Validating App Store Receipts

Important: This is a preliminary document. Although it has been reviewed for technical accuracy, it is not final. Apple is supplying this information to help you adopt the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be vetted against final documentation. For information about updates to this document, go to the Apple Developer website.

You can add receipt validation code to your application to prevent unauthorized copies of your application from running. Refer to the license agreement and the review guidelines for specific information about what your application may and may not do to implement copy protection.

Receipt validation requires an understanding of cryptography and a variety of secure coding techniques. It's important that you employ a solution that is unique to your application.

You should perform receipt validation immediately after your application is launched, before displaying any user interface or spawning any child processes. Ideally, this check should happen in `main`, before `NSApplicationMain` is called. For additional security, you may repeat this check periodically while your application is running.

Contents:

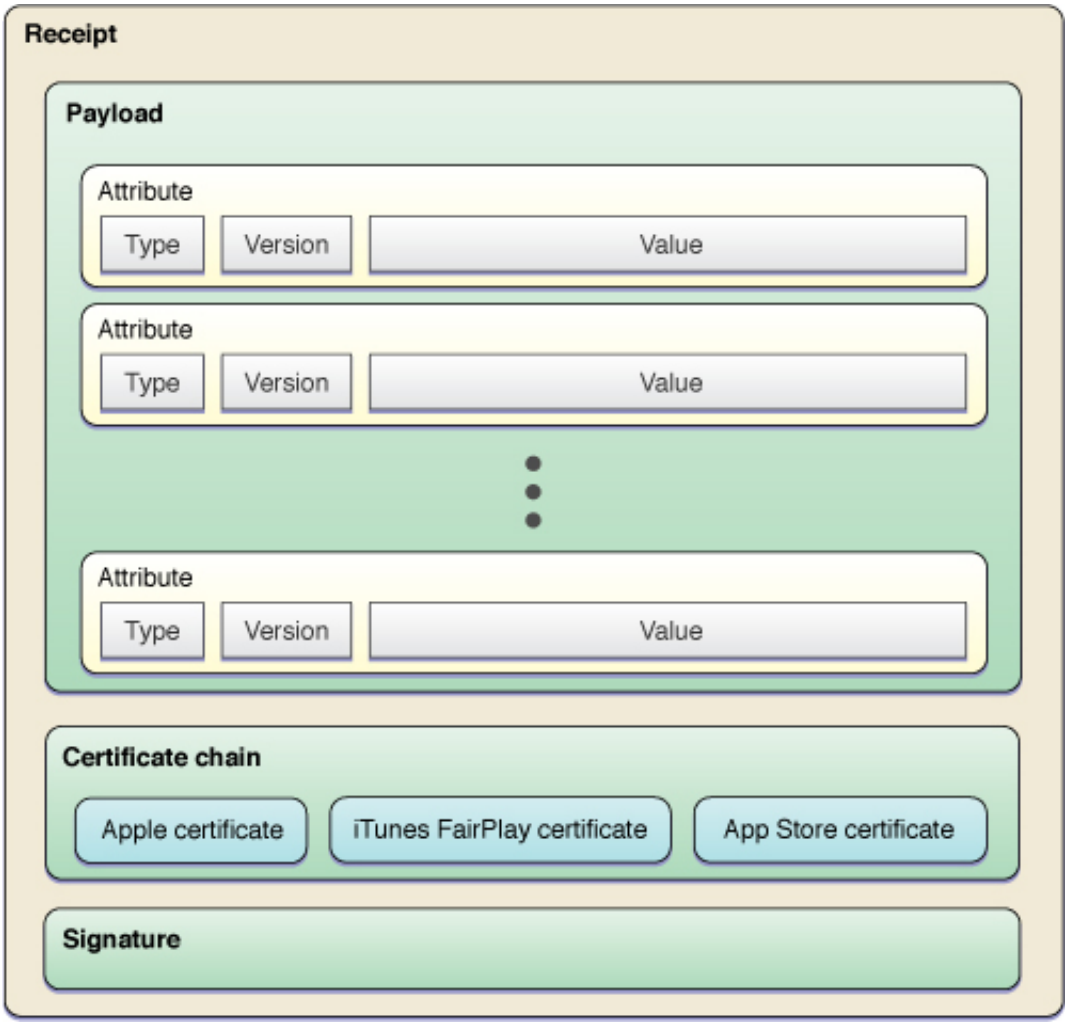
Locate and Parse the Receipt

When an application is installed from the App Store, it contains a receipt that is cryptographically signed, ensuring that only Apple can create valid receipts. The receipt is stored inside the application bundle. For example, if your application is named SampleApp, your receipt is located at:

```
.../SampleApp.app/Contents/_MASReceipt/Receipt
```

