the key according to its keyID.

Returns private key associated with ACID from the keyManager's table.

Return Value: privKey (GMessage)
Parameters: `acid` — ACID to find the key with.
See Also:
Author: Marek

7.1 attributes

Names

Table*	**keyTable**	*Table containing keys*

7.2 methods

Names

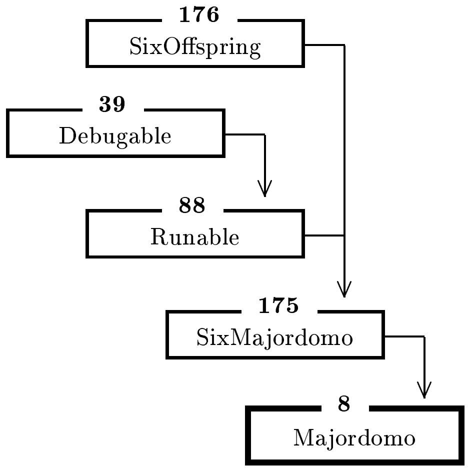
TableFieldQueue*
CreateDefinition ()
creates definition of acTable - used only in constructor

8

class **Majordomo** : public SixMajordomo

Class responsible for handling messages comming to Bank

Inheritance



Public Members

Class responsible for handling messages comming to Bank.

This descendant of SixMajordomo takes main functionality from its ancestor extending it with functionality typical from Bank in Eternity Service.

See Also: co_se_toho_tyka
Author: jmeno

8.2 methods

Names

Majordomo (Bank * parent,
LogFile * majordomoLogFile)

Bank Majordomo constructor

virtual ACManager* **GetPtrToACManager** (Char *type)
Overriden method that returns pointer to ACManager of specified type

Err **Initialize** () *Some cool initialization*

ESTime* **GetTime** () *Returns current time*

MsgField* **UnsealSAuth** (const ACID acid, const MsgField *sealSA)
Decrypts S_AUTH

8.3

Err **CommandTimeoutAllocation** (MsgField *dataFld)

Serves TO_ALLOC Processes timeout command for allocation

Serves TO_ALLOC

Processes timeout command for allocation. Deletes apropriate records in Allocator and Banker.

Return Value: error code
Parameters: `dataFld` — data comming with timeout
See Also: Allocator, Banker,
Author: Marek, Pavel

8.4

endif MsgField* **ComputeMAC** (MsgField *oAuth)

Computes MAC

Computes MAC.

Computes MAC from OAuths.

Return Value: MsgField computed MAC.
Parameters: `oAuth` — oAuth to compute MAC.
See Also:
Author: Pavel

8.5

ifndef NO_BANKER Err **CreateCipheredData** ()

Creates ciphered data

Creates ciphered data.

Modifies file for storage. Adds random string and encrypts with private key. <pre> The result looks like this: GM_PUBLIC_KEY: GM_DATA: (encrypted with Private Key) GM_RANDOM: (random string) GM_FILE: (decrypted file) </pre>

Return Value: error code
See Also: Cipherer, RandomGenerator
Author: pavel

8.6

GMessage* **CreateSimpleFileToStor** (GMessage *certificate)

Creates simple message with a file for storage

Creates simple message with a file for storage.

Creates a message with file for storage, that is going to be sent to Eso as a CE_FILE_FOR_STOR including encryption of the file with TCBPubKey of the Eso.

Return Value:	created messsage.
Parameters:	`certificate` — Certificate of the Eso with the TCBPubKey.
See Also:	GMessage, Cipherer
Author:	Pavel

8.7

Err **DRCAnswerHeader** (const MsgField *dataFld)

Serves CE_REPLY_HEADER message

Serves CE_REPLY_HEADER message.

Gets the header out of the GMessage and prints info about it to the stdout.

Return Value:	error code
Parameters:	`dataFld` — Data from the GMessage.
See Also:	
Author:	Marek

8.8

Err **DRCAnswerHeader** (const MsgField *dataField)

Serves CMD_ANSWER_HEADER message

Serves CMD_ANSWER_HEADER message.

processes the incoming found header - forwards found header to client

Return Value: error code

Parameters: `dataField` — encrypted request comming with message
`acid` — ID of this Eso access certificate which was used to address thid message

See Also: TCBWrapper, Finder

Author: Marek

8.9

Err **DRCDTSGetFields** (const MsgField *dataFld, GMessage **replyToAC, TID *replyToTID, TID *inReplyToTID, MsgField **ffid, MsgField **keywords, MsgField **dataToStoreFld)

Parses the GMessage comming with command CE_DATA_TO_STORE

Parses the GMessage comming with command CE_DATA_TO_STORE.

GMessage comming with command CE_DATA_TO_STORE should have this structure: <pre> Reply-To-ID: ID of a message assigned by sender (Client) FFID: Keywords: Data: </pre>

Return Value: error code

Parameters: `dataFld` — data comming with command
`replyToAC` — out parameter for Reply-To access certificate
`replyToTID` — out parameter for Reply-To message ID
`inReplyToTID` — out parameter for In-Reply-To message ID
`ffid` — out parameter for file find ID
`keywords` — out parameter for keywords
`dataToStoreFld` — out parameter for data to store

See Also: DRCDataToStore()

Author: Pavel

8.10

Err **DRCDataToStore** (const MsgField *dataFld, const ACID acid)

Serves CE_DATA_TO_STOR command

Serves CE_DATA_TO_STOR command.

Handles the CE_DATA_TO_STOR command, according to Eso-Others interface. <pre> In-Reply-To-TID: ID of a EC_REPLY_ALLOC message Reply-To-ID: ID of a message assigned by sender (Client) Reply-To-AC: AC to reply to FFID: Keywords: Data: </pre>

Return Value:	error code
Parameters:	`dataFld` — data comming with command
See Also:	FillReplyFileStored(), Allocator, Banker, Scheduler, TCBWrapper
Author:	Pavel

8.11

void **DownloadHeaders** ()

Downloads headers from Esos

Downloads headers from Esos.

Tries to find files according to FileAttributeMask given by user in command line.

Return Value:	void
See Also:	
Author:	Marek

8.12

Err **FillReplyAllocation** (GMessage *outMsg, TID inReplyToTID, MsgField *payPlan)

Sets the outMsg as ReplyAllocation

Sets the outMsg as ReplyAllocation.

Fills all needed fields in a message that is sent as a reply to CE_REQ_FOR_STOR command.

Return Value: error code
Parameters: `parametr` —
See Also: co_se_toho_tyka
Author: Pavel

8.13

Err **FillReplyFile** (GMessage *outMsg, TID inReplyToTID, MsgField *aFile)

Sets the outMsg as ReplyFile

Sets the outMsg as ReplyFile .

Fills all needed fields in a message that is sent as a CMD_REPLY_FILE command.

Return Value: error code
Parameters: `parametr` —
See Also: co_se_toho_tyka
Author: Marek

8.14

Err **FillReplyFileStored** (GMessage *outMsg, TID inReplyToTID)

Sets the outMsg as ReplyFileStored

Sets the outMsg as ReplyFileStored.

Fills all needed fields in a message that is sent as a reply to CE_DATA_TO_STOR command.

Return Value: error code
Parameters: `parametr` —
See Also: co_se_toho_tyka
Author: Pavel

8.15

GMessage* **LoadPrivateKey** ()

Returns Client's privKey

Returns Client's privKey.

Loads Client's privKey from file.

Return Value: client's privKey
See Also:
Author: Marek

8.16

GMessage* **LoadPublicKey** ()

Returns Client's pubKey

Returns Client's pubKey.

Loads Client's pubKey from file.

Return Value: client's pubKey
See Also:
Author: Marek

8.17

Majordomo (Acs * parent, LogFile * majordomoLogFile)

Acs Majordomo constructor

Acs Majordomo constructor.

Just initializes the parent classes as it has nothing else to do.

Parameters: `parent` — Pointer to the parent Acs.
`logFile` — Log file where to write messages
See Also: Acs (→1, *page 11*)
Author: Venca

8.18

include H majordomo.h **Majordomo** (Bank * parent, LogFile * majordomoLogFile)

Bank Majordomo constructor

Bank Majordomo constructor.

Initializes majordomo.

See Also: Bank, SixMajordomo
Author: Marek

8.19

void **WaitForAnswersToReqForHeaders** ()

Waits for answers to REQ_FOR_HEADERS

Waits for answers to REQ_FOR_HEADERS.

Waits for answers for number of seconds given in ConfigFile.

Return Value: void
See Also:
Author: Marek

8.20

void **WaitForAnswersToReqForStor** (int requestsSent)

Waits for answers to CE_REQUEST_FOR_STOR

Waits for answers to CE_REQUEST_FOR_STOR.

Waits for answers to CE_REQUEST_FOR_STOR in a loop until a timeout is over.

Return Value: void
Parameters: `requestSent` — Number of messages for wait for.
See Also:
Author: Pavel

8.21 ˜Majordomo ()

Destructor of Majordomo

Destructor of Majordomo.

Destructor of Majordomo in Client

Return Value: co
See Also:
Author: pavel

8.1 methods

Names

Methods managing access certificates

virtual MsgField* **DecryptByACPriv** (const ACID acid, const MsgField *dData)
Decrypting data from Service

virtual Err

SwitchDataReceivedCommand (const Char *command, const MsgField *dataFld, const ACID acid)

Handle data_received command

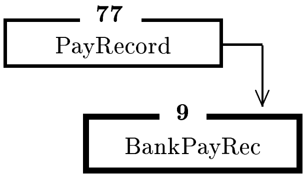
Data_received commands handling methods

9

class **BankPayRec** : public PayRecord

Bank Payment Record

Inheritance



Public Members

methods

9.1	include . H payManager.h	**BankPayRec** ()	*Bank Payment Record constructor*	60
9.2	Void	**Clear** ()	*Clear data*	60
9.3	Err	**FetchFromRecord** (Record *r)	*Fetches from given record*	61
9.4	Err	**FillRecord** (Record *r)	*Fills given record*	61
9.5	Char*	**GetAmount** ()	*Get Amount*	62
9.6	MsgField*	**GetSealSAuth** ()	*Get SealSAuthentification*	62
9.7	Int	**MACEqualTo** (MsgField *otherMAC)	*MAC is equal to given MAC*	63
9.8	Int	**SAuthEqualTo** (MsgField *otherSA)	*SAuth is equal to given sAuth*	63
9.9	Err	**SetSAuth** (MsgField *otherSA)	*Set SAuthentification*	63
9.10		**~BankPayRec** ()	*Bank Payment Record destructor*	64

Protected Members

attributes

Bank Payment Record.

An auxiliary class for one Payment record in Bank. Later it should have attribute account, which will be represent a real 'noname account' information in bank, from which the bank will pay to Eso.

Author: Ivana see PayManager

9.1

include . H payManager.h **BankPayRec** ()

Bank Payment Record constructor

Bank Payment Record constructor.

Bank Payment Record constructor initializes variables.

See Also: PayManager::StorePayments(), PayManager::AnswerFromEso()

Author: Ivana

9.2

Void **Clear** ()

Clear data

Clear data.

Clears data - including eCode.

Return Value: void
See Also: PayManager::StorePayments()
Author: Ivana

9.3

Err FetchFromRecord (Record *r)

Fetches from given record

Fetches from given record.

Fetches data from given record. A virtual method, that should be overwritten, to be able to read from table.

Return Value: error code
Parameters: `r` — record from which we take data
See Also: PayRecord (→77, *page 256*)
Author: Ivana

9.4

Err FillRecord (Record *r)

Fills given record

Fills given record.

Fills given record with data. A virtual method, that should be overwritten, to be able to write to or update table.

Return Value: error code
Parameters: `r` — record into which we store data
See Also: PayRecord (→77, *page 256*)
Author: Ivana

9.5

Char* **GetAmount** ()

Get Amount

Get Amount.

Returns our amount field in new string.

Return Value:	copy of amount
See Also:	PayManager::StorePayments(), PayManager::AnswerFromEso()
Author:	Ivana

9.6

MsgField* **GetSealSAuth** ()

Get SealSAuthentification

Get SealSAuthentification.

Returns our sealSAuth field in new MsgField.

Return Value:	copy of sealSAuth
See Also:	PayManager::StorePayments()
Author:	Ivana

9.7

Int **MACEqualTo** (MsgField *otherMAC)

MAC is equal to given MAC

MAC is equal to given MAC.

Finds wheather given mac is equal to this->mac.

Return Value:	YES/NO
Parameters:	`otherMAC` — a mac that is compared to ours
See Also:	PayManager::AnswerFromEso()
Author:	Ivana

9.8

Int **SAuthEqualTo** (MsgField *otherSA)

SAuth is equal to given sAuth

SAuth is equal to given sAuth.

Finds wheather given sAuth is equal to this->sAuth.

Return Value:	YES/NO
Parameters:	`otherSA` — an sAuth that is compared to ours
See Also:	PayManager::AnswerFromEso()
Author:	Ivana

9.9

Err **SetSAuth** (MsgField *otherSA)

Set SAuthentification

Set SAuthentification.

Sets our sAuth to given one.

Return Value: error code
Parameters: `otherSA` — sAuth for setting our data
See Also: PayManager::StorePayments()
Author: Ivana

9.10

~BankPayRec ()

Bank Payment Record destructor

Bank Payment Record destructor.

Bank Payment Record destructor deletes variables.

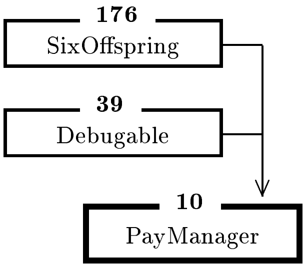
See Also: PayManager::StorePayments(), PayManager::AnswerFromEso()
Author: Ivana

10

class **PayManager** : public Debugable, public SixOffspring

Payment Manager

Inheritance



Public Members

Protected Members

Payment Manager.

A class which manages everything about payments in Bank: 1) stores payment plan sent by client (and transfer money) 2) reacts to eso's request for payment - if everything is OK transfer money to it's account

Author: Ivana see Bank

10.3 methods

Names

GMessage*
MoneyFromClient (GMessage *inMsg)
reacts to message from client

GMessage*
RequestFromEso (GMessage *inMsg)
reacts to a request for payment from Eso (prepares challenge for it)

GMessage*

AnswerFromEso (GMessage *inMsg)
reacts to an answer from Eso

10.4

GMessage* **AnswerFromEso** (GMessage *inMsg)

Reacts to an answer from Eso

Reacts to an answer from Eso.

If it finds the payment record and MAC and SAuth are OK, transfers money to Eso's account.

Return Value: message (answer) for eso
Parameters: `inMsg` — message (response to challenge) from eso
See Also: Majordomo::DRCAnswerFromEso()
Author: Ivana

10.5

GMessage* **CreateCommonMsg** (Err succ, Char *reason, ID *payId)

Create Common Message

Create Common Message.

Creates part of message common to all outgoing messages. If parametr payId is not specified, does not set it.

Return Value: created message
Parameters: `succ` — value of succes field
`reason` — value of reason field
`payId` — value of paymentId field

See Also: MoneyFromClient(), RequestFromEso(), AnswerFromEso()
Author: Ivana

10.6

Err **CreatePayments** ()

Create Payments Table

Create Payments Table.

Creates table Payments - a persistent PayManager's table.

Return Value: error code
See Also: PayManager()
Author: Ivana

10.7

Err **DeleteRecsInTable** (Table *t, Char *whichId, ID id)

Deletes all records in the table

Deletes all records in the table.

Deletes all records in the table with given ID. If whichId is not specified (NULL), clears given table.

Return Value: error code
Parameters: `t` — table in which we will delete
`whichId` — specifies ID, that is used for deleting
`id` — a value of the ID
See Also: AnswerFromEso()
Author: Ivana

10.8

Record* **FindInTable** (Table *t, Char *whichId, ID id)

Finds record in the table

Finds record in the table.

Finds record in the table with given ID.

Return Value:	found record or NULL
Parameters:	`t` — table in which we will search
	`whichId` — specifies ID, that is used for searching
	`id` — a value of the ID
See Also:	RequestFromEso()
Author:	Ivana

10.9

GMessage* **MoneyFromClient** (GMessage *inMsg)

Reacts to message from client

Reacts to message from client.

It first generate temporary table of sent PaymentPlan. Then stores modified (with unsealed SAuths) table to PayManager's payments table, transfer money from client account and generate message for him.

Return Value:	message (answer) for client
Parameters:	`inMsg` — message from client
See Also:	Majordomo::DRCMoneyFromClient()
Author:	Ivana

10.10

PayManager (LogFile *logFile, Bank *bank)

PayManager constructor

PayManager constructor.

Creates PayManager's persistent table.

Parameters: `logFile` — pointer to log file
`bank` — pointer to main class bank
See Also: Bank (→6, *page 37*)
Author: Ivana

10.11

GMessage* **RequestFromEso** (GMessage *inMsg)

Reacts to a request for payment from Eso

Reacts to a request for payment from Eso.

If it finds the payment record, the Eso ask for, and time of it is OK, sends to Eso OAuth (to be able to compute MAC).

Return Value: message (challenge) for eso
Parameters: `inMsg` — message from eso
See Also: Majordomo::DRCRequestFromEso()
Author: Ivana

10.12

Long **StorePayments** (Table *fromClient, ACID myACID)

Stores Payments

Stores Payments.

Stores payments from given table. For each record first unseal SAuth and then stores all useful information to 'payments' table. If something goes wrong, returns -1 instead of sum.

Return Value:	total sum of money from client
Parameters:	`fromClient` — table, from which payments are taken `myACID` — ID of bank's ac, which used for encrypting sAuth
See Also:	MoneyFromClient()
Author:	Ivana

10.13

~PayManager ()

PayManager destructor

PayManager destructor.

Destructor deletes PayManager's persistent table.

See Also:	Bank (→6, *page 37*)
Author:	Ivana

10.1 attributes

Names

Table* **payments** *PayManager's persistent table of payments*

10.2 methods

Names

Long **StorePayments** (Table *fromClient, ACID myACID)
stores PayPlan from given table to payments table

GMessage* **CreateCommonMsg** (Err succ, Char *reason=NULL, ID *payId=NULL)
creates outMsg with fields Success, Reson-Of-Fail and Paymet-ID

11 PayManagerDefines

Names

PayManager's files dir

PayManagers' files

12

include . .. Common H common.h int **main** (int argc, char ** argv)

Bank main function

Bank main function.

Parses options Bank can be given from command line and then creates an instance of class Bank which then runs the service.

See Also: Bank (→6, *page 37*)
Author: Venca

13 BankerDefines

Names

Banker's files dir

Bankers' files

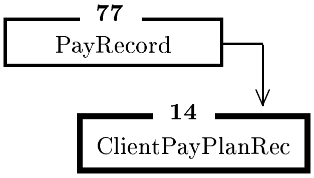
Banker's table fields' names

14

class **ClientPayPlanRec** : public PayRecord

Client Payment Plan Record

Inheritance



Public Members

Protected Members

attributes

Client Payment Plan Record.

An auxiliary class for one Payment record in Client.

See Also: Banker (→120, *page 355*)
Author: Ivana

14.1 Void **Clear** ()

Clear data

Clear data.

Clears data - including eCode.

Return Value: void
See Also: Banker::StorePayPlan()
Author: Ivana

14.2 **ClientPayPlanRec** ()

Client Paymen Plan Record constructor

Client Paymen Plan Record constructor.

Client Paymen Plan Record constructor initializes variables.

See Also: Banker::StorePayPlan()
Author: Ivana

14.3

Err **FetchFromRecord** (Record *r)

Fetches from given record

Fetches from given record.

Fetches data from given record. A virtual method, that should be overwritten, to be able to read from table.

Return Value: error code
Parameters: `r` — record from which we take data
See Also: PayRecord (→77, *page 256*)
Author: Ivana

14.4

Err **FillRecord** (Record *r)

Fills given record

Fills given record.

Fills given record with data. A virtual method, that should be overwritten, to be able to write to or update table.

Return Value: error code
Parameters: `r` — record into which we store data
See Also: PayRecord (→77, *page 256*)
Author: Ivana

14.5

MsgField* **GenerateOAuth** (Size lengthOfAuth)

Generate OAuthentification

Generate OAuthentification .

Generates OAuthentification - a random string used as a secret key between Client and Bank.

Return Value:	generated oAuth
Parameters:	`lengthOfAuth` — length of generated OAuth
See Also:	Banker::StorePayPlan()
Author:	Ivana

14.6

Char* **GetAmount** ()

Get Amount

Get Amount.

Returns our amount field in new string.

Return Value:	copy of amount
See Also:	
Author:	Ivana

14.7

GMessage* **GetBankAC** ()

Get bank's access certificate

Get bank's access certificate.

Returns bank's access certificate field in new gMessage.

Return Value: copy of bankAC
See Also:
Author: Ivana

14.8

ACID **GetBankACID** ()

Get bank's access certificate ID

Get bank's access certificate ID.

Returns bank's access certificate ID field.

Return Value: bankACID
See Also:
Author: Ivana

14.9

Err **SetMAC** (MsgField *mac)

Set MAC

Set MAC.

Sets our MAC to given one.

Return Value: error code
Parameters: `mac` — new mac for setting our data

See Also: Banker::StorePayPlan()
Author: Ivana

14.10

Void **SetTID** (TID tId)

Set transaction ID

Set transaction ID.

Sets transaction ID to given TID

Return Value: void
Parameters: `tId` — given transaction ID
See Also: Banker::StorePayPlan()
Author: Ivana

14.11

~ClientPayPlanRec ()

Client Paymen Plan Record destructor

Client Paymen Plan Record destructor.

Client Paymen Plan Record destructor deletes variables.

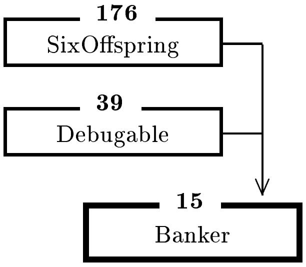
See Also: Banker::StorePayPlan()
Author: Ivana

15

class **Banker** : public Debugable, SixOffspring

Banker

Inheritance



Public Members

Protected Members

Banker.

A class which manages everything about payments in Client: 1) For each payment (in payment plan sent by Eso) generates OAuth and counts MAC. 2) New payment plan sends to given banks (along with money).

See Also: Client (→17, *page 100*)
Author: Ivana

15.2 methods

Names

Err	**DecideAcceptPPlan** (MsgField *pPlanFld)	*decides wheather client accepts sent payment plan*
Err	**MakePPlanForBanks** (MsgField *pPlanFld)	*finishes payment plan (generates oAuths and count MACs) for bank*
Queue*	**PrepareMsgsForBanks** (TID oldTID)	*prepares msgs for banks*

15.3

Void **AnswerFromBank** (GMessage *inMsg)

Reacts to an answer from bank

Reacts to an answer from bank.

If everything is OK, writes to log file and deletes records in temporary table. Otherwise writes to Not Paid Pays table and warn administrator of the eso.

Return Value: void
Parameters: `inMsg` — message (answer) from bank
See Also: Majordomo::DRCAnswerFromBank()
Author: Ivana

15.4

Void **AnswerKO** (CurrPay *p, Char *reason)

Reaction to a 'KO answer' from bank

Reaction to a 'KO answer' from bank.

Reacts to an answer from bank. Something is wrong - write to notPaidPays and logFile, deletes the record in current pays.

Return Value:	void
Parameters:	`p` — current payment structure `reason` — reason of fail
See Also:	AnswerFromBank()
Author:	Ivana

15.5

Err **AnswerOK** (CurrPay *p, GMessage *inMsg)

Reaction to a 'OK answer' from bank

Reaction to a 'OK answer' from bank.

Reacts to an answer from bank. Success is OK - bank knows the payment, MAC and SAuth are correct.

Return Value:	error code
Parameters:	`p` — current payment structure `inMsg` — message from bank
See Also:	AnswerFromBank()
Author:	Ivana

15.6

Banker (LogFile *logFile, Client *client)

Banker constructor

Banker constructor.

Banker constructor creates (and clears) bankers' persistent tables.

Parameters: `logFile` — pointer to log file
`client` — pointer to main class client
See Also: Client (→17, *page 100*)
Author: Ivana

15.7

include . H banker.h **Banker** (LogFile *logFile, Eso *eso)

Banker Constructor

Banker Constructor.

Banker Constructor creates bankers' persistent tables.

Parameters: `logFile` — pointer to log file
`eso` — pointer to main class eso
See Also: Eso (→131, *page 375*)
Author: Ivana

15.8

GMessage* **ChallengeFromBank** (GMessage *inMsg, TID tId, ACID *bankACID)

Reacts to a challenge from bank

Reacts to a challenge from bank.

As an answer to eso's 'request for payment' bank sends challenge. If something goes wrong (bank does not know the payment, eso requests for payment too early,...) bank does not send the key for computing MAC. If everything is OK (bank knows the payment and time is OK), an OAuthentification is in challenge. On this OAuth and stored client's file banker computes MAC. Then eso send this MAC along with SAuthentification as a 'response to challenge' to bank.

Return Value: message (response to challenge) for bank
Parameters: `inMsg` — message (challenge) from bank
`tId` — current transaction ID
`bankACID` — ID of bank's access certificate (return parametr)
See Also: Majordomo::DRCChallengeFromBank()
Author: Ivana

15.9

Void **ChallengeKO** (CurrPay *p, Char *reason)

Reaction to a 'KO challenge' from bank

Reaction to a 'KO challenge' from bank.

Reacts to a challenge from bank. Something is wrong - write to notPaidPays and logFile.

Return Value: void
Parameters: `p` — current payment structure
`reason` — reason of fail
See Also: ChallengeFromBank()
Author: Ivana

15.10

GMessage* **ChallengeOK** (CurrPay *p, GMessage *inMsg)

Reaction to a 'OK challenge' from bank

Reaction to a 'OK challenge' from bank.

Reacts to a challenge from bank; success is OK - bank knows the payment it counts MAC and writes it (with OAuth) to CurrPays.

Return Value:	message for bank
Parameters:	`p` — current payment structure
	`inMsg` — message from bank
See Also:	ChallengeFromBank()
Author:	Ivana

15.11

Void **ChallengeWrongTime** (CurrPay * , GMessage *)

Reaction to a 'BadTime challenge' from bank

Reaction to a 'BadTime challenge' from bank.

Reacts to a challenge from bank. Bank has different time - warn synchronizer (not implemented now).

Return Value:	void
Parameters:	`p` — current payment structure
	`inMsg` — message from bank
See Also:	ChallengeFromBank()
Author:	Ivana

15.12

Char* **CountPrice** (Size , RelTime)

Counts price for stored data

Counts price for stored data.

From size and timeToStore counts price for stored data. Returns it as a string.

Return Value:	amount of money
Parameters:	`size` — size of stored file
	`timeToStore` — period of storrage of sent file
See Also:	MakePayPlan()
Author:	Ivana

15.13

Err **CreateBankPPlan** ()

Create BankPPlan Table

Create BankPPlan Table.

Creates table BankPPlan - a persistent Banker's table.

Return Value:	error code
See Also:	Banker()
Author:	Ivana

15.14

Err **CreateClientPPlan** ()

Creates Client Payment Plan table

Creates Client Payment Plan table.

Creates (or connects to exist.) Client Payment Plan table. This table is common to eso and client so it calls CreateEsoToClientTable() function to create it.

Return Value: error code
Parameters: `void` —
See Also: Banker(), CreateEsoToClientTable()
Author: Ivana

15.15

Err **CreateCurrPays** ()

Creates Current Payments table

Creates Current Payments table.

Creates (or connects to exist.) Current Payments table.

Return Value: error code
Parameters: `void` —
See Also: Banker()
Author: Ivana

15.16

Err **CreateNotPaidPays** ()

Creates Not Paid Payments table

Creates Not Paid Payments table.

Creates (or connects to exist.) Not Paid Payments table.

Return Value: error code
Parameters: `void` —
See Also: Banker()
Author: Ivana

15.17

Err **CreatePPlans** ()

Creates Payment Plan table

Creates Payment Plan table.

Creates (or connects to exist.) Payment Plan table.

Return Value:	error code
Parameters:	`void` —
See Also:	Banker()
Author:	Ivana

15.18

include . H banker.h Err **CreatePayPlan** ()

Create PayPlan Table

Create PayPlan Table.

Creates table PayPlan - a persistent Banker's table.

Return Value:	error code
See Also:	Banker()
Author:	Ivana

15.19

Err **CreatePayments** ()

Creates Payments table

Creates Payments table.

Creates (or connects to exist.) Payments table.

Return Value: error code
Parameters: `void` —
See Also: Banker()
Author: Ivana

15.20

Err **DecideAcceptPPlan** (MsgField *)

Decide Wheather To Accept Payment Plan

Decide Wheather To Accept Payment Plan.

Decides wheather client accepts sent payment plan or not. (Now agrees with everything, later it should accept client's decision.) If sent payment plan is NULL, Eso store client's file for free.

Return Value: OK/KO
Parameters: `pPlanFld` — msgField of payment plan
See Also: Majordomo::StoreFile()
Author: Ivana

15.21

Err **Rollback** (TID tId)

Rollback transaction

Rollback transaction.

We don't have to cancel bank transactions, because they have TimeOuts. After some time they will be resent. We only must rollback generated payment plan (if any). TID is used as FileID!.

Return Value:	error code
Parameters:	`tId` — ID transaction to be rollbacked
See Also:	
Author:	Ivana

15.22

Err SchedulePays (TID tId, ID fileId)

Schedule Payments with given TID

Schedule Payments with given TID.

Client sent file and eso stored it. So the banker has to schedule payments. For each payment record from table generatedPayPlans with given TID: first, writes it to table payments; second, set time out for it.

Return Value:	error code
Parameters:	`tId` — transaction ID corresponding to the payment plan `fileId` — ID od file, which has been stored
See Also:	Majordomo::DRCDataToStore()
Author:	Ivana

15.23

Err SetCurrPayTimeOut (ID payId, Int state)

Sets current payment time out

Sets current payment time out.

Sets time out of current payment to specified time.

Return Value:	error code
Parameters:	`payId` — current payment ID
	`state` — state of payemnt (REQUEST/ANSWER)
See Also:	GenerateRequest(), GenerateAnswer()
Author:	Ivana

15.24

Err **SetPaymentTimeOut** (ID payId, ESTime time)

Sets payment time out

Sets payment time out.

Sets time out on the date the payment should be carried out.

Return Value:	error code
Parameters:	`payId` — payment ID
	`time` — timeOut time
See Also:	SchedulePays()
Author:	Ivana

15.25

Err **StorePayPlan** (Table *esoPPlan, TID tId)

Stores payments from given table

Stores payments from given table.

For each record from table esoPPlan generates OAuth and compute MAC. Then writes the record to table payPlan.

Return Value: error code
Parameters: `esoPPlan` — table of payment plan sent by Eso
`tId` — current transaction ID
See Also: MakePPlanForBanks()
Author: Ivana

15.26

GMessage* **TimeForPayment** (GMessage *inMsg, TID tId, ACID *bankACID)

Time For Some Payment is up

Time For Some Payment is up.

When time of some payment is up, finds payment record with given payId, generates request for bank including payment ID, sets timeout of current payment and fills in table currentPayments.

Return Value: message (request) for bank
Parameters: `inMsg` — message from scheduler
`tId` — current transaction ID
`bankACID` — ID of bank's access certificate (return parametr)
See Also: Majordomo::DRCTimeForPayment()
Author: Ivana

15.27

GMessage* **TimeOutCurrPay** (CurrPay *p)

TimeOut of some current payment

TimeOut of some current payment.

TimeOut of some current payment has gone.

Return Value: message for bank or NULL
Parameters: `p` — current payment structure
See Also: TimeOutPayment()
Author: Ivana

15.28

GMessage* **TimeOutPayment** (GMessage *inMsg, TID tId, ACID *bankACID, Int *request)

Reacts to a timeout of some payment

Reacts to a timeout of some payment.

When time of some current payment is up, finds payment record with given payId. If number of tries to get money from bank is still under limit, generates request or answer (depend on 'state' field) for bank and again sets timeout.

Return Value: message for bank
Parameters: `inMsg` — message from scheduler
`tId` — current transaction ID
`bankACID` — ID of bank's access certificate (return parametr)
`request` — wheather state is request or answer; parametr can have values YES or NO (retrun parametr)

See Also: Majordomo::DRCTimeOutPayment()
Author: Ivana

15.29

˜Banker ()

Banker destructor

Banker destructor.

Banker destructor deletes (and clears) bankers' persistent tables.

See Also: Client (→17, *page 100*)
Author: Ivana

15.1 attributes

Names

Table* **payPlan** *Bankers' persistent tables*

16

enum **CommandOption**

Command to perform

Command to perform.

This enum specifies the action to be taken - some test methods and Search upload or download

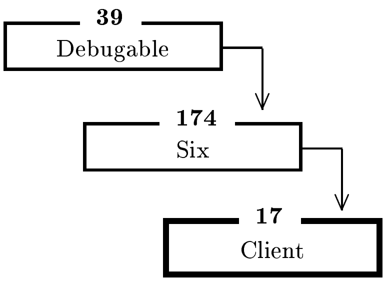
See Also: Client
Author: Marek Pavel Pechy Venca

17

class **Client** : public Six

Client main class

Inheritance



Public Members

		attributes		
		methods		
17.1		**Client** (string configName)	*Constructor*	101
17.2	void	**CreateMajordomo** ()	*Creates Majordomo*	101
17.3	Err	**InitializeRandomGenerator** ()	*Initializes random generator*	102
17.4		**˜Client** ()	*Destructor of Client*	102

Protected Members

methods

Client main class.

Consists of severaal objects that together perform all the actions necessary to make a user interface to Eternity service.

See Also: Six Eso Bank
Author: Marek Pavel Pechy Venca

17.1

Client (string configName)

Constructor

Constructor.

Creates all the objects necessary to run.

Parameters: `configName` — name of config flie
See Also: client Banker, client Majordomo
Author: Pavel

17.2

void **CreateMajordomo** ()

Creates Majordomo

Creates Majordomo.

Overrides inherited CreateMajordomo. Creates client type of majordomo.

Return Value: void
See Also: Majordomo from Client, SixMajordomo
Author: Pavel

17.3

Err **InitializeRandomGenerator** ()

Initializes random generator

Initializes random generator.

We suppose that we will use only random() function, because we seed with srandom(). We use /dev/urandom, but /dev/random would be better, but for simplicity and debugging purpose this is enough.

Return Value: error code
See Also: RandomGenerator
Author: Pechy

17.4

~Client ()

Destructor of Client

Destructor of Client.

Deletes all objects necessary to delete.

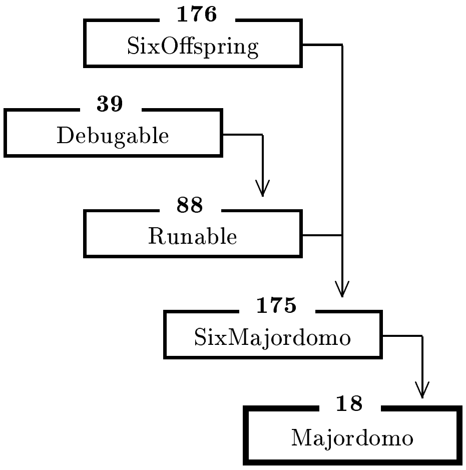
See Also: Banker
Author: Pavel

18

class **Majordomo** : public SixMajordomo

Client majordomo

Inheritance



Public Members

18.2		**methods**	106
18.3	Err	**AskForEsoCertificates** () *Asks for Eso certificates*	107
18.4	Err	**CommandCertificatesReceived** (MsgField *dataFld) *Processes CMD_CERTIFICATES_RECEIVED command*	107
18.5	Err	**CommandControllACCount** (MsgField *dataFld) *Serves TO_CHECK_AC_COUNT command*	108
18.6	include . .. .. Common H labels.h Err		

		GetPtrToACManager (Char *)	*Returns always NULL*	114
18.19	ESTime*	**GetTime** ()	*Gets current time*	114
18.20	Err	**HandleGeneratedAC** (GMessage *generatedAC)	*Saves the generated AC to AC-Manager*	115
18.21	Err	**Initialize** ()	*Initializes Bank Majordomo*	115
18.22	Err	**InsertToOutMsgQueue** (GMessage *msg)	*Inserts message to outMsgQueue*	115
18.23	void	**PublishCertificates** (GMessage * request)	*Publishes certificates to Acs servers*	116
18.24	void	**RequestStorage** (ArrayGMessage *certificates, File *fileToStore)	*Asks Esos to store the file*	116
18.25	void*	**Run** (void *)	*Acs majordomo Run() method*	117
18.26	MsgField*	**UnsealSAuth** (const ACID myACID, const MsgField *sealSA)	*Unseals sAuth*	118
18.27	void	**UploadFile** ()	*Uploads file*	118
18.28	void	**WaitForAnswersToFileToStor** (int requestsSent)	*Waits for answers to CE_FILE_TO_STOR*	119
18.29	void	**WaitForAnswersToReqForFile** ()	*Waits for answers to REQ_FOR_FILE*	119

Protected Members

		attributes		
18.1		**methods**		119

Client majordomo.

Implements all the tasks for communication with Eso Mix Bank and is an expert in protocols used between these entities.

