



Application Programmers Guide  
**SmartTrust Wib™ Plug-ins**



© 2009, 2010 SmartTrust AB. All rights reserved.

SmartTrust endeavors to ensure that the information in this document is correct and fairly stated, but does not accept liability for any error or omission. The development of SmartTrust products and services is continuous and published information may not be up to date. It is important to check the current position with SmartTrust. This document is not part of a contract or license save insofar as may be expressly agreed.

Unless otherwise noted, all names of companies, products, street addresses and persons contained herein are part of a completely fictitious scenario and are designed solely to document the use of the described product or service.

SmartTrust and SmartTrust WIB are trademarks of SmartTrust AB.

All the other trademarks are the property of their respective owners.



## Contents

1 Document profile	5
1.1 Purpose of the document	5
1.2 Target audience	5
1.3 Terms, acronyms and abbreviations	5
1.4 Symbols	6
1.5 References	6
2 Introduction	8
2.1 What is a Wib™ plug-In?	8
2.2 Organization of this document	9
3 RSA Based Security Plug-Ins	10
3.1 P7 - PKCS#7 Signature Plug-In	11
3.2 FP - Fingerprint Plug-In	13
3.3 AD - Asymmetric Decryption Plug-In	17
4 2nd Generation 3DES Security Plug-Ins	19
4.1 *DE- 3DES Encrypt Plug-In	20
4.2 *DD - 3DES Decrypt Plug-In	22
4.3 *DS - 3DES Sign Plug-In	24
4.4 *DU - 3DES Unwrap Key Plug-in	27
5 1st Generation 3DES Security Plug-Ins	31
5.1 ENCR - 3DES Encrypt Plug-In	32
5.2 DECR - 3DES Decrypt Plug-In	34
5.3 SIGN - 3DES Sign Plug-In	36
6 Card Management Plug-Ins	38
6.1 RTPROF - Retrieve Terminal Profile	39
6.2 RRFMS - Retrieve Remote File Management Status	40
6.3 CP – Change PIN	42
6.4 RP – Reset PIN	44
6.5 EM – Event Manager	48
6.6 *ICCID – Retrieve the ICCID of the card	50
7 Data Management Plug-Ins	51
7.1 DUDA - Display User Data	52
7.2 EUDA - Encrypted User Data	54
7.3 *PAD – Privileged Application Data	57
Appendix A: Triple DES modes	69
A.1 Triple encryption (TDEA_ENCR)	69



A.2 Triple decryption (TDEA\_DECR) ..... 70



## 1 Document profile

### 1.1 Purpose of the document

This document specifies the interfaces and functionality of the standard SmartTrust Wib™ plug-ins. The documentation supports Wib application development with examples given in WIG WML, [WIGWML]. The content of this documentation requires knowledge about the SmartTrust Delivery Platform and Wib.

### 1.2 Target audience

The document is intended for Wib application developers.

### 1.3 Terms, acronyms and abbreviations

<b>Term</b>	<b>Definition</b>
DF	Dedicated File
EF	Elementary File
GSM	Global System for Mobile communications
IV	Initialization Vector
LSB	Least Significant Byte
MAC	Message Authentication Code
MSB	Most Significant Byte
PIN1	Personal Identification Number 1 (CHV1 on SIM)
PKCS#7	Public-Key Cryptography Standard, specification number 7
RSA	Rivest-Shamir-Adleman, asymmetric encryption algorithm
SAT	SIM Application Toolkit
TAR	Toolkit Application Reference, part of GSM 03.48 specification
TLV	Tag-Length-Value, coding scheme
UG	Universal Gateway
WIB	SmartTrust Wib™
WIG	Wireless Internet Gateway
WML	Wireless Markup Language



## 1.4 Symbols

Symbol	Description
$K1, K2, K, K'$	DES keys.
$X // Y$	Concatenation of byte-strings X and Y (in that order).
<i>SHA1</i>	SHA-1 hash function. See [SHA1] for further reference.
<i>ISO_9797_ALG3</i>	ISO9797 MAC algorithm 3. See [ISO9797] section 7.3 for further reference.
<i>ISO_9797_PAD2</i>	ISO9797 padding method 2. See [ISO9797] section 6.1.2 for further reference.
<i>PKCS5_PAD</i>	PKCS#5 padding function. See [PKCS5] section 6.1.1.
<i>PKCS5_UNPAD</i>	Inverse of PKCS5_PAD. See [PKCS5] section 6.1.1.
<i>TDEA_ENCR</i>	Triple DES encryption algorithm. See “A.1 Triple encryption (TDEA_ENCR)” on page 69 for details regarding the algorithm.
<i>TDEA_DECR</i>	Triple DES decryption algorithm. See “A.2 Triple decryption (TDEA_DECR)” on page 70 for details regarding the algorithm.
$\langle i..j \rangle$	Sub-string extraction operator. Extracts bytes $i$ through $j$ . $1 \leq i \leq j$ .

## 1.5 References

Ref.	Title
[WIGWML]	WIG WML v.5 – Specification, SmartTrust
[WMLCLIB]	WAP WMLScript Crypto library, Version 05-Nov-1999, with specification changes as of 27-Dec-2000. Wireless Application Forum
[GSM03.38]	ETSI. GSM 03.38. Alphabets and language specific information. Version 7.2.0. Release 1998.
[GSM11.14]	ETSI. GSM 11.14. Specification of the SIM Application Toolkit for the Subscriber Identity Module - Mobile Equipment (SIM - ME) interface. Version 8.7.0. Release 1999.
[PKCS1]	“PKCS #1 v2.0: RSA Cryptography Standard”, RSA Laboratories
[PKCS5]	RSA Laboratories , “PKCS #5 v2.0: Password-Based Cryptography Standard”, <a href="http://www.rsalabs.com/pkcs/">http://www.rsalabs.com/pkcs/</a>
[PKCS7]	“PKCS #7 v1.5: Cryptographic Message Syntax”, RSA Laboratories
[CMS]	“Cryptographic Message Syntax”, RFC 2630, R. Housley
[WTLS]	Wireless Application Forum , “Wireless Transport Layer Security”, <a href="http://www.wapforum.org/">http://www.wapforum.org/</a>
[DEA]	ISO 8731-1, “Banking – Approved algorithms for message authentication – Part 1: DEA”
[ISO9797]	ISO/IEC 9797-1:1999(E) – Information technology – Security techniques – Message Authentication Codes (MACs)



<b>Ref.</b>	<b>Title</b>
[SHA1]	FIPS PUB 180-1, "Secure Hash Standard (SHS)"
[MODES]	ISO/IEC 10116 – Security Techniques – Modes of Operation for an n-bit Block Cipher Algorithm"



## 2 Introduction

The plug-ins included in this specification are specified by SmartTrust. The plug-ins are implemented by SIM vendors on a Wib card. The Card Issuer, which is normally a Mobile Operator, might define only a subset of the plug-ins for inclusion on a specific SIM card.

The Wib architecture allows for other entities than SmartTrust to specify and implement plug-ins on SIM cards. Therefore it is necessary to find out the current availability of plug-ins on a specific target SIM before building applications.

The WIG WML used in the examples is supported by SmartTrust Delivery Platform version 6.1 and later.

### 2.1 What is a Wib™ plug-In?

A Wib plug-in is a SIM located component providing additional features to SmartTrust Wib™. Such features may be cryptographic functions, file and data management or other functions not in the WIB native command set. Plug-ins can be used in Wib applications and be called from within WIG WML documents that implement installed wiblets as well dynamically loaded wiblets.

The general syntax for calling a plug-in from within a WIG WML document is defined in [WIGWML]<sup>1</sup> and is like the following.

```
<plugin name="PLUGINNAME" destvar="OUTPUT" params="INPUT" class="CLASS"/>
```

The arguments have the following meaning<sup>2</sup>:

Name	Value	Explanation	Var
<b>class</b> optional	SMS-DEFAULT   UCS2   binary	The Wib encoding of the plug-in output. Default is binary.	No
<b>destvar</b> mandatory	variable-name	Name of a variable that will contain the output data from the plug-in.	No
<b>name</b> mandatory	String	The name of the plug-in to call.	No
<b>params</b> mandatory	String	The input parameters to the plug-in.	Yes

<sup>1</sup> For backward compatibility, earlier versions of the plug-in calling syntax is supported, but the use of the syntax specified in this document is encouraged.

<sup>2</sup> For further details, refer to [WIGWML].



Each plug-in provides a specific interface shared between the plug-in and the application using the plug-in. The application interface is the interface provided by the content of the params and destvar arguments found in the generic interface. The application interface is transparent to UG and Wib. It is each plug-in that defines the application interface.

## 2.2 Organization of this document

The plug-ins described in this document are presented in chapters that group the plug-ins into functional groups.

- Security – This section presents plug-ins that provide specific security functions, such as encryption and decryption for end-to-end security.
- Data management – This section presents plug-ins that provide functions for data management to an application.
- Card management – This section presents plug-ins whose function is somehow related to generic management of the card.

Each plug-in is described using the following structure.

- Name – the name of the plug-in used when calling the plug-in
- Description – states the main functionality of the plug-in in question
- Input Parameters – plug-in specific input parameters in calling order. Parameters identified with description and length of bytes. In case of dynamic variable length, a notation *N* is used. Input parameters are passed to the plug-in through the params argument.
- Output Parameters – plug-in specific output parameters. The output parameters are returned by the plug-in in the variable named by the destvar argument.
- WIG WML Example – calling syntax for the plug-in with explanations.