The receipt is a binary file with the structure shown in Figure 1-1.

Figure 1-1 Structure of an App Store receipt



The outermost portion is a PKCS #7 container, as defined by RFC 2315, with its payload encoded using ASN.1 (Abstract Syntax Notation One), as defined by ITU-T X.690. The payload is composed of a set of **receipt attributes**. Each receipt attribute contains a type (an ASN.1 `INTEGER`), a version (an ASN.1 `INTEGER`), and a value (encoded as an ASN.1 `OCTET STRING`). Table 1-1 lists the documented receipt attribute types. Ignore all receipt attributes with types that do not appear in this table—their contents may change at any time.

Table 1-1 Receipt attribute types

Type	Definition	Value
2	Bundle identifier	Interpret as an ASN.1 `UTF8STRING`.
3	Application version	Interpret as an ASN.1 `UTF8STRING`.
4	Opaque value	Interpret as a series of bytes.
5	SHA-1 hash	Interpret as a 20-byte SHA-1 digest value.

The format of the payload is as shown in Listing 1-1. You can generate C code with data type declarations and functions for encoding/decoding the payload using the `asn1c` tool. (You may have to install `asn1c` first; it is available from several sources online.) Save the listing to a file and, in Terminal, run the following command:

```
asn1c -fnative-types filename
```

Listing 1-1 ASN.1 Description of the payload format

```
ReceiptModule DEFINITIONS ::=
BEGIN

ReceiptAttribute ::= SEQUENCE {
    type    INTEGER,
    version INTEGER,
    value   OCTET STRING
}

Payload ::= SET OF ReceiptAttribute

END
```

Compute the Hash of the GUID

Use the method described in “Get the Computer’s GUID” to fetch the computer’s GUID. To compute the hash, first concatenate the GUID value with the opaque value (the attribute of type 4) and the bundle identifier. Be sure you use the raw bytes from the receipt. Then compute the SHA-1 hash of this concatenated series of bytes.

Validate the Receipt

Perform the following tests, in order:

1. Look for the receipt. If no receipt is present, verification fails.
2. Verify that the receipt is properly signed by Apple. If it is not signed by Apple, verification fails.
3. Verify that the bundle identifier matches the value for `CFBundleIdentifier` in the `Info.plist` file. If they do not match, verification fails.
4. Verify that the version identifier string in the receipt matches the value for `CFBundleShortVersionString` in the `Info.plist` file. If they do not match, verification fails.
5. Compute the hash of the GUID as described in “Compute the Hash of the GUID”. If the result does not match the hash in the receipt, verification fails.

If all of the tests pass, verification passes.

> **Note:** Bundle identifiers and version identifier strings are UTF-8 strings, not just a series of bytes. Make sure you code your comparison logic accordingly.

Exit If Validation Fails

Validation can fail for a variety of reasons. For example, when users copy your application from one computer to another, the GUID no longer matches, causing receipt validation to fail.

If validation fails, call `exit` with a status of `173`. This status notifies the system that your application

has determined that its receipt is invalid. At this point, the system attempts to get a valid receipt and may prompt for the user's iTunes credentials.

If the system can get a valid receipt, it relaunches the application. Otherwise, it displays an error message to the user, explaining the problem.

Set a Minimum System Version

If you implement receipt validation, include the `LSMinimumSystemVersion` key in your application's `Info.plist` file with a value of 10.6.6 or greater. If receipt validation fails on older versions of Mac OS X, your application quits immediately after launch with no explanation to the user. Older versions of Mac OS X do not interpret the exit status of `173`, so they do not try to obtain a valid receipt or display any error message.