See Also: Mix Client Eso Bank

Author: Marek, Pavel, Pechy, Venca

18.2 methods

Names

Majordomo (Client * parent, LogFile * logFile)
Constructor

virtual **~Majordomo** () *destructor*

local methods

commands

data_received commands

Helper functions for commands

Tests and not finished versions of commands

18.2.1

overriden methods

Names

virtual void*
Run (void *) *Main thread method that handles all requests*

virtual ACManager*
GetPtrToACManager (Char *type)
Returns the AC Manager for the specified type.

virtual Queue*

GetACs (Char *type, const int count = 1,
Queue *notIn = NULL)
Get access certificates of specified type

18.3

Err **AskForEsoCertificates** ()

Asks for Eso certificates

Asks for Eso certificates.

Generates message for missing ACs and sends it to Mix.

Return Value: error code
See Also: Six, SixMajordomo
Author: Marek

18.4

Err **CommandCertificatesReceived** (MsgField *dataFld)

Processes CMD_CERTIFICATES_RECEIVED command

Processes CMD_CERTIFICATES_RECEIVED command.

Processes CMD_CERTIFICATES_RECEIVED command by putting received certificates to appropriate ACManager.

Return Value: error code
Parameters: `dataFld` — data comming along with command
See Also: InsertAC()
Author: Marek

18.5

Err **CommandControllACCount** (MsgField *dataFld)

Serves TO_CHECK_AC_COUNT command

Serves TO_CHECK_AC_COUNT command.

Processes timeout command for controlling count of ACs.

Return Value:	error code
Parameters:	`dataFld` — data comming with timeout
See Also:	ACManager, Scheduler, GenerateRequestForMissingACs(), CreateNextACTimeOutMsg()
Author:	Marek, Pavel

18.6

include . .. .. Common H labels.h Err **CommandDelFinderTbl** (MsgField *dataFld)

Serves the TO_DEL_FINDER_TBL command

Serves the TO_DEL_FINDER_TBL command.

Processes timeout command for deleting tables in Finder.

Return Value:	error code
Parameters:	`dataFld` — data comming with timeout
See Also:	Finder, Scheduler
Author:	Marek, Pavel

18.7

Err **DRCAnswerFile** (const MsgField *dataField)

Serves CMD_ANSWER_FILE command

Serves CMD_ANSWER_FILE command.

processes the incoming found file - forwards found file to client

Return Value: error code
Parameters: `dataField` — encrypted request comming with message
`acid` — ID of this Eso access certificate which was used to address thid message
See Also: TCBWrapper, Finder
Author: Marek

18.8

Err **DRCAnswerFromBank** (MsgField *dataInMsgFld)

DRCAnswerFromBank (cmd be_reply_to_resp) Dlouhy_komentar

18.9

Err **DRCAnswerFromEso** (MsgField *dataFld)

Serves Eso's answer to reply to its request for money

Serves Eso's answer to reply to its request for money.

Compares if Eso sent the same MAC as Bank holds for requested payment.

Return Value: error code
Parameters: `dataFld` — data comming along with command

See Also:
Author: Ivana

18.10

include . H majordomo.h MsgField* **DecryptByACPriv** (const ACID acid, const MsgField *eData)

Decrypts data with access private key

Decrypts data with access private key.

Identifies proper private key by access certificate ID and then decrypts given data. Decrypted data are returned.

Return Value: co
Parameters: `acid` — ID of access certificate which private key is to be used for decrypting process
`eData` — Encrypted data
See Also: SixMajordomo, KeyManager, Cipherer
Author: Marek

18.11

MsgField* **DecryptByACPriv** (const ACID acid, const MsgField *data)

Decrypts data by asymetric key

Decrypts data by asymetric key.

Decrypts data by asymetric key that is saved in a file.

Return Value: decrypted MsgField
Parameters: `acid` — ID of certificate associated with the key.
`data` — Data to decrypt.

See Also: Cipherer (→34, *page 150*)
Author: Marek

18.12

void **DownloadFile** ()

Downloads file from Esos

Downloads file from Esos.

Tries to find files according to FindFileID given by user in command line.

Return Value: void
See Also:
Author: Marek

18.13

void **FindHeaders** (ArrayGMessage *certificates)

Tries to find headers on Esos

Tries to find headers on Esos.

Creates request message, sends it to given number of Esos and waits for reply.

Return Value: void
Parameters: `certificates` — Certificates of Esos to ask.
See Also:
Author: Marek

18.14

include . H majordomo.h Err **GenerateACID** (ACID *acid)

Generates ID of certificate

Generates ID of certificate.

Generates random string of the length of ACID.

Return Value:	error code
Parameters:	`acid` — Generated acid.
See Also:	RandomGenerator (→78, *page 262*)
Author:	Marek

18.15

Err **GenerateACKeys** (ACID *acid, GMessage **acPubKey)

Generates certificate key pair

Generates certificate key pair.

Generates new access certificate key pair.

Return Value:	error code
Parameters:	`acid` — Generated ID of newly generated access certificate `acPubKey` — Public key of generated key pair
See Also:	Bank, SixMajordomo, Cipherer
Author:	Marek

18.16

Err **GenerateAsymetricKey** ()

Generates public and private keys

Generates public and private keys..

Calls the cipherer's method to generate public and private keys. Saves them in files.

Return Value:	error code
See Also:	Cipherer (→34, *page 150*)
Author:	Marek

18.17

ACManager* **GetPtrToACManager** (Char *type)

Returns pointer to ACManeger

Returns pointer to ACManeger.

Returns pointer to ACManager of given type, but does not create a copy of this manager, so DON'T EVER DELETE this value. All ACManagers are owned by Bank.

Return Value:	ACManager of given type
Parameters:	`type` — one of ACTYPE_... values identifying ACManager
See Also:	Bank, SixMajordomo, ACManager
Author:	Marek

18.18

ACManager* **GetPtrToACManager** (Char *)

Returns always NULL

Returns always NULL.

This is an overriden Six method. Because Client has no ACManagers, this method always returns NULL.

Return Value:	NULL
Parameters:	`char*` — - not used
See Also:	SixMajordomo (→175, *page 522*)
Author:	Marek

18.19

ESTime* **GetTime** ()

Gets current time

Gets current time.

Uses ESTime method to check current time.

Return Value:	error code
See Also:	
Author:	Marek

18.20

Err **HandleGeneratedAC** (GMessage *generatedAC)

Saves the generated AC to ACManager

Saves the generated AC to ACManager.

This is an overriden method from Six. In Bank it is necessary to set generated AC to ACManager.

Return Value:	error code
Parameters:	`generatedAC` — Generated AC to handle.
See Also:	Eso::HandleGeneratedAC(), Client::HandleGeneratedAC()
Author:	Marek

18.21

Err **Initialize** ()

Initializes Bank Majordomo

Initializes Bank Majordomo.

Certain (but really COOL) initialization.

Return Value:	error code
See Also:	
Author:	Marek

18.22

Err **InsertToOutMsgQueue** (GMessage *msg)

Inserts message to outMsgQueue

Inserts message to outMsgQueue.

Puts the message to queue that is periodically feeded to majordomoToSender MessageQueue, which is then emptied to Mix.

Return Value: error code
Parameters: `msg` — message to be sent
See Also: SixMajordomo (→175, *page 522*)
Author: Marek

18.23

void **PublishCertificates** (GMessage * request)

Publishes certificates to Acs servers

Publishes certificates to Acs servers.

Every server using Mixes probably wants to advertise itself, so this method publishes certificates it gets.

Return Value: void
Parameters: `request` — <pre> GM_PARAMETERS GM_CERTIFICATE_1 GM_CERTIFICATE_2 ... GM_CERTIFICATE_"n" </pre>
See Also: Acs, CertificateRepository
Author: Venca

18.24

void **RequestStorage** (ArrayGMessage *certificates, File *fileToStore)

Asks Esos to store the file

Asks Esos to store the file.

Contact all these Eso's and ask if it is possible to store file

Return Value:	void
Parameters:	`certificates` — Certificates of Esos to ask. `fileToStore` — file to try to store
See Also:	
Author:	Pavel

18.25

void* **Run** (void *)

Acs majordomo Run() method

Acs majordomo Run() method.

Gets message with the following structure: <pre> Command: data_received Parameters: Data: Command: get_certificates Request-ID: ... Mix's request ID Parameters: Onion: Certificate-Type: Count: Request-ID: </pre>

or

<pre> Command: data_received Parameters: Data: Command: publish_certificates Parameters: Certificate-1: Certificate-2: ... Certificate-n: </pre>

And sends back a reply: <pre> Command: send_data Parameters: Add-Hops: Onion: Data: Command: certificates_received Request-ID: ... Mix's request ID Parameters: Request-ID: Certificates: Certificae-1: Certificae-2: ... Certificae-n: </pre>

Return Value:	(void *)NULL - no result is returned
See Also:	Majordomo, GetCertificates(), PublishCertificates()
Author:	Venca

18.26

MsgField* **UnsealSAuth** (const ACID myACID, const MsgField *sealSA)

Unseals sAuth

Unseals sAuth.

Given seald sAuth returns unsealed one using DecryptByACPriv.

Return Value:	error code
Parameters:	`myACID` — ID of access certificate which public key was used to seal sAuth `sealSA` — Sealed sAuth
See Also:	DecryptByACPriv()
Author:	Marek

18.27

void **UploadFile** ()

Uploads file

Uploads file.

Tries to upload file on number of Esos specified by user. Upload is made in two phases. The first phase is request for allocation. The file is then sent to the Esos, whose reply is positive.

Return Value:	void
See Also:	
Author:	Pavel

18.28

void **WaitForAnswersToFileToStor** (int requestsSent)

Waits for answers to CE_FILE_TO_STOR

Waits for answers to CE_FILE_TO_STOR.

Waits for answers to CE_FILE_TO_STOR in a loop until a timeout is over.

Return Value: void
Parameters: `requestSent` — Number of messages for wait for.
See Also:
Author: Pavel

18.29

void **WaitForAnswersToReqForFile** ()

Waits for answers to REQ_FOR_FILE

Waits for answers to REQ_FOR_FILE.

Waits for answers for number of seconds given in ConfigFile.

Return Value: void
See Also:
Author: Marek

18.1

methods

Names

virtual Err
SwitchDataReceivedCommand (const Char *command, const MsgField *dataFld, const ACID acid)
Handle data received command

virtual Err

GenerateThisSixAC (MsgField *onion,
GMessage **generatedAC)
generating client's AC

Err **GenerateACID** (ACID *acid)
Generates uniqueu access certificate ID

Err **GetAsymetricKey** (GMessage **acPubKey)
returns Acs asymetric key

Err **GenerateAsymetricKey** ()
Generates asymetric key for making access certificate

virtual MsgField*
DecryptByACPriv (const ACID acid,
const MsgField *dData)
Decrypts data by Acs' private key

19

#define **CLIENT_CONFIG_FILE**

default config file, if not set by '-f' option

20

@name Global parameters for client

21

int **main** (int argc, char ** argv)

Client main function

Client main function.

Parses command line options and sets global variables according to switches specified It also creates Client's main object and executes his method Run()

See Also: Client, Client::Run()
Author: Pavel, Marek,Venca

22

void **printHelp** ()

Prints usage instructions

Prints usage instructions.

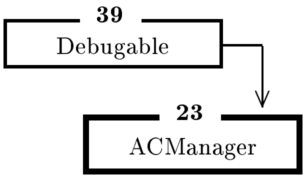
See Also: co_se_toho_tyka
Author: Pavel Marek

23

class **ACManager** : public Debugable

ACManager

Inheritance



Public Members

Protected Members

ACManager.

encapsulates the table of Access certificates

See Also:
Author: Marek

23.2 methods

Names

ACManager (const Char *aType,
Char* aTableName,
const Int minACCountP = 1,
const Int maxACCountP = 1,
LogFile *aLogFile = NULL,
const RelTime *countTestP = NULL,
const RelTime *gotCertifTestP = NULL)
Constructor

virtual **~ACManager** () *destructor*

Err **InsertAC** (GMessage* ac)
inserts AC into the table

Queue* **GetACs** (const Int count = 1,
Queue *notIn = NULL)

		returns queue of [count] GMessages (ACs), that are not in [notIn] queue if there are less then count
GMessage*	**GetACByACID** (ACID acID)	*finds the AC according to its ACID*
ESTime	**NextTimeOut** ()	*tells the Scheduler to set the timeout*
Err	**TestACCount** (Int *countOfMissingACs)	*controlling of count of ACs at timeout returns a GMessage for MIX, if it is necessary to get new ACs or NULL if number of ACs is between minACCount and maxACCount*

23.3

Err **ACCount** (Int *count)

Get count of certificates

Get count of certificates.

Returns the current count of certificates in the manager - used to keep the number in the range specified on creation of AC manager.

Return Value:	Error code
Parameters:	`count` — out parameter that gets filled with the count of certificates currently in AC manager.
See Also:	ACManager()
Author:	marek

23.4

ACManager (const Char *aType, Char* aTableName, const Int minACCountP, const Int maxACCountP, LogFile *aLogFile, const RelTime *countTestP, const RelTime *gotCertifTestP)

Access certificate manger constructor

Access certificate manger constructor.

Takes care of certificates of certain type. Keeps the number between specified ranges and downloads new certificates if necessary.

Parameters:
- `aType` — Type of certificates to store.
- `aTableName` — Table where to store the certificates.
- `minACCountP` — Minimum number of certificates to keep.
- `maxACCountP` — Maximum number of certificates to keep.
- `aLogFile` — Log file for debug messages.
- `countTestP` — Time to wait before checking if we have enough certificates
- `gotCertifTestP` — Time to wait before checking if certificates arrived

See Also: Six (→174, *page 514*)

Author: marek

23.5

TableFieldQueue* **CreateDefinition** ()

Creates a definition of Table

Creates a definition of Table.

This table is then used for storage of access certificates of type specified in constructor.

Return Value:	Array of fields that will be in the queue.
See Also:	ACManager(), Table
Author:	marek

23.6

GMessage* **GetACByACID** (ACID acID)

Finds and returns the AC according to its ACID

Finds and returns the AC according to its ACID.

Searches the internal table of certificates for an access certificate with the specified access certificate ID.

Return Value:	Access certificate that matches the query.
Parameters:	`acID` —
See Also:	Six (→174, *page 514*)
Author:	marek

23.7

Queue* **GetACs** (const Int count, Queue *notIn)

Get access certificates

Get access certificates.

Gets count ACs, that are not in notIn Queue if there are less then count of such of them, returns less.

Return Value: List of certificates that matchthe query.
Parameters: `count` — Number of certificates to return.
`notIn` — List of forbidden certificates.
See Also: Six (→174, *page 514*)
Author: marek

23.8

Err **InsertAC** (GMessage* ac)

Insert Access certificate

Insert Access certificate.

Inserts a GMessage into the table (or rewrites old record with same ACID)

Return Value: Error code
Parameters: `ac` — Access certificate that should be inserted.
Author: marek

23.9

ESTime **NextTimeOut** ()

Returns time of next timeout

Returns time of next timeout.

Works around scheduler for now - finds out the time of next timeout.

Return Value: Time when the timeout should occur.
See Also: Scheduler (→139, *page 405*)
Author: marek

23.10

Err **TestACCount** (Int *countOfMissingACs)

Finds out how many certificates are missing

Finds out how many certificates are missing.

Controlling the count of ACs at timeout returns a GMessage for mix, if necessary to get certificates.

Return Value:	Error code
Parameters:	`countOfMissingACs` — out parametr taht is filled with number of necessary access certificates.
See Also:	Tests if the access certificates count is in the range specified.
Author:	marek

23.11

~ACManager ()

Access certificates manager destructor

Access certificates manager destructor.

Deletes all certificates from memory.

See Also:	ACManager(), Table
Author:	marek

23.1

methods

Names

TableFieldQueue*
CreateDefinition ()
creates definition of acTable - used only in constructor

Err **ACCount** (Int *count)
returns count of not expired ACs, deletes expired ACs

24

typedef multimap < string, GMessage*, less<string> > **Certificates**

Array of certificates

Array of certificates.

Contains Certificate pairs of 'type -> Certificate' that allows for searching for certificates of specified type

25

typedef multimap < string, GMessage*, less<string> > iterator **CertificateIterator**

Certificates iterator

26

typedef multimap < string, GMessage*, less<string> > value_type **CertificateValue**

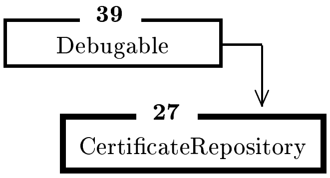
pair 'type -> Certificate'

27

class **CertificateRepository** : public Debugable

Repository for various certificates

Inheritance



Public Members

Protected Members

Repository for various certificates.

Stores certificates in memory and is able to seve them to disk and again load them. It is also able to return certificates of desired type.

See Also: AddressManager MessageCreator
Author: Venca

27.2 methods

Names

CertificateRepository (ConfigFile * cmConfigFile, LogFile *cmLogFile)
constructor

CertificateRepository (ConfigFile * cmConfigFile, LogFile *cmLogFile, set<string, less<string> > allowedTypes)
constructor

GMessage*

GetCertificates (Size count, string type)
Return certificates of specified type

GMessage*
GetCertificate (string type)
Return certificate of specified type

void **PutCertificates** (GMessage *certifcates)
Insert certificates into repository

bool **IsAllowedType** (string type)
Is allowed to store in repository

Err **LoadCertificates** ()
Load certificates from disk

Err **SaveCertificates** ()
Save certificates to disk

void **DeleteCertificates** ()
Free certificates from memory

Err **ClearDirectory** (string dir)
Delete certificates in certificates directory

27.3

CertificateRepository (ConfigFile * cmConfigFile, LogFile *cmLogFile)

Certificate repository constructor

Certificate repository constructor.

Certificate repository stores different types of certificates and is able to return certificates of specified type. Certificates are stored internally in a STL container called map that is implemented like lookup tree so the lookups of certificates efficient.

Parameters: `cmConfigFile` — Configuration file.
`cmLogFile` — Log file for debug messages.

See Also: AddressManager, GetCertificate()
Author: Venca

27.4

CertificateRepository (ConfigFile * cmConfigFile, LogFile *cmLogFile, set<string, less<string> > allowedTypes)

Certificate repository constructor

Certificate repository constructor.

Certificate repository stores different types of certificates and is able to return certificates of specified type. Certificates are stored internally in a STL container called map that is implemented like lookup tree so the lookups of certificates efficient.
 This version of constructor takes an additional parameter that specifies what certificates can be stored in this instance of certificate repository.

Parameters:
`cmConfigFile` — Configuration file.
`cmLogFile` — Log file for debug messages.
`allowedTypes` — list (STL list) of types that can be stored in repository.

See Also: AddressManager, GetCertificate()
Author: Venca

27.5

Err **ClearDirectory** (string dir)

Clears certificates from directory

Clears certificates from directory.

deletes all the certificates stored in this directory (cert0 ... certn) Useful before writting new certificates.

Return Value: OK if successful KO otherwise
Parameters: `dir` — Directory to clear
See Also: SaveCertificates()
Author: Venca

27.6

void **DeleteCertificates** ()

Delete certificates from memmory

Delete certificates from memmory.

Goes through all the certificates stored in repository and deletes them.

Return Value: void
See Also: LoadCertificates(), ~CertificateRepository()
Author: Venca

27.7

GMessage* **GetCertificate** (string type)

Returns random certificate

Returns random certificate.

Return random certificate of type ”type”.

Return Value: Pointer to a new GMessage containing sertificate of type ’type’
Parameters: `type` — Type of certificate to return
See Also: PutCertificate(), AddressManager

Author: Venca

27.8

GMessage* **GetCertificates** (Size count, string type)

Returns a bunch of certificates

Returns a bunch of certificates.

Get count of random certificates with type "type". Internally implemented as count of calls to GetCertificate()

Parameters: `count` — Number of certificates to return.
`type` — Type of certificates we want.
See Also: GetCertificate(), PutCertificates()
Author: Venca

27.9

bool **IsAllowedType** (string type)

Is the type allowed to be stored in repository

Is the type allowed to be stored in repository.

Checks if the type is between those allowed types that were specified in constructor - default is allow all.

Return Value: true if allowed false otherwise.
Parameters: `type` — Certificate type to check.
See Also: PutCertificate(), PutCertificates()
Author: Venca

27.10

Err **LoadCertificates** ()

Load certificates from disk

Load certificates from disk.

From directory specified in config file reads number of certificates and then inserts all the certififcates into its search tree.

Return Value: OK if siccessful KO otherwise.
See Also: SaveCertificates()
Author: Venca

27.11

void **PutCertificates** (GMessage *data)

Inserts certificate into certificate repository

Inserts certificate into certificate repository.

Checks the type against its list of allowed types and if it is there then the certificate is stored.

Parameters: `data` — Type of certificate to return
See Also: GetCertificate(), AddressManager
Author: Venca

27.12

Err **SaveCertificates** ()

Save certificates to disk

Save certificates to disk.

Clears the directory specified in configuration file and then writes all the certificates in repository to separate files. It also writes a certificate Count in a special file.

Return Value:	OK if successful KO otherwise.
See Also:	LoadCertificates()
Author:	Venca

27.13

~CertificateRepository ()

Certificate repository destructor

Certificate repository destructor.

Frees all stored certificates from memmory.

See Also:	PutCertificate(), CertificateRepository()
Author:	Venca

27.1

attributes

Names

Certificates	**certificates**	*Stored certificates*
set < string, less<string> >	**allowedTypes**	*list of allowed types*
bool	**allowAllTypes**	*Do we allow everything*
ConfigFile*	**configFile**	*Configuration file*

28

#define **SYM_KEY_LEN**

We assume DES symetric encryption only for now, for simplicity.

29

#define **RSA_DEFAULT_MODULUS_BIT_LEN**

Default modulus len for whole Eternity Service for now.

30

#define **RSA_BLOCK_LEN** (((RSA_DEFAULT_MODULUS_BIT_LEN - 1)

Count RSA block length from RSA_DEFAULT_MODULUS_BIT_LEN

31

#define **RSAREF_RESERVED_BYTES**

According to PKCS1 standard, RSA block needs these bytes to be unused by user

32

Encryption flags for asymetric coding for using only one method for both dectryption and encryption

33

#define **BITS_IN_BYTE**

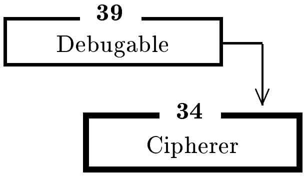
do not use magic numbers, please

34

class **Cipherer** : public Debugable

Class that handles encryption algorithms

Inheritance



Public Members

Private Members

Class that handles encryption algorithms.

It implements various algorithms like DES, RSA, MD5 ... All the data passed to it should be in a message field (MsgField). The data returned is also a message field. (data + length)

See Also: Translator Client Bank MessageCreator Eso
Author: Pechy

34.1 methods

Names

unsigned char **GenerateRandomByte** ()
Auxiliary public methods, not too efficient, but fully functional.

GMessage* **ReturnGMsgFullOfSymKeys** (Size hopCount, GMessage *params = NULL)
Auxiliary method for MessageCreator's methods.

34.1.1

General methods for asymetric encryption and decription, keys have type of the key information inside

General methods for asymetric encryption and decription, keys have type of the key information inside. For now, only RSA algorithm is implemented.

34.1.2

Hi-level methods for asymetric encryption of symetric keys

Hi-level methods for asymetric encryption of symetric keys. Symetric key can of any type, but symKey->SizeOf() can't be bigger than (RSA_BLOCK_LEN - RSAREF_RESERVED_BYTES).

See Also: cipherer.h

34.1.3

General methods for symetric encryption and decryption, keys have type of the key information inside

General methods for symetric encryption and decryption, keys have type of the key information inside. For now, only DES algorithm is implemented (because of RSAEuro toolkit).

34.1.4

General methods for symetric sealing and opening digital envelopes

General methods for symetric sealing and opening digital envelopes. For now, only RSA, DES and MD5 is implemented.

34.1.5

General methods for signing, verification and digesting

General methods for signing, verification and digesting. Only RSA and MD5 is implemented for now.

34.1.6

General methods for key generation of symetric and asymetric keys

General methods for key generation of symetric and asymetric keys. For now, only RSA and DES keys are implemented.

34.3

include . .. H common.h **Cipherer** (LogFile * ciphererLogFile)

Cipherer constructor

Cipherer constructor.

Cipherer constructor. For now, nothing is needed to run here (random generator is initialized in main.cc).

Parameters: `ciphererLogFile` — Log file.
See Also:
Author: Pechy

34.4

Err **DecryptBlockWithPrivateKey** (GMessage *gmPrivateKey, char *dataIn, Size dataInLen, char **dataOut, Size *dataOutLen)

Decrypts one PKCS1 block by private key

Decrypts one PKCS1 block by private key.

Decrypts one PKCS1 block by private key, the block is defined by PKCS1 standard. Inside we use RSAEuro function.

Return Value: KO if decryption fails, OK otherwise.

Parameters: `gmPrivateKey` — Private key.
`dataIn` — Input data octet stream.
`dataInLen` — Length of input data octet stream.
`dataOut` — Output data octet stream.
`dataOutLen` — Length of output data octet stream.
See Also: other encryption methods
Author: Pechy

34.5

Err **DecryptBlockWithPublicKey** (GMessage *gmPublicKey, char *dataIn, Size dataInLen, char **dataOut, Size *dataOutLen)

Decrypts one PKCS1 block by public key

Decrypts one PKCS1 block by public key.

Decrypts one PKCS1 block by public key, the block is defined by PKCS1 standard. Inside we use RSAEuro function.

Return Value: KO if decryption fails, OK otherwise.
Parameters: `gmPublicKey` — Public key.
`dataIn` — Input data octet stream.
`dataInLen` — Length of input data octet stream.
`dataOut` — Output data octet stream.
`dataOutLen` — Length of output data octet stream.
See Also: other encryption methods
Author: Pechy

34.6

Err **DecryptKeyWithPrivateKey** (GMessage *privKey, MsgField *eSymKey, GMessage **pSymKey)

Decrypts symetric key with private key

Decrypts symetric key with private key.

Due to efficiency and implementation, we use sealing, so this method decrypts a symetric key (of GMessage type) with private key.

Return Value:	KO if encryption failes, OK otherwise.
Parameters:	`privKey` — Private key. `eSymKey` — Symetric key to process. `pSymKey` — Decrypted symetric key to process.
See Also:	other cryptography methods.
Author:	Pechy

34.7

Err **DecryptWithPrivateKey** (GMessage * privateKey, MsgField * eData, MsgField ** pData)

Decrypts data with private key

Decrypts data with private key.

Decrypts data with private key. The data is in msgField, output is inserted in a msgField too. The msgField is freshly allocated.

Return Value: KO if an action fails, OK otherwise.
Parameters: `privateKey` — Private key.
`eData` — Encrypted data to process.
`pData` — Processed data.
See Also: other cryptography methods
Author: Pechy

34.8

Err **DecryptWithPublicKey** (GMessage * publicKey, MsgField * eData, MsgField ** pData)

Decrypts data with public key

Decrypts data with public key.

Decrypts data with public key. The data is in msgField, output is inserted in a msgField too. The msgField is freshly allocated.

Return Value: KO if an action fails, OK otherwise.
Parameters: `publicKey` — Public key.
`eData` — Encrypted data to process.
`pData` — Processed data.
See Also: other cryptography methods
Author: Pechy

34.9

Err **DecryptWithSymetricKey** (GMessage * symKey, MsgField * eData, MsgField ** dData)

Decrypt data using symetric key

Decrypt data using symetric key.

We use ,,fields", because it's quite easy to work with them in connection with gMessages. MsgField is abstract data type, so whenever we need create an instance of that, we use BytesMsgField. <p> Uses auxiliary method ProcessWithSymetricKey() defined in another module, in RSAEuro decrypting with DES is not the same as encrypting.

Return Value: KO if action fails, OK otherwise.
Parameters: `symKey` — What symetric key to use.
`eData` — Encrypted data
`pData` — Plain data to process.
See Also: other crypthographic methods
Author: Pechy

34.10

Err **DecryptWithSymetricKeys** (GMessage * symKeys, MsgField * eData, MsgField ** dData)

Decrypt multiple encrypted data using symetric keys

Decrypt multiple encrypted data using symetric keys.

When we use onions and chunks to make traffic analysis harder, we make a special peel for each hops the chunk goes through. When we make these chunks or process received data, we use encryption or decryption in a loop, each loop for one hop included.

Return Value: KO if action fails, OK otherwise.
Parameters: `symKeys` — What symetric keys to use.
`eData` — Encrypted data
`pData` — Plain data to process.
See Also: other crypthographic methods
Author: Pechy

34.11

Err **EncryptBlockWithPrivateKey** (GMessage *gmPrivateKey, char *dataIn, Size dataInLen, char **dataOut, Size *dataOutLen)

Encrypts one PKCS1 block by private key

Encrypts one PKCS1 block by private key.

Encrypts one PKCS1 block by private key, the block is defined by PKCS1 standard. Inside we use RSAEuro function.

Return Value: KO if encryption fails, OK otherwise.
Parameters: `gmPrivateKey` — Private key.
`dataIn` — Input data octet stream.
`dataInLen` — Length of input data octet stream.
`dataOut` — Output data octet stream.
`dataOutLen` — Length of output data octet stream.
See Also: other encryption methods
Author: Pechy

34.12

Err **EncryptBlockWithPublicKey** (GMessage *gmPublicKey, char *dataIn, Size dataInLen, char **dataOut, Size *dataOutLen)

Encrypts one PKCS1 block by public key

Encrypts one PKCS1 block by public key.

Encrypts one PKCS1 block by public key, the block is defined by PKCS1 standard. Inside we use RSAEuro function.

Return Value: KO if encryption fails, OK otherwise.
Parameters: `gmPublicKey` — Public key.
`dataIn` — Input data octet stream.
`dataInLen` — Length of input data octet stream.
`dataOut` — Output data octet stream.
`dataOutLen` — Length of output data octet stream.
See Also: other encryption methods
Author: Pechy

34.13

include . .. H common.h Err **EncryptKeyWithPublicKey** (GMessage *pubKey, GMessage *pSymKey, MsgField **eSymKey)

Encrypts symetric key with asymetric

Encrypts symetric key with asymetric.

Due to efficiency and implementation, we use sealing, so this method encrypts a symetric key (of GMessage type) with public key.

Return Value: KO if encryption failes, OK otherwise.
Parameters: `pubKey` — Public key.
`pSymKey` — Symetric key to process.
`eSymKey` — Encrypted symetric key to process.
See Also: other cryptography methods.
Author: Pechy

34.14

include . .. H common.h Err **EncryptWithPrivateKey** (GMessage * privateKey, MsgField * pData, MsgField ** eData)

Encrypts data with private key

Encrypts data with private key.

Encrypts data with private key. The data is in msgField, output is inserted in a msgField too. The msgField is freshly allocated.

Return Value:	KO if an action fails, OK otherwise.
Parameters:	`privateKey` — Private key.
	`pData` — Plain data to process.
	`eData` — Processed data.
See Also:	other cryptography methods
Author:	Pechy

34.15

Err **EncryptWithPublicKey** (GMessage * publicKey, MsgField * pData, MsgField ** eData)

Encrypt data with public key

Encrypt data with public key.

This method is not too clean, we count on the fact that RSAPublicEncrypt() function from RSAEuro encodes one chunk (ie. stream with current modulus in length) and symetric key's gMessage is short. <p> Currently we use 512 bits long modulus, so we can have symetric key's gMessages 64 bytes long (ie. gMessage converted to an octet stream can be maximally 64 bytes in length).

Return Value: Error code OK - KO
Parameters: `parametr` —
See Also: EncryptWithPrivateKey()
Author: Pechy

34.16

include . .. H common.h Err **EncryptWithSymetricKey** (GMessage * symKey, MsgField * pData, MsgField ** eData)

Encrypt data using symetric key

Encrypt data using symetric key.

We use ,,fields", because it's quite easy to work with them in connection with gMessages. MsgField is abstract data type, so whenever we need create an instance of that, we use BytesMsgField. <p> eData is encoded octet-stream createt from gMessage, ie. the whole gMessage is saved to an octet-stream, encoded and then BytesMsgField is initialized from it. <p> Uses auxiliary method ProcessWithSymetricKey() defined in another module, in RSAEuro decrypting with DES is not the same as encrypting.

Return Value: KO if action fails, OK otherwise.
Parameters: `symKey` — What symetric key to use.
`pData` — Plain data to process.
`eData` — Encrypted data.
See Also: other crypthographic methods
Author: Pechy

34.17

Err **EncryptWithSymetricKeys** (GMessage * symKeys, MsgField * pData, MsgField ** eData, int useAlsoKeyOne)

Encrypt data using multiple symetric keys

Encrypt data using multiple symetric keys.

When we use onions and chunks to make traffic analysis harder, we make a special peel for each hops the chunk goes through. When we make these chunks or process received data, we use encryption or decryption in a loop, each loop for one hop included.

Return Value:	KO if action fails, OK otherwise.
Parameters:	`symKeys` — What symetric keys to use.
	`pData` — Plain data.
	`eData` — Encrypted data to process.
See Also:	other crypthographic methods
Author:	Pechy

34.18

GMessage* **GenerateAsymetricKey** (GMessage *parameters = NULL)

Generate asymetric key pair

Generate asymetric key pair.

Generates one asymetric key pair, in parameters we can insert what type of key to generate. RSA key is generated by default (because of RSAEuro toolkit). For now, only RSA is supported

Return Value:	GMessage with generated asymetric key. <pre> GM_PUBLIC_KEY_INFO GM_PUBLIC_KEY - native structure from RSAEuro GM_ASYMETRIC_CIPHER_TYPE - ie. "RSA" GM_ASYMETRIC_CIPHER_VERSION - obsolete, still supported, not used GM_PRIVATE_KEY_INFO GM_PRIVATE_KEY - native structure from RSAEuro GM_ASYMETRIC_CIPHER_TYPE - ie. "RSA" GM_ASYMETRIC_CIPHER_VERSION - obsolete, still supported, not used <pre>

Parameters: `parameters` — What type of algorithm.
See Also: other crypthographic methods
Author: Pechy

34.19

char unsigned **GenerateRandomByte** ()

Very simple method for generation of one byte

Very simple method for generation of one byte.

Very simple method, uses standard random number generator, should be replaced in future by R_GenerateBytes(rsaeuro) and /dev/random at least. Relies * on the fact that somebody has already seeded generator with srandom(). For now, we seed srandom() with randomness got from /dev/urandom (it's good !)

Return Value: One random byte.
See Also: other cryptography methods.
Author: Pechy

34.20

include . .. H common.h GMessage* **GenerateSymetricKey** (GMessage *parameters = NULL)

Generate symetric key

Generate symetric key.

Generates one symetric key, in parameters we can insert what type of key to generate. DES key is generated by default (because of RSAEuro toolkit).

Return Value: GMessage Symetric key, NULL if generation failed: <pre> GM_SYMETRIC_KEY - ie. 8 random bytes of DES key GM_INITIALIZATION_VECTOR - ie. 8 random bytes of DES iv GM_SYMETRIC_CIPHER_TYPE - ie. "DES" by default </pre>
Parameters: `parameters` — What type of key to generate.
See Also: other crypthographic methods
Author: Pechy

34.21

include . .. H common.h Err **MessageDigest** (MsgField *dataToDigest, MsgField **dig, GMessage *parameters = NULL)

Creates a message digest over given data

Creates a message digest over given data.

Creates a message digest over given data. Inside, a RSAEuro function is used. We use only MD5 for now, but type of digest can be specified in parameters.

Return Value: KO if digest creation fails, OK otherwise.
Parameters: `dataToDigest` — Data to digest.
`dig` — Created digest.
`parameters` — Parameters - which digest to use.
See Also: other cryptographic methods
Author: Pechy

34.22

Err **OpenSealed** (GMessage *privateKeyInfo, GMessage *publicKeyInfo, MsgField *dataIn, MsgField **dataOut)

Opens sealed data

Opens sealed data.

Verifies whether given data is not tempered and returns it. For now, we use only RSA, DES and MD5 algorithms, but we can insert parameters with additional information which algs to use (in Seal() method).

Return Value: KO if opening fails, OK otherwise.
Parameters: `privKeyGM` — Private key used for data decryption.
`publicKeyInfo` — Public key used to data verification.
`dataIn` — Data to process.
`dataOut` — Processed data.
See Also: other cryptographic methods
Author: Pechy

34.23

Err **ProcessWithSymetricKey** (GMessage * symKey, MsgField * mfDataIn, MsgField ** mfDataOut, int actionFlag)

Generic method for symetric procession

Generic method for symetric procession..

Symetric encryption and decryption is almost identical, so we made this auxiliary method.

Return Value: KO if procession fails, OK otherwise
Parameters: `symKey` — Symetric key to use.
`mfDataIn` — Input data in a MsgField.
`mfDataOut` — Output data in a MsgField.
See Also: other crypthographic methods
Author: Pechy

34.24

Err **ProcessWithSymetricKey_RawVersion** (GMessage *symKey, char *dataIn, Size dataInLen, char **dataOut, Size *dataOutLen, int actionFlag)

Generic method for symetric procession

Generic method for symetric procession..

Symetric encryption and decryption is almost identical, so we made this auxiliary method. This method process octet stream.

Return Value:	KO if procession fails, OK otherwise
Parameters:	`symKey` — Symetric key to use. `dataIn` — Input data stream. `dataInLen` — Input data stream length. `dataOut` — Output data stream. `dataOutLen` — Output data stream length.
See Also:	other crypthographic methods
Author:	Pechy

34.25

include . .. H common.h GMessage* **ReturnGMsgFullOfSymKeys** (Size hopCount, GMessage *parameters = NULL)

Returns GMessage full of symetric keys

Returns GMessage full of symetric keys.