### 3 RSA Based Security Plug-Ins

This chapter introduces RSA based security plug-ins for Wib. The plug-ins are the following.

- P7 - PKCS#7 Signing
- FP - Fingerprint
- AD - Asymmetric Decryption

Note that the administration plug-ins, *Change PIN* and *Install PIN*, may be related to the same keys as the plug-ins in this chapter.



### 3.1 P7 - PKCS#7 Signature Plug-In

#### Name

P7

#### Description

The P7 plug-in is used to provide a digital signature based on a private RSA key stored on a SIM card. The output of this plug-in is compliant with the WMLScript Crypto Library SignText function, [WMLCLIB]. As such, P7 will also be compliant with other important (de-facto) standards such as [PKCS1], [PKCS7], [CMS].

The plug-in first shows the text to be signed to the user and then prompts for a signature PIN. The plug-in implements true WYSIWYS (What-You-See-Is-What-You-Sign).

#### Input Parameters

Parameter	Description	Length
Character Encoding Scheme	This field indicates what character encoding scheme is used for the TTBS and hence shall be used when displaying the TTBS.  The following values are valid: “U” – UCS2 “S” – SMS default alphabet, 7-bit unpacked	1
Format Options	This field specifies additional processing options to the plug-in.  <b>Bit 1:</b> CTFLG – Content flag – If set, the plug-in must include the TTBS in the output.  <b>Bit 2:</b> KHFLG – Key hash flag – If set, the plug-in must include the hash of the public key corresponding to the signature key in the output.  <b>Bit 3:</b> CEFLG – Certificate flag – If set, the plug in must include a URL to the public key certificate in the output.  <b>Bit 4:</b> ICCFLG – ICCID flag – If set, the plug-in must include the ICCID of the SIM in the output.  <b>Bit 5:</b> MDFLG – Message digest flag – If set, the plug-in must include the message digest of the TTBS in the output.  <b>Bits 6-8:</b> RFU.	1
Text To Be Signed (TTBS)	A valid TTBS must not exceed 160 bytes. The character encoding shall be as specified in the CES field.	1-160



## Output Parameters

Parameter	Description	Length
Signature or Error message	<p>The output from the P7 plug-in is one, and only one, of the following:</p> <ul style="list-style-type: none"> <li>- A byte string representing the SignedContent data structure as specified in [WMLCLIB].</li> <li>- An error message.</li> </ul> <p>It is an byte string where the bytes may take any value in the range:</p> <p><b>'0x00' – '0xFF'</b></p>	N

Note that if the total length of the output from the P7 plug-in exceeds the maximum length of a variable, 255 for Wib 1.3 and earlier or 8191 for Wib 2.0, this will lead to an error situation.

## WIG WML Example

This example illustrates the use of the P7 plug-in. The following WIG WML document is sent from a server and contains the text to be signed. The plug-in is called to sign the text and the signature is sent to a server.

The document to be signed is sent in SMS Default alphabet as given by the CES parameter. The OPTS parameter is set to request the document as well as the ICCID to be included in the signature message. This may be useful for an application verifying the signature.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
"http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card>
    <p>
      Signature request from The Bank
      <!-- set input parameters -->
      <setvar name="CES" value="S"/>
      <setvar name="OPTS" value="\x09"/>
      <setvar name="TTBS" value="Amount: 44 USD; Debit acc.no: 123456-7;
Credit acc.no: 9876-543210; Ref: The Insurance company" class="Binary"/>
      <!-- call the signature plug-in -->
      <plugin name="P7" params="$(CES)$(OPTS)$(TTBS)" destvar="STEXT"/>
      <!--send the signature to the host -->
      <go
href="http://dupond.smarttrust.com/sign.php?STEXT=$(STEXT)&RefID=5432
1"/>
    </p>
  </card>
</wml>
```



## 3.2 FP - Fingerprint Plug-In

### Name

FP

### Description

The FP plug-in is used to provide a digital signature based on a private RSA key stored on a SIM card. The output of this plug-in is a WrappedContent structure described below that includes a PKCS#1 compliant signature. As such, FP will also be compliant with other important (de-facto) standards such as [PKCS1], [PKCS7], [CMS].

FP may seem strikingly similar to the P7 plug-in. However, it operates in a different way. As opposed to the P7 plug-in, FP will not operate strictly according to the WYSIWYS (What-You-See-Is-What-You-Sign) paradigm, but instead work more as an alternative to a smart card in a “fixed” PKI scenario. This ensures that FP can be utilized in cases where P7 is clearly unsuitable such as the following.

- Signing of data that is larger than a few hundred bytes. This might be an email message, a word-processor document or some other data.
- Signing of data that is not displayable on a mobile phone such as word-processor documents, pictures or random nonce in a VPN set-up phase.

Other utilization is also easily imaginable.

### Input Parameters

Parameter	Description	Length
Key Usage identifier	This field determines which type of key the plug-in shall use in the forthcoming signing operation.  The following values are valid: “S” – sign “N” – nonRepudiation	1



Parameter	Description	Length
Format options	This field specifies additional processing options to the plug-in. Bit 1: RFU. Bit 2: KHFLG – Key hash flag – If set, the plug-in must include the hash of the public key corresponding to the signature key in the output. Bit 3: CEFLG – Certificate flag – If set, the plug in must include a URL to the public key certificate in the output. Bit 4: ICCFLG – ICCID flag – If set, the plug-in must include the ICCID of the SIM in the output. Bit 5-8: RFU.	1
Data To Be Signed (DTBS)	This field represents the data to be signed. To be truly PKCS#1 compliant, this should be a DER encoded value of the DigestInfo ASN.1 type, as specified in [PKCS1].	16-255

## Output Parameters

Parameter	Description	Length
Signature or Error message	The output from the FP plug-in is one (and only one) of the following: - A byte string representing the signature. The byte string follows the WrappedContent structure specified below. - An error message. It is a byte string where the bytes may take any value in the range: <b>'0x00' – '0xFF'</b>	N

If the total length of the output from the FP plug-in exceeds the maximum length of a variable, 255 for Wib 1.3 and earlier or 8191 for Wib 2.0, this will lead to an error situation.



## Format of WrappedContent

The output of the plug-in follows the WrappedContent format. The format includes both the signature and other related data. It is described using the same presentation language as used in [WTLS].

```
struct {
    opaque signature<0.. 2^16-1>;
} Signature;

enum { implicit(0), sha_key_hash(1), certificate_url(5), iccid
(128), key_usage_id(129), (255) } SignerInfoType;
```

Item	Description
Implicit	The signer is implied by the content.
sha_key_hash	The SHA-1 hash of the public key, encoded as specified in [WTLS].
certificate_url	A URL where the certificate is located.
key_usage_id	An ID revealing the key usage flag (in the PKCS#15 sense) of the signature key.

```
struct {
    SignerInfoType signer_info_type;
    switch (signer_info_type) {
        case implicit: struct{};
        case sha_key_hash: opaque hash[20];
        case certificate_url: opaque url<0..255>;
        case iccid: opaque iccid[10];
        case key_usage_id: uint8;
    };
} SignerInfo;
```

```
struct {
    uint8 version;
    Signature signature;
    SignerInfo signer_infos<0..2^16-1>;
} WrappedContent;
```

Item	Description
Version	Version of the WrappedContent structure. For this specification the version is 1.
Signature	Signature



Item	Description
signer_infos	Information on the signer. This may contain zero items (in case the signer is implicit). Also, there may be multiple items of SignerInfo present (public key hash and a certificate).

### WIG WML Example

This example illustrates the use of the FP plug-in. The following WIG WML document is received from a network application and contains the data to be signed. The data may for example origin from a word document. The plug-in signs the data and the signature is posted to the network application.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
"http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card>
    <p>
      Signature request from The Application, ref: 12345
      <!-- set input parameters -->
      <setvar name="KU" value="S"/>
      <setvar name="OPTS" value="\x00"/>
      <setvar name="DTBS"
value="\x02\x33\xF0\xA8\x89\x28\xF1\x80\x5A\x23\x33\xE3\x38\x61\x30\x11\x
19\x5A\x58\x12"/>
      <!-- call the signature plug-in -->
      <plugin name="FP" params="$(KU)$(OPTS)$(DTBS)"
      destvar="SDATA"/>
      <!-- send the signature to the host -->
      <go
href="http://dupond.smarttrust.com/sign.php?SDATA=$(SDATA)&RefID=1234
5"/>
    </p>
  </card>
</wml>
```



### 3.3 AD - Asymmetric Decryption Plug-In

#### Name

AD

#### Description

This plug-in is used for application-level asymmetric decryption. The decryption is performed according to RSADP. See [PKCS1] for further reference.

Just as in the case with the FP plug-in, the motivation for this plug-in is to serve as a replacement for a smart card in a “fixed” PKI scenario. While the FP plug-in is focused on digital signatures, AD is focused on the remaining private key operation, namely decryption.

Together, FP and AD form a complete replacement to the “PC attached” smart card, and in addition offer other benefits like end-user mobility, cost effectiveness and easy deployment.

If the output of the plug-in shall be used in a network application it is crucial that the plaintext is protected by some means, e.g. using "blinding".