Protect Your Validation Check

Remember that an attacker may try to circumvent the validation code by patching your application binary or altering the basic operating system routines that the validation code depends upon. Resilience against these types of attacks requires a variety of coding techniques, for example:

- Inline the code for cryptographic checks instead of using the APIs provided by the system.
- Avoid simple code constructions that provide a trivial target for patching the application binary. For example, avoid writing code like the following:

```
if (failedValidation) {
    exit(173);
}
```

- Implement code robustness techniques, such as obfuscation. If every application uses the exact same instructions for performing validation, this common code signature can be targeted by tools that patch application binaries.
- Ensure that your application stops running even if the `exit` function fails to terminate your application.

Validate During the Development Process

When the App Store is open for submissions, you will be able to download a tool to facilitate application testing. With this tool installed, perform the following actions to test your application:

1. Make sure you have Internet access so you can connect to Apple's servers.
2. Launch your application by double-clicking on it (or in some way cause Launch Services to launch it).

At this point, the following occurs:

- Your application fails to validate its receipt because there is no receipt present, and exits with a status of `173`.
- The system interprets the exit status and attempts to obtain a valid receipt. Assuming your application signing certificate is valid, the system installs a valid receipt for the application. The system may prompt you for your iTunes credentials.

- The system relaunches your application, and your application successfully validates the receipt.

With this development receipt installed, you can launch your application by any method—for example, with `gdb` or the Xcode debugger.

Implementation Tips

This section contains several code listings to help you implement receipt validation.

Get the Computer's GUID

Follow the steps in Listing 1-2 (or even use this exact same code), so that the method you use to get the GUID in your validation code is exactly the same as the method used when the application's receipt was created.

Listing 1-2 Get the computer's GUID

```
#import <IOKit/IOKitLib.h>
#import <Foundation/Foundation.h>

// Returns a CFData object, containing the computer's GUID.
CFDataRef copy_mac_address(void)
{
    kern_return_t             kernResult;
    mach_port_t               master_port;
    CFMutableDictionaryRef    matchingDict;
    io_iterator_t             iterator;
    io_object_t               service;
    CFDataRef                 macAddress = nil;

    kernResult = IOMasterPort(MACH_PORT_NULL, &master_port);
    if (kernResult != KERN_SUCCESS) {
        printf("IOMasterPort returned %d\n", kernResult);
        return nil;
    }

    matchingDict = IOBSDNameMatching(master_port, 0, "en0");
    if (!matchingDict) {
        printf("IOBSDNameMatching returned empty dictionary\n");
        return nil;
    }

    kernResult = IOServiceGetMatchingServices(master_port, matchingDict,
&iterator);
    if (kernResult != KERN_SUCCESS) {
        printf("IOServiceGetMatchingServices returned %d\n", kernResult);
        return nil;
```

```
    }

    while((service = IOIteratorNext(iterator)) != 0) {
        io_object_t parentService;

        kernResult = IORegistryEntryGetParentEntry(service, kIOServicePlane,
                &parentService);
        if (kernResult == KERN_SUCCESS) {
            if(macAddress) CFRelease(macAddress);

            macAddress = IORegistryEntryCreateCFProperty(parentService,
                    CFSTR("IOMACAddress"), kCFAllocatorDefault, 0);
            IOObjectRelease(parentService);
        } else {
            printf("IORegistryEntryGetParentEntry returned %d\n", kernResult);
        }

        IOObjectRelease(service);
    }

    return macAddress;
}
```

Parse the Receipt and Verify Its Signature

Use the following code listings as an outline of one possible implementation using OpenSSL and `asn1c`. These listings are provided to guide you as you write your own code by highlighting relevant APIs and data structures, not to use as a copy-and-paste solution. This process consists of the following tasks:

1. Verify the signature (Listing 1-3).
2. Parse the payload (Listing 1-4).
3. Extract the receipt attributes (Listing 1-5).
4. Compute the hash of the GUID (Listing 1-6).