When we need to create a new onion of a chunk, we have to use some symetric keys. So this method gives them to us.

Return Value:	GMessage full of symetric keys: <pre> GM_SYMETRIC_KEY_COUNT GM_SYMETRIC_KEY_INFO_1 ... ... GM_SYMETRIC_KEY_INFO_"hopCount" </pre>

Parameters: `hopCount` — How many keys do we need.
`parameters` — What types of keys do we need.
See Also: other cryptography methods.
Author: Pechy

34.26

include . .. H common.h Err **Seal** (GMessage *privateKeyInfo, GMessage *publicKeyInfo, MsgField *dataIn, MsgField **dataOut, GMessage *parameters = NULL)

Seals given data

Seals given data.

Seals given data into a digital envelope, using (for now) only RSA, DES and MD5 algorithms, but in parameters, there could be given customization.

Return Value: KO if sealing fails, OK otherwise.
Parameters: `privKeyGM` — Private key used to sign the data.
`dataToSign` — Data to sign.
`sig` — Created signature.
`parameters` — Parameters - which algorithms to use.
See Also: other cryptographic methods
Author: Pechy

34.27

Err **Sign** (GMessage *privKeyGM, MsgField *dataToSign, MsgField **sig, GMessage *parameters = NULL)

Signs given data

Signs given data.

Signs given data using message digest. For now, only MD5 is supported. Asymetric algorithm is RSA for now. In parameters, there could be given customization.

Return Value: KO if sealing fails, OK otherwise.
Parameters: `privKeyGM` — Private key used to sign the data.
`dataToSign` — Data to sign.
`sig` — Created signature.
`parameters` — Parameters - which algorithms to use.
See Also: other cryptographic methods
Author: Pechy

34.28

Err **VerifySignature** (GMessage *pubKeyGM, MsgField *dataToVerify, MsgField *sig, GMessage *parameters = NULL)

Verifies signed data

Verifies signed data.

Verifies given data, using (for now) only RSA and MD5 algorithms, but in parameters, there could be given customization.

Return Value: KO if verification fails, OK otherwise.
Parameters: `pubKeyGM` — Public key to verify signature.
`dataToVerify` — Data to verify.
`sig` — Given signature.
`parameters` — Parameters - which algorithms to use.
See Also: other cryptographic methods
Author: Pechy

34.29 ~Cipherer ()

Cipherer destructor

Cipherer destructor.

Cipherer destructor. For now, nothing is needed to run here.

See Also: Cipherer()
Author: Pechy

34.2 methods

Names

Err **ProcessWithSymetricKey** (GMessage * symKey, MsgField * inData, MsgField ** outData, int actionFlag)

Used by methods for symetric enctryption.

34.2.1 Err **ProcessWithSymetricKey_RawVersion** (GMessage *symKey, char *dataIn, Size dataInLen, char **dataOut, Size *dataOutLen, int flag)
Used by ProcessWithSymetricKey() 175

Process one PKCS1 block. Methods use RSAEuro's RSA implementation.

34.2.1

Err **ProcessWithSymetricKey_RawVersion** (GMessage *symKey, char *dataIn, Size dataInLen, char **dataOut, Size *dataOutLen, int flag)

Used by ProcessWithSymetricKey()

Used by ProcessWithSymetricKey(). Inside, RSAEuro's DES implementation is used.

35

typedef map <string, string, less<string> > **CfgVariables**

Config variable

36

typedef CfgVariables::value_type **CfgVariable**

Array of config variables

37 class **ConfigFile**

Configuration file

Public Members

Private Members

Configuration file.

Encapsulates work with configuration file. When created requires config name - reads it in memory and then is ready to answer questions about keys and values defined. It can also be reloaded.

See Also: Mix Acs Eso Client Bank
Author: Venca

37.3 methods

Names

	ConfigFile (string fileName)	*Opens and reads config from file.*
	~ConfigFile ()	*Destructor*
string	**GetValue** (string name)	*Returns value of variable called "name".*
Err	**Reload** ()	*Reloads settings from file.*
void	**PrintAll** ()	*prints all variables to stdout.*

37.4 ConfigFile (string name)

Config file constructor

Config file constructor.

Creates a mutex for protecting the STL map container. Reads configuration from specified file.

Parameters: `name` — Name of the file that contains configuration data.
See Also: co_se_toho_tyka
Author: Venca

37.5 string GetValue (string name)

Returns value asociated vith name Looks the key in STL map container and returns the associated value, that was specified in configuration file like this: <pre> key1 value1 key2 value2 </pre>

37.6

Err **Load** ()

Loads settings from config file

Loads settings from config file.

Opens the file with O_SHLOCK so that noone can modify the file as it is read. The configuration file is read into memory - it is really small and has to be loaded anyway. And then the strings are parsed and stored in a STL map container as pairs key -> value.

Return Value:	OK if success else KO
See Also:	ReLoad()
Author:	Venca

37.7

void **PrintAll** ()

Lists all variables to stdout

Lists all variables to stdout.

Prints all the pairs key -> value from STL map to stdout. Just for debugging purposes.

See Also:	Load()
Author:	Venca

37.8

Err **Reload** ()

Reloads settings from config file

Reloads settings from config file.

Calls method Load that once more processes the configuration file.

Return Value: OK if success else KO.
See Also: Load()
Author: Venca

37.9

˜**ConfigFile** ()

Config file destructor

Config file destructor.

Destroys mutex that was used to protect access to STL map container.

See Also: ConfigFile (→37, *page 178*)
Author: Venca

37.1

attributes

Names

string **fileName** *Name of configuration file*

pthread_mutex_t

	mutex	*Mutex to protect internal structures.*
CfgVariables	**cfgVariables**	*Array of pairs of name and value.*

37.2

methods

Names

Err	**Load** ()	*Loads settings from file.*
Err	**PrintVar** (CfgVariable *var)	*Prints variable var to stdout.*

38

Debugging macros

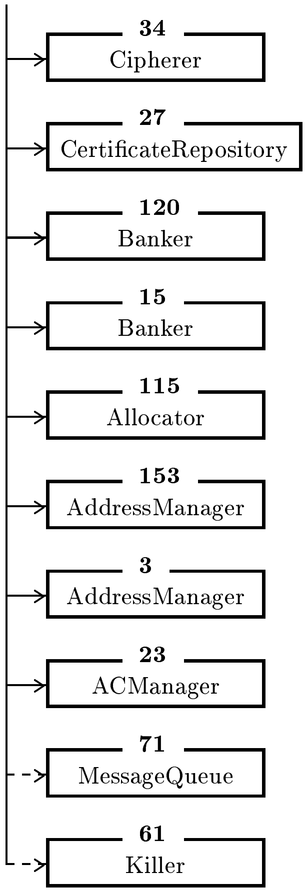
39

class **Debugable**

Debugging class

Inheritance

- **39** Debugable
  - **103** TransactionManager
  - **151** TimeSynchronizer
  - **147** TCBWrapper
  - **174** Six
  - **88** Runable
  - **78** RandomGenerator
  - **10** PayManager
  - **69** MessageIDRepository
  - **164** MessageCreator
  - **60** KeyManager
  - **7** KeyManager
  - **135** Finder



Public Members

Debugging class.

Has the ability to write its progress into log file - it has one and can write into it using its method WriteString.

See Also: Runable LogFile
Author: Venca

39.1 attributes

Names

int **debugOptions** *What messages should be logged*

LogFile* **logFile** *Log file for debug messages*

39.2 methods

Names

virtual int **SetDebugOptions** (int newLogOptions)
Change what is debugged

virtual void
WriteString (LogLevel debugLevel, char * format, ...)
Write message to log file

void **SetLogFile** (LogFile *aLogFile)

		Change log file
	Debugable (LogFile * debugLogFile = NULL)	*constructor*
virtual	**~Debugable** ()	*destructor*
string*	**CreateString** (char *firstStr, int firstInt, int secondInt)	*Helper method*

39.3

string* **CreateString** (char *firstStr, int firstInt, int secondInt)

Creates a string for numbering certificates

Creates a string for numbering certificates.

Return Value: Allocated string containing: <pre> "firstStr"-"firstInt"-"secondInt", if secondInt is equal to -1, only "firstStr"-"firstInt" is created. </pre>

Parameters: `firstStr` — String to take as base for building result
`firstInt` — First number to add
`secondInt` — Second number to add

See Also: Translator, MessageCreator

Author: Pechy

39.4

Debugable (LogFile * debugLogFile)

Debugable constructor

Debugable constructor.

Stores pointer to the log file and initializes debug options.

Parameters:	`debugLogFile` — Pointer to log file that should be used for debugging.
See Also:	WriteString, LogFile
Author:	Venca

39.5

int **SetDebugOptions** (int newLogOptions)

Change debug options of the debugable object

Change debug options of the debugable object.

Sets new value in debug options that are used to decide whether the message will be written into log file and returns the old value so that the caller can restore these settings.

Return Value:	Old value of debug options
Parameters:	`newLogOptions` —
See Also:	WriteString()
Author:	Venca

39.6

void **SetLogFile** (LogFile *aLogFile)

Changes the log file

Changes the log file.

Remembers pointer to the LogFile and inits debugOptions. Allows for log file switching - may be used to debug a certain method into a different file.

Return Value: void
Parameters: `aLogFile` —
See Also: WriteString()
Author: Pavel

39.7

void **WriteString** (LogLevel debugLevel, char * format, ...)

Writes a message to LogFile

Writes a message to LogFile.

Checks if the current debug options allow this log level to be written to log file and if so then it asks log file to write it there. This allows for different debug level for different objects.

Return Value: co
Parameters: `debugLevel` — log level of the message (warning, error ...)
`format` — Format string with the same semantics and syntax as the printf format string.
`...` — Additional parameters that should be incorporated into into format.
See Also: printf(), LogFile, LogLevel
Author: Venca

39.8

~Debugable ()

Debugable destructor

Debugable destructor.

Empty - no need to do anything.

See Also: Debugable()
Author: Venca

40 commands

Names

40.1

Mix commands

Names

#define **CMD_TEST** *Test command*

#define **CMD_SEND_DATA**
Command for mix to send data

#define **CMD_DATA_RECEIVED**
Data arrived

#define **CMD_GET_CERTIFICATES**
Request for certificates

#define **CMD_CERTIFICATES_RECEIVED**
Certificates arrived

#define **CMD_GENERATE_ONION**
Request for onion

#define **CMD_ONION_GENERATED**
Onion arrived

#define **CMD_PUBLISH_CERTIFICATES**
Request for publishing certificates to Acs

40.2

file storage:

Names

#define	**CE_REQ_FOR_STOR**	*Request if it is possible to store a file*
#define	**EC_REPLY_ALLOC**	*If allocation was successful*
#define	**CE_DATA_TO_STOR**	*When sending the whole data for storage*
#define	**EC_REPLY_STOR**	*If storage was successful*
#define	**CB_PAY**	*Client asks Bank to transfer money to Eso's account*
#define	**BC_ANSW_PAY**	
#define	**CE_PAY_DONE**	

40.3

finding file:

Names

#define	**CMD_REQ_FOR_HEADERS**	*Request for information about file*
#define	**CMD_ANSWER_HEADER**	*Headers arrived*
#define	**CMD_REQ_FOR_FILE**	

Give me that file

#define **CMD_ANSWER_FILE**
Answer to request for file

40.4

payment to ESO:

Names

#define **BE_CHALL** *Challlenge from Bank to Eso*

#define **EB_RESP_TO_CHALL**
Eso responds to Bank's challenge

#define **BE_REPLY_TO_RESP**
Bank tells Eso if he guessed it right

41

#define **_THREAD_SAFE**

We want to be safe in the threaded environment

42

#define **MIX_MIX_PROTOCOL_VERSION**

this will be the first field in the message as it goes through the Net

43

#define **NULL**

Invalid or uninitialized pointer

44

extern int **errno**

Extern errno variable

45

#define **NO_PARAM**

Numeric equivalent to NULL

46

#define **DIGITS_IN_NUMBER**

Maximum length of count of certificates in GMessage

47

Boolean values

Names

#define	**YES**	*Positive answer*
#define	**NO**	*Negative answer*

48 Error codes

49

#define **DELETE** (x)

Safe delete

Safe delete. For debugging purpose, delete pointer and set it to NULL Complain to stderr if it is already NULL

50

#define **ASYMETRICALLY_ENCODED_SYMETRIC_KEY_LEN**

Length of symetric key encrypted using asymetric cipher

51

Generic types for portability

Names

#define	**Int**	*Number*
#define	**Char**	*character*
#define	**Double**	*Real number*
#define	**Long**	*Long number*
#define	**Void**	*Empty type*

52

Some special types

Names

typedef unsigned long
Size — *Size of something, length, count*

typedef int
Err — *Error code type*

53

Generic types various identification numbers

Names

typedef unsigned long **ID** — *ID*

typedef unsigned long **TID** — *Transaction ID*

typedef unsigned long **ACID** — *Access certificate ID*

54

Values for config file options

Names

#define **CFG_DEBUG_RANDOM_LONG_SEED**
So that we can reconstruct some situations when debugging

#define **CFG_DEBUG_ADD_HOPS_TO_SEND_DATA**
For debugging purposes

#define **CFG_DEBUG_ONION_LENGTH**
For debugging purposes

#define **CFG_DEBUG_LOOPS**
For debugging purposes

#define **CFG_CERTIFICATES_DIRECTORY**
Where are the certificates stored

#define **CFG_START_INTERVAL_LENGTH**
For padding sending - is padding algorithm dependet

#define **CFG_START_LOAD**
The start rate at which messages are sent

#define **CFG_CERTIFICATE**
Filename in which to store the current service certificate

#define **CFG_ASYMETRIC_KEY_PAIR**
Both asymetric keys - public and private

#define **CFG_MIX_CERTIFICATES_POOL_SIZE**
Number of Mix certificates to keep

#define **CFG_MIX_CERTIFICATE**
Certificate of Mix

#define **CFG_ADD_HOPS**
With how many hops to protect itself by default

#define **CFG_DEFAULT_ONION_LENGTH**

		How long onion to generate by default
#define	**CFG_LOGFILE**	*In what file do we write debug messages, warnings and errors?*
#define	**CFG_MIX_PRIVATE_KEY**	*Private key of Mix*
#define	**CFG_NAME**	*Name of this service - used to identify Six*
#define	**CFG_FROM_MIX_PORT**	*Port where we listen to Mix*
#define	**CFG_FROM_SIX_PORT**	*Port where we listen to Six*
#define	**CFG_INCOMING_PORT**	*Where to listen for incoming messages*
#define	**CFG_OUTGOING_PORT**	*Where to send outgoing messages*
#define	**CFG_DEFAULT_REDUNDANCY**	*On how many Eso's to store the file by default*
#define	**CFG_NUMBER_OF_ESOS_TO_QUERY**	*How many Eso's to ask if they have the file*
#define	**CFG__ADDRESS**	*Address suffix*
#define	**CFG_MIX_ADDRESS**	*Address of mix (IP or hostname)*
#define	**CFG_SIX_**	*Six prefix for lookups in configuration file*
#define	**CFG_FROM_**	*From prefix for lookups in configuration file*
#define	**CFG__PORT**	*Port suffix for lookups in config.*
#define	**CFG_MIX_PORT**	*Port of Mix*
#define	**CFG_MIX_SIX_PROTOCOL_VERSION**	*Protocol version used to communicate between Mix and Six*

55 AC types

Names

#define	**ACTYPE_MIX**	*Certificate type Mix*
#define	**ACTYPE_BANK**	*Certificate type Bank*
#define	**ACTYPE_ACS**	*Certificate type Acs*
#define	**ACTYPE_ESO**	*Certificate type Eso*
#define	**ACTYPE_OTHER**	*General certificate type*
#define	**ACTYPE_THIS_SIX**	*Special certificate type of this Six*

56

GMessage labels

Names

#define	**GM_RECIPIENT_1**	*1st recipient that should get this message*
#define	**GM_RECIPIENT_2**	*2nd recipient that should get this message*
#define	**GM_ORIGIN**	*Where it came from?*
#define	**GM_ACID**	*Access certificate ID*
#define	**GM_ADDRESS**	*Address - IP or symbolic or special name*
#define	**GM_ADD_HOPS**	*Number of hops to add on onion*
#define	**GM_ASYMETRIC_CIPHER_TYPE**	*Which asymetric cipher is used - default is RSA*
#define	**GM_ASYMETRIC_CIPHER_VERSION**	*Which asymetric cipher is used*
#define	**GM_BODY**	*Body of message*
#define	**GM_CERTIFICATE**	*Certificate of a Six or Mix*
#define	**GM_CERTIFICATES**	*Certificates of a Sixes or Mixes*
#define	**GM_CERTIFICATE_1**	*1st certificate of a Six or Mix*
#define	**GM_CERTIFICATE_TYPE**	*Type of certificate - mix, acs, eso ...*
#define	**GM_CHUNK**	*A piece of data*
#define	**GM_CHUNK_ID**	*Unique ID for message chunk that identifies its position in message*
#define	**GM_COMMAND**	

		Specifies the action that ougth to be taken
#define	**GM_COUNT**	*Number of items*
#define	**GM_DATA**	*Message data*
#define	**GM_DATA_SIZE**	*Size of message data*
#define	**GM_DEBUG**	*For debugging purposes - should be 0 in real world*
#define	**GM_DESTINATION**	*Where to send this message?*
#define	**GM_EXPIRATION**	*Date after which the GMessage is not vlid anymore*
#define	**GM_FINAL_STOP**	*If we see this in a message then we can be sure that it is the last mix and we should process it*
#define	**GM_HOP_COUNT**	*Number of hops an onion should include*
#define	**GM_IN_REPLY_TO_TID**	*ID of a message this is a reply to*
#define	**GM_INITIALIZATION_VECTOR**	*Initialization vector used in CBC mode of DES*

#define	**GM_MESSAGE_ID**	*Unique identification of message.*
#define	**GM_MESSAGE_LENGTH**	*Total length of message (used by Linker)*
#define	**GM_MIX**	*Mix.*
#define	**GM_MIX_CERTIFICATE**	*Mix certificate.*
#define	**GM_MIX_CERTIFICATES**	

A GMessage of Mix certificates.

#define **GM_MIX_CERTIFICATE_1**
First Mix certificate.

#define **GM_MIX_CERTIFICATE_2**
Second Mix certificate.

#define **GM_MIX_CERTIFICATE_COUNT**
Number of Mix certificates in GMessage.

#define **GM_MIX_CERTIFICATE_TUPLE_1**
First pair of Mix certificates.

#define **GM_NAME** *Name.*

#define **GM_OAUTH** *Lable for random string used to proove wheter a money requesting Eso is really storing a file*

#define **GM_ONION** *A structure that represents encrypted routing information that is used in onion routing*

#define **GM_PARAMETERS**
GMessage containing additional parameters.

#define **GM_PAY_PLAN**
schedule that says when and how much should be paid

#define **GM_PORT** *u_short specifying the number of port to which to connect*

#define **GM_PRIVATE_KEY**
Private part of asymetric key

#define **GM_PRIVATE_KEY_INFO**
includes private key and its version

#define **GM_PROTOCOL_VERSION**
Version of protocol used

#define **GM_PUBLIC_KEY**
Public part of asymetric key

#define **GM_PUBLIC_KEY_INFO**

Includes public key and its version

#define **GM_RANDOM** *Random data*

#define **GM_RECIPIENT**
Address and port of recipient machine

#define **GM_REPLY_TO_AC**
Access certificate which to reply to

#define **GM_REPLY_TO_TID**
Message ID that should be included in reply as GM_IN_REPLY_TO_TID

#define **GM_REQUEST_ID**
ID of the request that should be returned so that we know what question is the answer for

#define **GM_REST_OF_PATH**
Encrypted part of onion that hides all the remaining hops

#define **GM_RESULT** *What happened?*

#define **GM_SYMETRIC_DATA_KEY_INFO**
GMessage containing symetric data key and its version

#define **GM_SYMETRIC_PATH_KEY_INFO**
GMessage containing symetric path key and its version

#define **GM_SYMETRIC_DATA_KEY_INFOS**
GMessage containing symetric data keys and their versions

#define **GM_SYMETRIC_KEY**
Key for a symetric cipher (DES)

#define **GM_SYMETRIC_KEY_COUNT**
Number of symetric keys

#define **GM_SYMETRIC_CIPHER_TYPE**
Type of symetric cipher - usually DES

#define **GM_SYMETRIC_KEY_INFO**

		GMessage containing symetric key and its version
#define	**GM_TCB_PUBLIC_KEY**	*Public key of trusted 'computing base'*
#define	**GM_TID**	*Transaction ID.*
#define	**GM_TO_AC**	*Access certificate of an Acs to which to send data.*
#define	**GM_TOTAL_CHUNKS**	*Linker and Chopper info about number of chunks in message*
#define	**GM_VERSION**	*Version*
#define	**GM_X_DESTINATION**	*Helper header for directing message inside Mix.*

#define	**GM_IV**	*Initialization vector for CBC mode*
#define	**GM_TRANSPORT_KEY**	*Key used to encrypt for transport security.*
#define	**GM_FFID**	*File find ID.*
#define	**GM_FILE**	*Data of file to store.*

56.1

@name Labels used when specifying file characteristics

Names

#define	**GM_FFID**	*Finder file identificator - used as result of file searches and as a download identificator*

#define	**GM_FID**	*File identificator - used to manage file in Eso*
#define	**GM_FILE_SIZE**	*Size of file in bytes*
#define	**GM_STORAGE_TIME**	*Relative time for which the file should be stored*

56.2

@name Labels used when searching for file

Names

#define	**GM_KEYWORD**	*A word describing the file*
#define	**GM_KEYWORDS**	*GMessage of keywords*
#define	**GM_MAX_SIZE**	*Maximum accepted size*
#define	**GM_MIN_SIZE**	*Minimum accepted size*
#define	**GM_SEARCH_DEPTH**	*Depth of the search to perform*
#define	**GM_SEARCH_WIDTH**	*Width of the search to perform*
#define	**GM_HEADER**	*Header info*

56.3

@name When is the certificate valid?

Names

#define **GM_NOT_VALID_AFTER**
Last date of validity.

#define **GM_NOT_VALID_BEFORE**
First day of validity.

56.4

@name GMessage fields for sealing and opening

Names

#define **GM_ENCRYPTED_CONTENT**
Encrypted data

#define **GM_ENCRYPTED_KEY**
Encrypted key

#define **GM_ENCRYPTED_SIGNATURE**
Encrypted result of a hash function over data.

57

Certificate fields

Names

#define **ACFLD_TYPE** *Certificate type*

#define **ACFLD_ACID** *Certificate unique ID*

#define **ACFLD_PUB_KEY**
Certificate Public key

#define **ACFLD_EXPIRATION**
Expiration date of certificate

#define **ACFLD_MIX_CERTIFICATE**
Mix certificate included in access certificate

#define **ACFLD_TCB_PUB_KEY**
Public key of trusted computing base

#define **ACFLD_NOT_VALID_BEFORE**
Valid from

#define **ACFLD_NOT_VALID_AFTER**
Valid till

58 GMessage fields' values

Names

#define	**LBL_MIX_LOCAL**	*This Mix*
#define	**LBL_MIX_REMOTE**	*Remote Mix*
#define	**LBL_MIX**	*Mix*
#define	**LBL_ACS**	*Recipient is Six named acs*
#define	**LBL_ESO**	*Recipient is Six named eso*
#define	**LBL_CLIENT**	*Recipient is Six named client*
#define	**LBL_YES**	*Positive reply - yes*
#define	**LBL_NO_DEBUG_NOTE**	*Helper label*
#define	**LBL_MIXSIX_PROTOCOL_VERSION_BETA**	*Protocol version used to communicate between Mix and Six*
#define	**LBL_ESO_OTHERS_PROTOCOL_VERSION_BETA**	*Esos' protocol*

59

#define **MAX_HOST_NAME_STRING_LEN**

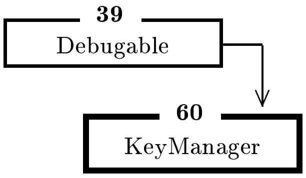
Maximum length of hostname

60

class **KeyManager** : public Debugable

Generic key manager

Inheritance



Public Members

Generic key manager.

Generates all kinds of keys and manages them - takes care of their expiration and renewal.

See Also: Cipherer AddressManager
Author: Pechy, Venca

60.1 attributes

Names

GMessage* **asymetricKeyPair** *asymetric keys*

ConfigFile* **configFile** *Configuration file*

Cipherer* **cipherer** *Encryption class*

MessageQueue* **messageQueue** *Message queue for publishing*

60.2 methods

Names

KeyManager::KeyManager (ConfigFile *configFile, LogFile * keyManagerLogFile, Cipherer *cipherer, MessageQueue *messageQueue)
Constructor

virtual **~KeyManager** () *Destructor*

GMessage* **GetMixPrivateKey** (char *asymCipherType, char *asymCipherType)
Returns this Mix's private key

GMessage*

	GetMixPublicKey (char *asymCipherType, char *asymCipherType)	*Returns this Mix's public key*
Err	**CreateMixCertificate** (u_short port, GMessage **certificate)	*Generates Mix certificate (for this Mix)*
Err	**LoadAsymetricKeyPairIntoMemory** ()	*Load keys from disk*

60.3

KeyManager (LogFile *aLogFile, Char* aTableName)

Constructor of keyManager

Constructor of keyManager.

Creates the keyManager table.

Parameters: `logFile` — keyManager logFile
`aTableName` — Name of file to create keyManager table in.

See Also:

Author: Marek

60.4

KeyManager (ConfigFile *keyManagerConfigFile, LogFile * keyManagerLogFile, Cipherer *keyManagerCipherer, MessageQueue *keyManagerMessageQueue)

Key manager constructor

Key manager constructor.

Stores pointers to objects that are need later for various tasks.
 It also loads keys saved on disk into memory in case someone wants it.

Parameters:
- `keyManagerConfigFile` — Pointer to cinfig that we ask for configuration options.
- `keyManagerLogFile` — Log file that we have to write debug messages to.
- `keyManagerCipherer` — Cipherer object that is used for key generation.
- `keyManagerMessageQueue` — Message queue that is used for publishing keys.

See Also: Cipherer (→34, *page 150*)

Author: Pechy

60.5

Err **LoadAsymetricKeyPairIntoMemory** ()

Loads asymetric keys from disk

Loads asymetric keys from disk.

If there is a problem loading keys then an error is reported and exit called.

Return Value: OK if success else KO.

See Also: GetMixPrivateKey(), GetMixPublicKey()

Author: Pechy

60.6

Err **SaveKeys** (ACID acid, GMessage *pubKey, GMessage *privKey)

Inserts keys into the table

Inserts keys into the table.

Saves public and private key together with the associated ACID.

Return Value: error code
Parameters: `acid` — Acid to save.
`pubKey` — PubKey to save.
`privKey` — PrivKey to save.
See Also:
Author: Marek

60.7

~KeyManager ()

Destructor of keyManager

Destructor of keyManager.

Deletes the keyManager's table.

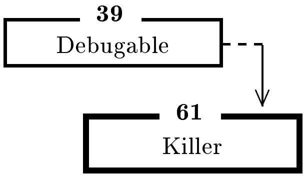
See Also:
Author: Marek

61

class **Killer** : Debugable

Shuts down all shutdownable objects

Inheritance



Public Members

Protected Members

Shuts down all shutdownable objects.

All Shutdownable objects register with Killer when created and he shuts them down when necessary. In fact it calls the Shutdown() method of all Shutdownable objects that have to implement it. Calls Shutdown() method of all shutdownable objects - that is why he stores pointers to all of them. They pass it to him when created.

See Also: Shutdownable (→99, *page 301*)
Author: Venca

61.2 methods

Names

	Killer (LogFile * logFile)	*Constructor*
void	**RegisterForShutdown** (Shutdownable * victim)	*Join the victims*
void	**ShutdownAll** ()	*Shutdown all victims*

61.3 Killer (LogFile * logFile)

Killer constructor

Killer constructor.

Initializes parent classes.

Parameters: `logFile` — Killer is Debugable so he needs a log file to write to.
See Also: Shutdownable, Shutdown()
Author: Venca

61.4 void RegisterForShutdown (Shutdownable * victim)

Register yourself for information about shutdown

Register yourself for information about shutdown.

Adds victim to a list of objects that should be notified by calling their method Shutdown when we are going down. Victim has to be Shutdownable - has to implement Shutdown().

Parameters: `victim` — object to notify.
See Also: Shutdownable (→99, *page 301*)
Author: Venca

61.5

void **ShutdownAll** ()

Call Shutdown() of all victims

Call Shutdown() of all victims.

Victims - objects that asked for it.

See Also: Shutdownable (→99, *page 301*)
Author: Venca

61.1

attributes

Names

list < Shutdownable * >
victims *list of objects to notify*

62 class **LeakDetector**

Memory leak detector

Public Members

Memory leak detector.

A parent of all classes that we want to track. It finds out if all instances are deleted correctly or if there are some memory leaks.

See Also: GMessage MsgField
Author: Venca

62.1 operators

Names

void* **operator new** (size_t size)
Overloaded new

void **operator delete** (void * what)
Overloaded delete

62.2

void **operator delete** (void * what)

Overridden operator delete

Overridden operator delete.

Frees memory for the object and writes a message into the log file including the ID that is associated with this pointer so that we can find leaks by comparing ID's and looking for those that were not deleted.

Return Value:	void
Parameters:	`what` — Pointer to object we want to delete
See Also:	GMessage, operator new
Author:	Venca

62.3

void* **operator new** (size_t size)

Overridden operator new

Overridden operator new.

Allocates memory for the object and remembers that along with a unique ID and writes a message into log file including that ID. Later we can find what was allocated and not freed.

Return Value:	Pointer to the newly allocated space.
Parameters:	`size` — Size of object to allocate.
See Also:	GMessage, operator delete
Author:	Venca

63

typedef map < void *, long, less<void *> > **Pointers**

Structure containing allocated pointers together with ID's

64

typedef Pointers::value_type **pointerRec**

Pair of pointer and ID

65

Pointers **pointers**

Variable containing allocated pointers together with ID's

66

enum **LogLevel**

Log level

Log level.

Type that specifies all the possible levels of debug messages.

See Also: LogFile, Debugable
Author: Venca

67

class **LogFile**

Log file

Public Members

Protected Members

Log file.

class that encapsulates file for logging debug messages. Has the ability to log just certain types of messages.

See Also: LogLevel, Debugable
Author: Venca

67.2

methods

Names

	LogFile (const char *name)	*create log called name*
int	**SetDebugOptions** (int options)	*set what to log xor-ed log levels*
virtual void	**WriteString** (LogLevel logLevel, char *format, ...)	*Print to log*
virtual void	**WriteStringVA** (LogLevel logLevel, char *format, va_list args)	*Print to log*

67.3

LogFile (const char *name)

Log file constructor

Log file constructor.

Opens the specified file for appending and writes start of session in there. <p> Also sets initial debug options.

Parameters: `name` — Log file name.
See Also: ~LogFile(), LogLevel
Author: Venca

67.4

int **SetDebugOptions** (int options)

Change what is logged in log file

Change what is logged in log file.

Sets debug options = what should be logged. Accepts any combination of logLevels. Returns old value of debug options.

Return Value:	Old value of debug options.
Parameters:	`options` — Log levels that should be logged ored together.
See Also:	WriteString(), Debugable
Author:	Venca

67.5

void **WriteString** (LogLevel logLevel, char *format, ...)

Write message to log file

Write message to log file.

Writes a string to log file if its logLevel matches debugOptions of the log file. <p> Also prepends time. Every log entry is on separate line.

Return Value:	void
Parameters:	`logLevel` — log level of message - warning, error ... `format` — format string similar to printf() `...` — Additional variables we want to print
See Also:	Debugable, printf(), LogLevel
Author:	Venca

67.6

void **WriteStringVA** (LogLevel logLevel, char *format, va_list args)

Write message to log file

Write message to log file.

Writes a string to log file if its logLevel matches debugOptions of the log file. <p> Also prepends time. Every log entry is on separate line.

Return Value: void

Parameters:
`logLevel` — log level of message - warning, error ...
`format` — format string similar to printf()
`args` — Additional variables we want to print

See Also: Debugable, printf(), LogLevel

Author: Venca

67.7

~LogFile ()

Log file destructor

Log file destructor.

Writes end of session to log file and closes log file.

See Also: LogLevel (→66, *page 233*)

Author: Venca

67.1

attributes

Names

map < int, char *, less<int> >	**debugMessages**	*Descriptions for types of messages*
char*	**fileName**	*Where to write all the messages*
int	**debugOptions**	*What to write*

68

class **RegistratorRecord**

Registrator record

Public Members

Registrator record.

Contains information about the entity that sent a request with the Request ID object or six name -> RequestID

See Also: Majordomo, MessageIDRepository
Author: Venca

68.1

attributes

Names

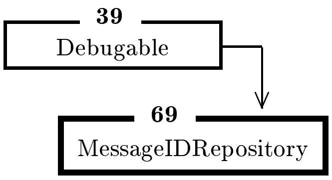
Registerable* **registrator**		*Pointer to object*
string	**destination**	*Six name*

69

class **MessageIDRepository** : public Debugable

Message Id repository

Inheritance



Public Members

Protected Members

attributes

Message Id repository.

Remembers the pairs (object or Six) and request ID so that reply can send reply to the correct requestor.

See Also: Majordomo, RegistratorRecord
Author: Venca

69.1 methods

Names

MessageIDRepository (LogFile * messageIDRepositoryLogFile)
Constructor

RegistratorRecord* **GetRegistratorRecord** (ID messageID)
Get who is registered

Registerable* **GetRegistrator** (ID messageID)
Get who is registered

Err **Register** (ID messageID, Registerable * registrator, string destination = "")
Register for Imessage ID

69.2 Registerable* **GetRegistrator** (ID messageID)

Reveal which object is waiting for this message

Reveal which object is waiting for this message.

Returns pointer to object that is waiting for this reply. Deletes this object from its table.

Return Value: Pointer to registerable object that registered for this message. NULL if noone found.
Parameters: `messageID` — ID of message we are inspecting.
See Also: Registerable (→86, *page 277*)
Author: Venca

69.3

RegistratorRecord* **GetRegistratorRecord** (ID messageID)

Reveal who is waiting for this message ID

Reveal who is waiting for this message ID.

Returns pointer to structure containing object that is waiting for this reply or the name of Six willing to receive the message. `<p>` Deletes this structure from its table

Return Value: pointer to structure that contains pointer to Registerable object or name of Six.
Parameters: `messageID` — ID of message we are deciding where to send
See Also: Registerable, Majordomo
Author: Venca

69.4

MessageIDRepository (LogFile * messageIDRepositoryLogFile)

Message ID repository

Message ID repository.

Initializes its parent classes.

Parameters:	`messageIDRepositoryLogFile` — Log file to write to.
See Also:	Registerable (→86, *page 277*)
Author:	Venca

69.5

Err **Register** (ID messageID, Registerable * registrator, string destination)

Register object registrator for reply to messageID

Register object registrator for reply to messageID.

Remembers pointer to requesting object together with ID so that is able to call them when reply arrives.

Return Value:	KO if someone is already registered for this ID.
Parameters:	`messageID` — ID of message registrator is waiting for reply to. `registrator` — Object that wants to be notified when reply arrives. `destination` — Six that wants to receive the reply.
See Also:	Registerable (→86, *page 277*)
Author:	Venca

70

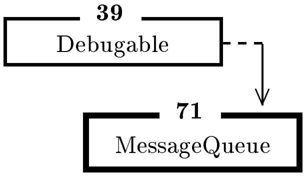
for requestType SOFT always tries whether the queue is full, in that case it returns QUEUE_FULL, HARD would go to sleep

71

class **MessageQueue** : Debugable

Queue of message pointers

Inheritance



Public Members

71.2		**methods**		245
71.3	Err	**Append** (GMessage *msg, int requestType = HARD)	*Append message to queue*	246
71.4	Size	**FreeSlotsCount** ()	*Number of messages it is possible to append to queue*	246
71.5	Err	**GetMsg** (GMessage **msg, Size number)	*et message from queue*	247
71.6	Err	**GetRandomMsg** (GMessage **msg)	*Get random message from queue*	247
71.7	bool	**IsQueueFull** ()	*Is the queue full?*	248
71.8		**MessageQueue** (LogFile * queueLogFile, Size size)	*Message queue constructor*	248
71.9	Size	**MsgCount** ()	*Returns the number of messages in queue*	249
71.10	Err	**ShowMsg** (GMessage **msg, Size number)	*Get message from queue*	249
71.11		**~MessageQueue** ()	*Message queue destructor*	249

Private Members

Queue of message pointers.

Holds a queue of pointers to messages. It provides access to stored messages in FIFO order and even random access.

See Also:
Author: Marek, Pechy, Venca

71.2 methods

Names

	MessageQueue (LogFile * queueLogFile, Size size = MSGQ_DEFAULT_BUFFER_LEN)	*Creates a message that can store up to size messages*
	~MessageQueue ()	*Destructor*
Err	**Append** (GMessage *msg, int requestType = HARD)	*Appends a message to the end of queue*
Err	**GetMsg** (GMessage **msg, Size number = 0)	*Returns the number-th message from queue (removes it from queue)*
Err	**GetRandomMsg** (GMessage **msg)	*Returns a random message from queue (removes it from queue)*
Err	**ShowMsg** (GMessage **msg, Size number = 0)	*Returns the number-th message from queue (leaves it in queue)*
Size	**MsgCount** ()	*Returns the number of messages in queue*

Size	**FreeSlotsCount** ()	*Returns the number of free slots in queue*
bool	**IsQueueFull** ()	*Is the queue full ?*

71.3

Err **Append** (GMessage *msg, int requestType = HARD)

Append message to queue

Append message to queue.