#### Input Parameters

Parameter	Description	Length
Key Usage identifier	This field determines which type of key the plug-in shall use in the forthcoming decryption operation. “D” = decrypt “U” = unwrap	1
Ciphertext	This field represents the ciphertext, a byte string of length k, where k is the length in bytes of the modulus n. Hence, for a 1024 bit key, it must be of length 128.	16-255

#### Output Parameters

Parameter	Description	Length
Plaintext or Error message	The output from the FP plug-in is one (and only one) of the following: - A byte string representing the decrypted data. - An error message. It is a byte string where the bytes may take any value in the range: ‘0x00’ – ‘0xFF’	N



If the total length of the output from the AD plug-in exceeds the maximum length of a variable, 255 for Wib 1.3 and earlier or 8191 for Wib 2.0, this will lead to an error situation.

### WIG WML Example

This example illustrates the use of the AD plug-in. The following WIG WML document is received from a network application and contains the data to be decrypted. The data may for example originate from a mail application. The plug-in decrypts the data and the data is posted to the network application. This example is using a modulus that is artificially small, since length of ciphertext is only 16 bytes. Nevertheless it suits its purpose as an example for the calling convention.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
"http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card>
    <p>
      Decryption request from The Application, ref: 135
      <!-- set input parameters -->
      <setvar name="KeyUsage" value="\x44"/>
      <setvar name="CipherText"
value="\xAC\x33\xF0\xA8\x28\xF1\x80\x23\x33\xE3\x61\x30\x11\x5A\x58\x13"
/>
      <!-- call the decryption plug-in -->
      <plugin name="AD" params="$(KeyUsage)$(CipherText)"
destvar="DDATA"/>
      <!-- send the decrypted data to the application -->
      <go
href="http://dupond.smarttrust.com/decrypted.php?DATA=$(DDATA) & RefID=
135"/>
    </p>
  </card>
</wml>
```



## 4 2nd Generation 3DES Security Plug-Ins

This chapter introduces the 2<sup>nd</sup> generation 3DES security plug-ins for Wib. The plug-ins offers the same basic functionality as the 1<sup>st</sup> generation plug-ins as well as significant improvements.

- Support for session keys. Enables a party to create keys dynamically, i.e. whenever they are needed and not only when the card is personalized.
- Key diversification. Enable the possibility to use different keys for different purposes. For example different keys can be used for digital signatures, data encryption/decryption and key transport.
- Small changes in the plug-in interface to improve flexibility.
- Possibility to associate secret keys and PINs.

The plug-ins are:

- \*DE - 3DES Encryption
- \*DD - 3DES Decryption
- \*DS - 3DES Signing
- \*DU - 3DES Unwrap



## 4.1 \*DE- 3DES Encrypt Plug-In

### Name

\*DE

### Description

The *3DES encrypt* plug-in is used to encrypt arbitrary application-level data. It is typically called from a WIG WML document to privacy-protect data before it is transmitted to a network application.

### Input Parameters

Parameter	Description	Length
Key identity	The key identity identifies the key to be used for the operation. Its value is given in hexadecimal form.  The following values are valid: <b>'0x01' – '0xFE'</b>	1
Options	Options bit-field with the following bit-values defined: <b>b1:</b> IV flag: 0 – IV=0 1 – Plaintext starts with IV (8 bytes) <b>b2:</b> Cipher spec: 0 – 3DES ECB EDE with two keys 1 – 3DES CBC EDE with two keys <b>b3 – b8:</b> RFU  The combination 'Plaintext starts with IV' and '3DES ECB EDE with two keys' is illegal.	1
Plaintext	This is the data to be encrypted. It is a byte string where the bytes may take any value in the range: <b>'0x00' – '0xFF'</b>	N

### Output Parameters

The output from the plug-in is a sequence of bytes with the following meaning and in the given order.

Parameter	Description	Length
Ciphertext	This is the encrypted plaintext. It is an byte string where the bytes may take any value in the range: <b>'0x00' – '0xFF'</b>	N



## WIG WML Example

This example illustrates the use of the *3DES encrypt* plug-in. The plug-in encrypts the text entered by the user and the ciphertext is posted to the host.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
"http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card>
    <p>
      To pay the ordered goods we need you to supply your Credit Card Nr
      (will be sent encrypted)
      <!-- Get credit card as input from the user -->
      <input title="Enter Credit Card Nr:" name="PlainText" format="*N"/>
      <!-- set input parameters -->
      <setvar name="KeyId" value="\x02"/>
      <setvar name="Opts" value="\x00"/>
      <!-- call the encryption plug-in -->
      <plugin name="*DE" params="\$(KeyId)\$(Opts)\$(PlainText)"
        destvar="CipherText"/>
      <!-- send the ciphertext to the host -->
      <go href="http://dupond.smarttrust.com/pay.php?CCNR=\$(CipherText)"/>
    </p>
  </card>
</wml>
```

## Decryption procedure

To decrypt the ciphertext received from the plug-in, do the following:

- 1 Select the appropriate Key ( $K_1$ ,  $K_2$ ).
- 2 Calculate the padded message  $PM = TDEA\_DECR(Ciphertext)$  using the appropriate cipher parameterization for  
 $K_1, K_2$             Key(s)  
Cipher mode    ECB or CBC according to cipherspec.  
IV                IV flag.
- 3 Calculate the plaintext  $M = PKCS5\_UNPAD(PM)$ .  
Where:  
 $PKCS5\_UNPAD$     See [PKCS5] section 6.1.1.  
 $TDEA\_DECR$         Triple DES decryption algorithm. See "A.2 Triple decryption (TDEA\_DECR)" on page 70 for details regarding the algorithm.



## 4.2 \*DD - 3DES Decrypt Plug-In

### Name

\*DD

### Description

The *3DES decrypt* plug-in is used to decrypt arbitrary application-level data. It is typically called from a Wib script to recover the data that has been privacy protected by a network application.

### Input Parameters

Parameter	Description	Length
Key identity	The key identity identifies the key to be used for the operation. Its value is given in hexadecimal form.  The following values are valid: <b>'0x01' – '0xFE'</b>	1
Options	Options bit-field with the following bit-values defined:  <b>b1:</b> IV flag: 0 – IV=0 1 – Ciphertext starts with IV (8 bytes)  <b>b2:</b> Cipher spec: 0 – 3DES EBC EDE with two keys 1 – 3DES CBC EDE with two keys  <b>b3 – b8:</b> RFU  The combination 'Ciphertext starts with IV' and '3DES ECB EDE with two keys' is illegal.	1
Ciphertext	This is the data to be decrypted. It is a byte string where the bytes may take any value in the range:  <b>'0x00' – '0xFF'</b>	N

### WIG WML Example

This example illustrates the use of the *3DES decrypt* plug-in. Information is sent to the user in encrypted form. The plug-in decrypts the information. The information is then displayed for the user.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
"http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card>
    <p>
      <!-- Set the encrypted data -->
      <setvar name="CipherText" value="\x2b\x85\x36\x05\x25\x67\x48\xe5"
```



```
        class="Binary"/>
<!-- set input parameters -->
<setvar name="KeyId" value="\x01"/>
<setvar name="Opts" value="\x00"/>
<!-- call the decryption plug-in -->
<plugin name="*DD" params="\$(KeyId)\$(Opts)\$(CipherText)"
  destvar="PlainText" />
<!-- Display the plaintext to the user -->
Your new PIN code is: $(PlainText)
</p>
</card>
</wml>
```

### Encryption procedure

To encrypt a plaintext to be sent to the plug-in, do the following:

- 1 Select the appropriate Key ( $K_1, K_2$ ).
- 2 Calculate the padded message  $PM = PKCS5\_PAD(Plaintext)$ .
- 3 Calculate encrypted message  $EM = TDEA\_ENCR(PM)$  using the following cipher parameterization:

$K_1, K_2$	Key(s)
Cipher mode	ECB or CBC
IV	IV flag.

- 4  $EM$  is the ciphertext that may be decrypted by the plug-in.

Where:

$PKCS5\_PAD$  PKCS#5 padding function. See [PKCS5] section 6.1.1.

$TDEA\_ENCR$  Triple DES encryption algorithm. See “A.1 Triple encryption (TDEA\_ENCR)” on page 69S for details regarding the algorithm.



### 4.3 \*DS - 3DES Sign Plug-In

#### Name

\*DS

#### Description

The *3DES sign* plug-in is used to calculate a message authentication code (MAC) for arbitrary application-level data. The MAC can be used as a data integrity mechanism to verify that data has not been altered in an unauthorized manner. It can also be used as a message authentication mechanism to provide assurance that a message has been originated by an entity in possession of the secret key.

The plug-in displays the text to be signed to the user and prompts for a PIN before calculating the MAC.

The cryptographic algorithm used is ISO9797 MAC algorithm 3, padding method 2. There is an option of using first 4 bytes (32 bits) or 8 bytes (64 bits) of the MAC calculation as output of this plug-in.

#### Input Parameters

Parameter	Description	Length
Key identity	The key identity identifies the key to be used for the operation. Its value is given in hexadecimal value.  The following values are valid: <b>'0x01' – '0xFE'</b>	1
Options	Options bit-field with the following bit-values:  <b>b1:</b> Truncation flag: 1 – 8 byte output 0 – 4 byte output  <b>b2 – b8:</b> RFU	1
Character encoding scheme	Character encoding scheme used in the TTBS. It shall contain one of the following values:  “U” = UCS2 “S” = SMS default alphabet, 7-bit unpacked	1
Text To Be Signed (TTBS)	This field contains the text used as input for the MAC calculation.	N



## Output Parameters

Parameter	Description	Length
Signature	The output from the plug-in is the signature (or more correctly, the MAC) on the text to be signed (TTBS). The length of the output is 4 or 8 bytes as indicated by the Truncation flag MSB of the MAC calculation. The output is an byte string where the bytes may take any value in the range: <b>'0x00' – '0xFF'</b>	N

## WIG WML Example

This example illustrates the use of the *3DES sign* plug-in. The following WIG WML document is received from a host and contains the text to be signed. The plug-in signs the text and the signature is posted to a host.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
"http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card>
    <p>
      Signature request from The Bank...
      <!-- set input parameters -->
      <setvar name="TTBS" value="Amount: 44 USD; Debit acc.no: 123456-7;
To: Credit acc.no: 9876-543210; Ref: The Insurance company"
class="Binary" />
      <setvar name="KeyID" value="\x03" class="Binary"/>
      <setvar name="OPTS" value="\x01" class="Binary"/>
      <setvar name="CES" value="S" class="Binary"/>
      <!-- call the signature plug-in -->
      <plugin name="*DS" params="\$(KeyID)\$(OPTS)\$(CES)\$(TTBS)"
      destvar="MAC"/>
      <!-- send the signature to the host -->
    </p>
    <go
href="http://dupond.smarttrust.com/pay.php?DSTEXT=\$(MAC) & TTBS=\$(TTBS)
"/>
  </card>
</wml>
```