Listing 1-3 Verify the signature using OpenSSL

```
/* The PKCS #7 container (the receipt) and the output of the verification. */
BIO *b_p7;
PKCS7 *p7;

/* The Apple Root CA, as raw data and in its OpenSSL representation. */
BIO *b_x509;
X509 *Apple;

/* The root certificate for chain-of-trust verification. */
X509_STORE *store = X509_STORE_new();
```

```
/* ... Initialize both BIO variables using BIO_new_mem_buf() with a buffer and
its size ... */


/* Initialize b_out as an output BIO to hold the receipt payload extracted
during signature verification. */

BIO *b_out = BIO_new(BIO_s_mem());


/* Capture the content of the receipt file and populate the p7 variable with
the PKCS #7 container. */

p7 = d2i_PKCS7_bio(b_p7, NULL);


/* ... Get the Apple Root CA from http://www.apple.com/certificateauthority
and load it into b_X509 ... */


/* Initialize b_x509 as an input BIO with a value of the Apple Root CA then
load it into X509 data structure. Then add the Apple Root CA to the structure.
*/

Apple = d2i_X509_bio(b_x509, NULL);

X509_STORE_add_cert(store, Apple);


/* Verify the signature. If the verification is correct, b_out will contain
the PKCS #7 payload and rc will be 1. */

int rc = PKCS7_verify(p7, NULL, store, NULL, b_out, 0);


/* For additional security, you may verify the fingerprints of the issuer
certificates in the receipt. */
```

Listing 1-4 Parse the payload using asn1c

```
#include "Payload.h" /* This header file is generated by asn1c. */


/* The receipt payload and its size. */

void *pld = NULL;

size_t pld_sz;


/* Variables used to parse the payload. Both data types are declared in
Payload.h. */

Payload_t *payload = NULL;

asn_dec_rval_t rval;


/* ... Load the payload from the receipt file into pld and set pld_sz to the
payload size ... */


/* Parse the buffer using the decoder function generated by asn1c.  The
payload variable will contain the receipt attributes. */

rval = asn_DEF_Payload.ber_decoder(NULL, &asn_DEF_Payload, (void **)&payload,
pld, pld_sz, 0);
```

Listing 1-5 Extract the receipt attributes

```
/* Variables used to store the receipt attributes. */
OCTET_STRING_t *bundle_id = NULL;
OCTET_STRING_t *bundle_version = NULL;
OCTET_STRING_t *opaque = NULL;
OCTET_STRING_t *hash = NULL;

/* Iterate over the receipt attributes, saving the values needed to compute
the GUID hash. */
size_t i;
for (i = 0; i < payload->list.count; i++) {
    ReceiptAttribute_t *entry;

    entry = payload->list.array[i];

    switch (entry->type) {
        case 2:
            bundle_id = &entry->value;
            break;
        case 3:
            bundle_version = &entry->value;
            break;
        case 4:
            opaque = &entry->value;
            break;
        case 5:
            hash = &entry->value;
            break;
        }
    }
```

Listing 1-6 Compute the hash of the GUID

```
/* The GUID returned by copy_mac_address() is a CFDataRef.  Use
CFDataGetBytePtr() and CFDataGetLength() to get a pointer to the bytes that
make up the GUID and to get its length. */
UInt8 *guid = NULL;
size_t guid_sz;

/* Declare and initialize an EVP context for OpenSSL. */
EVP_MD_CTX evp_ctx;
EVP_MD_CTX_init(&evp_ctx);

/* A buffer for result of the hash computation. */
UInt8 digest[20];

/* Set up the EVP context to compute a SHA-1 digest. */
EVP_DigestInit_ex(&evp_ctx, EVP_sha1(), NULL);
```

```
/* Concatenate the pieces to be hashed.  They must be concatenated in this
order. */
EVP_DigestUpdate(&evp_ctx, guid, guid_sz);
EVP_DigestUpdate(&evp_ctx, opaque->buf, opaque->size);
EVP_DigestUpdate(&evp_ctx, bundle_id->buf, bundle_id->size);

/* Compute the hash, saving the result into the digest variable. */
EVP_DigestFinal_ex(&evp_ctx, digest, NULL);
```

Sample Receipt

Listing 1-7 contains an example of a valid receipt for a machine with a GUID of `0017f2c4bcc0`. To save it to a file, select the entire listing and copy it. Then in Terminal, run the command `pbpaste | uudecode` to save it as a file named Receipt in the current directory.