Appends GMessage to the end of messaage queue.

Return Value: If requestType is SOFT and queue is full then KO else OK.

Parameters: `msg` — GMessage
`requestType` — Wheteher we should fall asleep when message queue is already full - default is yes.

See Also:

Author: Pechy

71.4

Size **FreeSlotsCount** ()

Number of messages it is possible to append to queue

Number of messages it is possible to append to queue.

Return Value: Number of free slots.

See Also:

Author: Pechy

71.5

Err **GetMsg** (GMessage **msg, Size number)

et message from queue

et message from queue.

Returns the number-th message and removes it from queue, the 1st message has number 0 !!!!

Return Value:	OK
Parameters:	`msg` — Gets filled with a pointer to the message. `number` — Index of message we want from queue.
See Also:	
Author:	Pechy

71.6

Err **GetRandomMsg** (GMessage **msg)

Get random message from queue

Get random message from queue.

Chooses a random message in queue and returns it. It also deletes the message from queue.

Return Value:	OK
Parameters:	`msg` — Out - gets filled with pointer to a GMessage.
See Also:	
Author:	Pechy

71.7

bool **IsQueueFull** ()

Is the queue full?

Is the queue full?.

If number of messages is the same as number of slots then the queue is full.

Return Value: true if queue is full false otherwise.
See Also:
Author: Pechy

71.8

MessageQueue (LogFile * queueLogFile, Size size)

Message queue constructor

Message queue constructor.

Implements a simle FIFO queue. Initializes semaphores and mutex for protecting its inner data and also initializes this inner data.

Parameters: `queueLogFile` — Log file that is used for debug messages.
`size` — Maximum number of messages in queue.
See Also: semaphores
Author: Pechy

71.9

Size **MsgCount** ()

Returns the number of messages in queue

71.10

Err **ShowMsg** (GMessage **msg, Size number)

Get message from queue

Get message from queue.

Returns the number-th message from queue but leaves it in there.

Return Value: OK
Parameters: `msg` — Gets filled with pointer to message retured.
`number` — Message offset in queue.
See Also:
Author: Pechy

71.11

~MessageQueue ()

Message queue destructor

Message queue destructor.

Deletes its internal data structures.

See Also:
Author: Pechy

71.1

attributes

Names

Size	**size**	
Size	**firstMsg**	
Size	**lastMsg**	
Size	**msgCount**	
pthread_mutex_t	**mutex**	*for synchronization*
GMessage**	**messages**	*array of pointers to messages*

72

GMessage fields' names

73

Table fields' names

74

Values of 'success' field

75

Value of "bad" or missing ID

76

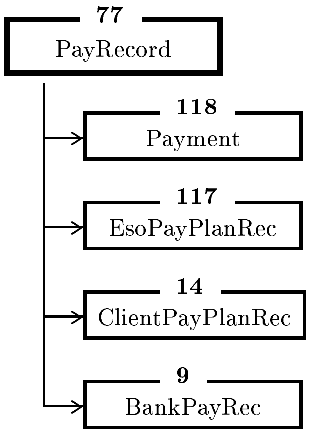
max length of authentification strings

77

class **PayRecord**

Payment Record

Inheritance



Public Members

Protected Members

attributes

Payment Record.

An auxiliary abstract class for one payment record. Each descendant of this class must overwrite FetchFromRecord and FillRecord methods to be able to fetch record from or write record to a table. Ancestor of all payment record classes (in bank, client and eso).

See Also: Bank, Client, Eso
Author: Ivana

77.1 methods

Names

	PayRecord ()	*constructor*
virtual	**~PayRecord** ()	*destructor*

77.2 Void **Clear** ()

Clears data

Clears data.

Sets class variables to initial values (including eCode).

Return Value: void
See Also:
Author: Ivana

77.3

Err **FetchAndDeleteFromTable** (Table *t, Char *whichId, ID value)

Fetches And Deletes Data From Table

Fetches And Deletes Data From Table.

Fetches and deletes data from given table which corresponds to given ID. If whichId is not specified (is NULL) returns first record.

Return Value: error code
Parameters: `t` — table in which we will search
`whichId` — name of ID we want to search
`value` — value of ID
See Also:
Author: Ivana

77.4

Err **FetchFromTable** (Table *t, Char *whichId, ID value)

Fetches Data From Table

Fetches Data From Table.

Fetches data from given table which corresponds to given ID. If whichId is not specified (is NULL) returns first record.

Return Value: error code
Parameters: `t` — table in which we will search
`whichId` — name of ID we want to search
`value` — value of ID
See Also:
Author: Ivana

77.5

Err **IsOK** ()

Returns eCode

Returns eCode.

Returns eCode - wheather the instance is consistent or not.

Return Value: eCode
See Also:
Author: Ivana

77.6

PayRecord ()

Payment Record constructor

Payment Record constructor.

Payment Record constructor initializes class variables.

See Also:
Author: Ivana

77.7

Err **UpdateTable** (Table *t)

Updates a record in given table

Updates a record in given table.

Updates a record in given table.

Return Value:	error code
Parameters:	`t` — table in which we will update
See Also:	
Author:	Ivana

77.8

Err **WriteToTable** (Table *t)

Writes a record to given table

Writes a record to given table.

Writes a record to given table.

Return Value:	error code
Parameters:	`t` — table into which we will write
See Also:	
Author:	Ivana

77.9

~PayRecord ()

Payment Record destructor

Payment Record destructor.

Payment Record destructor deletes class variables.

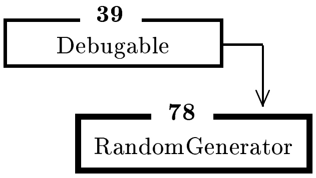
See Also:
Author: Ivana

78

class **RandomGenerator** : public Debugable

Generator of random data

Inheritance



Public Members

		methods		
78.1	MsgField*	**RandomBytes** (Size count)	*Generate random bytes*	263
78.2		**RandomGenerator** (LogFile *aLogFile)	*Random generator constructor* .	263
78.3	Size	**RandomSize** ()	*Generate random Size*	263

Generator of random data.

Generates a message field that contains

See Also:
Author: Pavel

78.1

MsgField* **RandomBytes** (Size count)

Generate random bytes

Generate random bytes.

Generates count bytes of random data and returns it in a MsgField. Used to generate data for padding size of messages and also for generating symetric and asymetric keys.

Return Value:	MsgField filled with number random bytes
Parameters:	`count` — Number of bytes to generate
See Also:	KeyManager, Cipherer
Author:	Pavel

78.2

RandomGenerator (LogFile *aLogFile)

Random generator constructor

Random generator constructor.

It opens device /dev/random and initializes its seed using the random data from it. /dev/random should provide us with suitable strength in randomness - if evene stronger randomness is required or bigger indenpendence on platform, we can modify this class to cooperate with some hardware random generator.

Parameters:	`aLogFile` — Log file for debug messages.
See Also:	KeyManager (→7, *page 41*)
Author:	Pavel

78.3

Size **RandomSize** ()

Generate random Size

Generate random Size.

Generates Size of random data and returns it. Used to generate random and unique ID's.

Return Value: Size filled with random data
See Also: KeyManager, Cipherer, UniqueIDGenerator
Author: Pavel

79

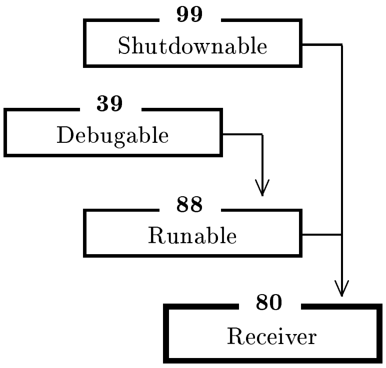
#define **RECEIVER_STACK_SIZE**

thread stack is not very large by default, so let's adjust that.

80 class **Receiver** : public Runable, public Shutdownable

Message receiver

Inheritance



Public Members

Protected Members

Message receiver.

a class for receiver, it is a object that is responsible for accepting messages from the Net and passing it to Translator. <PRE> In: byte stream from network Out: GMessage containing GM_DEBUG GM_PROTOCOL_VERSION GM_REST_OF_PATH GM_DATA GM_RECIPIENT </PRE>

See Also: Sender (→94, *page 288*)
Author: Pechy

80.2 methods

Names

	Receiver (ConfigFile * receiverConfigFile, LogFile * receiverLogFile, Killer * killer, MessageQueue * fromReceiver, string origin, u_short port)	*Constructor*
	~Receiver ()	*Destructor*
void*	**Run** (void *info)	*Thread method*
void	**RequestServant** (int sockIn)	*Handles one message*
virtual void	**Shutdown** ()	*Prepare for shutdown*

80.3 Receiver (ConfigFile * receiverConfigFile, LogFile * receiverLogFile, Killer * killer, MessageQueue * fromReceiver, string origin, u_short portWhereToListen)

Generic receiver constructor

Generic receiver constructor.

Remembers all the attributes. must get pointer to rcvToTran queue, because this queue is declared in Mix class so Receiver can't see it as we want to be generic.

Parameters: `receiverConfigFile` — Configuration file.
`receiverLogFile` — Log file for debug messages.
`killer` — Killer objectthat will call us when shutdown occurs.
`fromReceiver` — Output queue into which to insert received messages.
`origin` — GM_ORIGIN is set to origin unless it is empty string.
`portWhereToListen` — Port on which the receiver shoul listen to incomming messages.
See Also: Sender()
Author: Pechy

80.4

void **RequestServant** (int sockIn)

Receive one message from network

Receive one message from network.

Reads data from the specified socket and allocates memory for it. After this is done it makes a GMessage from it and inserts a field called GM_ORIGIN with value specified in constructor. Then the message is passed to the outgoing queue.

Return Value: void
Parameters: `sockIn` — Socket from which we can read tha incoming data.
See Also: Receiver (→80, *page 266*)
Author: Pechy

80.5

void* **Run** (void *)

Main receiver method

Main receiver method.

this function is the function that is run in thread. It listens on some port and whenever there comes a connection it disptches a new thread that takes care of receiving the whole message and passing it to the next object.

Return Value:	(void *)NULL
See Also:	Sender (→94, *page 288*)
Author:	Pechy

80.6

void **Shutdown** ()

Prepare for clean shut down

Prepare for clean shut down.

This method should implement all the tasks necessary for preparation for a clean shutdown like saving inconsistent data to disk etc.

Return Value:	void
See Also:	Shutdownable, Killer
Author:	Venca

80.7

~Receiver ()

Generic receiver destructor

Generic receiver destructor.

Empty for now.

See Also: Receiver()
Author: Pechy

80.1

attributes

Names

string	**origin**	*What to insert into message*
ConfigFile*	**configFile**	*Configuration file*
MessageQueue*	**fromReceiverQueue**	*Outgoing message queue*
u_short	**port**	*Port where to listen*
int	**receiverType**	*Type of receiver*

81 class **ArgsForRequestThread**

Receiver arguments for thread

Public Members

Receiver arguments for thread.

Contains all args needed by thread that handles retrieval of one message from net. Description: We need a class below for arguments that we need pass to a function for phthread_create. We have to use an auxiliary function and then to call the receiver->RequestServant() inside that function.

See Also: Receiver, Sender
Author: Pechy

81.1 attributes

Names

Receiver*	**receiver**	*Pointer to parent*
int	**sockIn**	*Socket to read from*

81.2 methods

Names

ArgsForRequestThread ()
Constructor

~ArgsForRequestThread ()
Destructor

82

void* **ReceiverRequestThread** (void *args)

thread that handles retrieval of one message

83

#define **RECEIVER_QUEUE**

length of listen()'s queue

84

#define **MAX_RECEIVER_BUF_SIZE**

Receiver buffer size - does not limit the total size of message

85

void* **ReceiverRequestThread** (void *args)

Take care of receiving one message

Take care of receiving one message.

This function is a parameter to the pthread_create() function call. It works around the fact that it is not possible to pass a method to pthread_create().

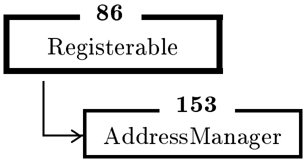
Return Value:	NULL - not used by now.
Parameters:	`args` —
See Also:	Receiver (→80, *page 266*)
Author:	Pechy

86

class **Registerable**

Registerable class

Inheritance



Public Members

Registerable class.

Descendands of this class are able to register with MessageIDRepository so that it can receive the reply.<p> This class forces its offsprings to implement method used for clean register.<p> All objects that have to save some info before terminating should inherit from this class

See Also: MessageIDRepository (→69, *page 239*)
Author: Venca

86.1

methods

Names

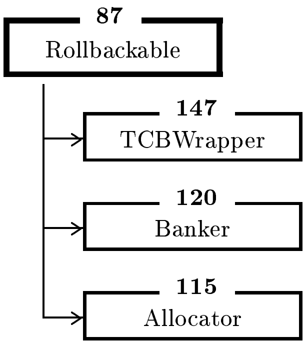
virtual void
HereYouAre (GMessage * data)
Pass data to object

87

class **Rollbackable**

Class taht is able to roll back its transaction

Inheritance



Public Members

Class taht is able to roll back its transaction.

In cooperation with TransactionManager is able to roll back some transactions that failed.

See Also: TransactionManager (→103, *page 308*)
Author: Pavel

87.1

methods

Names

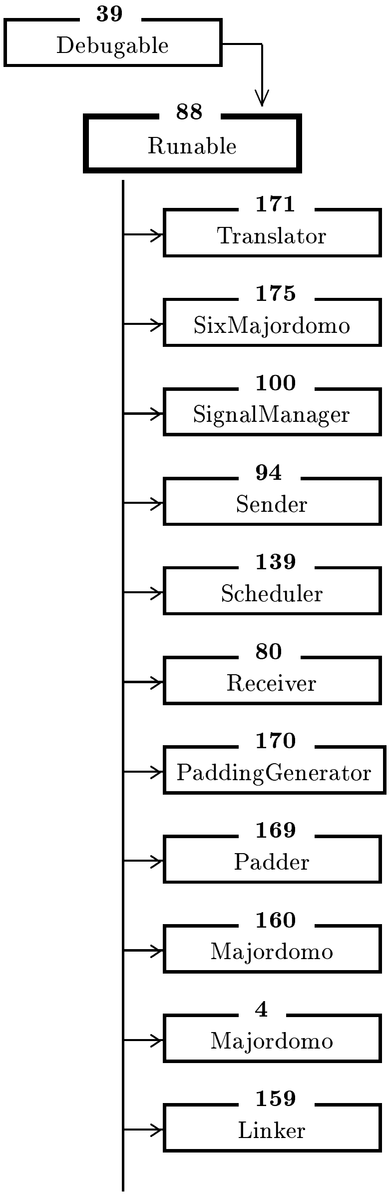
virtual Err
Rollback (TID) *Raoll back and return to a consistent state*

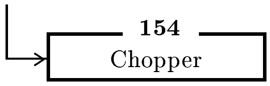
88

class **Runable** : public Debugable

Runable class

Inheritance





Public Members

88.1 **methods** 280

88.2 void **MaskAllSignals** ()
Mask all signals 281

88.3 **Runable** (LogFile * logFile)
Runable constructor 281

Runable class.

This class forces its offsprings to implement method Run used for execution in a separate thread. All objects that have to run as a thread should inherit from this class.

See Also: Debugable LogFile SignalManager
Author: Pechy Venca

88.1 methods

Names

Runable (LogFile * logFile = NULL)
Constructor

void **MaskAllSignals** ()
Do not send me signals (Signal manager will take care of them)

virtual void*
Run (void * param)
Main thread function

88.2

void **MaskAllSignals** ()

Mask all signals

Mask all signals.

All threads except SignalManager *have* to mask all signals for library to direct all signals to SignalManager.

Return Value: void
See Also: SignalManager, thread discussion, Shutdownable
Author: Pechy

88.3

Runable (LogFile * logFile)

Runable constructor

Runable constructor.

Initializes its parent class.

Parameters: `logFile` — Log file for writing debug messages.
See Also: Debugable, LogFile
Author: Venca

89

void* **RunObject** (void *object)

helper function

90

void* **RunObject** (void *object)

Run object in a separate thread

Run object in a separate thread.

Runs method Run of object. Used for creation of a new running object. Works around the problem with passing a method of an object to pthread_create().

Return Value:	what them method Run() of Runnable object returns.
Parameters:	`object` — Runable object to execute.
See Also:	pthread_create(), RunningObject, Run()
Author:	Venca

91

typedef enum **RECEIVER**

Name of running object (thread)

Name of running object (thread).

See Also: Receiver, Translator, Padder, Sender, Linker, Majordomo, Chopper, Scheduler, SignalManager

Author: Pechy

92 class **RunningObject**

Running objecti (thread)

Public Members

Protected Members

Running objecti (thread).

class that contains stuff for enabling waiting for running objects

See Also: Receiver, Translator, Padder, Sender, Linker, Majordomo, Chopper, Scheduler, SignalManager

Author: Pechy

92.2 methods

Names

RunningObject ()
constructor

~RunningObject ()
destructor

Err **SaveRunningObjectID** (RunningObjectName objectName, RunningObjectID objectID)
Store ID to be able to wait for the thread

Err **WaitForRunningObject** (RunningObjectName objectName)

Wait for thread to finish

92.1 attributes

Names

RunningObjectID*
runningObjectID
array for runnig objects ID's, for mix to be able to wait for them

93

class **Schedulable**

Schedulable class

Inheritance

93
Schedulable

169
Padder

Public Members

Schedulable class.

Schedulable forces its offsprings to implement HandleEvent method that is used as a callback function when the event we are waiting for happens. All objects that want to schedule some actions for the future have to inherit from this class.

See Also: Scheduler (→139, *page 405*)
Author: Venca

93.1

methods

Names

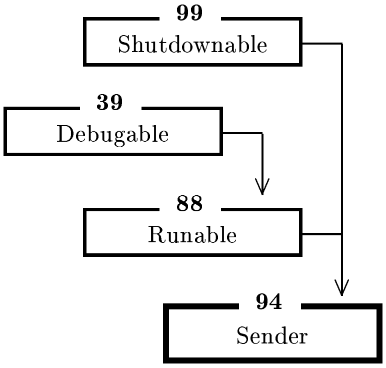
virtual void
HandleEvent (int handleID)
Notify about event that occured

94

class **Sender** : public Runable, public Shutdownable

Sender class

Inheritance



Public Members

94.3		**methods**	....	289
94.4	unsigned long	**ConvertToNumericIP** (char *addr)	*Converts string address into the unsigned long*	290
94.5	Err	**DispatchMsg** (GMessage *msg)	*Send a message over the network*	291
94.6	void	**DispatcherThread** (GMessage *msg)	*Connect to one of the recipients and send him the message*	291
94.7	GMessage*	**MakeMessageForSendingToMix** (GMessage *chunk)	*Add recipient to message*	292
94.8	void*	**Run** (void *)	*Generic Sender Run() method* ..	292
94.9		**Sender** (LogFile * senderLogFile, Killer * killer, MessageQueue * toSenderQueue, string origin, string IP = "", unsigned short port = 0)		

Protected Members

Sender class.

Sends messages across network to other Mix or Six Expects a GMessage containing: <PRE> GM_RECIPIENT_1 GM_RECIPIENT_2 GM_CHUNK GM_DEBUG GM_PROTOCOL_VERSION GM_REST_OF_PATH GM_DATA </PRE>

Makes a message like this and transmits it over network to IP from field Recipient. <PRE> GM_CHUNK GM_RECIPIENT GM_DEBUG PROTOCOL_VERSION GM_REST_OF_PATH GM_DATA </PRE>

The GM_RECIPIENT is set to GM_RECIPIENT_1 or GM_RECIPIENT_2 depending on which was used as destination address.

See Also: Receiver
Author: Pechy

94.3

methods

Names

Sender (LogFile * senderLogFile, Killer * killer,
MessageQueue * toSenderQueue,
string origin, string IP = "",
unsigned short port = 0)

		Constructor
virtual	**~Sender** ()	*Destructor*
void*	**Run** (void *)	*Main thread function*
Err	**DispatchMsg** (GMessage *msg)	*Take care of this one message*
void	**DispatcherThread** (GMessage *msg)	*method that runs in a separate thread and sends the message*
virtual void	**Shutdown** ()	*Prepare for clean shutdown*
GMessage*	**MakeMessageForSendingToMix** (GMessage *chunk)	*Insert default address and port to recipient*

94.4

unsigned long ConvertToNumericIP (char *addr)

Converts string address into the unsigned long

Converts string address into the unsigned long.

In the Path field in msg we can have both DNS or IP addresses. Both types are as a *string*. We need to convert the address to an unsigned long.

Return Value: IP address represented by a long value.
Parameters: `addr` — String representation of address (IP or hostname)
See Also: SenderDispatcher()
Author: Pechy

94.5

Err **DispatchMsg** (GMessage *msg)

Send a message over the network

Send a message over the network.

Sender requested for sending a message - This method creates a thread, that should do all the work. It takes GM_RECIPIENT and sends the message to the address specified in it and just before it he inserts GM_RECIPIENT into the GM_CHUNK.

Return Value:	OK if message was sent successfully.
Parameters:	`msg` — GMessage to send. <pre> GM_RECIPIENT GM_DDADDRESS GM_PORT GM_CHUNK </pre>
See Also:	SenderDispatcherThread()
Author:	Pechy

94.6

void **DispatcherThread** (GMessage *msg)

Connect to one of the recipients and send him the message

Connect to one of the recipients and send him the message.

In our implementation, we use threads for despatching msgs. This method is the body of that thread. It expects: <pre> GM_RECIPIENT_1 GM_ADDRESS GM_PORT [GM_RECIPIENT_2 GM_ADDRESS GM_PORT] GM_CHUNK </pre> The chosen recipient is inserted into the GM_CHUNK and that is transmitted over network.

Return Value:	void
Parameters:	`msg` — Message to send
See Also:	Sender (→94, *page 288*)
Author:	Pechy

94.7

GMessage* **MakeMessageForSendingToMix** (GMessage *chunk)

Add recipient to message

Add recipient to message.

Gets chunk and adds to it the GM_RECIPIENT field so that sender knows how to deliver it and where.

Return Value:	GMessage containing the original message with added GM_RECIPIENT field. <pre> GM_RECIPIENT GM_DDADDRESS GM_PORT GM_CHUNK </pre>
Parameters:	`chunk` — Message we want to send to Mix.
See Also:	SenderDispatcherThread()
Author:	Pechy

94.8

void* **Run** (void *)

Generic Sender Run() method

Generic Sender Run() method.

Neverending cycle, in each loop just one message passed from Padder is read and the method DispatchMsg() is called. This method should send message to the apropriate destination. Now DispatchMsg() creates thread for serving this.

Return Value:	never
See Also:	DispatchMsg()
Author:	Pechy

94.9

Sender (LogFile * senderLogFile, Killer * killer, MessageQueue * toSenderQueue, string origin, string IP = "", unsigned short port = 0)

Generic sender constructor

Generic sender constructor.

Remembers all the parameters for future use and initializes its parent classes.

Parameters: `senderLogFile` — Log file for debug messages.
`killer` — Who will notify us about shutdown.
`origin` — What will we insert as GM_ORIGIN into outgoing messages.
`IP` — Where to send messages by default (used when sending from Six to underlying Mix).
`port` — Port number to which to connect.

See Also: Receiver, Mix

Author: Pechy

94.10

void **Shutdown** ()

Prepare for clean shutdown

Prepare for clean shutdown.

This method should implement all the tasks necessary for preparation for a clean shutdown like saving inconsistent data to disk etc. Now it is empty as AddressManger is completely stateless as it stores certificate to disk immediatelly after it is generated.

See Also: Shutdownable, Killer

Author: Venca

94.11

Err **WriteMsgToSocket** (int sock, GMessage *msg)

Write message to connected socket

Write message to connected socket.

Writes body of the message to the socket, the socket must be already created and connected. This method is called from the thread is managing the actual sending.

Return Value: OK if successful else KO.
Parameters: `sock` — Socket to write to.
`msg` — GMessage to send.
See Also: SenderDispatcher()
Author: Pechy

94.12

~**Sender** ()

Generic Sender destructor

Generic Sender destructor.

Empty by now.

See Also: Sender()
Author: Pechy

94.1

attributes

Names

MessageQueue* **toSenderQueue**		*Incoming queue*
string	**origin**	*What set as GM_ORIGIN*
string	**IP**	*IP address to send by default*
unsigned short **port**		*Port to send by default*

94.2 methods

Names

unsigned long
ConvertToNumericIP (char *addr)
string -> long representation

Err **WriteMsgToSocket** (int sock, GMessage *msg)
Feed the raw data to socket

95

class **ArgsForDispatcherThread**

Thread that dispatches message

Public Members

Thread that dispatches message.

contains all args needed by function that is an argument in pthread_create and that takes care of transmitting one message to network. <p> Description: We need a class below for arguments that we need pass to a function for phthread_create. We have to use an auxiliary function and then to call the sender->DispatchMsg() inside that function.

See Also: Sender Receiver
Author: Pechy

95.1

attributes

Names

Sender*	**sender**	*Pointer to parent*
GMessage*	**msg**	*Gmessage to send*

95.2

methods

Names

ArgsForDispatcherThread ()
Constructor

~ArgsForDispatcherThread ()
Destructor

96

void* **SenderDispatcherThread** (void *args)

a prototype that will be passed to pthread_create()

97

#define **RECIPIENT_COUNT**

how many different recipients are there in one chunk?

98

void* **SenderDispatcherThread** (void *args)

Call Sender's method for dispatching one message

Call Sender's method for dispatching one message.

Works around the problem with passing method to pthread_create() by encapsulating this method into a normal function.

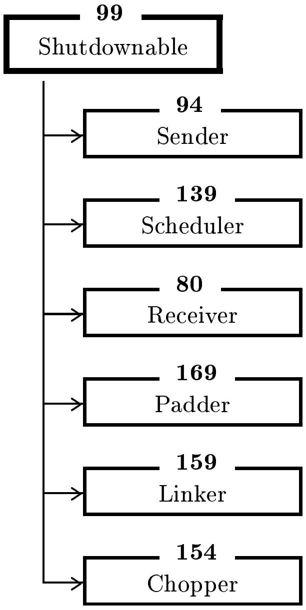
Return Value:	NULL - not used
Parameters:	`args` — Passed to sender->DispatcherThread()
See Also:	pthread_create(), DispatcherThread()
Author:	Pechy

99

class **Shutdownable**

Shutdownable class

Inheritance



Public Members

Shutdownable class.

This class forces its offsprings to implement method used for clean shutdown. All objects that have to save some info before terminating should inherit from this class

See Also: Killer (→61, *page 225*)
Author: Venca

99.1 methods

Names

Shutdownable (Killer * killer)
Constructor

virtual void
Shutdown () *This method is called by Killer.*

99.2 Shutdownable (Killer * killer)

Shutdownable constructor

Shutdownable constructor.

Reisters with killer so that he can notify us when shutdown comes. That means call our method Shutdown().

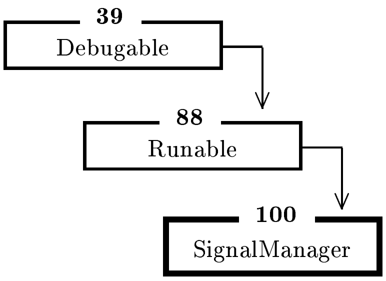
Parameters: `killer` — Killer that will notify us when shutdown comes.
See Also: Killer, ShutDown()
Author: Venca

100

class **SignalManager** : public Runable

Signal manger class

Inheritance



Public Members

Private Members

Signal manger class.

Implements system specific signal handling and dispatching signals to appropriate threads.

See Also: Scheduler Killer
Author: Pechy

100.1 methods

Names

	SignalManager (LogFile * signalManagerLogFile)	*Constructor*
void*	**Run** (void *)	*Main thread method*

100.3 void* **Run** (void *)

The main loop for Signal Manager thread

The main loop for Signal Manager thread.

Implementation of SignalManager thread, this is just a template here, we intercept SIGALRM signal for now only, to show how to do it. <p> Because lacking sigwait() function call, we have to do signal handling using classical handlers, so we twice lock mutex to fall asleep for ever.

Return Value: void *
Parameters: `void` — *
See Also: thread discussion
Author: Pechy

100.4 **SignalManager** (LogFile * signalManagerLogFile)

Signal Manager constructor

Signal Manager constructor.

Just initializes mutex.

Parameters: `signalManagerLogFile` — Log file.
See Also: thread discussion
Author: Pechy

100.5

~SignalManager ()

Signal Manager desstructor

Signal Manager desstructor.

Return Value: co
Parameters: `parametr` —
See Also: SignalManager()
Author: jmeno

100.2

attributes

Names

pthread_mutex_t
mutex *Synchronization*

101

void **processSIGALRM** (int)

Handler for SIGALRM signal

Handler for SIGALRM signal.

Because of lack of sigwait() call in thread library in former versions of FreeBSD, we have to use signal handlers. This should be changed in the future.

Return Value:	void
Parameters:	`int` —
See Also:	thread discussion, Shutdownable
Author:	Pechy

102

void **processSIGHUP** (int)

Handler for SIGQUIT signal Because of lack of sigwait() call in thread library in former versions of FreeBSD, we have to use signal handlers

Handler for SIGQUIT signal

Because of lack of sigwait() call in thread library in former versions of FreeBSD, we have to use signal handlers. This should be changed in the future.

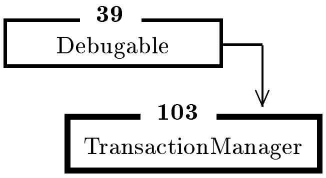
Return Value:	void
Parameters:	`int` —
See Also:	processSIGALRM()
Author:	Pechy

103

class **TransactionManager** : public Debugable

Transaction manager class

Inheritance



Public Members

103.2		**methods**	309
103.3	Err	**BeginTransaction** (TID *tid) *Kratky_komentar_s_teckou_na_konci*	310
103.4	Err	**EndTransaction** (TID tid) *Kratky_komentar_s_teckou_na_konci*	310
103.5	TID	**GenerateTID** () *Kratky_komentar_s_teckou_na_konci*	311
103.6	Err	**RegisterRollbackable** (Rollbackable *rbObject) *Kratky_komentar_s_teckou_na_konci*	311
103.7	Err	**Rollback** (TID tid) *Kratky_komentar_s_teckou_na_konci*	311
103.8		**TransactionManager** (Char *transLogName, LogFile *aLogFile) *Kratky_komentar_s_teckou_na_konci*	312
103.9	Err	**WriteToLog** (TID tid, TransactionState state) *Kratky_komentar_s_teckou_na_konci*	312
103.10		**~TransactionManager** ()	

Protected Members

Transaction manager class.

makes a log of transactions and enables some classes to be able to Roll back.

See Also: Rollbackable (→87, *page 278*)
Author: Pavel

103.2 methods

Names

CheckTransactionLog ()
goes through the transaction log and rolls back all not commited transactions

Rollback (TID tid)
calls Rollback(tid) for all rollbackable objects

Err **RegisterRollbackable** (Rollbackable *rbObject)
registers an rollbackable object

Err **BeginTransaction** (TID *tid)
returns new TID and writes to Transaction Log

Err **EndTransaction** (TID tid)
writes to Transaction Log

103.3

Err **BeginTransaction** (TID *tid)

Kratky_komentar_s_teckou_na_konci

Kratky_komentar_s_teckou_na_konci.

Dlouhy_komentar

Return Value: co
Parameters: `parametr` —
See Also: co_se_toho_tyka
Author: Pavel

103.4

Err **EndTransaction** (TID tid)

Kratky_komentar_s_teckou_na_konci

Kratky_komentar_s_teckou_na_konci.

Dlouhy_komentar

Return Value: co
Parameters: `parametr` —
See Also: co_se_toho_tyka
Author: Pavel

103.5

TID **GenerateTID** ()

Kratky_komentar_s_teckou_na_konci

Kratky_komentar_s_teckou_na_konci.

Dlouhy_komentar

Return Value:	co
Parameters:	`parametr` —
See Also:	co_se_toho_tyka
Author:	Pavel

103.6

Err **RegisterRollbackable** (Rollbackable *rbObject)

Kratky_komentar_s_teckou_na_konci

Kratky_komentar_s_teckou_na_konci.

Dlouhy_komentar

Return Value:	co
Parameters:	`parametr` —
See Also:	co_se_toho_tyka
Author:	Pavel

103.7

Err **Rollback** (TID tid)

Kratky_komentar_s_teckou_na_konci

Kratky_komentar_s_teckou_na_konci.

Dlouhy_komentar

Return Value: co
Parameters: `parametr` —
See Also: co_se_toho_tyka
Author: Pavel

103.8

TransactionManager (Char *transLogName, LogFile *aLogFile)

Kratky_komentar_s_teckou_na_konci

Kratky_komentar_s_teckou_na_konci.

Dlouhy_komentar

Return Value: co
Parameters: `parametr` —
See Also: co_se_toho_tyka
Author: Pavel

103.9

Err **WriteToLog** (TID tid, TransactionState state)

Kratky_komentar_s_teckou_na_konci

Kratky_komentar_s_teckou_na_konci.

Dlouhy_komentar

Return Value: co
Parameters: `parametr` —
See Also: co_se_toho_tyka
Author: Pavel

103.10

~**TransactionManager** ()

Kratky_komentar_s_teckou_na_konci

Kratky_komentar_s_teckou_na_konci.

Dlouhy_komentar

Return Value:	co
Parameters:	`parametr` —
See Also:	co_se_toho_tyka
Author:	Pavel

103.1

methods

Names

TID	**GenerateTID** ()	*generates new transaction ID*
Err	**WriteToLog** (TID tid, TransactionState state)	*writes to transactionLog - must be open*
Err	**ReadFromLog** (TID *tid, TransactionState *state)	*reads from transactionLog - must be open*

104

class **UniqueIDGenerator**

Unique ID Generator

Public Members

Protected Members

Unique ID Generator.

This class is responsible for generating unique ID's even in the case of an unexpected termination of the program.

Author: Pavel

104.3

methods

Names

	UniqueIDGenerator (Char *anIDTableName, ID startAt = 0)	*create ID Generator at the given path, starting with value startAt*
Err	**GetID** (ID *newID)	*get unique ID*

104.4

Err CreateDefinition (TableFieldQueue *definition)

Kratky_komentar_s_teckou_na_konci

Kratky_komentar_s_teckou_na_konci.

Dlouhy_komentar

Return Value: co
Parameters: `parametr` —
See Also: RandomGenerator (→78, *page 262*)
Author: Pavel

104.5

Err GetID (ID *newID)

Get unique ID

Get unique ID.

Dlouhy_komentar

Return Value: co
Parameters: `parametr` —

See Also: RandomGenerator (→78, *page 262*)
Author: Pavel

104.6

Err **InitLastIDFile** (Char *anIDTableName)

Kratky_komentar_s_teckou_na_konci

Kratky_komentar_s_teckou_na_konci.

Dlouhy_komentar

Return Value: co
Parameters: `parametr` —
See Also: RandomGenerator (→78, *page 262*)
Author: Pavel

104.7

UniqueIDGenerator (Char *anIDTableName, ID startAt)

Unique ID generator constructor

Unique ID generator constructor.

Dlouhy_komentar

Parameters: `anIDTableName` —
`startAt` —
See Also: RandomGenerator (→78, *page 262*)
Author: Pavel

104.1 attributes

Names

pthread_mutex_t **mutex** *Synchronization*

104.2 methods

Names

Err **CreateDefinition** (TableFieldQueue *definition)
create table definition for lastIDTable

Err **InitLastIDFile** (Char *lastIDTableName)
initialize the lastIDTable file

105

Table* **CreateEsoToClientTable** (Char *fileName)

Creates a table of payment plan sent by Eso to Client

Creates a table of payment plan sent by Eso to Client.

Creates a table definition and then the table of payment plan sent by Eso to Client.

Return Value: Pointer to generated table.
Parameters: `fileName` — File name in which to store table.
See Also: Banker, Eso, Client
Author: Ivanka

106

Table* **CreateClientToBankTable** (Char *fileName)

Creates a table of payment plan sent by Client to Bank

Creates a table of payment plan sent by Client to Bank.

Creates a table definition and then the table of payment plan sent by Client to Bank.

Return Value:	Pointer to generated table.
Parameters:	`fileName` — File name in which to store table.
See Also:	Banker, Eso, Client
Author:	Ivanka

107

int **mixMemCat** (char *one, int oneLen, char *two, int twoLen, char **result)

Join two memory areas into one

Join two memory areas into one.

Allocates memmory and copies the two regions in it.

Return Value: allocated memory that contains copies of the two regions

Parameters:
`one` — first region of memory
`oneLen` — its length
`two` — second memory region
`twoLe` — its lengthn

See Also: MessageCreator, Cipherer, Translator

Author: Pechy

108

char* **AddSizeHeaderAsDataSize** (char *data, Size dataLen)

Encapsulates data in a siple way

Encapsulates data in a siple way.

Take the data and add a header indicating the lenght of data part, the header is of type Size.

Return Value: allocated memory that contains the header and the data

Parameters: `data` — data we want to act on
`dataLen` — lenght of data

See Also: MessageCreator, Cipherer, Translator, ExtractDataFromSizeHeaderAnnotatedStream()

Author: Pechy

109

char* **ExtractDataFromSizeHeaderAnnotated-Stream** (char *dataIn, Size dataInLen, Size *dataOutLen)

Extracts data

Extracts data.

