

Формирование запроса PKCS#10

```
#include "rutoken/pki-core-cpp.h"
#include <iostream>

using namespace rutoken::pkicore;

class PkiCore
{
public:
    explicit PkiCore(const std::string& path)
    {
        initialize(path);
    }
    ~PkiCore() throw()
    {
        try {
            deinitialize();
        }
        catch (...) {
            //EXPECT_TRUE(false);
        }
    }
};

int main()
{
    try {
        PkiCore p("./");

        auto deviceList = Pkcs11Device::enumerate();

        for (auto& d : deviceList)
            std::cout << d.getLabel() << std::endl;

        deviceList.front().login("12345678");

        auto keyA = deviceList.front().generateKeyPair(Pkcs11Device::Gost34102001KeyGenParams
(Gost34102001Paramset::a));

        X500Dn x500Dn;
        // utf8: http://en.cppreference.com/w/cpp/language/string\_literal
        x500Dn.setRdn(X500Dn::RdnId::commonName, u8" ");
        x500Dn.setRdn(X500Dn::RdnId::surname, "Ivanov");
        x500Dn.setRdn(X500Dn::RdnId::givenName, "Ivan");
        x500Dn.setRdn(X500Dn::RdnId::title, "king");
        x500Dn.setRdn(X500Dn::RdnId::pseudonym, "Ivan");
        x500Dn.setRdn(X500Dn::RdnId::emailAddress, "example@example.com");
        x500Dn.setRdn(X500Dn::RdnId::countryName, "RU");
        x500Dn.setRdn(X500Dn::RdnId::localityName, "Moscow");
        x500Dn.setRdn(X500Dn::RdnId::stateOrProvinceName, "Moscow");
        x500Dn.setRdn(X500Dn::RdnId::organization, "Aktiv");
        x500Dn.setRdn(X500Dn::RdnId::organizationalUnit, "IT");
        x500Dn.setRdn(X500Dn::RdnId::street, "STREET");
        x500Dn.setRdn(X500Dn::RdnId::ogrn, "12345678987")
            .setRdn(X500Dn::RdnId::ogrnip, "12345678987")
            .setRdn(X500Dn::RdnId::snils, "12345678987")
            .setRdn(X500Dn::RdnId::inn, "12345678987");

        unsigned char customExtensionASN1Data[] = {
            0x0C, 0x33, 0x30, 0x30, 0x43, 0x41, 0x30, 0x31,
            0x37, 0x38, 0x69, 0xD0, 0x90, 0xD0, 0xA0, 0xD0,
            0x9C, 0x20, 0xD0, 0x98, 0xD0, 0xBD, 0xD0, 0xB8,
            0xD1, 0x86, 0xD0, 0xB8, 0xD0, 0xB0, 0xD0, 0xBB,
            0xD0, 0xB8, 0xD0, 0xB7, 0xD0, 0xB0, 0xD1, 0x86,
            0xD0, 0xB8, 0xD0, 0xB8, 0x20, 0xD0, 0xA2, 0xD0,
            0x95, 0xD0, 0xA1, 0xD0, 0xA2,
        };
    }
};
```

```

Pkcs10RequestInfo requestInfo;
requestInfo.setSubject(x500Dn
    .addKeyUsage(Pkcs10RequestInfo::X509KeyUsage::digitalSignature)
    .addKeyUsage(Pkcs10RequestInfo::X509KeyUsage::nonRepudiation)
    .addExtendedKeyUsage(Pkcs10RequestInfo::X509ExtendedKeyUsage::
emailProtection)
    .addExtendedKeyUsageByOid("1.2.643.2.2.34.6") // CryptoPro RA
user
    .addCustomExtension("1.2.643.3.123.3.1",
customExtensionASN1Data, sizeof(customExtensionASN1Data), false)
    .addSubjectSignTool());

Pkcs10Request request = createPkcs10Request(keyA, requestInfo);
auto data = request.toPem();

std::cout << data << std::endl;

return 0;
} catch (const std::exception& e) {
    std::cout << e.what() << std::endl;
}
}

```