### MAC calculation procedure

To calculate the MAC for verification do the following:

- 1 Select the Key (K, K')
- 2 Calculate the padded message  $PM = ISO\_9797\_PAD2(TTBS)$ .
- 3 Calculate  $MAC = ISO\_9797\_ALG3(PM)$  using the following cipher parameterization:  
K, K'            Key(s)  
Truncation      Truncation to either 4 MSB or 8.
- 4 For verification purposes the MAC is verified to be identical to the output from the plug-in.

Where:

*ISO\_9797\_ALG3*      ISO9797 MAC algorithm 3. See [ISO9797] section 7.3 for further reference.

*ISO\_9797\_PAD2*      ISO9797 padding method 2. See [ISO9797] section 6.1.2 for further reference.



## 4.4 \*DU - 3DES Unwrap Key Plug-in

### Name

\*DU

### Description

The *3DES Unwrap key* plug-in is a key-management plug-in that enables a party in possession of a certain secret key, called a *key encryption key*, to replace a key in the SIM based key file, EF<sub>SKKEY</sub>, at its own desire, under a set of well-defined security conditions.

### Input Parameters

Parameter	Description	Length
Key identity	The key identity identifies the secret key to be replaced/updated. Its value is given in hexadecimal value.  The following values are valid: <b>'0x01' – '0xFE'</b>	1
Algorithm identifier	Algorithm identifier. The following values are legal: <b>'0x01'</b> = 3DES + SHA-1 MDC <b>'0x02'</b> = 3DES + ISO 9797 MAC  All other values are RFU.	1
Encrypted key data	Key data, encrypted and integrity protected. The selected algorithm determines the length of the field.	N

### Output Parameters

None.

### WIG WML Example

This example illustrates the use of the 3DES unwrap key plug-in. The following WIG WML document is received from the host and contains the key to be unwrapped. The plug-in unwraps and stores the key on the card.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
"http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card>
    <p>
      <!-- set input parameters -->
      <setvar name="KeyID" value="\x02" class="Binary"/>
      <setvar name="AlgID" value="\x01" class="Binary"/>
    </p>
  </card>
</wml>
```



```
<setvar name="KeyData"  
value="\xaf\x4a\xbe\xd9\xbf\x5d\x55\x99\x2f\xa6\xc3\x18\x35\xf8\xa9\xbb\x  
f1\xf3\x0d\xfb\x77\xcd\xb9\x86\x18\x8b\xb9\xb3\x03\x32\xde\x9e"  
class="Binary"/>  
<!-- call the unwrap key plug-in -->  
<plugin name="*DU" params="$(KeyID)$(AlgID)$(KeyData)"  
  destvar="dummy"/>  
</p>  
</card>  
</wml>
```

### Encrypted Key Data Calculation Procedure

Creating the Encrypted Key Data to be sent to the Plug-in includes formatting, integrity protection and encryption of key data. The procedure for the two algorithms is described below.

Algorithm 1 is the preferred algorithm and may be the only one supported by the SIM unless specific reasons exist, e.g. like non-existing SHA-1 support on a SIM card.



**Algorithm 1: 3DES with SHA-1 MDC**

- 1 Create nonce and new DES key
- 2 Let *KD* be the Key Data and formatted according to:

Bytes	Description	M/O	Length
1 – 8	Random nonce.	M	8
9 – 24	Double length DES key	M	16
25 – 32	Key checksum (truncated SHA-1 MDC)	M	8

- 3 Let *Key ID* be the key identifier of the key to be replaced/updated on the SIM.
- 4 Calculate a message digest  

$$MD = SHA1(Key\ ID\ ||\ '0x01'\ ||\ KD<1..24>)$$
- 5 Calculate the Key Checksum  

$$KC = MD<1..8>$$

- 6 Encrypt the key data  $EKD = TDEA\_ENCR(KD)$  using the following cipher parameterization:

$K_1, K_2$	Key Encryption Keys.
Cipher mode	Outer CBC. See “A.1 Triple encryption (TDEA_ENCR)” on page 69 for details.
IV	0 (this is not a weakness since the nonce effectively becomes a randomly chosen IV).

- 7 *EKD* is the Encrypted Key Data to be sent to the plug-in.



### Algorithm 2: 3DES with ISO 9797 MAC

- 1 Create nonce and new DES key
- 2 Let *KD* be the Key Data and formatted according to:
 

Bytes	Description	M/O	Length
1 – 8	Random nonce.	M	8
9-24	Double length DES key	M	16
25 – 32	Key checksum (ISO 9797 MAC)	M	8
- 3 Let *Key ID* be the key identifier of the key to be replaced/updated on the SIM.
- 4 Calculate the padded message  

$$PM = ISO\_9797\_PAD2(Key\ ID\ ||\ '02h\ ||\ KD<1..24>)$$
- 5 Calculate the key checksum  

$$KC = ISO\_9797\_ALG3(PM)$$
- 6 Encrypt the key data  $EKD = TDEA\_ENCR(KD)$  using the following cipher parameterization:
 

$K_1, K_2$	Key Encryption Keys.
Cipher mode	Outer CBC. See “A.1 Triple encryption (TDEA_ENCR)” on page 69 for more details.
IV	0 (this is not a weakness since the nonce effectively becomes a randomly chosen IV).
- 7 *EKD* is the Encrypted Key Data to be sent to the plug-in.

**Note:** Using terminology from [ISO9797], keys *K* and *K'* shall be derived by complementing alternate sub-strings of four bits of *K<sub>1</sub>* and *K<sub>2</sub>* respectively, commencing with the first four bits.

**Note:** 8 bytes of output from the MAC calculation shall be used (i.e.  $m=64$  using ISO9797 terminology).



## 5 1st Generation 3DES Security Plug-Ins

This chapter describes the 1st generation 3DES security plug-ins for Wib. These plug-ins provide basic features of 3DES cryptography. They have the advantage of being implemented on most Wib cards. However, the 2nd generation 3DES plug-ins provide more flexibility and if present on the SIM card they should preferably be used, see chapter “2nd Generation 3DES Security Plug-Ins” on page 19. The 1st generation plug-ins are the following.

- ENCR - 3DES Encryption
- DECR - 3DES Decryption
- SIGN - 3DES Signing

The cryptographic algorithm used by the plug-in is triple DES with two keys (EDE2) in CBC mode, TCBC, also known as outer CBC mode, as opposed to the inner CBC mode.

Triple DES in EDE2, means that a block of data is passed through three cryptographic operations – encryption, decryption and encryption again. Since two keys are used, the first key (K1) will be used for the first and the last of these operations, while the second (K2) key will be used for the second operation.



## 5.1 ENCR - 3DES Encrypt Plug-In

### Name

ENCR

### Description

The *3DES encrypt* plug-in is used to encrypt arbitrary application-level data. It is typically called from a WIG WML document to privacy-protect data before it is transmitted to a network application. The cryptographic algorithm used is triple DES as described in [DEA], see also “Appendix A: Triple DES modes” on page 69.

Before encryption, the plug-in pads the data with 0-7 zero bytes to make the length divisible by 8.

### Input Parameters

Parameter	Description	Length
Encryptionkey	The key index to be used. Specified by <code>\xHH</code> syntax, where HH shall be replaced by the desired hexadecimal value.	1
Plaintext	This is the data to be encrypted. It is a byte string where the bytes may take any value in the range: <code>'0x00'</code> – <code>'0xFF'</code>	N

### Output Parameters

The output from the plug-in is a sequence of bytes with the following meaning and in the given order:

Parameter	Description	Length
Padding Information	The padding information gives the number of bytes added as padding to the end of the plaintext before encryption. The following values are valid: <code>'0x00'</code> – <code>'0x07'</code>	1
Ciphertext	This is the ciphertext, that is, the encrypted plaintext. It is a byte string where the bytes may take any value in the range: <code>'0x00'</code> – <code>'0xFF'</code>	N

For a network application the plaintext is received after decrypting bytes 2 to N and stripping the number of padding characters according to byte 1 of the output. Thus, the length of the output is always  $8n+1$ .



## WIG WML Example

This example illustrates the use of the ENCR plug-in. The plug-in encrypts the text entered by the user and the ciphertext is posted to the host. In the plug-in call the key index is set to 2.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
"http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card>
    <p>
      To pay the ordered goods we need you to supply your Credit Card Nr
      (will be sent encrypted)
      <!-- Get credit card as input from the user -->
      <input title="Enter Credit Card Nr:" name="PlainText" format="*N"/>
      <!-- call the encryption plug-in -->
      <plugin name="ENCR" params="\x02$(PlainText)" destvar="CipherText"/>
      <!-- send the ciphertext to the host -->
      <go href="http://dupond.smarttrust.com/pay.php?CCNR1=$(CipherText)"/>
    </p>
  </card>
</wml>
```



## 5.2 DECR - 3DES Decrypt Plug-In

### Name

DECR

### Description

The *3DES decrypt* plug-in is used to decrypt arbitrary application-level data. It is typically called from in a wiblet to recover the data that has been privacy protected by a network application. The cryptographic algorithm used is triple DES as described in [DEA], see also “Appendix A: Triple DES modes” on page 69.