Extracts data from the construct made by AddSizeHeaderAsDataSize()

Return Value: allocated memory that contains the data of header length

Parameters: `dataIn` — data we want to extract something from
`dataInLen` — lenght of dataIn
`dataOutLen` — the len of returned data

See Also: MessageCreator, Cipherer, Translator, AddSizeHeaderAsDataSize()

Author: Pechy

110

Table* **CreateEsoToClientTable** (Char *fileName)

Creates a table of payment plan sent by Eso to Client

Creates a table of payment plan sent by Eso to Client.

Creates a table definition and then the table of payment plan sent by Eso to Client.

Return Value:	Pointer to generated table.
Parameters:	`fileName` — File name in which to store table.
See Also:	Banker, Eso, Client
Author:	Ivanka

111

Table* **CreateClientToBankTable** (Char *fileName)

Creates a table of payment plan sent by Client to Bank

Creates a table of payment plan sent by Client to Bank.

Creates a table definition and then the table of payment plan sent by Client to Bank.

Return Value:	Pointer to generated table.
Parameters:	`fileName` — File name in which to store table.
See Also:	Banker, Eso, Client
Author:	Ivanka

112

include . .. H common.h int **mixMemCat** (char *one, int oneLen, char *two, int twoLen, char **result)

Join two memory areas into one

Join two memory areas into one.

Allocates memmory and concatenates the two regions in it.

Return Value: allocated memory that contains copies of the two regions

Parameters:
`one` — first region of memory
`oneLen` — its length
`two` — second memory region
`twoLe` — its lengthn

See Also: MessageCreator, Cipherer, Translator

Author: Pechy

113

char* **ExtractDataFromSizeHeaderAnnotatedStream** (char *data, Size dataLen, Size *dataOutLen)

Encapsulates data in a siple way

Encapsulates data in a siple way.

Take the data and add a header indicating the lenght of data part, the header is of type Size.

Return Value: allocated memory that contains the header and the data

Parameters: `data` — data we want to act on
`dataLen` — lenght of data

See Also: MessageCreator, Cipherer, Translator, ExtractDataFromSizeHeaderAnnotatedStream()

Author: Pechy

114

char* **AddSizeHeaderAsDataSize** (char *data, Size dataLen)

Extracts data

Extracts data.

Extracts data from the construct made by AddSizeHeaderAsDataSize()

Return Value: allocated memory that contains the data of header length

Parameters:
`dataIn` — data we want to extract something from
`dataInLen` — lenght of dataIn
`dataOutLen` — the len of returned data

See Also: MessageCreator, Cipherer, Translator, AddSizeHeaderAsDataSize()

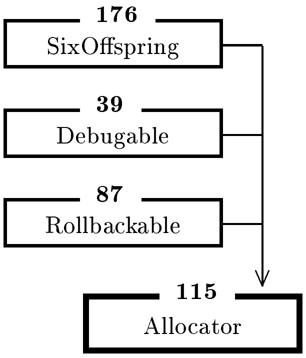
Author: Pechy

115

class **Allocator** : public Rollbackable, public Debugable, public SixOffspring

Class for allocating space on a disk

Inheritance



Public Members

115.3 **methods** 329

115.4 Err **AllocSpace** (TID tID, Size fileSize)
Allocates space 330

115.5 Err **AllocTableFind** (const TID tID, const Int shouldClose, Size *fileSize, Char **fileName)
Searches allocTable 331

115.6 Err **AllocTableInsert** (const TID tID, const Size fileSize, const Char *fileName)
Insert into allocTable 331

115.7 Err **AllocValid** (TID tID, Size fileSize)
Is allocation valid? 332

115.8 **Allocator** (Eso *anEso, LogFile *allocLogFile)
Constructor 332

115.9 Err **CreateAllocFile** (const Size fileSize, Char **anAllocFileName)

Protected Members

Class for allocating space on a disk.

This class is responsible for allocating space on the disk, when request to store a file arrives to Eso. It stores all information about allocations in a table with this structure:
(TID: ID, FileSize: Size, FileName: String).

See Also: Eso (→131, *page 375*)
Author: Pavel

115.3

methods

Names

Allocator (Eso *anEso, LogFile *allocLogFile)
Constructor.

~Allocator () *Destructor.*

Err **AllocSpace** (TID tID, Size fileSize)
Tries to allocate space for a specified time

Err **AllocValid** (TID tID, Size fileSize)

		Tests if there is an allocation (identified by tID) with greater or equal size to fileSize
Err	**FreeSpace** (TID tID)	*Frees specified allocation space.*
virtual Err	**Rollback** (TID tID)	*Rollbacks allocation created in tID transaction.*
void	**Test** ()	*A test function.*

115.4

Err **AllocSpace** (TID tID, Size fileSize)

Allocates space

Allocates space.

Tries to allocate space for a specified time

Return Value: error code
Parameters: `tID` — transaction ID
`fileSize` — size of allocated file
See Also: CreateAllocFile(), AllocTableInsert()
Author: Pavel

115.5

Err **AllocTableFind** (const TID tID, const Int shouldClose, Size *fileSize, Char **fileName)

Searches allocTable

Searches allocTable.

Finds first record with tID and returns fileSize and fileName, if desired. "shouldClose" says wether to close allocTable again.

Return Value:	error code
Parameters:	`tID` — transaction ID `shouldClose` — tells if the table should be closed after a successfull search `fileSize` — size of found file `fileName` — name of found file
See Also:	Table
Author:	Pavel

115.6

Err **AllocTableInsert** (const TID tID, const Size fileSize, const Char *fileName)

Insert into allocTable

Insert into allocTable.

Inserts a record to table of allocations

Return Value:	error code
Parameters:	`tID` — transaction ID `fileSize` — size of allocated file `fileName` — name of allocated file
See Also:	Table, Record
Author:	Pavel

115.7

Err **AllocValid** (TID tID, Size fileSize)

Is allocation valid?

Is allocation valid?.

Tests if there is an allocation (identified by tID) with greater or equal size to fileSize, which was not yet timed out.

Return Value:	error code
Parameters:	`tID` — transaction ID
	`fileSize` — checked size of file
See Also:	AllocTableFind
Author:	Pavel

115.8

Allocator (Eso *anEso, LogFile *allocLogFile)

Constructor

Constructor.

Creates allocation table and links to its parent Eso

Parameters:	`anEso` — pointer to Eso owning this class
	`allocLogFile` — pointer to log file where Allocator should log its work
See Also:	Eso, Majordomo, SixOffspring, Debugable
Author:	Pavel

115.9

Err **CreateAllocFile** (const Size fileSize, Char **anAllocFileName)

Creates allocation file

Creates allocation file.

Creates a dummy file in ALLOC_FILES_PATH directory of given size

Return Value:	error code
Parameters:	`fileSize` — size of file to allocate on disk `anAllocFileName` — out parameter that tells the name of newly created allocation file
See Also:	RandomGenerator, CreateTempFileName()
Author:	Pavel

115.10

TableFieldQueue* **CreateDefinition** ()

Creates definition of allocTable

Creates definition of allocTable.

Creates the definition of allocation table by inserting all needed fields in it.

Return Value:	definition of allocation table
See Also:	Table, TableField
Author:	Pavel

115.11

Err FreeSpace (TID tID)

Frees space

Frees space.

Frees specified allocation space

Return Value:	error code
Parameters:	`tID` — transaction ID
See Also:	AllocTableFind(), Table
Author:	Pavel

115.12

Err Rollback (TID tID)

Rolls back a transaction

Rolls back a transaction.

Rollbacks allocation created in tID transaction

Return Value:	error code
Parameters:	`tID` — transaction ID
See Also:	Rollbackable, FreeSpace()
Author:	Pavel

115.13

void Test ()

Allocator test

Allocator test.

Test method used only for debugging

Return Value: void
Author: Pavel

115.14

~Allocator ()

Destructor

Destructor.

Destroyes allocation table

See Also: Table
Author: Pavel

115.1

attributes

Names

Table*	**allocTable**	*Used to store information about allocated space.*

115.2

methods

Names

TableFieldQueue* **CreateDefinition** ()
Creates definition of allocTable - used only in constructor.

Err **CreateAllocFile** (const Size fileSize, Char **allocFileName)
Creates a dummy place holder file.

Err **AllocTableInsert** (const TID tID, const Size fileSize, const Char *fileName)
Inserts information about allocated space to allocTable.

Err **AllocTableFind** (const TID tID, const Int shouldClose = 1, Size *fileSize = NULL, Char **allocFileName = NULL)
Searches allocTable by transaction ID, enables not closing the table after the search

116

BankerDefines

Names

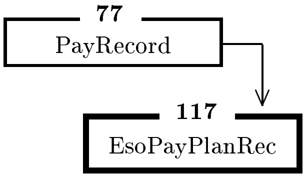
		banker's files dir
	bankers' files	
	table fields' names	
#define	**GM_STATE**	*gMessage fields' names*
		states of 'current payment'
	states of 'current payment'	

117

class **EsoPayPlanRec** : public PayRecord

Eso Payment Plan Record

Inheritance



Public Members

methods

117.1 **EsoPayPlanRec** (ID payId, TID tId, ESTime time, GMessage *bankAC, Char *amount, Size lenOfSA)
Eso Payment Plan Record Constructor 339

117.2 Err **FetchFromRecord** (Record *)
Fetches from given record 340

117.3 Err **FillRecord** (Record *r)
Fills given record 340

117.4 include . .. .. Common H labels.h Err **GenerateSAuth** (Size lengthOfSA)
Generate SAuthentification 340

117.5 Err **SealSAuth** () *Seals SAuthentification* 341

117.6 **~EsoPayPlanRec** ()
Eso Payment Plan Record destructor 341

Protected Members

attributes

methods

Eso Payment Plan Record.

An auxiliary class for one Payment Plan record in Eso.

See Also: Banker (→120, *page 355*)
Author: Ivana

117.1

EsoPayPlanRec (ID payId, TID tId, ESTime time, GMessage *bankAC, Char *amount, Size lenOfSA)

Eso Payment Plan Record Constructor

Eso Payment Plan Record Constructor.

Eso Payment Plan Record Constructor sets class variables. Generates SAuthentification and 'seals' it.

Parameters:
`payId` — payment ID
`tId` — transaction ID
`time` — time of storrage
`bankAC` — bank access certificate
`amount` — amount of money
`lenOfSA` — length of SAuthentification

See Also: Banker::MakePayPlan()
Author: Ivana

117.2

Err **FetchFromRecord** (Record *)

Fetches from given record

Fetches from given record.

Fetches data from given record. A virtual method, that should be overwritten, to be able to read from table. (We will not use this method.)

Return Value:	error code
Parameters:	`r` — record from which we take data
See Also:	PayRecord (→77, *page 256*)
Author:	Ivana

117.3

Err **FillRecord** (Record *r)

Fills given record

Fills given record.

Fills given record with data. A virtual method, that should be overwritten, to be able to write to or update table.

Return Value:	error code
Parameters:	`r` — record into which we store data
See Also:	PayRecord (→77, *page 256*)
Author:	Ivana

117.4

include . .. .. Common H labels.h Err **GenerateSAuth** (Size lengthOfSA)

Generate SAuthentification

Generate SAuthentification.

Generates SAuthentification - a random string used as a secret key between Eso and Bank.

Return Value: error code
Parameters: `lengthOfSA` — length of generated SAuth
See Also: EsoPayPlanRec()
Author: Ivana

117.5

Err **SealSAuth** ()

Seals SAuthentification

Seals SAuthentification.

Encrypts SAuthentification with bank's public key.

Return Value: error code
Parameters: `void` —
See Also: EsoPayPlanRec()
Author: Ivana

117.6

~EsoPayPlanRec ()

Eso Payment Plan Record destructor

Eso Payment Plan Record destructor.

Eso Payment Plan Record destructor deletes class variables.

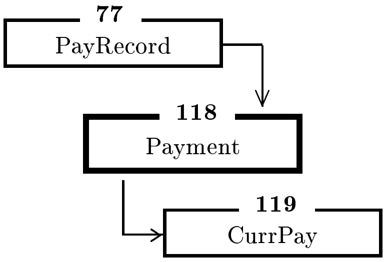
See Also: Banker::MakePayPlan()
Author: Ivana

118

class **Payment** : public PayRecord

Payment

Inheritance



Public Members

methods

118.1	Void	**Clear** ()	*Clear data*	343
118.2	Err	**FetchFromRecord** (Record *r)	*Fetches from given record*	343
118.3	Err	**FillRecord** (Record *r)	*Fills given record*	344
118.4	Char*	**GetAmount** ()	*Get Amount*	344
118.5	ID	**GetID** (Char *whichId)	*Returns requested ID*	345
118.6	ESTime	**GetTime** ()	*Returns time field*	345
118.7		**Payment** ()	*Payment constructor*	345
118.8	Err	**SetID** (Char *whichId, ID value)	*Sets given ID*	346
118.9		**~Payment** ()	*Payment destructor*	346

Protected Members

attributes

Payment.

An auxiliary class for one Payment record in Eso.

See Also:
Author: Ivana

118.1

Void **Clear** ()

Clear data

Clear data.

Clears data - including eCode.

Return Value: void
See Also: Banker::SchedulePays()
Author: Ivana

118.2

Err **FetchFromRecord** (Record *r)

Fetches from given record

Fetches from given record.

Fetches data from given record. A virtual method, that should be overwritten, to be able to read from table.

Return Value: error code
Parameters: `r` — record from which we take data

See Also: PayRecord (→77, *page 256*)
Author: Ivana

118.3

Err **FillRecord** (Record *r)

Fills given record

Fills given record.

Fills given record with data. A virtual method, that should be overwritten, to be able to write to or update table.

Return Value: error code
Parameters: `r` — record into which we store data
See Also: PayRecord (→77, *page 256*)
Author: Ivana

118.4

Char* **GetAmount** ()

Get Amount

Get Amount.

Returns our amount field in new string.

Return Value: copy of amount
See Also: Banker::SchedulePays()
Author: Ivana

118.5

ID **GetID** (Char *whichId)

Returns requested ID

Returns requested ID.

Returns 'whichId' ID value or BAD_ID.

Parameters: `whichId` — name of ID we want to get
See Also: Banker::SchedulePays()
Author: Ivana

118.6

ESTime **GetTime** ()

Returns time field

Returns time field.

Returns value of time field.

See Also: Banker::SchedulePays()
Author: Ivana

118.7

Payment ()

Payment constructor

Payment constructor.

Payment constructor initializes class variables.

See Also: Banker::SchedulePays()
Author: Ivana

118.8

Err **SetID** (Char *whichId, ID value)

Sets given ID

Sets given ID.

Sets 'whichId' ID value.

Return Value: error code
Parameters: `whichId` — name of ID we want to set
See Also: Banker::SchedulePays()
Author: Ivana

118.9

~Payment ()

Payment destructor

Payment destructor.

Payment destructor deletes class variables.

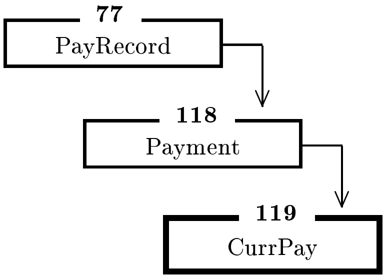
See Also: Banker::SchedulePays()
Author: Ivana

119

class **CurrPay** : public Payment

Current Payment

Inheritance



Public Members

		methods		
119.1		**CurrPay** ()	*Current Payment Costructor* ...	348
119.2	Err	**FetchFromRecord** (Record *r)	*Fetches from given record*	349
119.3	Err	**FillRecord** (Record *r)	*Fills given record*	349
119.4	MsgField*	**GetAuth** (Char *whichAuth)	*Returns requested authentification*	350
119.5	ID	**GetID** (Char *whichId)	*Returns requested ID*	350
119.6	MsgField*	**GetMAC** ()	*Get MAC*	351
119.7	Int	**GetState** ()	*Get State Of Current Payment* .	351
119.8	Int	**NumOfTries** (Char operation)	*Get number of tries*	351
119.9	Err	**SetAuth** (Char *whichAuth, MsgField *auth)	*Sets authentification*	352
119.10	Err	**SetID** (Char *whichId, ID value)		

Protected Members

attributes

Current Payment.

An auxiliary class for one currently taken payment in Eso.

See Also:
Author: Ivana

119.1

CurrPay ()

Current Payment Costructor

Current Payment Costructor.

Current Payment constructor initializes class variables.

See Also: Banker::TimeForPayment(), Banker::ChallengeFromBank(), Banker::AnswerFromBank(), Banker::TimeOutPayment(),
Author: Ivana

119.2

Err **FetchFromRecord** (Record *r)

Fetches from given record

Fetches from given record.

Fetches data from given record. A virtual method, that should be overwritten, to be able to read from table. (We will not use this method.)

Return Value:	error code
Parameters:	`r` — record from which we take data
See Also:	PayRecord (→77, *page 256*)
Author:	Ivana

119.3

Err **FillRecord** (Record *r)

Fills given record

Fills given record.

Fills given record with data. A virtual method, that should be overwritten, to be able to write to or update table.

Return Value:	error code
Parameters:	`r` — record into which we store data
See Also:	PayRecord (→77, *page 256*)
Author:	Ivana

119.4

MsgField* **GetAuth** (Char *whichAuth)

Returns requested authentification

Returns requested authentification.

Returns 'whichAuth' value or NULL.

Parameters: `whichId` — name of auhtentification we want to get
See Also: Banker::GenerateAnswer()
Author: Ivana

119.5

ID **GetID** (Char *whichId)

Returns requested ID

Returns requested ID.

Returns 'whichId' ID value or BAD_ID.

Parameters: `whichId` — name of ID we want to get
See Also: Banker (→120, *page 355*)
Author: Ivana

119.6

MsgField* **GetMAC** ()

Get MAC

Get MAC.

Returns our mac field in new MsgField.

Return Value: copy of mac
See Also: Banker::GenerateAnswer()
Author: Ivana

119.7

Int **GetState** ()

Get State Of Current Payment

Get State Of Current Payment.

Returns a value of state field.

Return Value: state value
See Also: Banker (→120, *page 355*)
Author: Ivana

119.8

Int **NumOfTries** (Char operation)

Get number of tries

Get number of tries.

If the operation is 'plus', first increase numOfTries and then return it. If the operation is 'one', sets numOfTries to 1, else return numOfTries.

Parameters: `operation` — which operation should be done before return numOfTries
See Also: Banker::PartialUpdateCurrPays(), Banker::TimeOutCurrPay(), Banker::TimeOutPayment()
Author: Ivana

119.9

Err **SetAuth** (Char *whichAuth, MsgField *auth)

Sets authentification

Sets authentification.

Sets 'whichAuth' authentification with new value.

Return Value: error code
Parameters: `whichId` — name of auhtentification we want to set
`auth` — a new value for the auhtentification
See Also: Banker::ChallengeOK()
Author: Ivana

119.10

Err **SetID** (Char *whichId, ID value)

Sets given ID

Sets given ID.

Sets 'whichId' ID value.

Return Value: error code
Parameters: `whichId` — name of ID we want to set
See Also: Banker (→120, *page 355*)

Author: Ivana

119.11

Err **SetMAC** (MsgField *mac)

Set MAC

Set MAC.

Sets our MAC to given one.

Return Value: error code
Parameters: `mac` — new mac for setting our data
See Also: Banker::ChallengeOK()
Author: Ivana

119.12

Err **SetState** (Int st)

Set State Of Current Payment

Set State Of Current Payment.

Set state of current payment field with given value.

Return Value: OK if st is ANSWER or REQUEST, KO otherwise
Parameters: `st` — a new value for the state
See Also: Banker::ChallengeOK()
Author: Ivana

119.13

Err **WriteToTable** (Table *t, Char *reason)

Writes record to given table

Writes record to given table.

Writes (inserts) a record to given table (it does not change eCode variable).

Parameters: `t` — table to which we will write new record
`reason` — reason of fail - an additional parametr for filling 'Reason-Of-Fail' field in some tables

See Also: Banker::TimeForPayment(), Banker::TimeOutPayment(), Banker::ChallengeOK(), Banker::ChallengeKO(), Banker::AnswerKO(), Banker::TimeOutCurrPay(),

Author: Ivana

119.14

˜CurrPay ()

Current Payment Destructor

Current Payment Destructor.

Current Payment destructor deletes class variables.

See Also: Banker::TimeForPayment(), Banker::ChallengeFromBank(), Banker::AnswerFromBank(), Banker::TimeOutPayment(),

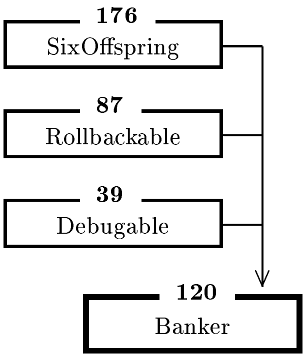
Author: Ivana

120

class **Banker** : public Debugable, public Rollbackable, SixOffspring

Banker

Inheritance



Public Members

methods

120.2 GMessage* **DecideWhichBank** ()
Chooses a bank 357

120.3 Void **DelPayPlan** (TID tId)
Deletes Payment Plan with given TID 357

120.4 Void **DeleteRecsFileID** (ID fileId)
Deletes all records with given file ID 358

120.5 Err **DeleteRecsInTable** (Table *t, Char *whichId, ID id)
Deletes all records in the table .. 358

120.6 Void **DeleteRecsPayID** (ID payId)

Protected Members

methods

Banker.

A class which manages everything about payments in Eso:

Makes Payment Plans for Clients Has control over all payment transactions between Bank and Eso (sends request for payment and reacts to a challenge from Bank)

Author: Ivana

120.2

GMessage* **DecideWhichBank** ()

Chooses a bank

Chooses a bank.

Decide which bank we use when generating PayPlan. Now randomly selects a bank AC (later it should be more sophisticated).

Return Value: bank access certificate
See Also: MakePayPlan()
Author: Ivana

120.3

Void **DelPayPlan** (TID tId)

Deletes Payment Plan with given TID

Deletes Payment Plan with given TID.

Deletes Payment Plan with given TID. The reason of it may be: first, client didn't send data (timeout from Scheduller), second, client sent "wrong" data (too big, ...), third, eso couldn't store sent file.

Return Value:	void
Parameters:	`tId` — transaction ID corresponding to the payment plan
See Also:	Majordomo::DRCDataToStore()
Author:	Ivana

120.4

Void **DeleteRecsFileID** (ID fileId)

Deletes all records with given file ID

Deletes all records with given file ID.

If administrator wants to delete a file (the reason can be, that there are no payments for it), this method is used. It deletes all records (in all tables) with given file ID.

Return Value:	void
Parameters:	`fileId` — ID of deleted file records
See Also:	DeleteRecsPayID
Author:	Ivana

120.5

Err **DeleteRecsInTable** (Table *t, Char *whichId, ID id)

Deletes all records in the table

Deletes all records in the table.

Deletes all records in the table with given ID. If whichId is not specified (NULL), clears given table.

Return Value: error code
Parameters: `t` — table in which we will delete
`whichId` — specifies ID, that is used for deleting
`id` — a value of the ID
See Also:
Author: Ivana

120.6

Void **DeleteRecsPayID** (ID payId)

Deletes all records with file ID which correspond to given payment ID

Deletes all records with file ID which correspond to given payment ID.

If administrator wants to delete a file, which correspond to given payment ID (the reason can be, that there are no payments for it), this method is used. It deletes all records (in all tables) with file ID which correspond to given payment ID.

Return Value: void
Parameters: `payId` — ID of payment that correspond to deleted file records
See Also: DeleteRecsFileID()
Author: Ivana

120.7

Char* **FillTableForBank** (Record **r, Table *bankTmpTab)

Fills given table with information about payments

Fills given table with information about payments.

From each record from table PayPlan with given bankACID and transaction ID strips TID and writes it to the table for bank.

Return Value: total sum of money to be paid to the bank
Parameters: `r` — initialize record, in which we find useful information
`bankTable` — table for bank
See Also: PrepareMsgsForBanks()
Author: Ivana

120.8

Record* **FindAndDelInTable** (Table *t, Char *whichId, ID id)

Finds and deletes record in the table

Finds and deletes record in the table.

Finds and deletes record in the table with given ID.

Return Value: found record or NULL
Parameters: `t` — table in which we will search
`whichId` — specifies ID, that is used for searching
`id` — a value of the ID
See Also: PrepareMsgsForBanks()
Author: Ivana

120.9

GMessage* **GenerateAnswer** (CurrPay *p)

Generates message for bank

Generates message for bank.

Generates answer for bank, including payId, MAC, SAuthentification.

Return Value: message for bank
Parameters: `p` — current payment structure
See Also: ChallengeOK(), TimeOutCurrPay()
Author: Ivana

120.10

GMessage* **GenerateMsgForBank** (GMessage *bankAC, Char *sum, Table *bankTab)

Generate message for bank

Generate message for bank.

Generate message for bank including bank access certificate, total sum of money to be paid and payment plan.

Return Value: message for bank or NULL
See Also: PrepareMsgsForBanks()
Author: Ivana

120.11

Err **GeneratePayId** (ID *payId)

Generates Pay ID

Generates Pay ID.

Generates unique payment ID.

Return Value: error code
Parameters: `payId` — return parametr
See Also: MakePayPlan()
Author: Ivana

120.12

GMessage* **GenerateRequest** (CurrPay *p)

Generates message for bank

Generates message for bank.

Generates request for bank including payId.

Return Value: message for bank
Parameters: `p` — current payment structure
See Also: TimeForPayment(), TimeOutCurrPay()
Author: Ivana

120.13

Err **MakePPlanForBanks** (MsgField *pPlanFld)

Makes Payment Plans For Banks

Makes Payment Plans For Banks.

Receiving Eso's payment plan, finishes it (by generating oAuths and computing MAC for each record) and stores it with current transaction ID.

Return Value: error code
Parameters: `pPlanFld` — msgField of payment plan
See Also: Majordomo::StoreFile()
Author: Ivana

120.14

MsgField* **MakePayPlan** (Size size, RelTime timeToStore, TID tId)

Generates payment plan for a client

Generates payment plan for a client.

First it check system time. If it is OK, count price for storrage and for each pay period chooses bank and generate payment record. After generating all pay records returns whole payment plan as new msgField.

Return Value: payment plan as a new msgField
Parameters: `size` — size of file, which will be stored
`timeToStore` — period for which is file stored
`tId` — current transactoin ID
See Also: Majordomo::DRCRequestForStorage()
Author: Ivana

120.15

Err **PartialUpdateCurrPays** (CurrPay *p)

Updates table Current Payments

Updates table Current Payments.

Updates a record in the table CurrentPayments with new TID and NumOfTries.

Return Value: error code
Parameters: `p` — current pay record
See Also: TimeOutCurrPay()
Author: Ivana

120.16

Queue* **PrepareMsgsForBanks** (TID oldTID)

Prepares queue of mesages for banks

Prepares queue of mesages for banks.

Finds all records with given TID. For each bank found in these records generate message including payment plan.

Return Value: queue of messages for banks
Parameters: `oldTID` — transaction ID with which the payments are connected
See Also: Majordomo (→160, *page 468*)
Author: Ivana

120.1

attributes

Names

Table*	**pPlans**	*bankers' persistent tables:*
Table*	**clientPPlan**	*generated payment plans*
Table*	**payments**	*pay plan that will be sent to client*
Table*	**currPays**	*all payments that should be done*
Table*	**notPaidPays**	*all payments that are carried out right now*
		payments that haven't been paid

121

Err **Majorodomo::DRCChallengeFromBank** (MsgField *dataInMsgFld)

DRCChallengeFromBank (cmd be_chall) Dlouhy_komentar

122

Information needed during Eso initialization

Names

#define **CFG_ESO_PATH**
Home directory for Eso server

#define **CFG_INITIALIZE**
Initialization flag

123

Max number of tries to get money from bank

Names

#define **CFG_FINDER_SEARCH_TIME_FOR_1_HOP**
Constants from cfg file

124

#define **ESOOTHERS_PROTOCOL_VERSION_BETA**

Version of protocol

125 Timeouts

Names

#define **TO_ALLOC_REL_TIMEOUT**
Time out period specification

#define **TO_BANKER_REL_TIMEOUT**
Time out period specification for Banker

126

Classes' paths relative to Eso path

Names

#define **ESO_PATH_ACMANAGER**
Path for data of ACManager

#define **ESO_PATH_ALLOCATOR**
Path for data of Allocator

127

ACManagers' tables

Names

#define **ACMGR_TABLE_BANK**

Path where Bank AC is stored

128

Eso timeout commands

Names

#define	**TO_ALLOC**	*Allocation timed out.*
#define	**TO_DEL_FINDER_TBL**	*Finder's forwarding tables obsolete.*
#define	**TO_CHECK_AC_COUNT**	*Time to check count of ACs of certian type.*
#define	**TO_PAYMENT**	*Time to ask a bank for money.*
#define	**TO_CURR_PAY**	*???.*
#define	**TO_TIME_TO_SYNCHRONIZE**	*It is time to start synchronization.*
#define	**TO_RESPONSE_TO_SYNCHRONIZE**	*Response to synchronization messages timed out.*

129

Eso synchronization commands

Names

#define **CMD_SYNCHRONIZER_REQUEST**
Request for synchronization.

#define **CMD_SYNCHRONIZER_RESPONSE**
Response to request for synchronization.

130

Commands from and to Service

Names

#define **CE_REQ_FOR_STOR**
Client's request for storage of a file.

#define **EC_REPLY_ALLOC**
Eso's reply to CE_REQ_FOR_STOR.

#define **CE_DATA_TO_STOR**
Client sending data to store.

#define **EC_REPLY_STOR**
Eso's reply to CE_DATA_TO_STOR .

#define **CE_REQ_FOR_FIND**
Client's search by keywords.

#define **EC_REPLY_HEADER**
Eso's reply to CE_REQ_FOR_FIND.

#define **CE_REQ_FOR_FILE**
Client's request to download a file.

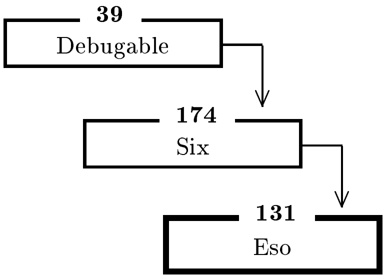
#define **EC_REPLY_FILE**
Eso sending a file.

131

class **Eso** : public Six

Main class that represents the whole Eso service

Inheritance



Public Members

Main class that represents the whole Eso service.

This class inherits its functionality from Six, so its capable of communicating with Mix servers. It holds all classes needed to run the storage service of ES.

See Also: Six, MessageQueu
Author: Venca

131.1 attributes

Names

TCBWrapper* **tcbWrapper** *Trusted computing base wrapper.*

ACManager* **bankACManager** *Manager of Banks' access certificates.*

ACManager* **esoACManager** *Manager of other Esos' access certificates.*

ACManager* **thisEsoACManager** *Manager of this Eso's access certificates.*

Allocator* **allocator** *Space allocating class.*

Banker* **banker** *Class responsible for money handling.*

Scheduler* **scheduler** *Timeouts generating class.*

Finder* **finder** *Search and download controler.*

TimeSynchronizer* **timeSynchronizer** *Class that keeps the right time and synchronizes current time with other Esos*

131.2 attributes

Names

	Eso (string configName = ESO_CONFIG_FILE)	*Constructor.*	
virtual	**~Eso** ()	*Destructor.*	
131.2.1 virtual void	**CreateRunningObjects** ()	*Creates all Runable objects, running them as threads*	377
131.2.2 virtual void	**WaitForRunningObjects** ()	*Waits for all running objects to stop*	377
131.2.3 virtual void	**CreateMajordomo** ()	*Sets Six::majordomo to new Eso's Majordomo*	378
Err	**Initialize** ()	*Does some really cool initializations needed especially when debugging*	

131.2.1

virtual void **CreateRunningObjects** ()

Creates all Runable objects, running them as threads

Creates all Runable objects, running them as threads. Overrides Sixes virtual method.

131.2.2

virtual void **WaitForRunningObjects** ()

Waits for all running objects to stop

Waits for all running objects to stop. Overrides Sixes virtual method.

131.2.3

virtual void **CreateMajordomo** ()

Sets Six::majordomo to new Eso's Majordomo

Sets Six::majordomo to new Eso's Majordomo. Overrides Sixes virtual method.

131.3

void **CreateMajordomo** ()

Eso Majordomo creator

Eso Majordomo creator.

Overrides Six::CreateMajordomo method to assign correct majordomo to this type of Six (Eso).

Return Value: void
See Also: Six (→174, *page 514*)
Author: Pavel

131.4

void **CreateRunningObjects** ()

Creates running objects

Creates running objects.

Calls methods that create all threads for both Six and Eso

Return Value: void
See Also: Six, Six::CreateRunningObjects()
Author: Pavel

131.5

Eso (string configName)

Eso constructor

Eso constructor.

Creates all the objects necessary to run

Parameters: `configName` — name of file with configuration information for Eso
See Also: Six, LogFile, Allocator, Banker, TCBWrapper, Finder, Scheduler, TimeSynchronizer, ACManager
Author: Marek, Pavel

131.6

Err **Initialize** ()

Initializes Eso

Initializes Eso.

Calls appropriate initializations, used on startup, mostly for debugging

Return Value: error code
See Also: Majordomo, Scheduler, TimeSynchronizer
Author: Marek, Pavel

131.7

void **WaitForRunningObjects** ()

Waites for running objects

Waites for running objects.

Waits for all threads to stop running, for both Six and Eso

Return Value:	void
See Also:	Six, Six::CreateRunningObjects()
Author:	Pavel

131.8

˜Eso ()

Eso destructor

Eso destructor.

Deletes all the objects - frees memory

Author: Pavel

132

class **EsoOffspring**

Eso offspring

Public Members

methods

Protected Members

attributes

Eso offspring.

Maintains a pointer to the object that created this one.

See Also: Eso (→131, *page 375*)
Author: Venca

133

General Finders' constants

Names

#define **FINDER_FWD_TABLE_NAME**
Name of forwarding table

#define **FINDER_REPLY_TABLE_NAME**
Name of replying table

134

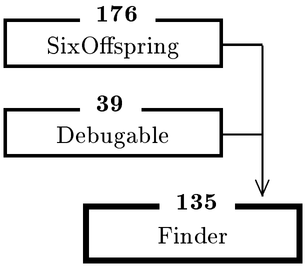
Names of Finder's tables' fields

135

class **Finder** : public Debugable, public SixOffspring

Manager of searching and downloading

Inheritance



Public Members

Protected Members

Manager of searching and downloading.

Encapsulates the TransactionForwardTable and makes easier the operations associated with forwarding found files and headers.

135.2

methods

Names

Finder (LogFile *aLogFile, Eso *eso)
Constructor.

~Finder () *Destructor.*

Err **SaveForwardInfo** (const TID aTID, GMessage *inMsg, const Int depth)
Save info about client.

Err **SaveReplyInfo** (const TID aTID, GMessage *aHeader, const Int depth)

Save info about sent file.

Err	**GetForwardInfo** (const TID inReplyToTID, GMessage **anAC, TID *replyToTID)	*Get forward info.*
Int	**AlreadySent** (const TID aTID, GMessage *aHeader)	*Return 0/1 if such a record isn't/is in replyTable.*
Err	**DeleteForwardInfo** (TID aTID)	*Delete fwd info according a TID.*
Err	**DeleteRecordsByExpiration** (ESTime currentTime)	*Delete records with by expiration.*

135.3

Int AlreadySent (TID aTID, GMessage *aHeader)

Returns 0/1 if the record wasn't/was already sent

Returns 0/1 if the record wasn't/was already sent.

Looks in the fwd table for the tuple [aTID, aHeader]. Returns 1/0 if found/found not.

Return Value: boolean value
Parameters: `aTID` — transaction ID
`aHeader` —
See Also:
Author: Marek

135.4

TableFieldQueue* CreateFwdTableDefinition ()

Creates definition of Finder fwd table

Creates definition of Finder fwd table.

Creates definition for fwd Table by putting all needed TableFields in it. Fwd table includes reply-to info [Reply-To-AC and Reply-To-TID] of requestor, TID of this Eso and time, when this information expires.

Return Value:	TableFieldQueue *
Parameters:	`void` —
See Also:	TableFieldQueue, TableField
Author:	Marek

135.5

TableFieldQueue* **CreateReplyTableDefinition** ()

Creates definition of Finder reply table

Creates definition of Finder reply table.

Creates definition for replyTable by putting all needed TableFields in it. Reply table includes the FFID together with ID of transaction, in which it was sent to requestor and time, in which this info can be deleted.

Return Value:	TableFieldQueue *
Parameters:	`void` —
See Also:	TableFieldQueue, TableField
Author:	Marek

135.6

Err **DeleteForwardInfo** (TID aTID)

Deletes forwarding information according to TID

Deletes forwarding information according to TID.

Finds the record in the fwd table and deletes it.

Return Value: error code
Parameters: `aTID` — transaction ID
See Also:
Author: Marek

135.7

Err **DeleteRecordsByExpiration** (ESTime currentTime)

Deletes records in tables according to expiration

Deletes records in tables according to expiration.

<pre> Looks in both tables for records with expiration < currentTime and deletes them. </pre>

Return Value: error code
Parameters: `currentTime` —
See Also:
Author: Marek

135.8

Finder (LogFile *aLogFile, Eso *eso)

Finder constructor

Finder constructor.