Listing 1-7 A sample valid receipt

```
begin 644 Receipt
M,((."08)*H9(AO<-`0<"H((-^C""#?8"`0$Q"S`)!@4K#@,"&@4`,((!I@8)
M*H9(AO<-`0<!H((!EP2"`9,Q@@&/,`L"`0X"`0$$`P(!`3`,`@$*`@$!!`06
M`C0K,`P"`0T"`0$$!`(")Q`P#0(!`0(!`00%`@,2UH<P#@(!!`(!`00&`@0`
MO&%.,`X"`0D"`0$$!@($43$P,#`.`@$+`@$!!`8"!`4Y?[$P#@(!#P(!`00&
M`@0ZWFBQ,`\"`0,"`0$$!PP%,2XP+C(P'`(!!0(!`004K>.[U=F*J:F(_T(R
MA_9?D3*-IE@P'@(!"`(!`006%A0R,#$P+3$P+3`W5#`S.C4W.C,U6C`>`@$,
M`@$!!!86%#(P,3`M,3`M,C%4,#(Z,CDZ,CA:,!\"`0("`0$$%PP58V]M+F5X
M86UP;&4N4V%M<&QE07!P,$$"`0<"`0$$.4,H%@(A!#M&7`K)^>$RQ"NG@^(F
MC/2B@;D.GA%O00+6;"=^KTRF-<!YR,GIOVA!2*,(@,QS_0?U5S!"`@$&`@$!
M!#K\!S"88)50(JY'V77EK.PIN=&5?Y'H!!X"'\PB;7PDK73:H,R:SF4N8V]M
M4AE*.)DF']R.:I&U"+LBH((+`3""`LDP@@(RH`,"`0("#3,SKQ`0%*\``:\`
M``$P#08)*H9(AO<-`0$%!0`P>S$+,`D&`U4$!A,"55,Q$S`1!@-5!`H3"D%P
M<&QE($EN8RXQ)C`D!@-5!`L3'4%P<&QE($-E<G1I9FEC871I;VX@075T:&]R
M:71Y,2\P+08#500#$R9!<'!L92!&86ER4&QA>2!#97)T:69I8V%T:6]N($%U
M=&AO<FET>3`>%PTQ,#$P,30P,#`S,3=:%PTQ-3$P,3,P,#`S,3=:,&PQ"S`)
M!@-5!`83`E53,1,P$08#500*$PI!<'!L92!);F,N,1<P%08#500+$PY!<'!L
M92!&86ER4&QA>3$O,"T&`U4$`Q,F36%C07!P4W1O<F4N,S,S,T%&,3`Q,#$T
M048P,#`Q048P,#`P,#$P@9\P#08)*H9(AO<-`0$!!0`#@8T`,(&)`H&!`+<?
M,?QZX\M"GX/7@\2&^2NB6\YRO-M:.OP2]9*"@V5`*NX&2#'(O4UX4:'%9'\J
M*PE42%^4D?^0NM]"<05=GRQ-/G3^UXFXC^G:0*8H*]7QYME(S#Q;7)#[B2I>
M"C)+3I+"K!LUD^4P2AXQ6K"`%GB%$C+DN=[4XO`\##L%Q^R]`@,!``&C8#!>
M,`X&`U4=#P$!_P0$`P(#N#`,!@-5'1,!`?\$`C``,!T&`U4=#@06!!3=$8!3
M733D,VNTN(RDVODRHI2(K#`?!@-5'2,$&#`6@!3Z#=01D1OFLDX>!DF4$=UC
M8@=99#`-!@DJADB&]PT!`04%``.!@0!/(X$X\36F=0%T1`=_.$UDZ?,C%;SM
MPYD0IHBWS[#E%*$.4J2"#C<BVT'YMM;AER2[-SFYQ"8U75$O#?UD<Y*06#)$
M_G"D::4_<((U6B&CB04FP1!4_K4]+5HM<0V75BX@$O"U(TA,Z"U!:>P8?.N4
```

```