### Input Parameters

Parameter	Description	Length
Encryptionkey	The key index to be used. Specified by <code>\xHH</code> syntax, where HH shall be replaced by the desired hexadecimal value.	1
Padding Information	The padding information gives the number of bytes added as padding to the end of the plaintext before encryption. The following values are valid: ‘0x00’ – ‘0x07’	1
Ciphertext	This is the data to be decrypted. It is a byte string where the bytes may take any value in the range: ‘0x00’ – ‘0xFF’	N

### Output Parameters

The output from the plug-in is a sequence of bytes with the following meaning and in the given order:

Parameter	Description	Length
Plaintext	This is the plaintext received from decrypting the ciphertext. It is a byte string where padding bytes have been removed and where the bytes may take any value in the range: ‘0x00’ – ‘0xFF’	N



## WIG WML Example

This example illustrates the use of the DECR plug-in. Information is sent to the user in encrypted form. The plug-in decrypts the information. The information is then displayed for the user. In the plug-in call the key index is set to 1.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
"http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card>
    <p>
      <!-- Set the encrypted data -->
      <setvar name="CipherText"
value="\x04\x2b\x85\x36\x05\x25\x67\x48\xe5"
      class="Binary"/>
      <!-- call the decryption plug-in -->
      <plugin name="DECR" params="\x01$(CipherText)" destvar="PlainText" />
      <!-- Display the plaintext to the user -->
      Your new PIN code is: $(PlainText)
    </p>
  </card>
</wml>
```



### 5.3 SIGN - 3DES Sign Plug-In

#### Name

SIGN

#### Description

The 3DES sign plug-in is used to calculate a message authentication code (MAC) for arbitrary application-level data. The MAC can be used as a data integrity mechanism to verify that data has not been altered in an unauthorized manner. It can also be used as a message authentication mechanism to provide assurance that a message has been originated by an entity in possession of the secret key.

The plug-in displays the text to be signed to the user and prompts for a PIN before calculating the MAC.

The cryptographic algorithm used is triple DES with two keys (EDE2) in outer CBC mode. The first 4 bytes (32 bits) of the MAC calculation are used as output of this plug-in.

#### Input Parameters

Parameter	Description	Length
Encryptionkey	The key index to be used. Specified by <code>\xHH</code> syntax, where HH shall be replaced by the desired hexadecimal value.	1
Text To Be Signed (TTBS)	This field contains the text used as input for the MAC calculation.	1-160

#### Output Parameters

Parameter	Description	Length
Signature	The output from the plug-in is the signature (or more correctly, the MAC) on the text to be signed (TTBS). The length of the output is 4 bytes and is the 4 MSB of the MAC calculation. It is a byte string where the bytes may take any value in the range: <code>'0x00'</code> – <code>'0xFF'</code>	N



## WIG WML Example

This example illustrates the use of the SIGN plug-in. The following WIG WML document is received from the host and contains the text to be signed. The plug-in signs the text and the signature is posted to the host. In the plug-in call the key index is set to 3.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
"http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card>
    <p>
      Signature request from The Bank...
      <!-- set input parameters -->
      <setvar name="TTBS" value="Amount: 44 USD; Debit acc.no: 123456-7;
To: Credit acc.no: 9876-543210; Ref: The Insurance company"
class="Binary"/>
      <!-- call the signature plug-in -->
      <plugin name="SIGN" params="\x03$(TTBS)" destvar="MAC"/>
      <!-- send the signature to the host -->
      <go
href="http://dupond.smarttrust.com/pay.php?STEXT1=$(MAC)&TTBS=$(TTBS)
"/>
    </p>
  </card>
</wml>
```



## 6 Card Management Plug-Ins

This section describes the management plug-ins for Wib. These are:

- Retrieve Terminal Profile, RTPROF
- Remote File Management, RRFMS
- Change PIN, CP
- Reset PIN, RP
- Event Manager, EM



## 6.1 RTPROF - Retrieve Terminal Profile

### Name

RTPROF

### Description

The terminal profile indicates the SIM Toolkit capabilities of the ME. It is downloaded to the SIM as part of the SIM initialization procedure if the terminal supports SIM Toolkit.

The terminal profile could be very useful for the application to, for example customize the user dialog. The Retrieve Terminal Profile plug-in enables the application to retrieve this information from the SIM.

### Input Parameters

None.

### Output Parameters

Parameter	Description	Length
Terminal Profile	This field holds the profile as a sequence of bytes where each bit indicates the presence of a certain SIM Application Toolkit facility. The Terminal Profile is detailed in [GSM 11.14].	N

### WIG WML Example

This example illustrates the use of the RTPROF plug-in. The plug-in is used to fill the Profile variable that is then sent to a server host. The field for input value is left blank, as there are no input parameters for this plug-in.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
"http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card>
    <p>
      <!-- call the plug-in to retrieve the terminal profile-->
      <plugin name="RTPROF" params="" destvar="Profile"/>
      <!-- get the IMEI as well to connect the profile -->
      <providelocalinfo cmdqualifier="imei" destvar="IMEI"/>
      <!-- send the profile to the host -->
      <go
href="http://dupond.smarttrust.com/profile/submit.php?P=${Profile}&I=
${IMEI}"/>
    </p>
  </card>
</wml>
```

Document number: 90-292  
Revision: 1.4 2010-02-04



## 6.2 RRFMS - Retrieve Remote File Management Status

### Name

RRFMS

### Description

The *Retrieve Remote File Management Status* plug-in is used for retrieving content of a special dedicated file on the SIM. The information can be used to report to the user, the outcome of the GSM 03.48 Remote File Operation.

The plug-in gets three bytes as an input value and according to those seeks the file EF 6F09 in the directory DF 2900 and reads the required information and returns it in the output variable.

### Input Parameters

Parameter	Description	Length
Offset high	This field identifies the high-order byte of offset. Legal values are: <b>'0x00'</b> to <b>'0xFF'</b>	1
Offset low	This field identifies the low-order byte of offset. Legal values are: <b>'0x00'</b> to <b>'0xFF'</b>	1
Length	This field sets the length of data to be retrieved	1

The following content and their plug-in input values are defined:

Retrieve status code:

Offset high: '0x00'

Offset low: '0x00'

Length: '0x02'

Retrieve status description:

Offset high: '0x00'

Offset low: '0x02'

Length: '0x29'



## Output Parameters

Parameter	Description	Length
Content	This field give the content requested from file EF 6F09.	N

When using the input values defined for retrieving the status code or status description the output will be 1 byte or 40 bytes respectively. The maximum size of the status description, i.e. the space that is reserved for it in EF (6F09), is 40 bytes.

### WIG WML Example

This is an example of an application retrieving the status description after having performed an OTA activity. The description is stored in the variable OUT. Here the offset is '0002h', and the number of bytes to fetch is '29h' (41 bytes). The plug-in will then read a length-value pair, starting at the third byte in EF (6F09). It will store the value part in the variable OUT.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
"http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card>
    <p>
      <!-- set the input parameters to read -->
      <!-- the third byte and the following 40 bytes -->
      <setvar name="IN" value="\x00\x02\x29"/>
      <!-- call the plug-in -->
      <plugin name="RRFMS" params="\$(IN)" destvar="OUT"/>
      <!-- Show the status to the user -->
      \$(OUT)
    </p>
  </card>
</wml>
```



## 6.3 CP – Change PIN

### Name

CP

### Description

The *Change PIN* plug-in is used for requesting change of a PIN. The new PIN value is specified by the user through the ME keypad. The user is requested to enter the new PIN twice.

### Input Parameters

Parameter	Description	Length
ID Type	This field identifies the type of the ID supplied in the next parameter. Legal values are: “U” – Asymmetric key usage “S” – Symmetric key ID  Type of the parameter is SMS default character.	1
Private object ID	This field identifies the ID of the private object who’s PIN shall be changed.  Legal values for ID type = "U" are: “N” – ‘non repudiation’ key. “S” – ‘sign’ key “D” – ‘decrypt’ key “U” – ‘unwrap’ key  Legal values for ID type = "S" are: ‘0x01’ to ‘0xFE’ – key ID of the selected symmetric key.	1

### Output Parameters

None.



