**Секреты разработки CSP**

Юрий Зырянов

**Введение**

Эта статья для тех, кто по тем или иным причинам решил написать собственный крипто-провайдер для OC семейства Windows. Если Вы хотите реализовать в вашем провайдере нестандартные алгоритмы, то вам предстоит столкнуться с определенными трудностями. Трудности могут возникнуть, например, при попытках использования вашего крипто-провайдера для проверки сертификатов в MS Internet Explorer.

Под нестандартными алгоритмами здесь понимаются не всемирные DES, RSA, DSA и т.д, а, например, алгоритмы семейства ГОСТ. Дело в том, что для RSA и подобных алгоритмов все необходимые идентификаторы уже зашиты в систему, а для ГОСТ-ов (или многих других алгоритмов) надо отдельно позаботиться о том, чтобы система их “увидела”.

Для примеров кода используется Си. Все примеры кода служат только для иллюстрации принципов изложенных в статье и не являются полноценными рабочими программами.

Также подразумевается, что у читателя есть базовые знания в области прикладной криптографии и термины “открытый ключ”, “ASN.1” и подобные для него не являются загадкой.

**Интеграция провайдера в Windows**

Помимо наличия библиотеки самого провайдера дополнительно требуется:

зарегистрировать провайдер и крипто-алгоритмы в системе, прописав определенные параметры в реестре;

создать библиотеку с функциями конвертирования ключей из форматов криптопровайдера во внешние форматы и зарегистрировать эти функции в реестре;

заменить функцию I\_CryptGetDefaultCryptProvider в библиотеке crypt32.dll

Только после выполнения этих действий провайдер нормально интегрируется в систему и вы сможете, например, генерировать сертификаты при помощи вашего провайдера на основе стандартного компонента ОС Windows Server - Сertification services или на тестовом УЦ КриптоПро http://www.cryptopro.ru/certsrv

Корректно будет отображаться статус проверки подписи у сертификатов, подписанных вашим “нестандартным” алгоритмом и т.п.

Далее подробно рассмотрим каждый из упомянутых выше шагов, предполагая при этом что библиотека с Вашим CSP уже имеется и корректно работают все функции провайдера.

**Регистрация крипто-провайдера и алгоритмов в системе**

Когда у вас уже есть готовая библиотека с реализацией функций CSP, необходимо зарегистрировать ее в системе, для того чтобы новый крипто-провайдер стал доступен различным приложениям.

Процесс регистрации самого CSP подробно описан в MDSN, и повторять эту информацию здесь смысла нет. Все это подробно описано в [1]. Также здесь мы не будем останавливаться на проблеме подписи нового CSP в Microsoft и путях ее обхода. Эта проблема уже многократно обсуждалась на различных форумах, например смотрите [3]. Гораздо интереснее рассмотреть регистрацию криптографических алгоритмов. Каждый алгоритм имеет свой уникальный ASN.1 идентификтор Оbject Identifier - OID. Например, алгоритм подписи ГОСТ-34.10-2001 имеет такой OID (представленный в виде строки) – “1.2.643.2.2.3”. Идентификатор каждого поддерживаемого вашим CSP алгоритма следует занести в реестр. Помимо OID у каждого крипто-алгоритма в Windows существует еще идентификатор в виде четырех-байтового числа – AlgID, по которому алгоритмы идентифицируются в провайдере. Этот идентификатор заноситься в CSP и его можно узнать перечислив алгоритмы посредством вызова CPGetProvParam. В КриптоПро, например, для алгоритма хеширования ГОСТ-34.11-94 AlgID используется значение 0x801e

Пусть нам необходимо зарегистрировать алгоритм подписи ГОСТ-34.10-2001. Тогда в реестре необходимо прописать следующие идентификаторы:

“1.2.643.2.2. 9!1” – Хэш ГОСТ-34.11-94

“1.2.643.2.2.19!3” – Ключ подписи ГОСТ-34.10-2001

“1.2.643.2.2.3!4” – Подпись ГОСТ-34.10-2001 – Алгоритм Хэша + Алгоритм Ключа

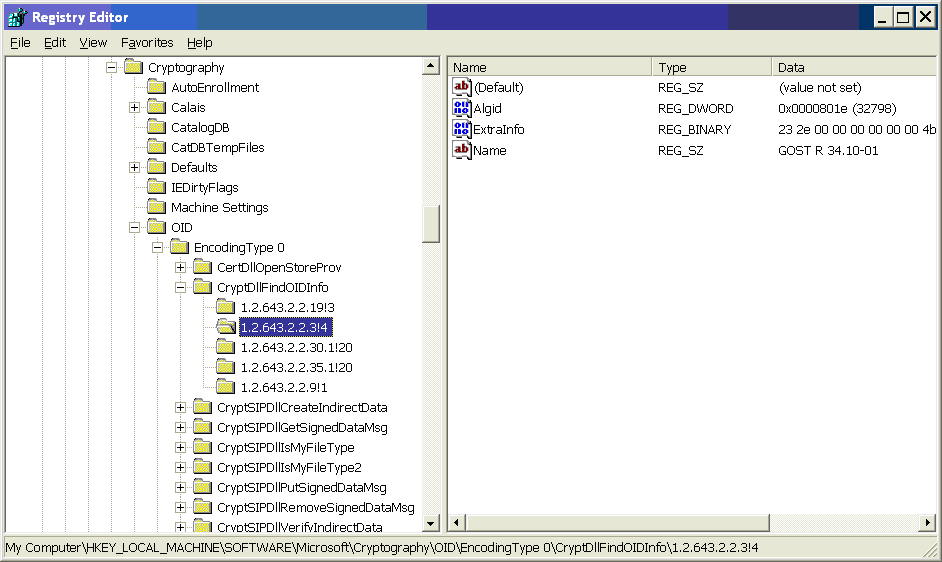


Рисунок 1 - Идентификаторы алгоритмов

Далее приведен пример кода регистрации OID алгоритма ГОСТ-34.11-94

// Регистрация GOST-3411-94 HASH OID

//

CRYPT\_OID\_INFO oidInfo;

int rc = 0;

memset(&oidInfo, 0, sizeof(CRYPT\_OID\_INFO));

oidInfo.cbSize = sizeof(CRYPT\_OID\_INFO);

oidInfo.pszOID="1.2.643.2.2.9";

oidInfo.pwszName= L"GOST-3411-94 HASH";

oidInfo.dwGroupId = CRYPT\_HASH\_ALG\_OID\_GROUP\_ID;

oidInfo.Algid = 0x801e;

oidInfo.ExtraInfo.cbData=0;

oidInfo.ExtraInfo.pbData=0;

rc = CryptRegisterOIDInfo(

&oidInfo,

0

);

if(rc)

printf("nHash algorithm registered”);

else

printf("nError registering hash algorithm”);

Аналогично регистрируются и остальные алгоритмы. Подробную информацию о структуре CRYPT\_OID\_INFO можно найти в MSDN: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/seccrypto/security/crypt\_oid\_info.asp

Для того чтобы провайдер вызывался для проверки сертификата, подписанного нашим “нестандартным” алгоритмом необходимо еще одно дополнительное действие.

Дело в том, что Windows определяет какой провайдер использовать для проверки по полю ExtraInfo (см. ссылку в предыдущем абзаце для описания этого поля) в ключе реестра соответвующему алгоритму подписи – такой ключ мы создаем, вызывая функцию CryptRegisterOIDInfo. Поэтому надо указать системе наш провайдер в качестве провайдера по умолчанию для типа, который занесен в ExtraInfo алгоритма подписи.

Следующий код устанавливает провайдер по умолчанию для определенного типа.

#define YOUR\_PROV\_NAME "MY\_PROV"

#define YOUR\_PROV\_TYPE 75

rc = CryptSetProvider(

YOUR\_PROV\_NAME,

YOUR\_PROV\_TYPE

);

Сценарии применения нашего CSP

Рассмотрим здесь два сценария применения CSP.