MZ?)H)HZN,R/X)`S^NAKP$#""`W$P@@)9H`,"`0("`1$P#08)*H9(AO<-`0$%
M!0`P8C$+,`D&`U4$!A,"55,Q$S`1!@-5!`H3"D%P<&QE($EN8RXQ)C`D!@-5
M!`L3'4%P<&QE($-E<G1I9FEC871I;VX@075T:&]R:71Y,18P%`8#500#$PU!
M<'!L92!2;V]T($-!,!X7#3`W,#(Q-#$Y,C`T,5H7#3$R,#(Q-#$Y,C`T,5HP
M>S$+,`D&`U4$!A,"55,Q$S`1!@-5!`H3"D%P<&QE($EN8RXQ)C`D!@-5!`L3
M'4%P<&QE($-E<G1I9FEC871I;VX@075T:&]R:71Y,2\P+08#500#$R9!<'!L
M92!&86ER4&QA>2!#97)T:69I8V%T:6]N($%U=&AO<FET>3"!GS`-!@DJADB&
M]PT!`0$%``.!C0`P@8D"@8$`LF<\72KGC_)WS^\QO-$$_>J?T(8ID`JC_V1*
M[VGT"KGOQ,1CRCHM(3U2C".)*:K;%]S+*,1IS7(DPA#!]'@=,1;%YNH9'2HS
MOH"KW!%(!BH?#F&!S7F:71+TA)#(?/9XFU+X^O&(;A!R^%W+B.RV5127I$T+
MP4\G9YFFI+/7VJT"`P$``:.!G#"!F3`.!@-5'0\!`?\$!`,"`88P#P8#51T3
M`0'_!`4P`P$!_S`=!@-5'0X$%@04^@W4$9$;YK).'@9)E!'=8V('660P'P8#
M51TC!!@P%H`4*]!I1Y1V"?[T:XTN0*;W1TU_"%XP-@8#51T?!"\P+3`KH"F@
M)X8E:'1T<#HO+W=W=RYA<'!L92YC;VTO87!P;&5C82]R;V]T+F-R;#`-!@DJ
MADB&]PT!`04%``."`0$`P*!S^!WJ'-'-Q89UI+OK0,5J+0S^6_QNP5O72'U`
MJ6$M2S=P..\&D4NO$<(?E>Z(,V]?<NKVU7:U5UAQ\#X0P]4NNR^F.G/",F4*
M5@06&9A-"WAMT*-T9)A4]*?7':+_*L-Y`+*9Y^>[U0:BZTDS:6=!`9FM:+#3
M>O25X/TDB"H0YZ-H_ZU[^MFZ\:5_RI.BS@,W`V:%DP4;,?9U@I,OT_#:.3<]
M7_ZZS9IPP`:+(NZIS@1N)&D[%E*EP/+`KD-PA+LAD4/+N[?J6[J3X+=WB>]-
M5@0S7<Y<63&GUSM%5A'-KLN;#J#/W).HRL58>``%V9FKUEZ'ZI)S-%2CDQBD
M@S""!+LP@@.CH`,"`0("`0(P#08)*H9(AO<-`0$%!0`P8C$+,`D&`U4$!A,"
M55,Q$S`1!@-5!`H3"D%P<&QE($EN8RXQ)C`D!@-5!`L3'4%P<&QE($-E<G1I
M9FEC871I;VX@075T:&]R:71Y,18P%`8#500#$PU!<'!L92!2;V]T($-!,!X7
M#3`V,#0R-3(Q-#`S-EH7#3,U,#(P.3(Q-#`S-EHP8C$+,`D&`U4$!A,"55,Q
M$S`1!@-5!`H3"D%P<&QE($EN8RXQ)C`D!@-5!`L3'4%P<&QE($-E<G1I9FEC
M871I;VX@075T:&]R:71Y,18P%`8#500#$PU!<'!L92!2;V]T($-!,(((!(C`-
M!@DJADB&]PT!`0$%``."`0\`,(((!"@*""`0$`Y)&I"1^1VQY'4.L%[5YYA"WK