## WIG WML Example

This is an example of an application requesting the user to change a PIN. The *ID Type* is set to 'Asymmetric key usage' and *Private ID object* is a 'sign' key. The prompts are held on the SIM card and used by the plug-in.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
"http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card>
    <p>
      <!-- set the input parameters -->
      <!-- use value="US" or value="\x55\x53" -->
      <setvar name="IN" value="US"/>
      <!-- call the plug-in to request change pin -->
      <plugin name="CP" params="\$(IN)" destvar="OUT1"/>
    </p>
  </card>
</wml>
```



## 6.4 RP – Reset PIN

### Name

RP

### Description

This plug-in offers a means for a trusted party to reset a PIN in the SIM over-the-air and set a new value.

### Input Parameters

Parameter	Description	Length
ID Type	This field identifies the type of the ID supplied in the next parameter. Legal values are: “U” – Asymmetric key usage “S” – Symmetric key ID  Type of the parameter is SMS default character.	1
Private object ID	This field identifies the ID of the private object who’s PIN shall be changed.  Legal values for ID type = "U" are: “N” – ‘non repudiation’ key. “S” – ‘sign’ key “D” – ‘decrypt’ key “U” – ‘unwrap’ key  Legal values for ID type = "S" are: ‘0x01’ to ‘0xFE’ – key ID of the selected symmetric key.	1
Encrypted PIN block header	Header specifying the algorithm used for wrapping. The following values are legal:  ‘0x01’ = 3DES + SHA-1 MDC ‘0x02’ = 3DES + ISO9797 MAC	1
Encrypted PIN block payload	PIN value, encrypted and integrity protected. Detailed format according to section below. The algorithm specified in the header (the first byte) of this field determines the length.	N

### Output Parameters

None.



### Encrypted PIN block payload calculation

Creating the Encrypted PIN Block data to be sent to the Plug-in includes formatting, integrity protection and encryption of data. The procedure for the two algorithms is described below.

Algorithm 1 is the preferred algorithm and may be the only one supported by the SIM unless specific reasons exist, e.g. like non-existing SHA-1 support on a SIM card.

#### Algorithm 1: 3DES with SHA-1 MDC

- 1 Create nonce and PIN value.
- 2 Let *PB* be the PIN Block and formatted according to:

Bytes	Description	M/O	Length
1 – 8	Nonce. 8 bytes of random data.	M	8
9 – 16	PIN value. Each digit in the PIN shall be encoded with it's corresponding GSM default alphabet value. All unused digits at the end shall be encoded as '0xFF'.	M	8
17 – 36	SHA-1 MDC	M	20

Table 1: PIN Block for algorithm 1

- 3 Calculate a message digest  $MD = SHA1('0x01' // PB<1..16>)$ .
- 4 Encrypt the PIN Block data  $EPB = TDEA\_ENCR(PB)$  using the following cipher parameterization:

Keys	$K_1, K_2$
Mode	Triple encryption in outer CBC mode using two keys in DED operation. See "Appendix A: Triple DES modes" on page 69 for more details.
IV	0 (this is not a weakness since the nonce effectively becomes a randomly chosen IV).

- 5 *EPB* is the Encrypted PIN Block data to be sent to the plug-in.

**Note:** Due to the DES block-size, the encrypted PIN payload will be 40 bytes (5 DES blocks).



**Algorithm 2: 3DES with ISO9797 MAC**

- 1 Create nonce and PIN value.
- 2 Let *PB* be the PIN Block and formatted according to:

Bytes	Description	M/O	Length
1 – 8	Nonce. 8 bytes of random data.	M	8
9 – 16	PIN value. Each digit in the PIN shall be encoded with it's corresponding GSM default alphabet value. All unused digits at the end shall be encoded as '0xFF'.	M	8
17 – 24	ISO 9797 MAC	M	8

Table 2: PIN Block data for algorithm 2

- 3 Create a padded PIN Block  $PPB = ISO\_9797\_PAD2('0x02' || PB<1..16>)$ .
- 4 Calculate  $MAC = ISO\_9797\_ALG3(PPB)$ .  
**Note:** 8 bytes of output from the MAC calculation shall be used (i.e.  $m=64$  using ISO9797 terminology).
- 5 Encrypt the PIN Block data  $EPB = TDEA\_ENCR(PB)$  using the following cipher parameterization:

Keys	$K_1, K_2$
Mode	Triple encryption in outer CBC mode using two keys in DED operation. See “Appendix A: Triple DES modes” on page 69 for more details.
IV	0 (this is not a weakness since the nonce effectively becomes a randomly chosen IV).

**Note:** Using terminology from [ISO9797], keys  $K$  and  $K'$  shall be derived by complementing alternate sub-strings of four bits of  $K_1$  and  $K_2$  respectively, commencing with the first four bits.



## WIG WML Example

An example of a call to the RP plug-in looks like this:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
"http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card>
    <p>
      <!-- set the input parameters -->
      <setvar name="ID" value="US"/>
      <setvar name="Algorithm" value="\x01"/>
      <setvar name="EncrPINBlock"
value="\xc0\x42\x7e\x1a\x52\xcf\x6\x75\xec\xe5\x4f\x52\xb2\x07\x41\x6f\x
13\xde\x88\xaa\xb5\x37\xd0\x34\x7e\x61\xbd\x25\x03\xa7\x4f\x55\x6f\xf1\x8
8\xc3\x32\x69\x69\xe0"/>
      <!-- call the plug-in to reset the pin to a new value -->
      <plugin name="RP" params="\$(ID)\$(Algorithm)\$(EncrPINBlock)"
destvar=" OUT1"/>
    </p>
  </card>
</wml>
```

In above shown example, the *ID Type* is 'Asymmetric key usage', *Private object ID* is 'sign' key and *Encrypted PIN Block header* is set to value 1.



## 6.5 EM – Event Manager

### Name

EM

### Description

The *Event Manager* plug-in allows an application to enable/(refresh)/disable the event mechanisms supported in Wib 1.2 (and later), both handset events and internal SIM events.

When the plug-in is called with the DISABLE flag, Wib is set to ignore all incoming events, both internal and handset events. This is valid either until the plug-in is called again with the ENABLE flag, or until next SIM Initialization, whichever occurs first.

When the plug-in is called with the ENABLE/REFRESH flag, the plug-in examines the EF (EventConfig), 6F0B. For all internal events, a coupling from those events to the corresponding event-activated scripts in EF (6F03) is established. If handset events occur in the file, and the handset supports events, the SAT command SET UP EVENT LIST is issued (if needed).

### Input Parameters

Parameter	Description	Length
Mode of operation flag	This field indicates the type of operation mode to be set for the <i>EventManager</i> . Legal values are: <b>'0x00'</b> – Disable event mechanism <b>'0x01'</b> – Enable/Refresh event mechanism	1

### Output Parameters

None.



## WIG WML Example

This example illustrates the use of the EM plug-in.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
"http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card>
    <p>
      <!-- Disable event mechanism -->
      <setvar name="IN" value="\x00"/>
      <plugin name="EM" params="\$(IN)" destvar="OUT"/>
    </p>
  </card>
</wml>
```



## 6.6 \*ICCID – Retrieve the ICCID of the card

### Name

\*ICCID

### Description

The \*ICCID plug-in provides functionality for the retrieval of the ICCID of the card where Wib is executing.

### Input Parameters

None

### Output Parameters

Parameter	Description	Length
ICCID	As a result the *ICCID plug-in stores the ICCID value, as stored in EF <sub>ICCID</sub> , in the output variable.	N

### WIG WML Example

This example illustrates the use of the \*ICCID plug-in and the ICCID is sent to a remote server.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
"http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card>
    <p>
      <!-- Read ICCID of the card -->
      <plugin name="*ICCID" params="" destvar="ICCID"/>
      <!-- send the signature to the host -->
      <go href="http://dupond.smarttrust.com/demo.php?ICCID=$( ICCID )" />
    </p>
  </card>
</wml>
```



## 7 Data Management Plug-Ins

This chapter describes the following plug-ins.

- Display User Data, DUDA
- Encrypted User Data, EUDA
- Privileged Application Data, \*PAD



## 7.1 DUDA - Display User Data

### Name

DUDA

### Description

The *Display User Data* plug-in is called from WIG WML to let the user display and/or update the value of user specific data.

When the plug-in is called, the requested data on the SIM is displayed to the user together with a descriptive text. The user will then have the opportunity to update the value through the normal editing functions in the terminal.

The displaying of the user data is protected with a PIN1 that needs to be entered before the data is shown. The PIN1 will be blocked in case it has been entered incorrectly too many times. If the PIN1 is disabled Wib execution is aborted.

User data is contained within user data objects stored in a file on the SIM. An object contains a tag, length, data coding scheme, value and descriptive text. The object and its usage is defined by the mobile operator and the user may only change the value part through the DUDA plug-in.

The data managed by the DUDA plug-in may be retrieved by an application by means of the EUDA plug-in.

### Input Parameters

Parameter	Description	Length
User Data Object Tag	This field identifies the user data to be requested. The mobile operator defines the content of the objects on SIM and their usage. Object Tags may take the values within the interval:  '0x01' to '0xFE'.	1

### Output Parameters

None.