Первый сценарий – проверка подписи сертификата. Для проверки подписи система загружает открытый ключ из сертификата, которым подписан проверяемый. Затем по OID алгоритма подписи проверяемого сертификата, так как описано в предыдущем разделе статьи, определяется требуемый провайдер. Чтобы проверить подпись, нужно импортировать открытый ключ в CSP и можно было бы подумать что Windows сразу вызывает функцию нашего провайдера CPImportKey. Но не тут-то было!

Второй сценарий – генерация ключевой пары и отправка запроса на сертификат на Удостоверяющий Центр. Windows загружает наш CSP, генерирует ключевую пару и экспортирует к себе наверх открытый ключ вызывая функцию CPExportKey. Все хорошо. Вроде бы надо взять и поместить полученный буфер с ключом в PKCS#10 запрос, который затем будет отправлен на УЦ. И тут все опять не совсем так.

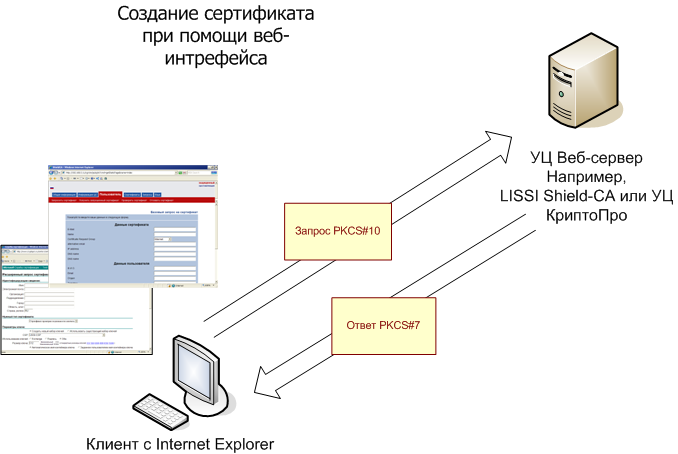


Рисунок 2 - Запрос на сертификат в УЦ

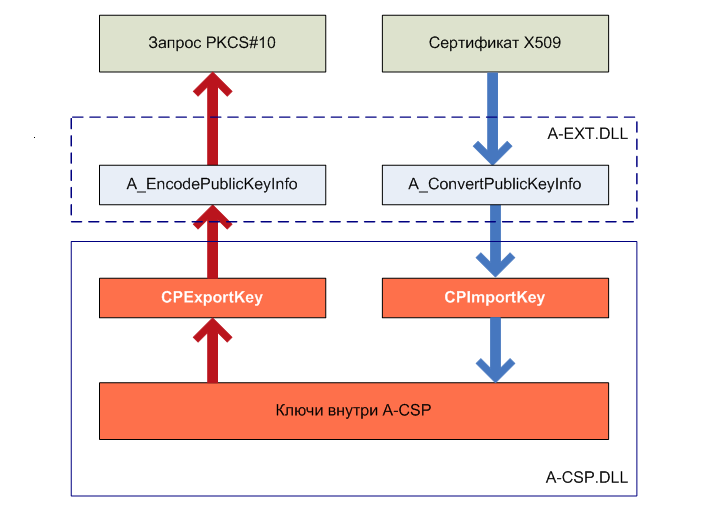


Рисунок 3 - Процесс создания сертификата

Оказывается, существуют промежуточные функции для экспорта/импорта открытых ключей, и без их реализации ничего хорошего с нашим CSP, в упомянутых выше двух сценариях, не получиться. Беда еще и в том, что функции эти недокументированные и найти информацию по ним крайне сложно. Их описанию посвящен следующий раздел.

**Функции конвертирования ключей**

Архитектура круговорота открытых ключей для “нестандартных” алгоритмов в Windows представлена на картинке:

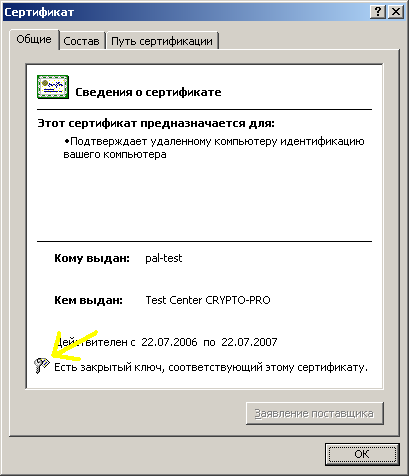


Рисунок 4 - Импорт и экспорт открытых ключей в/из CSP

Функция A\_ConvertPublicKeyInfo – на входе принимает ключ в формате ASN.1 DER и должна преобразовать его в формат, который “понимает” функция CSP CPImportKey