M-J)73%7LBQF)WOE+;/4'JR(P`N@8/OA0"=-_0:B8^='*9IPD:Q'0H[OD&RK#
M'Y6>>@RD1XM;U!8W,\O$#TW.%&G1R1ER]5T.U7]?F_(E`[I5CTU=#?%D-2,5
M2Q59';.4]_:<GL]0NL%84&>/"+0@]\NL+"!O<+8_`3",MT//#YT]\RM)*!K(
M_LZUN0[97AS6RSVU.JWT#PX`D@NQ(18N=-4\#=MB%JNC<9)'4U7!KR]!L_C[
MXW#-YJ-,17X?3&M0ED&)Q'1B"Q"#08<SBH&Q,%CL6@0RC&BSCQW>97/_9UYE
MO$G8=I\S%&6A=Y3)+0(#`0`!HX(!>C""`78P#@8#51T/`0'_!`0#`@$&,`\&
M`U4=$P$!_P0%,`,!`?\P'08#51T.!!8$%"O0:4>4=@G^]&N-+D"F]T=-?PA>
M,!\&`U4=(P08,!:`%"O0:4>4=@G^]&N-+D"F]T=-?PA>,((!$08#51T@!(((!
M"#""`00P@@$`!@DJADB&]V-D!0$P@?(P*@8(*P8!!04'`@$6'FAT='!S.B\O
M=W=W+F%P<&QE+F-O;2]A<'!L96-A+S"!PP8(*P8!!04'`@(P@;8:@;-296QI
M86YC92!O;B!T:&ES(&-E<G1I9FEC871E(&)Y(&%N>2!P87)T>2!A<W-U;65S
M(&%C8V5P=&%N8V4@;V8@=&AE('1H96X@87!P;&EC86)L92!S=&%N9&%R9"!T
M97)M<R!A;F0@8V]N9&ET:6]N<R!O9B!U<V4L(&-E<G1I9FEC871E('!O;&EC
M>2!A;F0@8V5R=&EF:6-A=&EO;B!P<F%C=&EC92!S=&%T96UE;G1S+C`-!@DJ
MADB&]PT!`04%``."`0$`7#:93"UXM^V,F]SS=YOR=M)W,$_!'X6#A1N9/4<W
M\JF;0(XLU+&0$MB^]'.;[M)D#\MY3S38HC[Y>/]KR`?L?3F#BU,@TSC$L;^:
M3PIK_RO\6:<%"7P70%81'G33MXLC.T>CU6\DXNO1MW#?#T7A)\KQ;7CMY[47
```

```
M%ZC<?B(URB75V0_6:]2B)",1]Z&LCW.!8,8;6PDODK+X1$CP8#B>%?4])F<@
MBC-J]PV"S][KHR_Y4VI;9,!C,W?W.@<L5NO:#R$.VKIS&4^UV39_P8=5V:>9
MN3)"^]C5<9Y^H5*W&[V30B02*L</';9-G%YCR$N`%U"JBM7:Y/S0"0<WL'5U
M(3&"`3,P@@$O`@$!,(&,,'LQ"S`)!@-5!`83`E53,1,P$08#500*$PI!<'!L
M92!);F,N,28P)`8#500+$QU!<'!L92!#97)T:69I8V%T:6]N($%U=&AO<FET
M>3$O,"T&`U4$`Q,F07!P;&4@1F%I<E!L87D@0V5R=&EF:6-A=&EO;B!!=71H
M;W)I='D"#3,SKQ`0%*\``:\```$P"08%*PX#`AH%`#`-!@DJADB&]PT!`0$%
M``2!@)>5O)5./4]D/:>>4;A0>[+ZO1)F]<ETTB.%QP)X_!!T'BHT#3#W0=TX
M4O#\]'LNP/NRZ5O>\.Z;RGB(BSH;DUW]FNAZ^*(:5-*R#4#%K@G_"#Y>_!A@
J5Q:14S52R3Y<'YPQJ%?R"IL>,XF4;>>EF`C,"*[+;?&7PB'`$T6G]/F7
`
end
```

Last updated: 2010-11-01