Guide

[A. The relation between test module(simple\_client) and SDKlib. 2](#_Toc11866)

[B. API unit test 3](#_Toc1684)

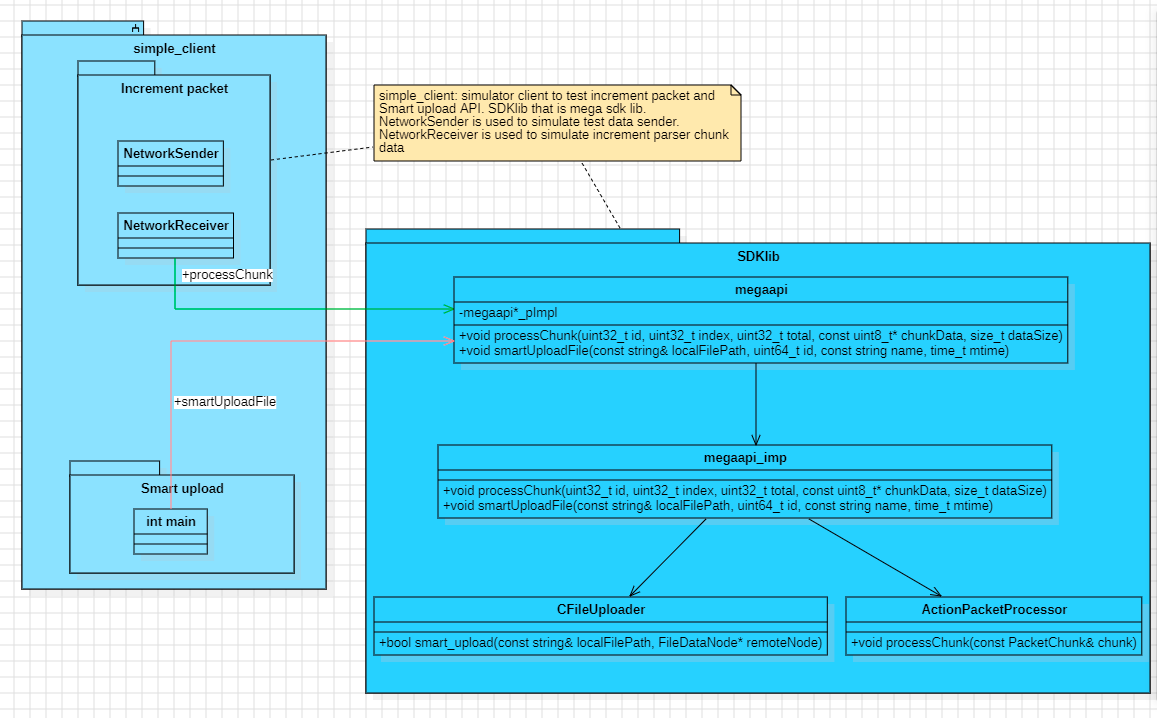
[1. generate data(increment packet data) 3](#_Toc18712)

[2. Invoke API in main(): 4](#_Toc32615)

[3. Test result 6](#_Toc1172)

[4. Special Note 6](#_Toc22446)

## The relation between test module(simple\_client) and SDKlib.



1. Simple\_client: simulator client to test increment packet and Smart upload API.
2. NetworkSender is used to simulate test data sender.
3. NetworkReceiver is used to simulate increment parser chunk data
4. SDKlib that is mega sdk lib. API:
5. Increment packet processing API

// precess the received chunk

void processChunk(uint32\_t id, uint32\_t index, uint32\_t total, const uint8\_t\* chunkData,size\_t dataSize);

// set a callback for when a packet is complete

void setPacketCompleteCallback(

std::function<void(uint32\_t, const std::vector<uint8\_t>&)> callback);

// set a callback for packet progress updates

void setPacketProgressCallback(std::function<void(uint32\_t, double)> callback);

// clear all pending packets

void clearPendingPackets();

// get the state of a specific packet

size\_t getPendingPacketCount() const;

1. smart upload file API

void smartUploadFile(const std::string& localFilePath,

uint64\_t id,

const std::string name,

time\_t mtime,

const std::string fingerprint,

const std::string parenthandle,

const std::string encryption\_key,

const std::string nonce,

const std::string mac);

## API unit test

### generate data(increment packet data)

void simulatePacketData::simulateDataToUnitTest(MegaApi\* megaApi)

{

std::cout << "=== ActionPacket Incremental Processing Demo ===" << std::endl;

// create sender and receiver instances

NetworkSender sender(1024, 0.1, megaApi); // chunk size of 1024 bytes, 10% packet loss

NetworkReceiver receiver(megaApi);

// start the receiver to process incoming packets

receiver.start();

// create a vector to hold test packets

std::vector<std::vector<uint8\_t>> testPackets;

// test data packets1: small text data

std::string textData = "Hello, this is a test packet for incremental processing demonstration!";

testPackets.push\_back(std::vector<uint8\_t>(textData.begin(), textData.end()));

// test packets2: large binary data

std::vector<uint8\_t> binaryData(8192);

for (size\_t i = 0; i < binaryData.size(); ++i)

{

binaryData[i] = static\_cast<uint8\_t>(i % 256);

}

testPackets.push\_back(binaryData);