Функция A\_EncodePublicKeyInfoAndParameters на входе принимает ключ в том формате, в котором ключ экспортирует CPExportKey. На выходе A\_EncodePublicKeyInfoAndParameters должна сформировать ASN.1 DER структуру с ключом – ту же самую которая в случае импорта передается в A\_ConvertPublicKeyInfo – на вход. Надеюсь, вы не запутались во всех этих входах ϑи выходах

Вот сигнатуры и краткие описания параметров этих функций:

BOOL WINAPI A\_ConvertPublicKeyInfo(

DWORD dwCertEncodingType, // IN -

VOID \*EncodedKeyInfo, // IN – буфер с ключом - указатель

// на структуру CERT\_PUBLIC\_KEY\_INFO

DWORD dwAlg, // IN – AlgId ключа

DWORD dwFlags, // IN – обычно 0

BYTE \*\*ppStructInfo, // OUT – двойной указатель на структуру

// в заголовке структуры идет сначала PUBLICKEYSTRUC, затем DSSPUBKEY,

// а затем сам ключ с параметрами

DWORD \*StructLen // OUT – длинна структуры

);

BOOL WINAPI A\_EncodePublicKeyAndParameters(

DWORD dwCertEncodingType, // IN

LPCSTR lpszStructType, // IN – OID алгоритма

const void\* pvStructInfo, // IN – такая же структура как

// на выходе ConvertPublicKeyInfo

DWORD nStructLen, // IN – длинна входной структуры

DWORD dwFlags, // IN – обычно 0

DWORD Unk, // ? - неизвестно

BYTE \*\*pbPubKey, // OUT – открытый ключ в ASN.1 DER

DWORD\* pcPubKeyLen, // OUT – длинна открытого ключа

BYTE \*\*pbParams, // OUT – параметры открытого ключа

DWORD\* pcParamsLen // OUT – длинна параметров

);

Форматы ключей зависят от крипто-провайдера и используемых алгоритмов.

Функция I\_CryptGetDefaultCryptProvider из crypt32.dll

По непонятной для меня причине Windows часто не пытается искать нужный крипто- провайдер по идентификатору алгоритма с которым требуется работать. В таких случаях она просто вызывает недокументированную функцию I\_CryptGetDefaultCryptProvider, и если тот провайдер, который вернула эта функция, не умеет работать с данным алгоритмом, то процесс завершается с ошибкой. Так происходит, например, при разборе в Internet Explorer PKCS#7 ответа в сценарии с запросом сертификата на тестовом УЦ.

HCRYPTPROV WINAPI I\_CryptGetDefaultCryptProv(ALG\_ID algid);

Необходимо заменить эту функцию таким образом, чтобы при нулевом параметре algid на входе она возвращала наш провайдер, который уже в отличии от штатного провайдера легко справиться с “нестандартными алгоритмами”.

Обсуждение способов замены функций в системной dll выходит далеко за рамки данной статьи. Могу лишь, как один из способов решения, предложить библиотеку Microsoft Detours: http://research.microsoft.com/sn/detours

**Привязка закрытого ключа к сертификату**

В отличие от открытого ключа закрытые ключи никогда не покидают крипто-провайдер и поэтому когда вы видите что для данного сертификата есть закрытый ключ (как на рисунке) это значит что в контексте этого сертификата существует явная ссылка на закрытый ключ.