## WIG WML Example

In this example, the DUDA plug-in is called to display/update the Credit Card Number. The user data object tag for the Credit Card Number is here '01'. The name of the plug-in is written with capital letters. The output variable "DUMMY" is just a dummy value and is not used.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
"http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card>
    <p>
      <!-- set the input parameters -->
      <setvar name="IN" value="\x01"/>
      <!-- call the plug-in -->
      <plugin name="DUDA" params="\$(IN)" destvar="DUMMY"/>
    </p>
  </card>
</wml>
```



## 7.2 EUDA - Encrypted User Data

### Name

EUDA

### Description

The Encrypted User Data plug-in is called to fetch Encrypted User Data from the SIM.

The plug-in fetches user data objects from the user data object file. To secure the data for transportation, the list of objects is padded and encrypted before it is stored in the output variable. The outcome of the plug-in is always an encrypted user data object value string. The maximum length for the output of the plug-in is 255 bytes. The input parameters include a key id to be used for fetching an encryption key.

User data objects are objects stored in a file on the SIM. An object contains a tag, length, data coding scheme, value and descriptive text. The object and its usage is defined by the mobile operator and its data may be retrieved with the EUDA plug-in.

### Input Parameters

Parameter	Description	Length
Encryptionkey	The key index to be used. Specified by \xHH syntax, where HH shall be replaced by the desired hexadecimal value.	1
User Data Object Tags	This field is a list (byte sequence) of tags for the user data objects to be requested. The mobile operator defines the values of the object tags and their usage. Object Tags may take the values within the interval:  <b>'0x01' to '0xFE'.</b>	N

### Output Parameters

The output data is an encrypted string containing a list of requested User Data Object values. The data objects are stored in the same order as they are defined in the input parameter of the plug-in call. The encrypted output is formatted according to following table:



Parameter	Description	Length
Padding Information	The padding information gives the number of bytes added as padding to the end of the plaintext before encryption.  The following values are valid: <b>'0x00' – '0x07'</b>	1

Ciphertext	This is the encrypted plaintext. It is a byte string where the bytes may take any value in the range: <b>'0x00' – '0xFF'</b>  The decrypted ciphertext contains the list of user data objects and is formatted as:	N
------------	---	---

Parameter	Description	Length
Length	The length of the following list (including padding bytes).	1
User Data Objects	This field is a list (byte sequence) of user data objects.	N
Padding Bytes	This is the bytes used for padding the plaintext before encryption. All bytes have the value '0x00'.  Every each user data object is formatted according to:	N

Parameter	Description	Length
User Data Object Tag	This field identifies the user data. The mobile operator defines the objects tags and their usage. Object Tags may take the values within the interval: <b>'0x01' to '0xFE'</b>	1
Length	The length of the following fields.	1
Character encoding scheme	Character encoding scheme used in the User Data field. It shall contain one of the following values: <b>"0x00"</b> = SMS default alphabet, 7-bit unpacked <b>"0x08"</b> = UCS2	1
User Data	User specific data.	N

If the length of the list of objects is longer than the maximum output length allowed, only the first objects that can be fully added into the output list are included. The succeeding objects are ignored.



## WIG WML Example

In this example, the EUDA plug-in is called to retrieve two User Data Object values: "Credit card number" and "Home address". The user data object tag for the Credit Card Number is '01' and for the Home Address '03'.

The input value is stored into the variable "IN". By a call to the EUDA plug-in, the data is fetched, formatted to output format, encrypted with the 3DES key with index 2 (indicated by using "\x02" first in the params argument), and then stored in the variable CCD. The output of the plug-in is then sent to the Content Provider.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
"http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card>
    <p>
      <!-- set the input parameters -->
      <setvar name="IN" value="\x01\x03"/>
      <!-- call the plug-in -->
      <plugin name="EUDA" params="\x02$(IN)" destvar="CCD"/>
      <go href="http://www.shop.com?CREDITCARDATA=$(CCD)"/>
    </p>
  </card>
</wml>
```



## 7.3 \*PAD – Privileged Application Data

### Name

\*PAD

The plug-in call can use the plug-in name "\*PAD" or an alternate name according to the naming rule "\*PADXX". When using the name "\*PAD" the default default storage area shall be used.

If "\*PADXX" is used an alternate PAD default storage area is used. XX consists of a hexadecimal number in the range '01'h to 'FF'h. The range "\*PADF0" to "PADFF" is reserved by SmartTrust when using alternate plug-in names.

Alternate default storage areas are a mechanism to create multiple databases. In that way an operator can specify cards to have separate databases for different application providers. Doing so reduces the need to coordinate identity usage and space management.

### Description

The \*PAD plug-in is a powerful data management plug-in for privileged applications. It was originally intended to be used by applications defined by SmartTrust. However, provisions have been made for using it even in applications defined by others and approved by the operator.

The \*PAD implements a tag-value storage that can be seen as an on-card database. Since memory is limited, it is important that an application using \*PAD clearly states its maximum memory need to ensure that applications do not fail due to lack of storage space. Since \*PAD stores data items that are identified by a tag, it is also important to ensure that applications use separate tag-ranges.

The plug-in is used to create, read, update, and delete privileged application data stored in a SIM card. This is implemented as separate operations of the plug-in. The data object is an octet string with any hexadecimal value.

The create operation is used to reserve storage space for new application data. The read operation is used to read application data. The update operation is used to update application data. The delete operation is used to delete a single or all application data.

Privileged application data is by default stored in a \*PAD default storage area. More than one \*PAD default storage areas may be used on the same card. A PAD implementation shall select the \*PAD default storage area to be used for an operation depending on the plug-in name used in the plug-in call.



A \*PAD implementation may<sup>3</sup> also support tag-dependent storage areas meaning that data is stored in other storage areas than the PAD default storage area for specific tag values.

In case of an error the normal error handling of Wib. If a faulty operation type or tag value '00'h exist in the plug-in input, generic Wib error handling is used with error code '60'h "invalid input parameter(s)" to inform the user before Wib shall terminate.

Legal tag values are in the range, '01'h...'FE'h. The tag ranges for plug-in calls using the plug-in name "\*PAD" are defined in the below table.

Tag (Hex)	Content
'00'h	Not used
'01'h...'7F'h	Reserved for SmartTrust specified generic applications
'80'h...'DF'h	Reserved for SmartTrust specified customer specific applications
'E0'h...'FE'h	Reserved for customer specific applications
'FF'h	In the delete operation 'FF'h is used to address all data objects in the PAD default storage area. In Annex A 'FF'h is used as an endmark of data storage object string.

Table 3 - Description of tag values.

<sup>3</sup> For \*PAD implementations on WIB 2.0 cards and later, tag-dependent storage areas are made mandatory.



The range '01'h-'0F'h is reserved for system values. If any of these tag are used they MUST contain the below values. There is no guarantee that the values are present.

Tag (Hex)	Content
01	IMEI
02	ICCID in format as in EF_ICCID
03	ICCID in plaintext format
04	MSISDN in human readable format, e.g. 0046701234567
05	MSISDN in format as in EF_MSISDN

Table 4 – System Tag Values

### PAD default storage area

The default memory area where the privileged application data is stored is called the PAD default storage area.

The plug-in can be used in default format where the literal plug-in name \*PAD is used. It can also be used in alternate format where a suffix is added as in \*PAD03. In this case, an alternate PAD default storage area 3 shall be used

The total amount of available memory for the privileged application data stored in the PAD default storage area(s) shall be configurable when creating the plug-in. It is an implementation issue if the size of a PAD default storage area can be even larger than the guaranteed size.

The PAD default storage area is considered as internal to \*PAD. This means that it shall not be modified in other ways than by using the \*PAD plug-in.

### Tag-dependent storage areas

In addition to the PAD default storage area a PAD may<sup>4</sup> support tag-dependent storage areas. A PAD implementation supporting tag-dependent storage areas shall use dedicated storage areas separated from the PAD default storage area for certain data object tag ranges. The mapping between tags and assigned storage areas shall be configurable.

---

<sup>4</sup> For \*PAD implementations on WIB 2.0 cards and later, tag-dependent storage areas are made mandatory.



A PAD supporting tag-dependent storage areas shall use the configuration data to select storage area depending on the tag value supplied in the input of the plug-in call.

Tag-dependent storage areas is applicable only if the plug-in call uses the plug-in name “\*PAD”. If a storage area has been configured for the tag value supplied in the input of the plug-in call the operation shall be executed using that storage area. If no storage area has been configured for the tag value supplied in the input of the plug-in call the operation shall be executed using the PAD default storage area.

It shall also be possible to configure a storage area as read-only, meaning that only read operations shall be allowed when accessing the storage area.

Tag-dependent storage areas have some restrictions compared to the PAD default storage area. It is not possible to dynamically allocate or release the physical memory used by a tag-dependent storage area. This means that the operations create and delete are not allowed for tags using tag-dependent storage areas. It also means that the size of a tag-dependent storage area can not be changed.

The maximum number of tag-dependent storage area configuration items shall be configurable when creating the plug-in. A \*PAD implementation shall support that the mapping between tags and storage areas is reconfigured at any time.

### General Input Parameter Description

The below table shows how the general input for \*PAD is structured.

Parameter	Description	Length
Operation type	This field identifies the type of operation that is requested. The possible values are the following. "C" = '43'h, Create "D" = '44'h, Delete "R" = '52'h, Read "U" = '55'h, Update	1
Data Object tag		1
Operation type dependent data	This field depends on the operation according to the below. Create: Storage length Delete: Not used Read: Not used Update: Data object (value to be stored)	A

Table 5 - General Input Values



The following sections explain the details of each of the operations: Create, Update, Read and Delete.