Creates fwd and reply Tables

Parameters: `aLogFile` — Log file.
`eso` — Pointer to Eso object.
See Also: Table, CreateFwdTableDefinition(), CreateReplyTableDefinition(), Six
Author: Marek

135.9

Err **GetForwardInfo** (TID aTID, GMessage **anAC, TID *replyToTID)

Gets info about where to forward

Gets info about where to forward.

According to TID gets the Reply-To-AC and Reply-To-TID.

Return Value: error code
Parameters: `aTID` — transaction ID
`anAC` — AC got from fwd table
`replyToTID` — Reply-To-TID got from fwd Table
See Also:
Author: Marek

135.10

Err **SaveForwardInfo** (TID aTID, GMessage *inMsg, const Int depth)

Saves info about where to forward the reply

Saves info about where to forward the reply.

Gets Reply-to-AC and Reply-to-TID fields from inMsg and saves them together with aTID and default search Time multiplied by depth of search..

Return Value: error code
Parameters: `aTID` — transaction ID
`inMsg` — message storing the information
`depth` — depth of search
See Also:
Author: Marek

135.11

Err **SaveReplyInfo** (TID aTID, GMessage *aHeader, const Int depth)

Saves info about what was forwarded

Saves info about what was forwarded.

Gets the FFID from header and saves it together with ID of transaction and default expiration time multiplied by depth of search.

Return Value: error code
Parameters: `aTID` — transaction ID
`aHeader` — a GMessage storing the header
`depth` — of search
See Also:
Author: Marek

135.12

~Finder ()

Finder destructor

Finder destructor.

Destroyes tables

See Also:
Author: Marek

135.1

attributes

Names

Table*	**fwdTable**	*Tables for forwarding transactions.*
TableFieldQueue*	**CreateFwdTableDefinition** ()	*Method that creates fwdTable description.*
TableFieldQueue*	**CreateReplyTableDefinition** ()	*Method that creates replyTable description.*

136

Labels for Eso's configuration file

Names

#define **CFG_MAX_SEARCH_DEPTH**
Maximum search depth allowed by this Eso.

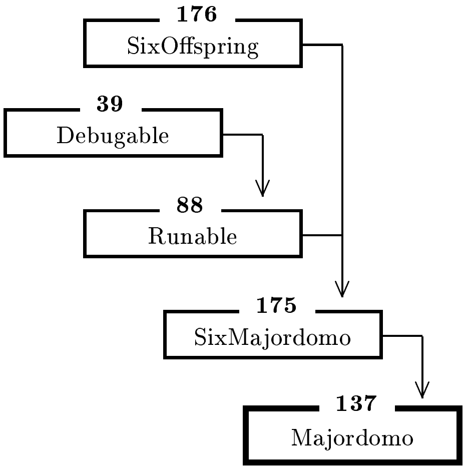
#define **CFG_MAX_SEARCH_WIDTH**
Maximum search width allowed by this Eso.

137

class **Majordomo** : public SixMajordomo

Class responsible for handling messages comming to Eso

Inheritance



Public Members

Majordomo (Eso * parent, LogFile * majordomoLogFile)
Eso's majordomo constructor.

Functions for ACManagers

137.2 Err **DRCRequestForFile** (const MsgField *dataField, const ACID acid)
Serves CMD_REQ_FOR_FILE message 396

137.3 Err **DRCRequestForHeaders** (const MsgField *dataField, const ACID acid)
Serves CMD_REQ_FOR_HEADERS message 396

137.4 include . .. Eso H eso.h Err

Protected Members

Class responsible for handling messages comming to Eso.

This class encapsulates main Eso's functionality. Its behaviour is inherited from general SixManager. Majordomo is capable of receiving messages from Mix and other Sixes in the Eternity Service, taking appropriate actions (such as to store or find a file) and creating replies.

See Also: SixMajordomo, Six, Eso
Author: Ivanka, Marek, Pavel

137.2

Err **DRCRequestForFile** (const MsgField *dataField, const ACID acid)

Serves CMD_REQ_FOR_FILE message

Serves CMD_REQ_FOR_FILE message.

<pre>processes the incoming request for file - searches for file on this server - if not found, continues the search in the search tree - if found, sends the file back to the client</pre>

Return Value: error code <pre> - KO in the case of bad structure of dataField or if fatal error comes (ex. NULL ptr to Finder) - OK otherwise (that means even in case of mistake in any step of the algorithm [ex. cannot send the message back to client] - these situations are only written to the logFile)</pre>

See Also: TCBWrapper, Finder, ACManager

Author: Marek

137.3

Err **DRCRequestForHeaders** (const MsgField *dataField, const ACID acid)

Serves CMD_REQ_FOR_HEADERS message

Serves CMD_REQ_FOR_HEADERS message.

<pre>processes the incoming request for headers - continues the search in the search tree - looks for the file on this server - sends the headers found on this server back to the client </pre>

Return Value: error code <pre> - KO in the case of bad structure of dataField or if fatal error comes (ex. NULL ptr to Finder) - OK otherwise (that means even in case of mistake in any step of the algorithm [ex. cannot send the message back to client] - these situations are only written to the logFile)</pre>

Parameters: `dataField` — encrypted request comming with message
`acid` — ID of this Eso access certificate which was used to address thid message

See Also: TCBWrapper, Finder, ACManager
Author: Marek

137.4

include . .. Eso H eso.h Err **DRCRequestForStorage** (const MsgField *dataFld)

Serves CE_REQ_FOR_STOR command

Serves CE_REQ_FOR_STOR command.

Handles the CE_REQ_FOR_STOR command, according to Eso-Others interface. dataFld comming has this structure: <pre> Reply-To-ID: ID of a message assigned by sender (Client) Reply-To-AC: AC to reply to File-Size: Size of file Client whishes to store Storage-Time: RelTime for which the file should be stored </pre>

Return Value: error code
Parameters: `dataFld` — data comming with command
See Also: FillReplyAllocation(), Allocator, Banker, Scheduler
Author: Pavel

137.5

Err **DRCRequestFromEso** (MsgField *dataFld)

Serves Eso's request for money

Serves Eso's request for money.

Finds appropriate oAuth to send back to Eso to verify it is the correct file owner not faking file storage.

Return Value: error code
Parameters: `dataFld` — data comming along with command

See Also: co_se_toho_tyka
Author: Ivana

137.6

Err **DRCTimeForPayment** (MsgField *dataInMsgFld)

DRCTimeForPayment (cmd to_payment) Dlouhy_komentar

137.7

include . H majordomo.h Err **DRCTimeForPayment** (MsgField *dataInMsgFld)

DRCTimeForPayment (cmd to_payment)

DRCTimeForPayment (cmd to_payment).

Dlouhy_komentar

Return Value: error code
Parameters: `parametr` —
See Also: co_se_toho_tyka
Author: Ivana

137.8

Err **DRCTimeOutPayment** (MsgField *dataInMsgFld)

DRCTimeOutPayment (cmd to_curr_pay) Dlouhy_komentar

137.9

Err **GetAsymetricKey** (GMessage **acPubKey)

Gets Asymetric key from file

Gets Asymetric key from file.

Gets Client's pubKey from file.

Return Value:	error code
Parameters:	`acPubkey` — Returned pubKey.
See Also:	
Author:	Marek

137.10

GMessage* **GetCertificates** (string type, Size count)

Asks mix for some certificates and then returns them

Asks mix for some certificates and then returns them.

Constructs message of this type: <pre> Command: get_certificates Parameters: Count: Certificate-Type: Request-ID: </pre>

Return Value:	GMessage - got certificates.
Parameters:	`type` — Type of certificates to get `count` — Count of certificates to get.
See Also:	
Author:	Venca, Pechy

137.11

void **GetCertificates** (GMessage * request)

Returns certificates from Mix's repository (Certificate Repository)

Returns certificates from Mix's repository (Certificate Repository).

Mix returns all certificates from its repository. When there is not enough certificates of requested type, Mix asks for them Acs. When certificates come to Mix from Acs, they are returned to the Six that wanted them.

Return Value:	void
Parameters:	`request` — Request's GMessage: <pre> GM_PARAMETERS CERTIFICATE_TYPE GM_COUNT </pre>
See Also:	CertificateRepository, Acs
Author:	Venca

137.12

GMessage* **GetOnion** (Size onionLen, Size requestID)

Asks Mix for onion

Asks Mix for onion.

Creates GenerateOnion message for Client and waits for reply.

Return Value:	GMessage - the onion.
Parameters:	`onionLen` — Requested length of onion. `requestID` — RequestID to send to client
See Also:	
Author:	Pechy, Venca

137.13

include . H majordomo.h Err **SwitchDataReceivedCommand** (const Char *command, const MsgField *dataFld, const ACID acid)

Chooses a method to handle the command with

Chooses a method to handle the command with.

According to the received command chooses a method to handle the command with.

Return Value: error code
Parameters: `command` — Command in the incoming message.
`dataFld` — Data in the incoming message.
`acid` — ID of the certificate to which the data was sent.

See Also:
Author: Marek

137.14

Err **SwitchDataReceivedCommand** (const Char *command, const MsgField *dataFld, const ACID acid)

Chooses the method to handle the command

Chooses the method to handle the command.

Decidec which method to use according to got command.

Return Value: error code
Parameters: `command` — got command
`dataFld` — got data
`acid` — ID of certificate to which the message was sent

See Also:
Author: Pavel

137.15

include . H majordomo.h Err **SwitchMixCommand** (MsgField *commandFld, MsgField *dataFld)

Chooses serving method for Mix command

Chooses serving method for Mix command.

According to field Command (in message poped from outMsgQueue) choose handling method.

Return Value: error code
Parameters: `commandFld` — command from Mix
`dataFld` — data comming along with command
See Also: SixMajordomo::SwitchMixCommand()
Author: Marek

137.1

methods

Names

Functions for commands got from receiverToMajordomo queue (Mix cmds and timeouts)

Filling messages for Service

Err **SendToEsos** (Char *command, GMessage *msg, Int count)
Sends messages to other Esos through Mix

137.1.1

Functions for commands from Service

Names

Saving files

Finding files

137.1.2

virtual MsgField* **DecryptByACPriv** (const ACID acid, const MsgField *dData)

Decrypting data from Service

Decrypting data from Service. Overrides pure virtual SixMajordomo::DecryptByACPriv

138

Err **Majorodmo::DRCChallengeFromBank** (MsgField *dataInMsgFld)

DRCChallengeFromBank (cmd be_chall)

DRCChallengeFromBank (cmd be_chall).

Dlouhy_komentar

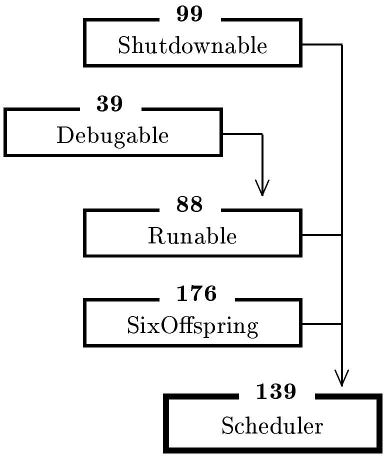
Return Value:	error code
Parameters:	`parametr` —
See Also:	co_se_toho_tyka
Author:	Ivana

139

class **Scheduler** : public SixOffspring, public Runable, public Shutdownable

Schedules time's operation

Inheritance



Public Members

Protected Members

Schedules time's operation.

This is implemented by sending messages (timeouts), when would start some operation. Scheduler runs in stand-alone thread and provides setting timeouts and getting of current time. <p> Timeout has three items: <pre> - command - name of timeouts - info - some additional information - time of expiration </pre>

See Also: Eso (→131, *page 375*)
Author: Jirka

139.3

Names

void*	**Run** (void*)	*Starts in stand-alone thread when is Eso starting*
virtual void	**Shutdown** ()	*Sets state shutdown to true*
Err	**SetTimeOut** (char *command, GMessage *info, RelTime time)	*Sets timeout to relative time from current time*
Err	**SetTimeOut** (char *command, GMessage *info, ESTime time)	*Sets timeout to absolute time*
bool	**IsSetTimeOut** (char *command)	*Returns true if is timeout set*
ESTime*	**GetTime** ()	*Return current time if is time ok or NULL if is time wrong*

139.1

Names

TimeSortedTable*	**tbl**	*sorted table for storing timeouts*
pthread_mutex_t	**mutexTbl**	*mutex to work with sorted table tbl*
bool	**TimeOk**	*state if time is ok or wrong*

Table*	**tblTimeOk**	*persistent storing of state*
pthread_mutex_t	**mutex**	*mutex to work with table tblTimeOk*
bool	**shutdown**	*state if thread would shutdown*

139.2

Names

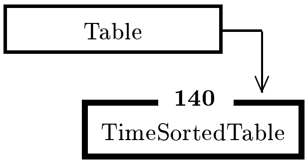
Err	**SetTimeOk** (bool state)	*Sets state TimeOk*
bool	**GetTimeOk** ()	*Gets state TimeOk*
void	**ProcessTimeOut** ()	*Searchs table tbl if is any expired timeout.*
Err	**CommonSetTimeOut** (char *command, GMessage *info, ESTime *pTime)	*Common method for setting time-out.*

140

class **TimeSortedTable** : public Table

Sorted table by time

Inheritance



Public Members

Sorted table by time.

Sorts table by first column which have to be TimeField.

See Also: Scheduler (→139, *page 405*)
Author: Jirka

140.1

Names

virtual Err
InsertRecord (Record *aRecord)
Sorted insert

141

@name TCBWrapper paths

142

@name File names of files TCBWrapper uses

143

@name Ring key names

144

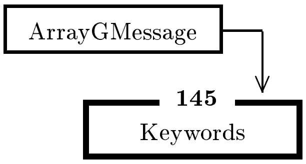
@name StorageTable and acTable field names

145

class **Keywords** : public ArrayGMessage

GMessage used for storing array of keywords

Inheritance



Public Members

GMessage used for storing array of keywords.

This class simply extends the functionality of ArrayGMessage, making it simple to work with an array of keywords. In a fact an ArrayGMessage using prefix GM_KEYWORD.

See Also: ArrayGMessage, TCBWrapper, Client
Author: Pavel

145.1

methods

Names

	Keywords (LogFile *aLogFile = NULL)	*Constructor.*
Char*	**Keyword** (Int i)	*Method returning i-th keyword from an array.*
Int	**MatchesPattern** (Char *pattern)	*Given a pattern (search string) decides whether keywords match it.*
Err	**AddKeyword** (Char *keyword)	*Adds a keyword.*

145.2

Err **AddKeyword** (Char *keyword)

Adds a keyword

Adds a keyword.

Simply calls ArrayGMessage::AddField with converted arguments.

Return Value: error code
Parameters: `keyword` — keyword to add
See Also:
Author: Pavel

145.3

Char* **Keyword** (Int i)

Gets i-th keyword label

Gets i-th keyword label.

Creates and returns i-th label for keyword.

Return Value: created label
Parameters: `i` — position of label to return
See Also:
Author: Pavel

145.4

Keywords (LogFile *aLogFile)

Constructor

Constructor.

Simply calls from ArrayGMessage inherited constructer.

Parameters: `aLogFile` — log file for debugging messages
See Also: ArrayGMessage
Author: Pavel

145.5

Int **MatchesPattern** (Char *pattern)

Decides wheter pattern matches keywords

Decides wheter pattern matches keywords.

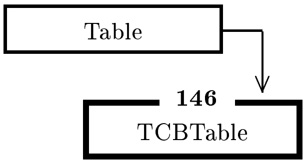
This version accepts only a single word and looks, if there is an exactly same keyword stored. This method is virtual and expects further modifications to accept even more complex queries.

Return Value: 0 if successfull, non 0 otherwise
Parameters: `pattern` — string with a query
See Also:
Author: Pavel

146 class **TCBTable** : public Table

Encrypted (database like) table

Inheritance



Public Members

146.1		**methods**		418
146.2	Err	**Close** ()	*Closes TCBTable*	418
146.3	Err	**Open** ()	*Opens TCBTable*	419
146.4	include . H tcbTable.h	**TCBTable** (const Char *tableName, TableFieldQueue *definition, const GMessage *tableRingKey, LogFile *aLogFile)	*TCBTable constructor*	419
146.5		**~TCBTable** ()	*TCBTable destructor*	420

Protected Members

attributes

Encrypted (database like) table.

The same like Table, but the file into which the data is stored is saved encrypted with the symetric key. This implementation works with it in a simple way:
 Open - decrypts the table, all operations then work on the decrypted file. Close - encrypts the table.

See Also: Table, TCBWrapper
Author: Pavel

146.1

methods

Names

	TCBTable (const Char *tableName, TableFieldQueue *definition, const GMessage *tableRingKey, LogFile *aLogFile)	*Constructor.*
virtual	**~TCBTable** ()	*Destructor.*
virtual Err	**Open** ()	*Opens the TCBTable - decrypts it temporarily*
virtual Err	**Close** ()	*Closes the TCBTable - encrypts it*

146.2

Err **Close** ()

Closes TCBTable

Closes TCBTable.

Uses ringKey to encrypt previously decrypted table.

Return Value: error code
See Also: Table, Cipherer, TCBWrapper
Author: Pavel

146.3

Err **Open** ()

Opens TCBTable

Opens TCBTable.

Uses ringKey to decrypt the table to allow all common table operations.

Return Value: error code
See Also: Table, Cipherer, TCBWrapper
Author: Pavel

146.4

include . H tcbTable.h **TCBTable** (const Char *tableName, TableFieldQueue *definition, const GMessage *tableRingKey, LogFile *aLogFile)

TCBTable constructor

TCBTable constructor.

Creates name for decrypted table and a file to access it.

Parameters:
`tableName` — name of encrypted table
`definition` — definition of table structure
`tableRingKey` — symetric key used to keep table encrypted
`aLogFile` — log file for debugging messages

See Also: Table, TableFieldQueue

Author: Pavel

146.5

~TCBTable ()

TCBTable destructor

TCBTable destructor.

Destroyes everything created.

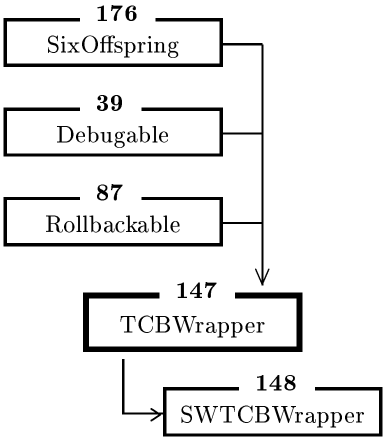
See Also: co_se_toho_tyka
Author: Pavel

147

class **TCBWrapper** : public Rollbackable, public Debugable, public SixOffspring

General Trusted Computing Base wrapper

Inheritance



Public Members

147.1 **methods** 422

147.2 include . H tcbWrapper.h
TCBWrapper (Eso *anEso, LogFile *aLogFile)
TCBWrapper constructor 425

147.3 **~TCBWrapper** () *TCBWrapper destructor* 425

General Trusted Computing Base wrapper.

This is an abstract generally designed class that enables any realization of TCB (either software or hardware). To each such TCB it would be needed to create appropriate TCBWrapper that would do all that stuff defined in this class.

See Also: Eso (→131, *page 375*)

Author: Pavel

147.1 methods

Names

TCBWrapper (Eso *anEso, LogFile *aLogFile = NULL)
Costructor.

virtual **~TCBWrapper** () *Destructor.*

virtual Err
Rollback (const TID tid)
Ability to be rollbacked.

147.1.1 virtual Err
SaveFile (const ACID acid, const ID fid, const MsgField *ffid, const MsgField *keywords, const MsgField *data)
Stores file kept in "data" with ID "fid"

virtual Err
DeleteFileByFileID (const ID fid)
Deletes a file.

virtual Err
ComputeMAC (const ID fid, const MsgField *oAuth, MsgField **mac)
Computes MAC by finding a file and concatenating it with OAuth.

147.1.2 virtual Queue*
GetFileHeadersByFAM (const ACID acid, const MsgField *fam, const GMessage *clientPubKey)
Finds all files matching the File Atribute Mask, and returns

147.1.3 virtual MsgField*

GetFileByFinderFID (const ACID acid,
const MsgField *ffid, const
GMessage *clientPubKey)
Finds the file with "ffid", decrypts it, reencrypts it with "clientPubKey" 424

virtual MsgField*
DecryptByACPriv (const ACID acid,
const MsgField *data)
Finds aproprite private key and returns decrypted "data".

virtual Err
GenerateACKeys (ACID *acid,
GMessage **acPubKey,
GMessage **tcbPubKey)
Generates and stores AC keys (both AC and TCB).

virtual Err
GetACPubKeys (const ACID,
GMessage **acPubKey = NULL,
GMessage **tcbPubKey =
NULL)
Returns pub keys to ACID.

virtual Err
RemoveACKeys (const ACID acid)
Removes AC keys (both AC and TCB).

virtual Err
RefreshMasterKey ()
Generates new Master Key and does all reencryption needed (table of ring keys)

virtual Err
RefreshRingKeys ()
Generates new Ring Keys and does all reencryption needed (table of files and Storage keys, table of certificates, table of transport keys)

147.1.1

virtual Err **SaveFile** (const ACID acid, const ID fid, const MsgField *ffid, const MsgField *keywords, const MsgField *data)

Stores file kept in "data" with ID "fid"

Stores file kept in "data" with ID "fid". "ffid", "keywords", "data" encrypted with TCBPubKey.

147.1.2

virtual Queue* **GetFileHeadersByFAM** (const ACID acid, const MsgField *fam, const GMessage *clientPubKey)

Finds all files matching the File Atribute Mask, and returns

Finds all files matching the File Atribute Mask, and returns. their headers that include FinderFileID, FileSize. FAM encrypted by TCB Public Key.

147.1.3

virtual MsgField* **GetFileByFinderFID** (const ACID acid, const MsgField *ffid, const GMessage *clientPubKey)

Finds the file with "ffid", decrypts it, reencrypts it with "clientPubKey"

Finds the file with "ffid", decrypts it, reencrypts it with "clientPubKey". and returns it as a MsgField.

147.2

include . H tcbWrapper.h **TCBWrapper** (Eso *anEso, LogFile *aLogFile)

TCBWrapper constructor

TCBWrapper constructor.

Initializes all ancestors.

Parameters: `anEso` — pointer to owner of this class
`aLogFile` — log file for debugging messages
See Also: SWTCBWrapper (→148, *page 426*)
Author: Pavel

147.3

~TCBWrapper ()

TCBWrapper destructor

TCBWrapper destructor.

Does nothing special.

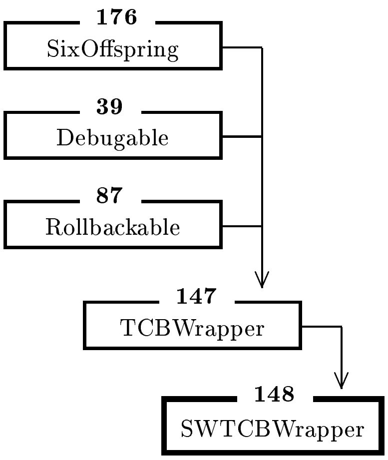
See Also: SWTCBWrapper (→148, *page 426*)
Author: Pavel

148

class **SWTCBWrapper** : public TCBWrapper

Software Trusted Computing Base wrapper

Inheritance



Public Members

Computes MAC 434

148.7 GMessage*
CreateReplyFileHeader (const GMessage *clientPubKey, const MsgField *ffid, const Size size)
Creates a header to reply 434

148.8 MsgField* **DecryptByACPriv** (const ACID acid, const MsgField *data)
Decrypts by private key 435

148.9 MsgField* **DecryptByTCBPriv** (const ACID acid, const MsgField *data)
Decrypts data with private key . 435

148.10 Err **DeleteFileByFileID** (const ID fid)
Deletes file by file ID 436

148.11 Err **GenerateACKeys** (ACID *acid, GMessage **acPubKey, GMessage **tcbPubKey)
Generates keys for access certificate 436

148.12 Err **GetACPubKeys** (const ACID acid, GMessage **acPubKey, GMessage **tcbPubKey)
Returns public keys 437

148.13 MsgField* **GetFileByFinderFID** (const ACID acid, const MsgField *ffid, const GMessage *clientPubKey)
Pops file with ffid 438

148.14 Queue* **GetFileHeadersByFAM** (const ACID acid, const MsgField *fam, const GMessage *clientPubKey)
Searches for file by file attribute mask 438

148.15 Err **GetRingKeys** (GMessage ** ringKeys)
Loads ring keys 439

148.16 Err **InstantiateTCBTables** (const GMessage *ringKeys)
Initializes all TCBTables 439

148.17 Err **LoadMasterKey** ()

Protected Members

Software Trusted Computing Base wrapper.

This class realizes all TCB tasks purely programmaticaly without need of any hardware.

See Also: TCBWrapper, Eso
Author: Pavel

148.3 methods

Names

	SWTCBWrapper (Eso *anEso, LogFile *aLogFile = NULL)	*Software TCBWrapper constructor.*
virtual	**~SWTCBWrapper** ()	*Hardware TCBWrapper constructor.*
virtual Err		

Rollback (const TID tid)
Ability to be rollbacked.

148.3.1 virtual Err
SaveFile (const ACID acid, const ID fid,
const MsgField *ffid,
const MsgField *keywords,
const MsgField *data)
Stores file kept in "data" with ID "fid" 431

virtual Err
DeleteFileByFileID (const ID fid)
Deletes a file.

virtual Err
ComputeMAC (const ID fid,
const MsgField *oAuth,
MsgField **mac)
Computes MAC by finding a file and concatenating it with OAuth.

148.3.2 virtual Queue*
GetFileHeadersByFAM (const ACID acid,
const MsgField *fam,
const GMessage
*clientPubKey)
Finds all files matching the File Atribute Mask, and returns their headers that include FinderFileID, FileSize 432

virtual MsgField*
GetFileByFinderFID (const ACID acid,
const MsgField *ffid, const
GMessage *clientPubKey)
Finds the file with "ffid", decrypts it, reencrypts it with "clientPubKey" and returns it as a MsgField

virtual MsgField*
DecryptByACPriv (const ACID acid,
const MsgField *data)
Finds aproprite private key and returns decrypted "data".

virtual Err

	GenerateACKeys (ACID *acid, GMessage **acPubKey, GMessage **tcbPubKey)	*Generates and stores AC keys (both AC and TCB).*
virtual Err	**GetACPubKeys** (const ACID, GMessage **acPubKey = NULL, GMessage **tcbPubKey = NULL)	*Returns pub keys to ACID.*
virtual Err	**RemoveACKeys** (const ACID acid)	*Removes AC keys (both AC and TCB).*
virtual Err	**RefreshMasterKey** ()	*Generates new Master Key and does all reencryption needed (table of ring keys)*
virtual Err	**RefreshRingKeys** ()	*Generates new Ring Keys and does all reencryption needed (table of files and storage keys, table of certificates, table of transport keys)*
Void	**Test** ()	*Test method for debugging purposes only.*

148.3.1

virtual Err **SaveFile** (const ACID acid, const ID fid, const MsgField *ffid, const MsgField *keywords, const MsgField *data)

Stores file kept in "data" with ID "fid"

Stores file kept in "data" with ID "fid". "ffid", "keywords", "data" encrypted with TCBPubKey.

148.3.2

virtual Queue* **GetFileHeadersByFAM** (const ACID acid, const MsgField *fam, const GMessage *clientPubKey)

Finds all files matching the File Atribute Mask, and returns their headers that include FinderFileID, FileSize

Finds all files matching the File Atribute Mask, and returns their headers that include FinderFileID, FileSize. FAM encrypted by TCB Public Key.

148.4

Err **CertTableFindByACID** (const ACID acid, const shouldClose, GMessage **acPubKey, GMessage **acPrivKey, GMessage **tcbPubKey, GMessage **tcbPrivKey)

Finds record in certifTable

Finds record in certifTable.

Searches open table for appearance of record with appropriate search file ID. Enables to close certifTable after successfull search.

Return Value: error code

Parameters:
`acid` — Access certificate ID
`shouldClose` — Should I close the certifTable, after I'm finished?
`acPubKey` — Access certificate public key (out parameter)
`acPrivKey` — Access certificate private key (out parameter)
`tcbPubKey` — TCB public key (out parameter)
`tcbPrivKey` — TCB private key (out parameter)

See Also:

Author: Pavel

148.5

Err **CertTableInsert** (const ACID acid, const GMessage *acPubKey, const GMessage *acPrivKey, const GMessage *tcbPubKey, const GMessage *tcbPrivKey)

Inserts record to certifTabel

Inserts record to certifTabel.

Creates new Record for certifTable, fills it with given arguments and inserts it to certifTable.

Return Value: error code

Parameters:
`acid` — Access certificate ID
`acPubKey` — Access certificate public key
`acPrivKey` — Access certificate private key
`tcbPubKey` — TCB public key
`tcbPrivKey` — TCB private key

See Also:

Author: Pavel

148.6

Err **ComputeMAC** (const ID fid, const MsgField *oAuth, MsgField **mac)

Computes MAC

Computes MAC.

Computes hash from stored file and given oAuth.

Return Value:	error code
Parameters:	`fid` — ID of a file MAC should be counted from `oAuth` — String sent by bank to verify file owner `mac` — Counted hash (out parameter)
See Also:	Cipherer, CreateTempFileName()
Author:	Pavel

148.7

GMessage* **CreateReplyFileHeader** (const GMessage *clientPubKey, const MsgField *ffid, const Size size)

Creates a header to reply

Creates a header to reply.

This method creates header by encrypting given file size and search file ID with client's public key and giving them into GMessage.

Return Value:	Header encrypted by clientPubKey
Parameters:	`clientPubKey` — Public key to encrypt header with `ffid` — Search file ID to include in header `size` — File size to include in header

See Also:
Author: Pavel

148.8

MsgField* **DecryptByACPriv** (const ACID acid, const MsgField *data)

Decrypts by private key

Decrypts by private key.

Decrypts data with private key identified by access certificate ID. Private key is found in certifTable.

Return Value: Decrypted data if successfull, NULL otherwise
Parameters: `acid` — Access certificate ID identifying access public key that was used to encrypt data
`data` — Data whished to decrypt
See Also: Cipherer, CertTableFindByACID()
Author: Pavel

148.9

MsgField* **DecryptByTCBPriv** (const ACID acid, const MsgField *data)

Decrypts data with private key

Decrypts data with private key.

Decrypts data with TCB private key identified by access certificate ID. Proper key is found in certifTable by acid.

Return Value: Decrypted data
Parameters: `acid` — Access certificate ID
`data` — Data to be encrypted by TCB private key
See Also:
Author: Pavel

148.10

Err **DeleteFileByFileID** (const ID fid)

Deletes file by file ID

Deletes file by file ID.

What more should be said?

Return Value: error code
Parameters: `fid` — ID of a file to be deleted
See Also:
Author: Pavel

148.11

Err **GenerateACKeys** (ACID *acid, GMessage **acPubKey, GMessage **tcbPubKey)

Generates keys for access certificate

Generates keys for access certificate.

Generates asymetric key pairs for access certificate and stores them in certifTable to be able to use them automatically later given only access certificate ID, that is generated as well.

Return Value: error code

Parameters:
`acid` — Generated access certificate ID (out parameter)
`acPubKey` — Generated access public key (out parameter)
`tcbPubKey` — Generated access public key (out parameter)

See Also: Cipherer (→34, *page 150*)

Author: Pavel

148.12

Err **GetACPubKeys** (const ACID acid, GMessage **acPubKey, GMessage **tcbPubKey)

Returns public keys

Returns public keys.

Returns both TCB and access certificate public keys identified by access certificate ID. The keys are searched for in certifTable.

Return Value: error code

Parameters:
`acid` — Access certificate ID to search by
`acPubKey` — Access public key (out parameter)
`tcbPubKey` — TCB public key (out parameter)

See Also: CertTableFindByACID()

Author: Pavel

148.13

MsgField* **GetFileByFinderFID** (const ACID acid, const MsgField *ffid, const GMessage *clientPubKey)

Pops file with ffid

Pops file with ffid.

Searches storageTable for files that have search file ID equal to ffid.

Return Value: File encrypted with clientPubKey, NULL otherwise
Parameters: `acid` — Access certificate ID identifying TCB private key used to encrypt search file ID
`ffid` — Encrypted search file ID
`clientPubKey` — Client's public key to encrypt replies with
See Also: Cipherer (→34, *page 150*)
Author: Pavel

148.14

Queue* **GetFileHeadersByFAM** (const ACID acid, const MsgField *fam, const GMessage *clientPubKey)

Searches for file by file attribute mask

Searches for file by file attribute mask.

Searches through storageTable for files that are matching all criteria expressed in FAM - File Attribute Mask. This was designed to create simple extensions even for complex masks. Returns file descripting headers encrypted by clientPubKey.

Return Value: Queue with headers encrypted by clientPubKey, NULL otherwise
Parameters: `acid` — Access certificate ID identifying TCB private key used to encrypt File Attribute Mask
`fam` — Encrypted File Attribute Mask to search by
`clientPubKey` — Client's public key to encrypt replies with

See Also: Keywords, DecryptByTCBPriv()
Author: Pavel

148.15

Err **GetRingKeys** (GMessage ** ringKeys)

Loads ring keys

Loads ring keys.

Initializes ring keys for tables loading it from files, that are encrypted by master key (according to rules of hierarchical key management).

Return Value: error code
Parameters: `ringKeys` — out parameter returning ring keys if successfull
See Also: Cipherer, MsgField, GMessage
Author: Pavel

148.16

Err **InstantiateTCBTables** (const GMessage *ringKeys)

Initializes all TCBTables

Initializes all TCBTables.

Creates instances for both storageTable and certifTable generating its definitions and using appropriate ring keys.

Return Value: error code
Parameters: `ringKeys` — ring keys for TCBTable constructors
See Also: TCBTable, TableFieldQueue
Author: Pavel

148.17

Err **LoadMasterKey** ()

Loads master key

Loads master key.

Initializes masterKey of key hierarchy loading it from a file.

Return Value: error code
See Also:
Author: Pavel

148.18

Err **RefreshMasterKey** ()

Create new master key and reencrypt the file that stores ring keys

Create new master key and reencrypt the file that stores ring keys.

Create new master key and reencrypt the file that stores ring keys.

Return Value: error code
See Also: Cipherer (→34, *page 150*)
Author: Pavel

148.19

Err **RefreshRingKeys** ()

Create new ring keys and reencrypt certifTable and certifTable

Create new ring keys and reencrypt certifTable and certifTable.

Create new ring keys and reencrypt certifTable and certifTable.

Return Value: error code
See Also: Cipherer (→34, *page 150*)
Author: Pavel

148.20

Err RemoveACKeys (const ACID acid)

Removes access certificate keys

Removes access certificate keys.

Removes both TCB and access certificate key pairs from certifTable.

Return Value: error code
Parameters: `acid` — Access certificate ID identifying keys to remove

See Also: CertTableFindByACID()
Author: Pavel

148.21

Err Rollback (const TID tid)

Rolls back a transaction

Rolls back a transaction.

Rollback of a transaction means deletion of file stored with same ID as is ID of that transaction.

Return Value: error code
Parameters: `tid` — ID of a transaction to rollback
See Also: Rollbackable, DeleteFileByFileID()
Author: Pavel

148.22

SWTCBWrapper (Eso *anEso, LogFile *aLogFile)

SWTCBWrapper constructor

SWTCBWrapper constructor.

Restores the MasterKey, decrypts ringKeys, instantiates TCBTables.

Parameters: `anEso` — pointer to owner of this class
`aLogFile` — log file for debugging messages
See Also: Eso (→131, *page 375*)
Author: Pavel

148.23

Err **SaveFile** (const ACID acid, const ID fid, const MsgField *ffid, const MsgField *keywords, const MsgField *data)

Saves file

Saves file.

Prepares file for saving: decrypting it by TCB private key and encrypting with a newly generated symetric storage key. Then it puts all information (prepared the same way) together to storageTable.

Return Value:	error code
Parameters:	`acid` — Access certificate ID `fid` — file ID of file to save `ffid` — search file ID of file to save `keywords` — keywords for file to save `data` — to save
See Also:	Cipherer, StorTableInsert()
Author:	Pavel

148.24

Err **StorTableFindByFFID** (const MsgField *ffid, Char **fileName, GMessage **storageKey, Keywords **keywords, Size *size)

Finds record in storageTable

Finds record in storageTable.

Searches open table for appearance of record with appropriate search file ID.

Return Value:	error code
Parameters:	`ffid` — search file ID to search by `fileName` — name of save file (out parameter) `storageKey` — symetric key used to encrypt saved file (out parameter) `keywords` — keywords for saved file (out parameter) `size` — size of saved file (out parameter)
See Also:	
Author:	Pavel

148.25

Err **StorTableFindByFID** (const ID fid, const Int shouldClose, Char **fileName, GMessage **storageKey)

Finds record in storageTable

Finds record in storageTable.

Searches open table for appearance of record with appropriate file ID. Enables to close storageTable after successfull search.

Return Value: error code
Parameters: `fid` — file ID to search by
`shouldClose` — Should I close the storageTable, after I'm finished?
`fileName` — name of save file (out parameter)
`storageKey` — symetric key used to encrypt saved file (out parameter)
See Also:
Author: Pavel

148.26

Err **StorTableInsert** (const ID fid, const MsgField *ffid, const Char *fileName, const GMessage *storageKey, const MsgField *keywords, const Size fileSize)

Inserts record to storageTable

Inserts record to storageTable.

Creates new Record for storageTable, fills it with given arguments and inserts it to storageTable.