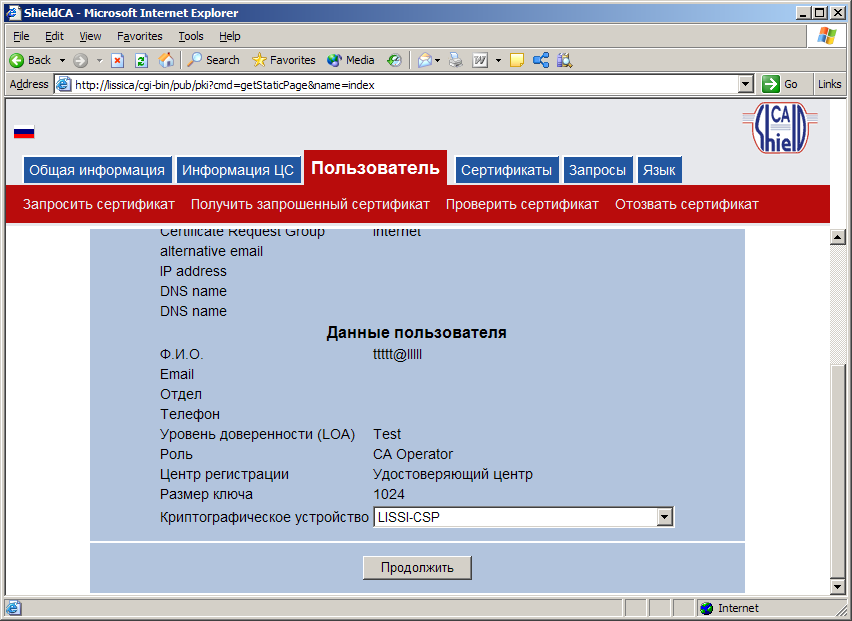


Рисунок 5 - Закрытый ключ сертификата

Контекст сертификата это набор дополнительных атрибутов сертификата, которые находятся не в теле сертификата, а хранятся отдельно от него. Одним из таких атрибутов и является ссылка на закрытый ключ, которая состоит из имени провайдера и имени ключевого контейнера.

Пример кода для привязки закрытого ключа к сертификату:

PCCERT\_CONTEXT pCert;

CRYPT\_KEY\_PROV\_INFO prov\_info;

…

prov\_info.cProvParam = 0;

prov\_info.rgProvParam = 0;

prov\_info.dwFlags = 0;

prov\_info.dwKeySpec = AT\_SIGNATURE;

prov\_info.dwProvType = 0;

prov\_info.pwszContainerName = L"key-kont-name";

prov\_info.pwszProvName = L"A-CSP";

CertSetCertificateContextProperty(

pCert,

CERT\_KEY\_PROV\_INFO\_PROP\_ID,

0,

&prov\_info

);

Успехов в разработке крипто-провайдера!

http://www.realcoding.net/

Формы в HTML документах

Некоторые WWW browser позволяют пользователю, заполнив специальную форму, возвращающую полученное значение, выполнять некоторые действия на вашем WWW - сервере. Когда форма интерпретируется WEB - броузером, создается специальные экранные элементы GUI, такие, как поля ввода, checkboxes, radiobuttons, выпадающие меню, скроллируемые списки, кнопки и т.д. Когда пользователь заполняет форму и нажимает кнопку "Подтверждение" (SUBMIT - специальный тип кнопки, который задается при описании документа), информация, введенная пользователем в форму, посылается HTTP-серверу для обработки и передаче другим программам, работающим под сервером, в соответствии с CGI (Common Gateway Interface) интерфейсом.

Когда вы описываете форму, каждый элемент ввода данных имеет тэг <INPUT>. Когда пользователь помещает данные в элемент формы, инфоромация размещается в разделе VALUE данного элемента.

Синтаксис

Все формы начинаются тэгом <FORM> и звершаются тэгом </FORM>.

<FORM METHOD="get|post" ACTION="URL"> Элементы\_формы\_и\_другие\_элементы\_HTML </FORM>

METHOD

Метод посылки сообщения с данными из формы. В зависимости от используемого метода вы можете посылать результаты ввода данных в форму двумя путями:

GET: Информация из формы добавляется в конец URL, который был указан в описании заголовка формы. Ваша CGI-программа (CGI-скрипт) получает данные из формы в виде параметра переменной среды QUERY\_STRING. Использование метода GET не рекомендуется.