// test packets3: random data

std::vector<uint8\_t> randomData(4096);

std::random\_device rd;

std::mt19937 gen(rd());

std::uniform\_int\_distribution<> dis(0, 255);

for (size\_t i = 0; i < randomData.size(); ++i)

{

randomData[i] = static\_cast<uint8\_t>(dis(gen));

}

testPackets.push\_back(randomData);

// send each packet using the sender

for (const auto& packet: testPackets)

{

sender.sendPacket(packet,

[&](const PacketTestChunk& chunk)

{

receiver.receiveChunk(chunk);

});

// data loss simulation

std::this\_thread::sleep\_for(std::chrono::milliseconds(200));

}

// wait for all packets to be processed

std::this\_thread::sleep\_for(std::chrono::seconds(2));

// stop the receiver

receiver.stop();

std::cout << "=== Demo completed ===" << std::endl;

}

### Invoke API in main():

int main(){

//Test Increment action packet interface

std::cout << "Test Increment action packet interface begin:" << std::endl;

simulatePacketData simPacketData;

simPacketData.simulateDataToUnitTest(megaApi);

std::cout << "Test Increment action packet interface end!" << std::endl;

std::cout << std::endl;

std::cout << std::endl;

std::cout << "Test smartuploadFile interface begin:" << std::endl;

// Test smartuploadFile interface

std::string localFilePath = "/path/to/local/file";

uint64\_t id = 10000;

std::string name = "localfilename";

time\_t mtime = 0;

std::string fingerprint = "fingerprint";

std::string parenthandle = "parenthandle";

std::string encryption\_key = "encryption\_key";

std::string nonce = "nonce tag";

std::string mac = "mac adr";

megaApi->smartUploadFile(localFilePath,

id,

name,

mtime,

fingerprint,

parenthandle,

encryption\_key,

nonce,

mac);

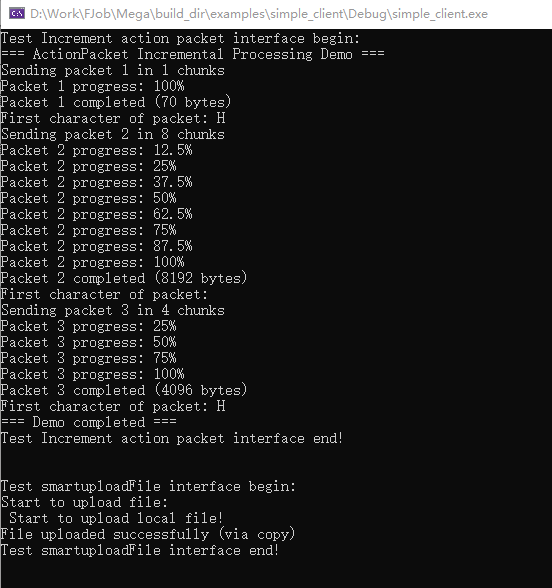
std::cout << "Test smartuploadFile interface end!" << std::endl;

std::cout << std::endl;

std::cout << std::endl;

}

### Test result



### Special Note

In the remote copy file function, this remote copy action should be copied from server side. From client that just need to check the validate of localfile and create new file id, mac etc. Then send message(new file id, mac etc.) to server side to ask server copy file by-self. So the function apicopyremotefile is an api that is provide by server(assumption in yellow background color).

// Copy remote files

bool CFileUploader::copy\_remote\_file(const std::string& localFilePath, FileDataNode\* remoteNode)

{

if (!client)

{

std::cerr << "MegaClient is not initialized." << std::endl;

return false;

}

if (!remoteNode)

{

std::cerr << "Invalid remote node." << std::endl;

return false;

}

std::cout << "Starting remote file copy for node: " << remoteNode->get\_nonce() << std::endl;

// get remote node's encryption parameters

std::string sourceKey = remoteNode->get\_encryption\_key();

std::string sourceNonce = remoteNode->get\_nonce();

std::string sourceMac = remoteNode->get\_mac();

if (sourceKey.empty() || sourceNonce.empty() || sourceMac.empty())

{

std::cerr << "Missing encryption parameters for source node." << std::endl;

return false;

}

// create new encryption parameters for the target node

std::string targetKey = client->rng.genstring(SymmCipher::KEYLENGTH); // CryptoUtils::generate\_new\_key();

std::string targetNonce = client->rng.genstring(SymmCipher::KEYLENGTH);

// real implementation should generate a new key and nonce

std::string parentHandle = remoteNode->get\_parenthandle(); // auuxiliary function to get the parent folder handle

std::string newName = remoteNode->get\_name() + "\_copy"; // copy file name

// validate the source MAC

bool macValid = CryptoUtils::verify\_mac(remoteNode->get\_file\_size(),

localFilePath,

sourceKey,

sourceNonce,

sourceMac);

if (!macValid)

{

std::cerr << "Source file MAC verification failed. Possible data corruption." << std::endl;

return false;

}

// excute the remote copy operation

// apicopyremotefile: should be provided by client and will ask copy file on server by theget

// key and nonce

std::string newNodeHandle;

bool success = true;

/\* bool success = client->apicopyremotefile->copy(remoteNode->get\_handle(),

parentHandle,

newName,

targetKey,

targetNonce,

&newNodeHandle);\*/

if (!success)

{

std::cerr << "Failed to copy remote file." << std::endl;

return false;

}