Return Value: error code
Parameters:
`fid` — file ID of saved file
`ffid` — search file ID of saved file
`fileName` — name of save file
`storageKey` — symetric key used to encrypt saved file
`keywords` — keywords for saved file
`fileSize` — size of saved file
See Also: TCBTable (→146, *page 417*)
Author: Pavel

148.27

Err **StorTableParseRecord** (const Record *storRec, ID *fid, MsgField **ffid, Char **fileName, GMessage **storageKey, Keywords **keywords, Size *fileSize)

Parses record got from storageTable

Parses record got from storageTable.

Parses record that was got from storageTable filling out parameters.

Return Value: error code
Parameters:
`storRec` — record read from storTable
`fid` — file ID of saved file (out parameter)
`ffid` — search file ID of saved file (out parameter)
`fileName` — name of save file (out parameter)
`storageKey` — symetric key used to encrypt saved file (out parameter)
`keywords` — keywords for saved file (out parameter)
`fileSize` — size of saved file (out parameter)
See Also: Record, TableField

Author: Pavel

148.28

~SWTCBWrapper ()

SWTCBWrapper destructor

SWTCBWrapper destructor.

Destroyes all created objects.

See Also:
Author: Pavel

148.1

attributes

Names

GMessage* **masterKey** — *Symmetric key used as a master key for hierarchical key management.*

Table with information about stored files, encrypted with a ring

TCBTable* **storageTable** — *key from a level below master key in key hierarchy.*

Table with Access Certificates keys, encrypted with its ring key

TCBTable* **certifTable** — *from a level below master key in key hierarchy.*

148.2 methods

Names

Methods used only by constructor

TCBTables management methods

MsgField* **DecryptByTCBPriv** (const ACID acid, const MsgField *data)
Finds aproprite TCB private key and returns decrypted "data".

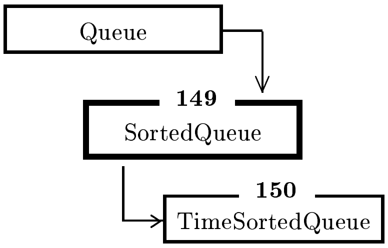
GMessage* **CreateReplyFileHeader** (const GMessage *clientPubKey, const MsgField *ffid, const Size size)
Creates encrypted header. Used when searching by keywords.

149

class **SortedQueue** : public Queue

Ancestor of all sorted queues

Inheritance



Public Members

virtual Int
Compare (Void *obj1, Void *obj2)
abstract method for comparing two objects

virtual Err
Insert (Void *obj) *Sorted insert*

Ancestor of all sorted queues.

This class is a basic class for classes suporting sorting and priorities.

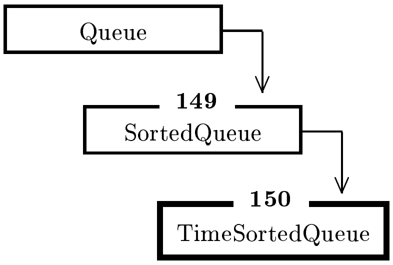
Author: Jirka

150

class **TimeSortedQueue** : public SortedQueue

Queue sorted by ESTime

Inheritance



Public Members

virtual Int	**Compare** (Void *obj1, Void *obj2)	*Compare two ESTime*
void	**Print** ()	*Print sorted queue to output*

Queue sorted by ESTime.

This queue class supports sorting by ESTime.

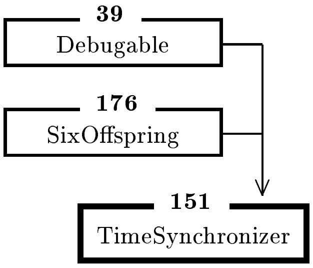
See Also: TimeSynchronizer (→151, *page 450*)
Author: Jirka

151

class **TimeSynchronizer** : public SixOffspring, public Debugable

This class is responsible for time synchronization

Inheritance



Public Members

Protected Members

This class is responsible for time synchronization.

Sends requests to other Esos, processes responses of other Esos and informs when time is wrong.

See Also: Eso (→131, *page 375*)
Author: Jirka

151.3

Names

Err	**Init** ()	*Initiate some attributes and set timeout to begin synchronization*
		Sends requests to other Esos and set timeouts
Err	**TO_TimeToSynchronize** ()	*to begin of synchronization and running of synchronization*
Err	**TS_Request** (MsgField *dataField)	*Responses to request for current time*
		Saves responses to table and then computes divergence from
Err	**TS_Response** (MsgField *dataField)	*current time. Informs when time is wrong.*
Err	**TO_Response** (MsgField *dataField)	*Ends synchronization.*

151.1

Names

ID	**ts_id**	*identifier of synchronization*
bool	**enableSynch**	*state if synchronization is running*
int	**numberItem**	*number of responses saved in table*
ESTime	**beginSynch**	*starting time of synchronization*
Table*	**tbl**	*table for responses from other Esos*

UniqueIDGenerator*
idGen *persistent identifier of synchronization*

151.2

Names

void **SetTimeOk** (Scheduler *sch, bool state)
Sets state of time (if is ok or wrong)

152

include . .. Common H common.h int **main** (int argc, char ** argv)

Eso main function

Eso main function.

Parses options Eso can be given from command line and then creates an instance of class Eso which then runs the service.

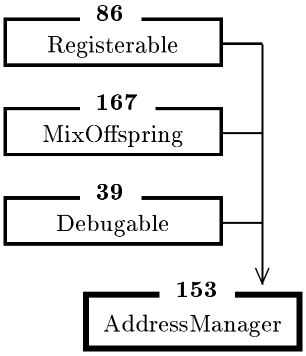
See Also: Eso (→131, *page 375*)
Author: Venca

153

class **AddressManager** : public Debugable, public MixOffspring, public Registerable

Mix address manager

Inheritance



Public Members

methods

Protected Members

attributes

Mix address manager.

Takes care of generation storage and expiration of mix's access certificates. This includes access certificates of Acs' and other Mixes.

See Also: Mix, CertificateRepository
Author: Pechy, Venca

153.1

methods

Names

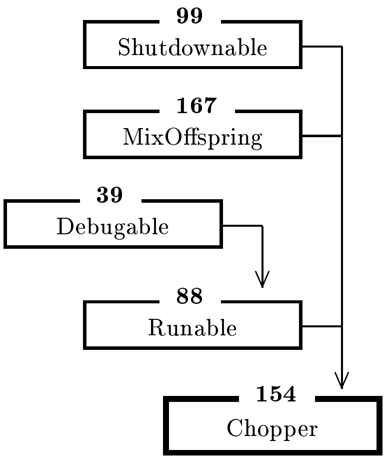
void	**DownloadCertificates** (Size count, string type)	*downloading certificates from ACS*
Err	**SaveCertificates** ()	*Storing certificates to disk*
Err	**LoadCertificates** ()	*Loading certificates from disk*

154

class **Chopper** : public Runable, public MixOffspring, public Shutdownable

Messages chopper

Inheritance



Public Members

methods

154.1		**Chopper** (Mix * parent, LogFile * chopperLogFile)	*Mix chopper constructor*	457
154.2	list < GMessage * >	**CreateChunks** (GMessage * parameters)	*Create chunks from message*	457
154.3	void*	**Run** (void *)	*Mix chopper Run() method*	458
154.4	void	**Shutdown** ()	*Prepare for clean shutdown*	459

Protected Members

attributes

Messages chopper.

Chops large messages into uniform chunks that can be untraceably sent across network.

See Also: Linker (→159, *page 464*)
Author: Venca

154.1

Chopper (Mix * parent, LogFile * chopperLogFile)

Mix chopper constructor

Mix chopper constructor.

Initializes lastMessageID with a random number so that probability of conflicting Message ID's is minimized. This Message ID is later used for marking chunks of one message as that they belong together. It also initializes its parent classes.

Parameters: `parent` — Pointer to Mix objects.
`chopperLogFile` — Log file pointer.
See Also: ~Chopper()
Author: Venca

154.2

list < GMessage * > **CreateChunks** (GMessage * parameters)

Create chunks from message

Create chunks from message.

Chops a message.

<PRE> [Onion—Mix]: AddHops: Data: </PRE>

and splits it into multiple chunks of the form:

<PRE> [Onion—Mix] AddHops Data: Message-ID: Chunk-ID: Total-Chunks: Message-Length: Data: </PRE>

Return Value:	list of chunks of the structure: <PRE> [Onion—Mix] AddHops Data: Message-ID: Chunk-ID: Total-Chunks: Message-Length: Data: </PRE>
Parameters:	`parameters` — The GMessage has the following structure: <PRE> [Onion—Mix]: AddHops: Data: </PRE>
See Also:	Linker, StoreChunk(), traffic analysis discussion
Author:	Venca

154.3

void* **Run** (void *)

Mix chopper Run() method

Mix chopper Run() method.

This methods processes all messages from the incoming queue. If the message is for local Mix then it is passed to Majordomo for further processing. If it is destined to be sent over the network then the message has to be prepared for transmission. That means splitting it into uniform chunks so that they cannot be distinguished and then wraping some additional peels onto its onion - in this way the sender is protected against traffic analysis too.

See Also:	MessageCreator, Linker, traffic analysis discussion
Author:	Venca

154.4

void **Shutdown** ()

Prepare for clean shutdown

Prepare for clean shutdown.

This method should implement all the tasks necessary for preparation for a clean shutdown like saving inconsistent data to disk etc. Now it is empty as Chopper is completely stateless as the loss of some partially received messages is not fatal to the system.

See Also:	Shutdownable, Killer
Author:	Venca

155

class **ChunkData**

Chunk of data

Public Members

Chunk of data.

Represents one chunk of data as it was created by Chopper

See Also: Linker, PartialMessage, PartialMessages, Chopper
Author: Venca

155.1

ChunkData ()

Chunk of Data constructor

Chunk of Data constructor.

Initializes length of chunk data to 0 present sign to false and pointer to data to NULL.

See Also: PartialMessage (→157, *page 462*)
Author: Venca

156

typedef vector <ChunkData *> **PresentChunks**

Chunks that already arrived

Chunks that already arrived.

Stores those chunks of message that alredy arrived. When the message is complete it can be joined together.

See Also: ChunkData, Linker
Author: Venca

157

class **PartialMessage**

Partially received message

Public Members

	ChunkData** **presentChunks**	*received chunks*	
	Size **messageSize**	*Info about complete message*	
157.1	**PartialMessage** (Size msgSize, Size numOfChunks)	*Partial message constructor*	462

Partially received message.

Represents all the received chunks, size of complete message, total number of chunks.

See Also: Linker, ChunkData, PartialMessage, PartialMessages
Author: Venca

157.1

PartialMessage (Size msgSize, Size numOfChunks)

Partial message constructor

Partial message constructor.

Constricts an internal structure for linker that will hold all the chunks of a message.

Parameters: `msgSize` — Total size of the original message
`numOfChunks` — Number of chunks into which the original message was split.
See Also: PartialMessages, StoreChunk()
Author: Venca

158

typedef map <ID, PartialMessage *, less<ID> > **PartialMessages**

Partially received messages

Partially received messages.

Represents all the received messages.

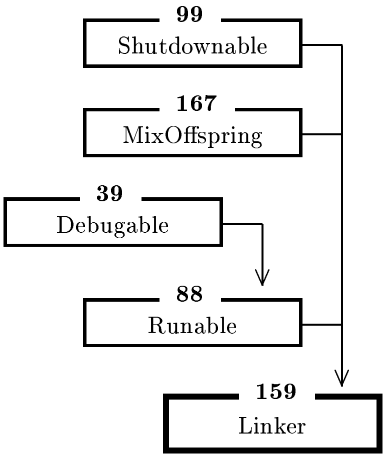
See Also: Linker, ChunkData, PartialMessage, PartialMessages
Author: Venca

159

class **Linker** : public Runable, public MixOffspring, public Shutdownable

Messages linker

Inheritance



Public Members

methods

159.1		**Linker** (Mix * parent, LogFile * linkerLogFile)	*Mix linker constructor*	465
159.2	void*	**Run** (void *)	*Mix linker Run() method*	465
159.3	void	**Shutdown** ()	*Prepare for clean shutdown*	466
159.4	GMessage*	**StoreChunk** (GMessage * chunk)	*Store chunk of message into internal structures*	466

Protected Members

attributes

Messages linker.

Collects chunks of messages (created by chopper) as they arrive, stores them until the message is complete and then joins the chunks so that it reconstruct the original message.

See Also: ChunkData, PartialMessage, PartialMessages
Author: Venca

159.1

Linker (Mix * parent, LogFile * linkerLogFile)

Mix linker constructor

Mix linker constructor.

Allocates a new object of type PartialMessages for storing chunks for different messages.

Parameters: `parent` — Pointer to Mix object.
`parent` — linkerLogFile Log file.
See Also: PartialMessage (→157, *page 462*)
Author: Venca

159.2

void* **Run** (void *)

Mix linker Run() method

Mix linker Run() method.

Processes all messages from the incoming queue and inserts them into its internal structures. Has to watch if a message is already completed and then it kinks all its chunks together and passes the final message further to Majordomo for processing.

Return Value: void pointer
See Also: Chopper, Majordomo
Author: Venca

159.3

void **Shutdown** ()

Prepare for clean shutdown

Prepare for clean shutdown.

This method should implement all the tasks necessary for preparation for a clean shutdown like saving inconsistent data to disk etc. Now it is empty as Linker is completely stateless as the loss of some partially received messages is not fatal.

return void

See Also: Shutdownable, Killer
Author: Venca

159.4

GMessage* **StoreChunk** (GMessage * chunk)

Store chunk of message into internal structures

Store chunk of message into internal structures.

Actually performs the task of storing message into its internal structures, checking for Message ID conflicts and checking if the message is already complete. If that is true than a completely linked message is returned.

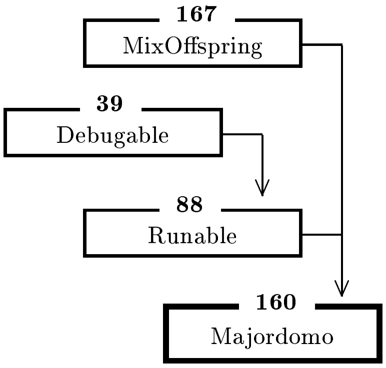
Return Value:	Completely linked message if any else NULL.
Parameters:	`chunk` — Message from network that represents one chunk of original message and as such has to be linked with all the others to reconstruct it.
See Also:	Majordomo::Run(), Chopper, Chopper::CreateChunks()
Author:	Venca

160

class **Majordomo** : public Runable, public MixOffspring

Mix majordomo

Inheritance



Public Members

methods

160.1 Err **DRCMoneyFromClient** (MsgField *dataFld)
Serves command from client 470

160.2 Err **DRCRFSGetFields** (const MsgField *dataFld,
GMessage **replyToAC,
TID *replyToTID,
Size *fileSize,
RelTime **storageTime)
Parses GMessage comming with CE_REQ_FOR_STOR command 470

160.3 Err **DRCReplyAllocation** (const MsgField *dataFld)
Serves the CE_REPLY_ALLOCATION message 471

160.4 Err **DRCReplyStorage** (const MsgField *dataFld)

Private Members

methods

Mix majordomo.

Processes various requests comming from network and from SIX also distributes messages to apropriate objects. <p> Commands that it handles are: generate_onion get_certificates publish_certificates.

See Also: MessageCreator, GM_COMMAND

Author: Pechy, Venca

160.1

Err **DRCMoneyFromClient** (MsgField *dataFld)

Serves command from client

Serves command from client.

Handles command that Client sends after successfull storage of his file on Eso, to inform Bank about the stored file owner.

Return Value: error code
Parameters: `dataFld` — Data comming along with command
See Also:
Author: Ivana

160.2

Err **DRCRFSGetFields** (const MsgField *dataFld, GMessage **replyToAC, TID *replyToTID, Size *fileSize, RelTime **storageTime)

Parses GMessage comming with CE_REQ_FOR_STOR command

Parses GMessage comming with CE_REQ_FOR_STOR command.

GMessage comming with command CE_REQ_FOR_STOR should have this structure: <pre> Reply-To-ID: ID of a message assigned by sender (Client) File-Size: Size of file Client whishes to store Storage-Time: RelTime for which the file should be stored </pre>

Return Value: error code
Parameters: `dataFld` — data comming with command
`replyToAC` — out parameter for Reply-To access certificate
`replyToTID` — out parameter for Reply-To message ID
`fileSize` — out parameter for size of file whished to stored
`storageTime` — out parameter for time for which the file should be stored
See Also: DRCRequestForStorage()
Author: Pavel

160.3

Err **DRCReplyAllocation** (const MsgField *dataFld)

Serves the CE_REPLY_ALLOCATION message

Serves the CE_REPLY_ALLOCATION message.

Looks into the messages, whether or not the allocation was succesful. If yes, adds the Eso certificate to yesCertificates ArrayGMessage.

Return Value: error code
Parameters: `dataFld` — data from the incoming message
See Also: Banker
Author: Pavel

160.4

Err **DRCReplyStorage** (const MsgField *dataFld)

Serves the reply to CE_DATA_TO_STOR Looks into the message, whether the storage was successful and prints the result to the stdout

Serves the reply to CE_DATA_TO_STOR

Looks into the message, whether the storage was successful and prints the result to the stdout.

Return Value: error code
Parameters: `dataFld` — Data from the incoming message.
See Also: Banker
Author: Pavel

160.5

include . H majordomo.h Err **GenerateThisSixAC** (MsgField *onion, GMessage **generatedAC)

Generates access certificate for this Bank

Generates access certificate for this Bank.

Creates asymetric keys and generates access certificate of this Bank using the keys and the onion.

Return Value: error code
Parameters: `onion` — Onion to use in AC.
`generatedAC` — AC generated in this method.
See Also: Six Eso::GenerateThisSixAC
Author: Marek

160.6

Err **GenerateThisSixAC** (MsgField *onion, GMessage **generatedAC)

Generates this Six AC

Generates this Six AC.

This Six AC is generated every time Client wants to communicate. New tuple of keys is generated and saved.

Return Value: error code
Parameters: `onion` — Onion to include to AC.
`generatedAC` — AC that was generated.
See Also: SixMajordomo (→175, *page 522*)
Author: Marek

160.7

Queue* **GetACs** (Char *type, const int count = 1, Queue *notIn = NULL)

Gets ACs

Gets ACs.

Overriden Six method. Client has always only one THIS_SIX AC, that is generated every time Client wants to communicate. The getting of THIS_SIX AC is thus overriden, otherwise the inherited method is called.

Return Value: Queue of ACs of given type.
Parameters: `type` — Type of ACs to get.
`count` — Count of ACs to get.
`notIn` — Queue of ACs not wanted.
See Also: SixMajordomo (→175, *page 522*)
Author: Marek

160.8

void* **Run** (void *)

Decides what to do with messages for which Mix is a final point

Decides what to do with messages for which Mix is a final point.

When a message ends in Mix, it can have different recipients - Six or Mix. We use GM_X_DESTINATION and GM_ORIGIN to decide which type of the message it is. Message can go from Mix to Mix, from Six to Mix and from Mix to Six. According to this, Majordomo passes chunks along the way or gives them to the object that is the real recipient in Mix.

See Also: GM_X_DESTINATION, GM_ORIGIN
Author: Venca, Pechy

160.9

void **SendData** (MsgField * data, GMessage * destination, string destinationType, Size hops)

Send data to onion

Send data to onion.

Sends data to onion or mix - method used by test methods in Client.

Return Value: void
Parameters: `data` — Data to send.
`destination` — Onion to send the data to.
`hops` — number of hops to add.
See Also:
Author: Pechy, Venca

160.10

include . H majordomo.h Err **SendToEsos** (Char *command, GMessage *msg, Int count)

Sends message to number of Esos

Sends message to number of Esos.

Sends the message to count randomly chosen Esos (creates count "send_data" mixMsgs and puts them into outMsgQueue)

Return Value: error code
Parameters: `command` — command to be sent to Esos
`msg` — message to be sent to Esos
`count` — count of Esos to send the message to
See Also: SixMajordomo (→175, *page 522*)
Author: Marek

160.11

void **StoreFile** (ArrayGMessage *certificates)

Stores file on Esos

Stores file on Esos.

Creates messages to store file on Esos with given certificates and waits for answers to them.

Return Value: void
Parameters: `certificates` — Certificates of Esos to store the file on.
See Also: Banker
Author: Pavel

161

Rough estimation about the maximum length of onion bottom and one onion peel

Rough estimation about the maximum length of onion bottom and one onion peel. Estimation is oversized a bit.

162

#define **HALF_REST_OF_PATH_LEN** (((BOTTOM_ONION_MAX_LENGTH_ESTIMATION \
+ (ONE_ONION_PEEL_MAX_LENGTH_ESTIMATION \
* MAX_HOPS_TO_ASK_FOR)

Because of fixed size of the ,,rest of path length", we have to limit the onion to (aproximately) half of FULL_REST_OF_PATH_LEN

Because of fixed size of the ,,rest of path length", we have to limit the onion to (aproximately) half of FULL_REST_OF_PATH_LEN.

See Also: FULL_REST_OF_PATH_LEN

163

#define **FULL_REST_OF_PATH_LEN** (((HALF_REST_OF_PATH_LEN \ + (ONE_ONION_PEEL_MAX_LENGTH_ESTIMATION \ * MAX_HOPS_TO_ASK_FOR)

Due to security reason, onion („the rest of path") should be always the same length

Due to security reason, onion („the rest of path") should be always the same length. The enemy cat't estimate how many hops is chunk still to go. So the macro is the fixed onion length.

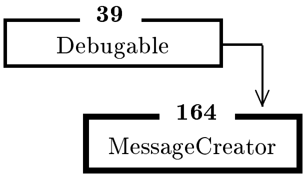
See Also: HALF_REST_OF_PATH_LEN

164

class **MessageCreator** : public Debugable

Message Creator

Inheritance



Public Members

164.2		**methods**		481
164.4	Err	**AddLocalCertifs** (GMessage *callBackOnionInfo)	*Adds local certificates to the certificates that are used for onion creation*	482
164.5	void	**AnnotateWith4ByteLength** (GMessage *onion, char **onionBottomStream, Size *onionBottomStreamLen)	*Add Size integer as a header denoting the lenght of data*	483
164.6	GMessage*	**CreateChunk** (GMessage * chunkProperties)	*Creates a chunk*	483
164.7	include . .. .. Common H common.h GMessage*	**CreateOnion** (GMessage *onionProperties)	*Creates the onion with owner identification in the bottom*	484
164.8	include . .. .. Common H common.h GMessage*			

164.15 include . .. .. Common H common.h Err

Protected Members

Private Members

Message Creator.

Specializes in creation of messages, chunks onions and similar things. Understands structure of onions and messages.

See Also: PaddingGenerator, Mix
Author: Pechy

164.2

methods

Names

MessageCreator (ConfigFile * mcConfigFile, LogFile * messageCreatorLogFile, CertificateRepository * certificateRepositorya, Cipherer * cipherer, KeyManager * keyManager)
Constructor.

GMessage*

CreateChunk (GMessage *message)
Creates a chunk in a format that is suitable for Sender.

GMessage*
CreateOnion (GMessage *request)
Creates an onion. The recipient is in the bottom.

GMessage*
CreateOnionFromMixCertificate (GMessage *request)
For the need of Mix only. When asking for certificates for Acs.

~MessageCreator ()
Desctructor.

164.4

Err **AddLocalCertifs** (GMessage *callBackOnionInfo)

Adds local certificates to the certificates that are used for onion creation

Adds local certificates to the certificates that are used for onion creation.

When we create the onion, the onion's bottom identifies the recipient. For not to create the onion in the two steps, we add local mix certificates to the certificates that are used for onion creation. The new local certificates are the first certificates that are used when the onion is created (identifiing the Mix as a final Mix).

Return Value: KO when Mix certificate can't be returned, OK otherwise.
Parameters: `callBackOnionInfo` — Contains the certificates.
See Also: MakeMoreOnionLayers()
Author: Pechy

164.5

void **AnnotateWith4ByteLength** (GMessage *onion, char **onionBottomStream, Size *onionBottomStreamLen)

Add Size integer as a header denoting the lenght of data

Add Size integer as a header denoting the lenght of data.

When we layer particular peels, we need convert the upper new onion peel (the peel without GM_REST_OF_PATH) into a stream. To be able to recognize how long is the onion, we have to include 4 bytes header as a length of the onion.

Return Value:	void
Parameters:	`onion` — This onion is converted to an octet stream and annotated with the header. `onionBottomStream` — The stream that is returned. `onionBottomStreamLen` — Size of returned stream.
See Also:	MakeMoreOnionLayers()
Author:	Pechy

164.6

GMessage* **CreateChunk** (GMessage * chunkProperties)

Creates a chunk

Creates a chunk. Chunk is a part of message, is of fixed size.

When anybody wants to send a message, the message can be of any size. To make traffic analysis harder, we send the message in fix sized chunks. The method creates one chunk.

Return Value: Cooked chunk: <pre> GM_RECIPIENT_1 - see CreateOnion() above for discussion on on recipients in a peel GM_RECIPIENT_2 GM_CHUNK GM_DEBUG - some string, for debugging purpose GM_PROTOCOL_VERSION - Mix-Mix protocol identification GM_DATA - encrypted data GM_REST_OF_PATH - rest of path </pre>

Parameters: `chunkProperties` — Information needed to create the chunk: <pre> GM_ONION - chunk's recipient idenfification GM_ADD_HOPS - how many peels to add on the top of the onion GM_DATA - the data thas are to be transmitted inside the chunk, we are not interested in its structure here. </pre>

See Also: CreateOnion(), MessageIDRepository, Receiver, Sender.

Author: Pechy

164.7

include . .. .. Common H common.h GMessage* **CreateOnion** (GMessage *onionProperties)

Creates the onion with owner identification in the bottom

Creates the onion with owner identification in the bottom.

Onion is a parallel for ,,an address". It is the identification of some service that is running somewhere. The onion carries the information about the chain of Mixes through the chunk has to go to the recipient. In each peel, there can be several recipients (in the last one, only the final recipient has to be included). This redundancy is useful for reliability (when one Mix in a peel failes, we have still some Mixes which we can try to connect to) as well as for security enhancement. This security enhancement is in the fact that the message consisted of many chunks is not feeded through the one exact channel only, but is spreaded a bit. It is more difficult to tail this stream of chunks for enemy.

Return Value: Onion.

Parameters: `onionProperties` — What is needed for onion creation: <pre> GM_HOP_COUNT - how many peels we are to create GM_ORIGIN - whose idenfification will be in the very bottom (eg. eso_johny, joes_acs etc., the important fact is that this string identifies the Six above the final Mix, so is is not the Internet address) </pre>

See Also: CreateChunk(), CreateOnionFromMixCertificate(), certificates, sixs' identification, image of onion and message on our www site

Author: Pechy

164.8

include . .. .. Common H common.h GMessage* **CreateOnionBottom** (char *xOrigin, GMessage *symKeys, GMessage **pubKeyInfo)

Creates an onion bottom with Six's identification (or Mix's sometimes)

Creates an onion bottom with Six's identification (or Mix's sometimes).

In the bottom there is an identification of the recipient running above the final Mix (sometimes the recipient can be Mix itself). The bottom also includes all symetric keys that were used to encrypt the data part of chunk as the chunk went along the Mixes from this onion. The sym keys are included because we needn't assign a unique ID to the onion and store symetric keys with this ID. Freshly generated public key is also included to the bottom, to make future middle attack harder.

Return Value: Bottom GMessage: <pre> GM_FINAL_STOP - a nonempty field, identifies that the peel is tha last one GM_RECIPIENT_1 GM_ADDRESS - a name of the recipient GM_PRIVATE_KEY_INFO - private key for data decryption GM_SYMETRIC_DATA_KEY_INFOS - symetric keys for data decryption </pre>

Parameters:
`xOrigin` — The name of the recipient.
`symKeys` — Symetric keys we have to insert into the bottom.
`pubKeyInfo` — Symetric keys we have to insert into the bottom.

See Also: CreateOnion(), CreateChunk(), middle attack discussion.

Author: Pechy

164.9

GMessage* **CreateOnionFromMixCertificate** (GMessage *onionProperties)

Creates the onion with Mix's identification in the bottom

Creates the onion with Mix's identification in the bottom.

Sometimes Mix is the one that asks for something another server (eg. Acs, on the behalf of a Six or Mix himself), so we need an onion, whose recipient in the bottom is directly Mix.

Return Value:	Onion.
Parameters:	`onionProperties` — What is needed for the onion creation. <pre> GM_DESTINATION - which Mix object will get the answer GM_MIX_CERTIFICATE - local identification of Mix M_HOP_COUNT - how many peels to create </pre>
See Also:	CreateOnion(), MessageIDRepository
Author:	Pechy

164.10

Err **MakeMoreOnionLayers** (GMessage *properties, GMessage *callBackOnionInfo, GMessage *onion)

Add some peels on the top of given onion

Add some peels on the top of given onion.

It gets an onion (recipient's identification) and should add some peels. The onion an be a real onion (real Six's identifification) or an onion bottom created with CreateOnionBottom() method (in this case onion bottom simulates actuall onion. <p> We add layers in a loop, in each loop we do: <ul> add symetric data key to the onion (this key is used to encode the data), this __data__ key is taken from a sym keys' message (symDataKeys) taken from symetric keys' gMsg included in callBackOnionInfo <li> remove GM_REST_OF_PATH from onion, but convert it to stream and keep it aside <li> convert onion (i.e. already without GM_REST_OF_PATH) to a stream and add 4 bytes as a length indicator header <li> cat both streams (onion-stream first, then GM_REST_OF_PATH) and encrypt it with symetric __path__ key, taken from symetric keys' gMsg included in callBackOnionInfo <li> add recipients, each will include symetric path key (from 4.) encrypted with respective public key taken from respecive certificates <p> Note: if onion is just an onionBottom, we add in the first step a symetric key that is never used.. This is just that we didn't want to add any additional if, the best programmers are lazy programmers :-).

Return Value: Onion. <pre> GM_RECIPIENT_1 - see MakeRecipientGMessage() GM_RECIPIENT_2 GM_REST_OF_PATH - is an encrypted GMessage <<GM_RECIPIENT_1>> <<GM_RECIPIENT_2>> <<GM_SYMETRIC_DATA_KEY_INFO>> <<remained octet-stream>> <<GM_RECIPIENT_1>> <<GM_RECIPIENT_2>> <<GM_SYMETRIC_DATA_KEY_INFO>> <<remained octet-stream>> ... ... </pre>

Parameters:
`callBackOnionInfo` — <pre> GM_SYMETRIC_DATA_KEY_INFOS - used for encryption in particular peels GM_MIX_CERTIFICATES - used for creation recipients' GMessages in particular peels </pre>
`properties` — Contains: <pre> GM_ADD_HOPS - when we create a chunk </pre> <p> or <pre> GM_HOP_COUNT - when we create a onion </pre>
`onion` — Onion on which we have to add additional peels.

See Also: CreateOnion(), CreateChunk, Cipherer.

Author: Pechy

164.11

GMessage* **MakeRecipientGMessage** (GMessage *mixCertif, GMessage *symKey)

Makes GMessage with recipient's identification

Makes GMessage with recipient's identification.

When creating onions and chunks, is each peel there are two recipients' GMessages. Each can be used to send the chunk to.

Return Value: Recipient's GMessage: <pre> GM_PUBLIC_KEY_INFO GM_SYMETRIC_PATH_KEY_INFO GM_ADDRESS GM_PORT </pre>

Parameters:
`mixCertif` — Contains the recipient's data (address and port).
`symKey` — This symKey is used somewhere else for path encryption. For recipient to be able to use it, we encrypt the symetric key with recipient's public key.

See Also: MakeMoreOnionLayers()

Author: Pechy

164.12

MessageCreator (ConfigFile *mcConfigFile, LogFile * messageCreatorLogFile, CertificateRepository * certificateRepository, Cipherer * cipherer, KeyManager * keyManager)

Message Creator constructor

Message Creator constructor.

Saves object pointers he gets in parameters for future use.

Parameters:
`mcConfigFile` — Configuration file.
`messageCreatorLogFile` — Log file.
`certificateRepository` — Certificate repository object.

See Also: Translator, Cipherer, ~MessageCreator()

Author: Pechy

164.13

char* **PaddOnionRestOfPath** (char *restOfPath, Size restOfPathLen, Size fixedSize)

Padd the rest of path field to the original length

Padd the rest of path field to the original length.

When the chunk goes through the chain of Mixes, the recipient's identification (rest of path) is going down in size. This fact would enable the onion length attack (the enemy could estimate how many hops is the chunk to go to the final Mix), so this method padds the rest of path field with randomly generated bytes.

Return Value:	fixedSize Size to which we padd the restOfPath parameter.
Parameters:	`restOfPath` — Rest of path, downsized by the current peel. `restOfPathLen` — Rest of path length.
See Also:	MakeMoreOnionLayers(), Translator methods
Author:	Pechy

164.14

Err **ReturnCallBackOnionInfo** (GMessage *onionProperties, GMessage **callBackOnionInfo)

Returns GMessage full of useful stuff for creating onions or adding new peels onto the top of them

Returns GMessage full of useful stuff for creating onions or adding new peels onto the top of them.

When we create new onions or add some peels onto the of them, we have to symetric keys for encryption and Mixs' certificates to make peels. Generating of new symetric keys and returning of mixs' certificates should be mutually related. Recipients in one tuple (peel) has to support the same symetric cipher and on the contrary, the particular symetric key cipher has to be supported by all recipients in the tuple. We do *NOT* honor this rule for now - but this method (and several others) could be easily modified to support this (onionProperties would carry more information needed).

Return Value: GMessage with the stuff: <pre> GM_SYMETRIC_DATA_KEY_INFOS GM_MIX_CERTIFICATES </pre>

Parameters: `onionProperties` — How many peels will be created: <pre> GM_HOP_COUNT </pre>
`callBackOnionInfo` — Returned parameter.

See Also: MakeMoreOnionLayers(), CreateChunk(), CreateOnion()

Author: Pechy

164.15

include . .. .. Common H common.h Err **ReturnGMsgFullOfMixCertifs** (int tupleWidth, Size hopCount, GMessage **certificates)

Returns Mixs' certificates

Returns Mixs' certificates.

When we create chunks or onions, we add some peels onto the top of the particular onion. Each peel means to have 2 recipients, ie. 2 Mixs' certificates.

Return Value: KO if requested count can't be satisfied, OK otherwise. <pre> GM_MIX_CERTIFICATE_COUNT GM_MIX_CERTIFICATE_TUPLE_1 GM_MIX_CERTIFICATE_1 GM_MIX_CERTIFICATE_2 ... ... GM_MIX_CERTIFICATE_TUPLE-"Hop-Count" </pre>

Parameters: `tupleWidth` — How many recipients are in one peel.
`hopCount` — How many certificates' tuples to return.
`certificates` — Returned certificates.

See Also: CreateOnion(), CreateChunk()
Author: Pechy

164.16

˜MessageCreator ()

Message Creator destructor

Message Creator destructor.

Now does nothing since Message Creator is stateless.

See Also: MessageCreator()
Author: Pechy

164.1

attributes

Names

KeyManager* **keyManager** *We ask him for Mix's asymetric keys.*

Cipherer* **cipherer** *Needed when creating chunks and onions.*

CertificateRepository* **certificateRepository** *Repository containing certificates.*

ConfigFile* **configFile** *Configuration file.*

164.3

methods2

Names

Err **ReturnGMsgFullOfMixCertifs** (int tupleWidth, Size hopCount, GMessage **cerfifsGMsg)

Returns Mix's certificates that we use for onion or chunk creation.

GMessage* **CreateOnionBottom** (char *xOrigin, GMessage *symKeys, GMessage **pubKeyInfo)

Creates the onion bottom with symetric keys and recipient identification

GMessage* **MakeRecipientGMessage** (GMessage *mixCertif, GMessage *SymKey)

Creates recipient's GMessage that is in each onion peel.

void **AnnotateWith4ByteLength** (GMessage *onion, char **onion-BottomStream, Size *onionBottomStreamLen)

Add Size type integer before the data (it's denotes the data length).

Err **MakeMoreOnionLayers** (GMessage *properties, GMessage *callBackOnionInfo, GMessage *onion)

Add additional peels on the top of an onion.

Err **ReturnCallBackOnionInfo** (GMessage *onionProperties, GMessage **callBackOnionInfo)

Returns certificates and symetric keys for onion and chunk creation.

char* **PaddOnionRestOfPath** (char *restOfPath, Size restOfPathLen, Size requestedLen)

Padd „ rest of path" field to the original length.

165

#define **THREAD_STACK_SIZE**

thread stack is not very large by default, so let's adjust that

166

class **Mix**

Mix server

Public Members

Private Members

methods

Mix server.

Implements the ability to route messages in the spirit of onion routing slightly enhanced. The message that mix receives is decrypted so that it is possible to find out where to send the message next. The piece of information about where to send the message consists of two (that is what we chose but any finite number would go) recipients. This allows Mix for a much bigger reliability when delivering messages routed through many hops.

It also implements some techniques suitable for making traffic analysis harder. These techniques are: <ul> Header padding <li> Data padding <li> Onion routing <li> Traffic padding

See Also: Acs, Eso, Client, Bank, Translator, Padder, PaddingGenerator

Author: Pechy, Venca

166.1

attributes

Names

RunningObject*
runningObject *object needed for waiting for running objects (now threads, but possibly processes)*

166.2

Err **InitializeRandomGenerator** ()

Initializes a random generator

Initializes a random generator.