POST: Данный метод передает всю информацию о форме немедленно после обращения к указанному URL. Ваша CGI-программа получает данные из формы в стандартный поток ввода. Сервер не будет пересылать вам сообщение об окончании пересылки данных в стандартный поток ввода; вместо этого используется переменная среды CONTENT\_LENGTH для определения, какое количество данных вам необходимо считать из стандартного потока ввода. Данный метод рекомендуется к использованию.

ACTION

ACTION описывает URL, который будет вызываться для обработки формы. Данный URL почти всегда указывает на CGI-программу, обрабатывающую данную форму.

Тэги Формы

TEXTAREA

Тэг <TEXTAREA> используется для того, чтобы позволить пользователю вводить более одной строки информации (свободный текст). Вот пример использовани тэга <TEXTAREA>:

<TEXTAREA NAME="address" ROWS=10 COLS=50>Москва,Дмитровкое шоссе,д.9Б, офис 448 </TEXTAREA>

Атрибуты, используемые внутри тэга <TEXTAREA> описывают внешний вид и имя вводимого значения. Тэг </TEXTAREA> необходим даже тогда, когда поле ввода изначально пустое. Описание атрибутов:

NAME - имя поля ввода

ROWS - высота поля ввода в символах

COLS - ширина поля ввода в символах

Если вы хотите, чтобы в поле ввода по умолчанию выдавался какой-либо текст, то необходимо вставить его внутри тэгов <TEXTAREA> и </TEXTAREA>.

INPUT

Тэг <INPUT> используется для ввода одной строки текста или одного слова. Атрибуты тэга:

CHECKED - означает, что CHECKBOX или RADIOBUTTON будет выбран.

MAXLENGTH - определяет количество символов, которое пользователи могут ввести в поле ввода. При превышении количества допустимых символов броузер реагирует на попытку ввода нового символа звуковым сигналом и не дает его ввести. Не путать с атрибутом SIZE. Если MAXLENGTH больше чем SIZE, то в поле осуществляется скроллинг. По умолчанию значение MAXLENGTH равно бесконечности.

NAME - имя поля ввода. Данное имя используется как уникальный идентификатор поля, по которому, впоследствии, вы сможете получить данные, помещенные пользователем в это поле.

SIZE - определяет визуальный размер поля ввода на экране в символах.

SRC - URL,. указывающий на картинку (используется совместно с атрибутом IMAGE).

VALUE - присваивает полю значение по умолчанию или значение, которое будет выбрано при использовании типа RADIO (для типа RADIO данный атрибут обязателен)

TYPE - определяет тип поля ввода. По умолчанию это простое поле ввода для одной строки текста. Остальные типы должны быть явно,их полный список приведен ниже:

CHECKBOX

Используется для простых логических (BOOLEAN) значений. Значение, ассоциированное с именем данного поля, которое будет передаваться в вызываемую CGI-программу, может принимать значение ON или OFF.

HIDDEN

Поля данного типа не отображаются броузером и не дают пользователю изменять присвоенныеданному полю по умолчанию значение. Это поле используетс для передачи в CGI-программу статической информации, как то ID прользователя, пароля или другой информации.

IMAGE

Данный тип поля ввода позволяет вам связывать графический рисунок с именем поля. При нажатии мышью на какую-либо часть рисунка будет немедленно вызвана ассоциированная форме CGI-программа. Значения, присвоенные переменной NAME будут выглядеть так - создается две новых переменных: первая имеет имя, обозначенное в поле NAME с добавлением .x в конце имени. В эту переменную будет помещена X-координата точки в пикселах ( считая началом координат левый верхний угол рисунка), на которую указывал курсор мыши в момент нажатия, а переменная с именем, содержащимся в NAME и добавленным .y, будет содержать Y-координату. Все значения атрибута VALUE игнорируются. Само описание картинки осуществляется через атрибут SRC и по синтаксису совпадает с тэгом <IMG>.

PASSWORD

То же самое, что и атрибут TEXT, но вводимое пользователем значение не отображается броузером на экране.

RADIO

Данный атрибут позволяет вводить одно значение из нескольких альтернатив. Для создания набора альтернатив вам необходимо создать несколько полей ввода с атрибутом TYPE="RADIO" с разными значениями атрибута VALUE, но с одинаковыми значениями атрибута NAME. В CGI-программу будет передано значение типа NAME=VALUE, причем VALUE примет значение атрибута VALUE того поля ввода, которое в данный момент будет выбрано (будет активным). При выборе одного из полей ввода типа RADIO все остальные поля данного типа с тем же именем (атрибут NAME) автоматически станут невыбранными на экране.

RESET

Данный тип обозначает кнопку, при нажатии которой все поля формы примут значения, описанные для них по умолчанию.

SUBMIT

Данный тип обозначает кнопку, при нажатии которой будет вызвана CGI-программа (или URL), описанная в заголовке формы. Атрибут VALUE может содержать строку, которая будет высвечена на кнопке.

TEXT

Данный тип поля ввода описывает однострочное поле ввода. Используйте атрибуты MAXLENGTH и SIZE для определения максимальной длинны вводимого значения в символах и размера отображаемого поля ввода на экране (по умолчанию принимается 20 символов).

Меню выбора в формах

Под меню выбора в формах понимают такой элемент интерфейса, как LISTBOX. Существует три типа тэгов меню выбора для форм:

Select - пользователь выбирает одно значение из фиксированного списка значений, представленных тэгами OPTION. Даннй вид представляется как выпадающий LISTBOX.

Select single - то же самое, что и Select, но на экране пользователь видит одновременно три элемента выбора. Если их больше, то предоставляется автоматический вертикальный скроллинг.

Select multiple - позволяет выбрать несколько элементов из LISTBOX.

SELECT

Тэг SELECT позволяет пользователю выбрать значение из фиксированного списка значений. Обычно это представлено выпадающим меню.

Тэг SELECT имеет один или более параметр пежду стартовым тэгом <SELECT> и завершающим </SELECT>.По умолчанию, первый элемент отображается в строке выбора. Вот пример тэга <SELECT>:

<FORM>

<SELECT NAME=group>

<OPTION> AT 386

<OPTION> AT 486

<OPTION> AT 586

</SELECT>

</FORM>

SELECT SINGLE

Тэг SELECT SINGLE - это то же самое, что и Select, но на экране пользователь видит одновременно несколько элементов выбора (три по умолчанию). Если их больше, то предоставляется автоматический вертикальный скроллинг. Количество одновременно отображаемых элементов определяется атрибутом SIZE. Пример:

<FORM>

<SELECT SINGLE NAME=group SIZE=4>

<OPTION> AT 386

<OPTION> AT 486

<OPTION> AT 586

<OPTIONS> Pentium PRO

</SELECT>

</FORM>

SELECT MULTIPLE

Тэг SELECT MULTIPLE похож на тэг SELECT SINGLE, но пользователь может одновременно выбрать более чем один элемент списка. Атрибут SIZE определяет количество одновременно видимых на экране элементов, атрибут MULTIPLE - максимальное количество одновременно выбранных элементов. Пример:

<FORM>

<SELECT SINGLE NAME=group SIZE=4 MULTIPLE=2>

<OPTION> AT 386

<OPTION> AT 486

<OPTION> AT 586

<OPTIONS> Pentium PRO

</SELECT>

</FORM>

Если выбрано одновременно несколько значений, то серверу передаютс соответствующее выбранному количество параметров NAME=VALUE с одинаковыми значениями NAME, но разными VALUE.

**Отправление файлов при помощи форм**

Формы можно использовать для отправки не только небольших информационных сообщений ввиде параметров, а также и для отправки файлов.

Внимание! Поскольку данная возможность требует поддержки получения файлов WEB - сервером, то, соответственно, необходимо, чтобы сервер поддерживал получение файлов!

Например:

<FORM ENCTYPE="multipart/form-data" ACTION="url" METHOD=POST>

Отправить данный файл: <INPUT NAME="userfile" TYPE="file">

<INPUT TYPE="submit" VALUE="Отправить файл">

</FORM>

<FORM action="http://pandemonium.cs.nstu.ru/~gun/docs/sites.htm" encType="multipart/form-data" method="post">

Отправить данный файл: <INPUT name="userfile" type="file">

<INPUT type="submit" value="Отправить файл">

</FORM>