We use normal random() system function. This function is not the best one, but is still useable. We use /dev/urandom for seeding the generator (srandom()). The /dev/urandom should be good enough, if not, we can easily modify the method to use /dev/random.

Return Value: KO if the generator can't be initialized, OK otherwise.
Parameters: `void` —
See Also: random(4), srandom(3), random(3) man pages.
Author: Pechy

166.3

Mix (string configName)

Mix constructor

Mix constructor.

Creates all the objects necessary to run Mix server.

Parameters:	`configName` — Config file name form where to read configuration information.
See Also:	All other objects
Author:	Venca, Pechy

166.4

void **Run** (void)

Creates and starts all running objects (ie

Creates and starts all running objects (ie. threads).

Just creates all runnig objects and then falls asleep waiting for them.

Return Value:	void
See Also:	All running objects
Author:	Venca, Pechy

166.5

RunningObjectID **StartObject** (Runable *object)

Starts one running object

Starts one running object.

In this method, the running object is started. For now, the thread is started, but the method could be modified to start normal process (in the system that doesn't support threads). Be portable!

Return Value: RunningObjectID Running object ID, we can then wait for it.
Parameters: `object` — Running object created by Run().
See Also: Run(), threads discussion
Author: Pechy

166.6

~Mix ()

Mix destructor

Mix destructor.

Deletes all objects created in constructor. This should happen only after catching SIGTERM signal indicating the shutdown.

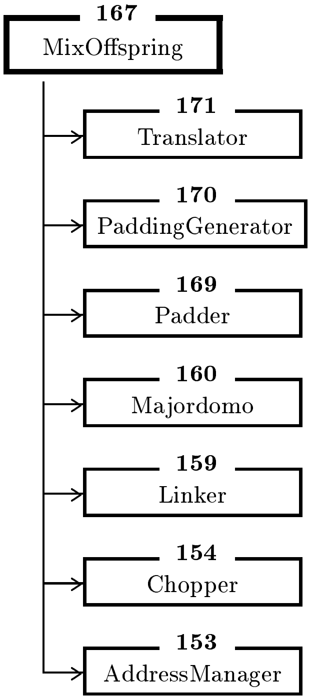
See Also: All other objects
Author: Venca, Pechy

167

class **MixOffspring**

Mix offspring

Inheritance



Public Members

methods

167.1 **MixOffspring** (Mix * mix)
Remembers pointer to its father 500

Protected Members

attributes

Mix offspring.

Maintains a pointer to it's parent so that descendands of this class can access some members of Mix. It is useful for objects that communicate heavily with other objects inside of Mix.

See Also: Mix, Majordomo
Author: Venca

167.1

MixOffspring (Mix * mix)

Remembers pointer to its father

Remembers pointer to its father.

Remembers pointer to its father. Some objects need pointers to almost all of other objects, so we made them mix's offsprings for not to have to include all pointers as parameters.

Parameters: `mix` — Mix main object.
See Also: Mix (→166, *page 495*)
Author: Venca

168

enum **MessageType**

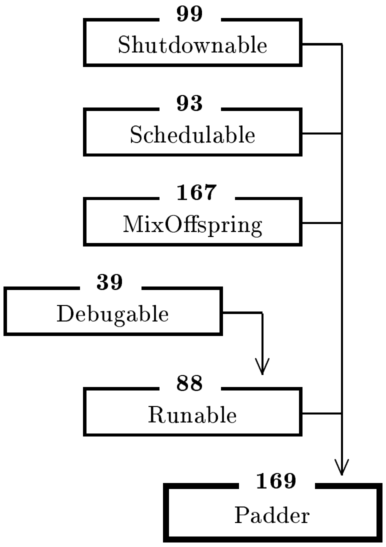
Message type

169

class **Padder** : public Runable, public MixOffspring, public Schedulable, public Shutdownable

Mix traffic padder

Inheritance



Public Members

attributes

methods

169.1		**Padder** (Mix * parent, LogFile *padderLogFile)	*Padder constructor*	503
169.2	void*	**Run** (void *)	*Padds the stream of chunks that goes through the Mix*	503
169.3	void	**Shutdown** ()	*Prepare for clean shutdown*	504
169.4		**~Padder** ()	*Padder destructor*	504

Mix traffic padder.

Pads the traffic (generates additional messages, delays despatching messages, ...) All these techniques make traffic analysis harder.

See Also: PaddingGenerator (→170, *page 505*)
Author: Pechy, Venca

169.1

Padder (Mix * parent, LogFile *padderLogFile)

Padder constructor

Padder constructor.

Initializes himself with some information taken from the config file.

Parameters: `parent` — Mix object.
`padderLogFile` — Log file.
See Also: padding discussion, traffic analysis discussion, ~Padder()
Author: Venca

169.2

void* **Run** (void *)

Padds the stream of chunks that goes through the Mix

Padds the stream of chunks that goes through the Mix.

Simple strategy (the simpler, the best -> see unix...) for now.

See Also: discussion on padding generation, traffic analysis discussion
Author: Venca

169.3

void **Shutdown** ()

Prepare for clean shutdown

Prepare for clean shutdown.

This method should implement all the tasks necessary for preparation for a clean shutdown like saving inconsistent data to disk etc. Now it is empty as Padder is completely stateless as the loss of some partially received messages is not fatal to the system.

Return Value: void
See Also: Shutdownable, Killer
Author: Venca

169.4

~Padder ()

Padder destructor

Padder destructor.

Does nothing for now, since Padder is stateless.

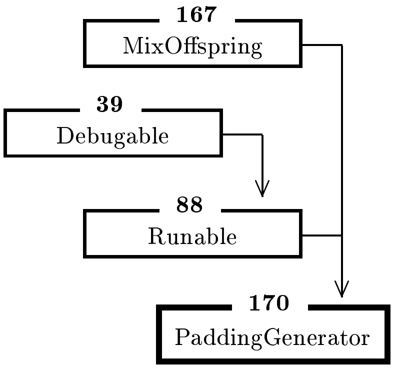
Author: Venca

170

class PaddingGenerator : public Runable, public MixOffspring

Padding messages generator

Inheritance



Public Members

methods

170.1		**PaddingGenerator** (Mix * parent, LogFile * logFile)	*PaddingGenerator constructor* ..	506
170.2	void*	**Run** (void *)	*The main loop for Padder thread*	506
170.3	void	**SendPaddingMessage** ()	*Send a padding message*	507

Protected Members

methods

Padding messages generator.

It just generates padding messages that are used to fill gaps between traffic peaks. Makes a reservoir of messages for Padder to send when there are no messages waiting.

See Also: Padder, traffic analysis discussion, padding discussion
Author: Venca

170.1

PaddingGenerator (Mix * parent, LogFile * logFile)

PaddingGenerator constructor

PaddingGenerator constructor.

Saves parameters that gets.

Parameters: `parent` — Pointer to Mix main object.
`logFile` — Log file.
See Also: Padder, traffic analysis discussion
Author: Venca

170.2

void* **Run** (void *)

The main loop for Padder thread

The main loop for Padder thread.

In a loop, cycles and sends padding messages.

Return Value: void pointer
Parameters: `void` — pointer
See Also: Padder, traffic analysis discussion
Author: Venca

170.3

void **SendPaddingMessage** ()

Send a padding message

Send a padding message.

Following a simple algorithm, creates and sends a padding message. The algoritm, though simple, is very powerful and hard to break by the enemy.

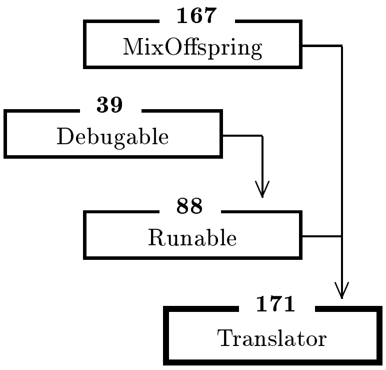
Return Value:	void
Parameters:	`void` —
See Also:	Padder, traffic analysis discussion
Author:	Venca

171

class **Translator** : public Runable, public MixOffspring

Message translator

Inheritance



Public Members

171.1	char*	**PaddRestOfPath** (char *path, Size pathLen, Size peelLen)	*Padds rest of path field to the original length*	509
171.2	void*	**Run** (void *)	*The main method that do the things*	509
171.3	include . .. .. Common H common.h	**Translator** (Mix * parent, LogFile * translatorLogFile)	*Translator constructor*	510
171.4		**~Translator** ()	*Translator destructor*	511

Private Members

methods

Message translator.

Decrypts one peel from message that is sent using onion routing. In the decrypted peel it finds the information necessary to decide whether the message is at its destination or whether it should be send to further to another Mix.

See Also: Cipherer, MessageCreator
Author: Venca, Pechy

171.1

char* **PaddRestOfPath** (char *path, Size pathLen, Size peelLen)

Padds rest of path field to the original length

Padds rest of path field to the original length.

When the chunk goes through the chain of Mixes, the rest of path field is downsizing. This fact could be used to try the onion length attack and recognize, how many hops the chunk is to go yet. The enemy then could try to chose what is easier to break (the Mixes that are close).

Return Value: Padded rest of path.
Parameters: `path` — Path to be padded.
`pathLen` — Current path length.
`peelLen` — Length to which to padd the path.
See Also: Translator, onion length attack discussion
Author: Pechy

171.2

void* **Run** (void *)

The main method that do the things

The main method that do the things.

WARNING: the method is long. <p> This method translates the incoming chunk into the outcoming one. First, the method decrypts the GMessage where is some information useful for further chunk translation. The GMessage is encrypted with public key of this Mix (the public key that was included into the Mix's certificate during certification creation). By decrypting this, Run() gets a symetric key that is used to decrypt information about next possible recipients. The combination of asymetric and symetric cryptography is used because we want to be able to have more than one recipient in each peel. <p> After decryption by symetric key, we have to check whether there is a final stop indication: <ul> Yes: we have to decrypt multiple encrypted data (as the data went through the chain of Mixes related to the recipient's onion). The symetric keys should be already in the dectypted onion bottom (ie. rest of path bottom). After the dectyption the data has to be dectypted by a private asymetric key included also in the bottom. See middle attack discussion for more info. That the data are passed to Linker. No (final stop not found): we strip from the decrypted rest of path field (decrypted by symetric key that was in turn decrypted by Mix's private key). The stripped data part is indicated by the first sizeof(Size) bytes in the decrypted rest of path field. From this part we make a GMessage. This GMessage contains needed information about the next possible recipients. After stripping the rest of path field, we have to padd it to the original length. Then we have to encrypt the data by another symetric key found in the stripped GMessage. Next, we take the data, padded rest of path, several recipients (got from stripped GMessage) and make a new chunk, suitable for Sender.

Return Value: void pointer
See Also: CreateChunk(), CreateOnion(), Cipherer
Author: Pechy, Venca

171.3

include . .. .. Common H common.h **Translator** (Mix * parent, LogFile * translatorLogFile)

Translator constructor

Translator constructor.

Saves object pointers he gets as parameters.

Parameters: `parent` — Parent, ie. Mix object.
`translatorLogFile` — Log file.

See Also: ~Translator()

Author: Venca

171.4

~**Translator** ()

Translator destructor

Translator destructor.

Now does nothing.

Author: Pechy or Venca

172

include . .. Common H common.h int **main** (int argc, char ** argv)

Mix main function

Mix main function.

Parses command line options and sets global variables according to switches specified It also creates Mix's main object and executes his method Run()

See Also: Mix (→166, *page 495*)
Author: Venca

173

Six constants

Names

#define **CFG_SEND_DATA_HOPS**
hops in sent data

#define **CFG_HOPS_IN_AC**
hops in AC

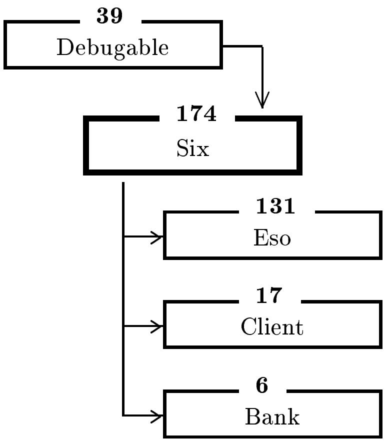
#define **CFG_SIX_PATH**
label for SixPath

174

class **Six** : public Debugable

Ancestor of some Services running above Mix

Inheritance



Public Members

174.3		**attributes**	....	515
174.4		**methods**	....	516
174.5	void	**CreateMajordomo** ()	*Creates majordomo*	517
174.6	void	**CreateRunningObjects** ()	*Creates running objects*	517
174.7	Char*	**GetMixSixVersion** ()	*Returns SixMix protocol version*	517
174.8	Char*	**GetPath** (Char *pathSuffix)	*Returns the six path with appended pathSuffix*	518
174.9	void	**Run** ()	*Function running Six*	518
174.10		**Six** (string configName)	*Creates all the objects necessary to run*	519
174.11	RunningObjectID			

Ancestor of some Services running above Mix.

Abstract class that provides capability to communication with Mix server. Sender, Receiver and msgQueues to and from majordomo are created and common information from the SixConfigFile is got.

See Also: Sender, Receiver, MessageQueue, ConfigFile, LogFile, Debugable

Author: Pavel, Marek

174.3

attributes

Names

MessageQueue*	**receiverToMajordomo**	*Queue connecting receiver to majordomo.*
MessageQueue*	**majordomoToSender**	*Queue connecting majordomo to sender.*
Cipherer*	**cipherer**	*Cipherer for ciphering in- and outcoming messages.*

Killer*	**killer**	*Class responsible for shutdown of all objects.*
ConfigFile*	**configFile**	*Log file.*
Receiver*	**receiver**	*Class responsible for receiving commands from Mix.*
SixMajordomo*	**majordomo**	*Class responsible for handling all messages.*
Sender*	**sender**	*Class responsible for sending commands to Mix.*

174.4

methods

Names

	Six (string configName)	*Constructor.*
virtual	**~Six** ()	*Destructor.*
virtual void	**Run** ()	*Method that runs as a thread and processes all requests.*
Char*	**GetMixSixVersion** ()	*Method that returns MixSix communication protocol version got from ConfigFile.*
Char*	**GetPath** (Char *pathSuffix = NULL)	*Method that adds path prefix got from ConfigFile before pathSuffix got as a parameter.*

174.5

void **CreateMajordomo** ()

Creates majordomo

Creates majordomo.

This method must be overloaded in Six descendants.

Return Value: void
See Also: SixMajordomo (→175, *page 522*)
Author: Pavel

174.6

void **CreateRunningObjects** ()

Creates running objects

Creates running objects.

Brings to life all running objects: Sender, Receiver, Majordomo.

Return Value: void
See Also: Receiver, SixMajordomo, Sender
Author: Pavel

174.7

Char* **GetMixSixVersion** ()

Returns SixMix protocol version

Returns SixMix protocol version.

Returns the protocol version got from the configFile in constructor.

Return Value: char* the version
See Also: ConfigFile (→37, *page 178*)
Author: Pavel

174.8

Char* **GetPath** (Char *pathSuffix)

Returns the six path with appended pathSuffix

Returns the six path with appended pathSuffix.

Creates string concatenating the sixPath (the path to SystemData of Six directory) and the path suffix and returns it.

Return Value: concatenated pathSuffix to sixPath
Parameters: `pathSuffix` — suffix to append behind the six path
See Also: ConfigFile
Author: Marek, Pavel

174.9

void **Run** ()

Function running Six

Function running Six.

Brings all the objets, that have to run, to life. (Each in a separate thread.)

Return Value: void
See Also: RunningObject
Author: Pavel

174.10

Six (string configName)

Creates all the objects necessary to run

Creates all the objects necessary to run.

In Six constructor the information from ConfigFile is read and the basic Six structure <pre> Receiver receiverToMajordomo queue

majordomoTosender queue Sender </pre> is created. Majordomo must be created in the descedants of Six.

Parameters:	`configName` — name of the ConfigFile
See Also:	Receiver, Sender, MessageQueue, SixMajordomo
Author:	Pavel

174.11

RunningObjectID **StartObject** (Runable *object)

Starts runable object

Starts runable object.

This function is used to start objects running in separate threads.

Return Value:	runningObjectID
Parameters:	`object` — object to start (must be descendant of Runable)
See Also:	Runable, RunningObject, RunningObjectID
Author:	Pavel

174.12

void **WaitForRunningObjects** ()

Waits for running objects forever

Waits for running objects forever.

Calls method of runningObject WaitForRunningObject, that waits for running object forever.

Return Value:	void
See Also:	RunningObject (→92, *page 285*)
Author:	Pavel

174.13

~Six ()

Six destructor

Six destructor.

Deletes all members.

See Also:	Receiver, Sender, MessageQueue
Author:	Pavel

174.1

attributes

Names

string **mixSixProtoVer** *MixSix protocol version got from ConfigFile*

string **path** *Path to the Six SystemData directory got from ConfigFile.*

RunningObject* **runningObject** *Object needed for waiting for running objects (now threads, but possibly processes).*

174.2 methods

Names

virtual void **CreateMajordomo** ()
In this method majordomo must be assigned a valid SixMajordomo ancestor pointer.

virtual void **CreateRunningObjects** ()
In this method running objects are created.

virtual void **WaitForRunningObjects** ()
In this method waiting for running objects to stop is started.

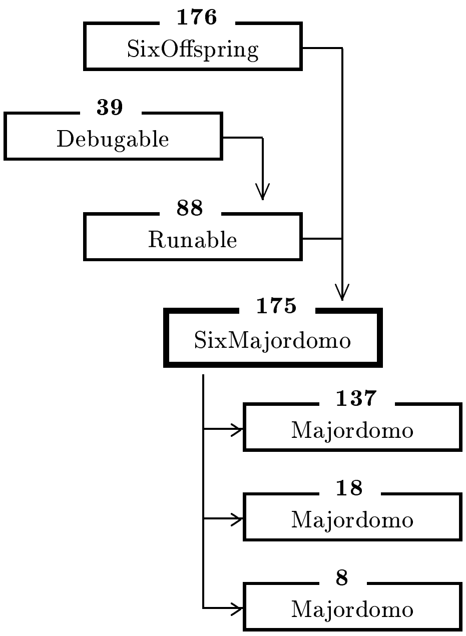
RunningObjectID **StartObject** (Runable *object)
Returns ID of a running object.

175

class **SixMajordomo** : public Runable, public SixOffspring

Object processing various requests and commands

Inheritance



Public Members

175.3 **methods** 526

175.4 Err **AskForOnion** () *Asks for onion, if necessary* ... 528

175.5 Err **BeginTransaction** (ID *newTID)
Starts the transaction 528

175.6 include . H sixMajordomo.h Err
CommandDataReceived (MsgField *paramFld)
Handles the Data_received command 529

175.7 Err **CommandOnionGenerated** (MsgField *dataFld)

Protected Members

Object processing various requests and commands.

Processes various requests comming from network and from Mix. Also distributes messages to apropriate objects.

See Also: Runable, SixOffSpring, Mix
Author: Pavel

175.3

methods

Names

	SixMajordomo (Six * parent, LogFile * majordomoLogFile)	*Constructor.*
virtual	**~SixMajordomo** ()	*Destructor.*
virtual void*	**Run** (void *)	*Method that runs the whole Eso - processes messages from receiverToMajordomo.*
GMessage*	**CreateSendDataMsg** (Char *commandSer, GMessage *dataSer, GMessage *ac, Size hops = CFG_SEND_DATA_HOPS)	*Creates "send_data" message.*
Err	**BeginTransaction** (ID *newTID)	*Starts new transaction and returns its ID.*
Err	**EndTransaction** (ID tid)	*Ends transaction with ID tid.*
ID	**GetTID** ()	*Returns ID of the most recently started transaction.*

175.3.1

Functions for getting information from ACManagers

Names

virtual ACManager*
GetPtrToACManager (Char *type)
Gets pointer to ACManager.

virtual Err
InsertAC (Char *type, GMessage *ac)
Inserts AC into ACManager of given type.

virtual Queue*
GetACs (Char *type, const int count = 1, Queue *notIn = NULL)
Gets given count of ACs of given type not included in notIn queue.

virtual GMessage*
GetACByACID (Char *type, ACID acID)
Gets AC of given type by ACID.

virtual Err
NextACTimeOut (Char *type, ESTime *t)
Returns next timeout of ACManager of given type.

virtual GMessage*
CreateNextACTimeOutMsg (Char *type)
Creates messages that will be returned in next TimeOut.

virtual Err
TestACCount (Char *type, Int *missing)
Tests AC count in ACMgr of given type.

virtual Err
GenerateRequestForMissingACs (Char *acType)
Generates request for missing ACs.

virtual Err
AskForOnion () *Asks Mix for onion.*

175.4

Err **AskForOnion** ()

Asks for onion, if necessary

Asks for onion, if necessary.

Test count of ACs of THIS_SIX. If there are some missing, generates request for onion and puts it into outMsgQueue.

Return Value:	error code
Parameters:	`type` — type of the AC
	`ac` — ac to save
See Also:	ACManager (→23, *page 125*)
Author:	Marek

175.5

Err **BeginTransaction** (ID *newTID)

Starts the transaction

Starts the transaction.

Calls the transactionManager method and thus begins new transaction.

Return Value:	newTID returned by the method
Parameters:	`parametr` —
See Also:	TransactionManager (→103, *page 308*)
Author:	error code

175.6

include . H sixMajordomo.h Err **CommandDataReceived** (MsgField *paramFld)

Handles the Data_received command

Handles the Data_received command.

Handling the Data_received command including decryption of the field Body contained in field Data. <pre> Handles this message: Data: ACID: Body: (encrypted by ACPubKey identified by ACID) Version: (eso-others protocol version) Command: Data: </pre>

Return Value: error code
Parameters: `paramFld` — parameters field from the inMsg
See Also:
Author: Pavel

175.7

Err **CommandOnionGenerated** (MsgField *dataFld)

Serves "onion_generated" command from Mix

Serves "onion_generated" command from Mix.

Gets the onion and creates the AC of this Six.

Return Value: error code
Parameters: `dataFld` — contents of dataMsg
See Also: GMessage, MsgField
Author: Marek

175.8

Err **CommandTest** (MsgField *dataFld)

Serves command Test from Mix

Serves command Test from Mix.

Handles command Test got from Mix.

Return Value: error code
Parameters: `dataField` — contents of data
See Also: GMessage, MsgField
Author: Marek

175.9

GMessage* **CreateGenerateOnionMsg** (Size hopCount = CFG_HOPS_IN_AC)

Creates "generate_onion" message for Mix

Creates "generate_onion" message for Mix.

Creates and fills the fields of "generate_onion" message for Mix. <pre> End situation (what is returned): [mixMsg]Command: generate_onion Parameters: [msg] Certificate_Type: Count: hops Request_ID: </pre>

Return Value: created GMessage
Parameters: `hopCount` — count of hops in AC wanted
See Also:
Author: Marek

175.10

GMessage* **CreateGetACsMsg** (Char *type, Int count)

Creates "get_certificates" message for Mix

Creates "get_certificates" message for Mix.

Creates and fills the fields of "get_certificates" message for Mix. <pre> [mixMsg]Command: get_certificates Parameters: [msg] Certificate_Type: Count: hops Request_ID: </pre>

Return Value: created GMessage
Parameters: `type` — type of ACs wanted
`count` — count of ACs wanted
See Also:
Author: Marek

175.11

GMessage* **CreateNextACTimeOutMsg** (Char *type)

Cretates timeout message

Cretates timeout message.

Calls method of ACManager of given type to save ACs of given type.

Creates timeout message understood by TO_CHECK_AC_COUNT command serve.

Return Value: acTimeOutMsg
Parameters: `type` — type of the AC
See Also: ACManager (→23, *page 125*)
Author: Marek

175.12

GMessage* **CreatePublishCertificateMsg** (GMessage *ac)

Creates "publish_certificates" message for Mix

Creates "publish_certificates" message for Mix.

Creates and fills the fields of "publish_certificates" message for Mix. <pre> End situation (what is returned): [mixMsg]Command: publish_certificates Parameters: [msg] Certificate-1: </pre>

Return Value: created GMessage
Parameters: `ac` — ac to publish
See Also:
Author: Marek

175.13

GMessage* **CreateSendDataMsg** (Char *commandSer, GMessage *dataSer, GMessage *ac, Size hops)

Creates "send_data" message for Mix

Creates "send_data" message for Mix.

Creates and fills the fields of "send_data" message for Mix. <pre> end situation (what is returned): [outMsg]Command: send_data Parameters: [msg3] AC: ac Add-Hops: hops Data: [msg2] ACID: Body: CIPHERED WITH PUB_KEY OF ac [msg1] Command: commandSer Data: dataSer Version: eso_others protocol version </pre>

Return Value: created GMessage
Parameters: `commandSer` — command to send to Service
`dataSer` — data to send to Service
`ac` — ac of recipient
`hops` — number of hops to add
See Also: Cipherer, GMessage, MsgField
Author: Marek

175.14

include . .. .. Common GMessage H arrayGMessage.h GMessage* **CreateTestMsg** (MsgField *dataFld)

Creates "test" message for Mix

Creates "test" message for Mix.

Creates and fills the fields of Test message for Mix. <pre> End situation (what is returned): [mixMsg]Command: test Parameters: dataFld </pre>

Return Value: created GMessage
Parameters: `dataFld` — dataField to insert to message
See Also:
Author: Marek

175.15

Err **DRCTest** (const MsgField *dataFld)

Handles Test message from service

Handles Test message from service.

Replies to the Test message got from service.

Return Value: error code
Parameters: `dataFld` — data got from service
Author: Pavel

175.16

Err **DecryptDataMsg** (const GMessage *dataMsg, MsgField **decryptedBodyFld, ACID *acid)

Decrypts the field Body

Decrypts the field Body.

Decrypts the field Body (stored in given GMessage) according to field ACID. Returns the decrypted field Body.

Return Value: error code
Parameters: `dataMsg` — given GMessage
`decryptedBodyField` — returned decrypted field
`acid` — acid to decrypt with
See Also: GMessage
Author: Pavel

175.17

Err **EndTransaction** (ID tid)

Finishes transaction

Finishes transaction.

Calls the transaction manager to finish the transaction.

Return Value: error code
Parameters: `tid` — tid of transaction to finish
See Also: TransactionManager (→103, *page 308*)
Author: Pavel

175.18

Err **FeedSenderWithMsgForMix** (GMessage *mixMsg)

Puts one message to the majordomoToSender queue

Puts one message to the majordomoToSender queue.

Appends parameter into the majordomoToSenderQueue.

Return Value: error code
Parameters: `mixMsg` — message to append
See Also: GMessage, Sender, MessageQueue
Author: Marek

175.19

Err **FeedSenderWithOutMsgQueue** ()

Puts the messages from outMsgQueue to the majordomoToSender queue

Puts the messages from outMsgQueue to the majordomoToSender queue.

If outMsgQueue is not null and not empty, takes all messages from it and puts into majordomoToSender queue.

Return Value: error code
See Also: Queue
Author: Marek

175.20

Err **FillReplyToInfo** (GMessage *gMsg)

Sets replyToInfo of GMessage

Sets replyToInfo of GMessage.

Sets two fields of the gMsg (replyToTID, replyToAC).

Return Value: error code
Parameters: `gMsg` — message to fill
See Also: GMessage
Author: Marek

175.21

include . H sixMajordomo.h Err **FillTest** (GMessage *gMsg, MsgField *msgFld)

Sets GMessage as Test

Sets GMessage as Test.

Sets the Parameters-field of GMessage to given msgField.

Return Value: error code
Parameters: `gMsg` — message to fill
`msgFld` — field tu put in gMsg
See Also:
Author: Marek

175.22

Err **GenerateRequestForMissingACs** (Char *acType)

Generates request for ACs of given type

Generates request for ACs of given type.

Test count of ACs of given type. If there are some missing, generates request for missing them and puts it into outMsgQueue.

Return Value: error code
Parameters: `type` — type of the AC
See Also: ACManager (→23, *page 125*)
Author: Marek

175.23

GMessage* **GetACByACID** (Char *type, ACID acID)

Get AC of given type and ACID

Get AC of given type and ACID.

Calls method of ACManager of given type to get AC of given ACID.

Return Value: found ac
Parameters: `type` — type of the AC
`acid` — acid to find
See Also: ACManager (→23, *page 125*)
Author: Marek

175.24

Queue* **GetACs** (Char *type, const int count, Queue *notIn)

Returns ACs of given type

Returns ACs of given type.

Calls method of ACManager of given type to return count ACs of given type not included in notIn queue.

Return Value: queue of returned ACs
Parameters: `type` — type of the AC
`count` — count of ACs wanted
`notIn` — queue of ACs not wanted to include
See Also: ACManager (→23, *page 125*)
Author: Marek

175.25

ID **GetTID** ()

Returns current TID

Returns current TID.

Returns the value of the attribute tID.

Return Value: ID currentTID.
Author: Pavel

175.26

Err **HandleDataReceivedBody** (const MsgField *bodyFld, const ACID acid)

Handles the command receiver from Service

Handles the command receiver from Service.

Handles the command received from the receiver (not from a Mix) according to Eso-Others protocol.

Return Value: err
Parameters: `bodyFld` — body of the message
`acid` — acid to which the message was sent
See Also:
Author: pavel

175.27

Err **HandleGeneratedAC** (GMessage *)

Handles generated AC

Handles generated AC.

In Six does nothing. Descendants may save or publish the AC by overriding this method.

Return Value:	co
Parameters:	`generatedAC` —
See Also:	co_se_toho_tyka
Author:	jmeno

175.28

include . H sixMajordomo.h Void **HandleMixMessage** (GMessage *inMsg)

Handles the message from Mix

Handles the message from Mix.

Extracts the command and data fields from the inMsg and calls switch methods.

Return Value:	void
Parameters:	`inMsg` — message that came from receiverToMajordomo queue
See Also:	GMessage, Debugable
Author:	Pavel, Marek

175.29

Err **InsertAC** (Char *type, GMessage *ac)

Saves ACs of given type

Saves ACs of given type.

Calls method of ACManager of given type to save ACs.

Return Value: error code
Parameters: `type` — type of the AC
`ac` — ac to save
See Also: ACManager (→23, *page 125*)
Author: Marek

175.30

Err **NextACTimeOut** (Char *type, ESTime *t)

Returns next timeout of ACManager of given type

Returns next timeout of ACManager of given type.

Calls method of ACManager of given type to return next timeout.

Return Value: error code
Parameters: `type` — type of the AC
`t` — returned time
See Also: ACManager (→23, *page 125*)
Author: Marek

175.31

void* **Run** (void *)

Main running loop of SixMajordomo

Main running loop of SixMajordomo.

In a loop gets a message from receiverToMajordomo queue, creates outMsgQueue, processes the message and puts all the messages (if there are any) from outMsgQueue into the majordomoToSender queue.

Return Value:	void
See Also:	MessageQueue (→71, *page 244*)
Author:	Pavel

175.32

include H sixMajordomo.h **SixMajordomo** (Six * parent, LogFile * majordomoLogFile)

Constructor of SixMajordomo

Constructor of SixMajordomo.

Creates transaction manager and calls its ancestors' constructors.

Parameters:	`parent` — pointer to Six `majordomoLogFile` — logFile to write to
See Also:	TransactionManager, Runable, SixOffspring
Author:	Pavel

175.33

include . H sixMajordomo.h Err **SwitchDataReceivedCommand** (const Char *command, const MsgField *dataFld, const ACID acid)

Incoming message handling

Incoming message handling.

Chooses a method to handle the command from the incomming message.

Return Value: error code
Parameters: `command` — command received from service
`dataFld` — data received from service
`acid` — ID of AC to which the data was sent
Author: Pavel

175.34

Err **SwitchDataReceivedVersion** (const Char *version, const GMessage *body, const ACID acid)

Switches a method according to the version

Switches a method according to the version.

Chooses a method to handle the incomming message according to its version.

Return Value: error code
Parameters: `version` — version of received GMessage
`body` — body of the received GMessage
`acid` — ID of certificate to which the message came
See Also:

Author: Pavel

175.35

Err **SwitchMixCommand** (MsgField *commandFld, MsgField *dataFld)

Chooses handling method

Chooses handling method.

According to field Command (in message poped from receiverToMajordomo) chooses and calls handling method.

Return Value: error code
Parameters: `commandFld` — command extracted from the message
`dataFld` — data extracted from the message
Author: Pavel

175.36

Err **TestACCount** (Char *type, Int *missing)

Test count of ACs of given type

Test count of ACs of given type.

Calls method of ACManager of given type to test count of ACs.

Return Value: error code
Parameters: `type` — type of the AC
`missing` — returned count of missing ACs
See Also: ACManager (→23, *page 125*)
Author: Marek

175.37

˜SixMajordomo ()

Destructor of SixMajordomo

Destructor of SixMajordomo.

Deletes transaction manager.

See Also: TransactionManager (→103, *page 308*)
Author: Pavel

175.1

attributes

Names

TID	**tID**	*transaction ID.*
Queue*	**outMsgQueue**	*Queue with messages prepared to send to Mix.*
TransactionManager*	**transactionManager**	*Object for managing transactions.*

175.2

methods

Names

175.2.1

Help functions for putting the messages to the majordomoToSender queue

Names

Err **FeedSenderWithOutMsgQueue** ()
Puts the messages from outMsgQueue to majordomoToSender queue.

Err **FeedSenderWithMsgForMix** (GMessage *mixMsg)
Puts one message to majordomoToSender queue.

175.2.2

Functions for commands got from IIQ (Mix cmds)

Names

virtual Void
HandleMixMessage (GMessage *inMsg)
gets the Command and Data fields from inMsg and calls SwitchMix-Command.

virtual Err
SwitchMixCommand (MsgField *commandFld, MsgField *dataFld)
calls appropriate method for Mix command.

virtual Err
CommandTest (MsgField *dataFld)
Serves Test command from MIX.

virtual Err
CommandDataReceived (MsgField *dataFld)
Serves DataReceived command from MIX.

virtual Err
CommandOnionGenerated (MsgField *dataFld)
Serves OnionGenerated command from MIX.

175.2.3

Functions needed for serving OnionGenerated command from Mix

Names

virtual Err
GenerateThisSixAC (MsgField *onion, GMessage **generatedAC)
Generates AC of this six having the Onion (Mix certificate).

virtual Err
HandleGeneratedAC (GMessage *generatedAC)
Handles the generated AC according to the type of Six.

175.2.4

Functions needed for serving DataReceived command from Mix

Names

virtual MsgField*
DecryptByACPriv (const ACID acid, const MsgField *dData)
Decrypting data by privateKey associated with ACID.

Err **DecryptDataMsg** (const GMessage *dataMsg, MsgField **decryptedBodyFld, ACID *acid)
Decrypting data that came from Service.

Err **HandleDataReceivedBody** (const MsgField *bodyFld, const ACID acid)
Handles the body of the message (the decrypted request from Service).

virtual Err
SwitchDataReceivedVersion (const Char *version, const GMessage *body, const ACID acid)
Chooses the way to continue according to the protocol version of the message.

175.2.5

Functions that handle various commands from Service

Names

virtual Err **SwitchDataReceivedCommand** (const Char *command, const MsgField *dataFld, const ACID acid)
Initializing switch.

Err **DRCTest** (const MsgField *dataFld)
Serving Test command.

175.2.6

Filling the fields into the messages to Service

Names

Err **FillTest** (GMessage *gMsg, MsgField *msgFld)
Filling fields of TestMessage.

Err **FillReplyToInfo** (GMessage *gMsg)
Filling Reply-to-AC and Reply-To-TID fields of a GMessage.

175.2.7

Functions that create messages for Mix

Names

GMessage* **CreateTestMsg** (MsgField *dataFld)
Creates testing message.

GMessage*

CreateGetACsMsg (Char *type, Int count)
Creates "get_certificates" message.

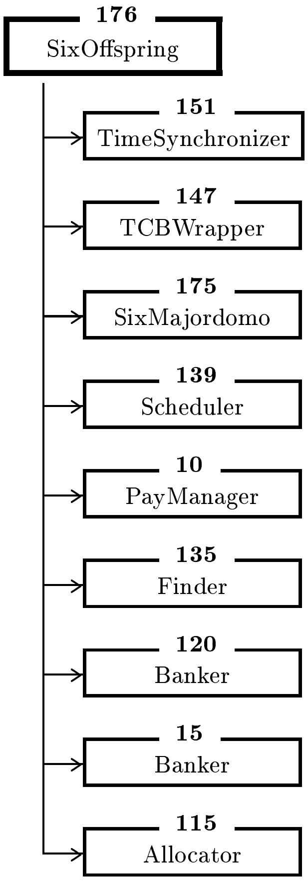
GMessage*
CreateGenerateOnionMsg (Size hopCount = CFG_HOPS_IN_AC)
Creates "generate_onion" message.

176

class **SixOffspring**

Maintains a pointer to the object that created this one

Inheritance



Public Members

Protected Members

Maintains a pointer to the object that created this one.

Maintains a pointer to it's parent so that descendands of this class can access some members of Six. It is useful for objects that communicate heavily with other objects inside of Six.

See Also: Six
Author: Pavel

176.2 methods

Names

SixOffspring (Six *six)
Constructor.

176.3 SixOffspring (Six * aSix)

Constructor of SixOffSpring

Constructor of SixOffSpring.

Remembers pointer to its father.

Parameters: `six` — parent
See Also: Six (→174, *page 514*)
Author: Pavel

176.1 attributes

Names

Six*	**six**	*Pointer to Six.*

Class Graph