### Create Operation

The purpose of the create operation is to reserve a defined amount of memory for the data object to ensure the available storage space and to minimise the need of de-fragmentation procedures.

The create operation is able to create a data storage object to the PAD default storage area. It is possible to create one data storage object with one Plug-In Wib command. The data storage object is named with a tag value in the range '01'h...'FE'h.

The create operation is not be allowed for tags that have been assigned a tag-dependent storage area.

Input for create operation includes the Tag of data object and the length of storage space. The result of create procedure is returned to the defined output variable.

New data storage object is created if a tag-dependent storage area has not been configured for the tag and the requested tag does not already exist in the PAD default storage area and if there is enough space available in the PAD default storage area. The length of storage space is reserved according to the given input value. Output value "create executed successfully" is stored in the output variable.

Data storage object is not created if a tag-dependent storage area has been configured for the tag or if the tag already exists in the PAD default storage area or if there is not enough space available for the new data storage object

### Input to Create Operation

When calling the create operation, the Input of the Plug-In Wib command is coded according to the below table.

Parameter	Description	Length
Operation type	"C" = '43'h, Create	1
Data Object tag	'01'h - 'FE'h	1
Length of storage space <sup>5</sup>	'00'h - 'FE'h or '0000'h - '1FFE'h	1 or 2

Table 6 - Input to Create Operation

Document number: 90-292  
Revision: 1.4 2010-02-04

<sup>5</sup> Length on two bytes is supported by \*PAD implementations on WIB 2.0 cards and later.



### Output of create operation

As a result for the create operation one of the following output values is stored in the output variable.

Value	Description
'00'h	Create executed successfully
'01'h	Not enough memory available in the PAD default storage area
'02'h	Data storage object tag is already reserved in the PAD default storage area
'03'h	Operation not allowed
'04'h...'FF'h	Reserved for Future Use

Table 7 - Output from Create Operation

### WIG WML Example of Create Operation

The following creates a storage space of 100 bytes for a data object with tag value 'E0'h in \*PAD. Note that the example thereafter uses the convert command to convert the result to a displayable format. This command is available in Wib 1.3 and later and is only used for cosmetic reasons for the example.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
"http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card>
    <p>
      Create 100 byte storage space for tag E0 in *PAD
      <!-- Create storage -->
      <plugin name="*PAD" params="C\xE0\x64" destvar="Result"/>
      <convert srcvar="Result" destvar="DispResult" type="bin-to-hexbin"/>
      $(DispResult)
    </p>
  </card>
</wml>
```

### Update Operation

Update operation is used to update a data object. It is possible to update one data object with one Plug-In Wib command. The data object is identified with the tag in the plug-in input.

If no tag-dependent storage area has been configured for the tag in the plug-in input and the corresponding data storage object does not exist in the PAD default storage area, a new data storage object will be created in the PAD default storage area according to the given input parameters.

The input for update operation includes a data object tag and a data object to be stored.

The result of update operation is returned to the defined output variable.



### **PAD default storage area**

The update operation for cases where no tag-dependent storage area has been configured for the tag in the plug-in input is defined in this section.

If the requested data storage object exists in the PAD default storage area and the new data object value fits to the storage space, data storage object is updated with the given data object. Output value "Update executed successfully" is stored in the defined output variable.

If the requested data storage object exists in the PAD default storage area, but the length of storage space is too short for the new data object value, the old data storage object is deleted. A new data storage object is created according to the given input values (tag, length of data object and data object) if there is enough space available in the PAD default storage area. The length of new storage space is set to be equal with the length of data object in the plug-in call.

After creation the data object in the plug-in input is stored in the new data storage object and output value "Update executed successfully" is returned to the output variable.

If there is not enough space available for the new data storage object, result value "No memory space available" is returned to the output variable.

If the requested data storage object does not already exist in the PAD default storage area, but there is enough space available, a new data storage object is created and updated with the given data object as described above. Output value "Update executed successfully" is stored in the defined output variable.

### **Tag-dependent storage area**

The update operation for cases a tag-dependent storage area has been configured for the tag in the plug-in input is defined in this section.

If a tag-dependent read-only storage area has been configured for the tag in the plug-in input, the update operation shall not be allowed and "Operation not allowed" shall be stored in the output variable.

If the new data object value fits in the tag-dependent storage area, the given data object is stored in the storage area from the beginning of the area. If the size of the tag-dependent storage area is larger than the size of the given data object the remaining part of the storage area is unchanged. Output value "Update executed successfully" is stored in the defined output variable.

If the size of the tag-dependent storage area is too small for the new data storage object, result value "No memory space available" is returned to the output variable.

If a tag-dependent storage area has been configured for the tag in the plug-in input but the configuration data does not identify a valid storage area according to the implemented tag-dependent storage area configuration method "No memory space available" shall be stored in the output variable.



### Input to Update Operation

When calling update operation the Input of plug-In Wib command is coded according to the below.

Parameter	Description	Length
Operation type	"U" = '55'h, Update	1
Data object tag	'01'h – 'FE'h	1
Data object	Octet string	X

### Output of Update Operation

As a result for an update operation one of the following output values is stored in the output variable.

Value	Description
'00'h	Update executed successfully
'01'h	No memory space available
'03'h	Operation not allowed
'02'h, '04'h... 'FF'h	Reserved for Future Use



### WIG WML Example of Update Operation

The following sets the value of the data object with tag 'E0'h to "Monday" and 'E1'h to "27".

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
"http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card>
    <p>
      Set the value of E0 to Monday and E1 to 27
      <!-- Update storage -->
      <plugin name="*PAD" params="U\xE0Monday" destvar="Result"/>
      <plugin name="*PAD" params="U\xE127" destvar="Result"/>
    </p>
  </card>
</wml>
```

### Read operation

The read operation is used to read a data object. It is possible to read one data object with one Plug-In Wib command. The data object is identified with the tag value.

The input for read operation includes the tag of requested data object. The data object is returned to the output variable.

If a tag-dependent storage area has been configured for the tag in the plug-in input, the entire contents of the tag-dependent storage area is stored in the output variable. Note that an Update operation may update only the beginning of a tag-dependent storage area, while the size of the data returned by a Read operation using a tag for which a tag-dependent storage area has been configured will always be the same as the size of the storage area.

An empty value is stored in the output variable if the requested data object doesn't exist in the PAD default storage area and no tag-dependent storage area has been configured for the tag.

If the requested data object exists in the PAD default storage area and the data object is not corrupted ( $LD \leq LS$ ), the data object is stored in the defined output variable.

If the data object is corrupted ( $LD > LS$ ), it is updated with an empty value ( $LD = 0$ ) and an empty output value (length is zero) is returned to the output variable.

If a tag-dependent storage area has been configured for the tag in the plug-in input but the configuration data does not identify a valid storage area according to the implemented tag-dependent storage area configuration method an empty value shall be stored in the output variable.



## Input to Read Operation

Parameter	Description	Length
Operation type	"R" = '52'h, Read	1
Data Object tag	'01'h – 'FE'h	1

## Output of Read Operation

The requested data object is stored in the output variable.

If no tag-dependent storage area has been configured for the tag and the requested data object does not exist in the PAD default storage area or the data object is corrupted, an empty (length is zero) output value is stored in the output variable.

## WIG WML Example of Read Operation

The following read the value of the data object with tag 'E0'h and 'E1'h and display the results.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
"http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
  <card>
    <p>
      Read the value of E0 and E1
      <!-- Read data -->
      <plugin name="*PAD" params="R\xE0" destvar="Value1"/>
      <plugin name="*PAD" params="R\xE1" destvar="Value2"/>
      I read $(Value1) - $(Value2)
    </p>
  </card>
</wml>
```

## Delete operation

The purpose of the delete operation is to delete the data object and free the storage space which is reserved for the data object. Alternatively the whole PAD default storage area can be emptied with one delete operation.

Delete operation deletes one data storage object or all data storage objects in the PAD default storage area. Output of delete operation is stored in the output variable defined in the Plug-In Wib command.

The delete operation shall not be allowed for tags that have been assigned a tag-dependent storage area.

The input for delete operation includes only the tag of data storage object to be deleted. Tag value in the range '01'h... 'FE'h defines the data storage object to be deleted. Tag value 'FF'h means that all data storage objects in the PAD default storage area are deleted.

Result value of the delete operation is returned to the output variable.



When calling delete operation the plug-in will first check if a tag-dependent storage area has been configured for the tag. If so, the operation is not allowed and "Operation not allowed" is returned in the output variable. If not, the plug-in shall check if the tag value is 'FF'h in the input. If yes, all data storage objects in the PAD default storage area are deleted and the output value "Delete executed successfully" is returned to the output variable.

If the tag value is in the range '01'h...'FE'h and no tag-dependent storage area has been configured for the tag, the plug-in will check if the requested data storage object exist in the PAD default storage area. If it exists, the requested data storage object is deleted and the output value "Delete executed successfully" is returned to the output variable. If the requested data storage object doesn't exist in the PAD default storage area and no tag-dependent storage area has been configured for the tag, output value "Delete executed successfully" is returned to the output variable.

### Input to Delete Operation

When calling delete operation the Input of the Plug-In Wib command is coded according to the following.

Parameter	Description	Length
Operation type	"D" = '44'h, Delete	1
Data object tag	'01'h - 'FE'h or 'FF'h	1

### Output of Delete Operation

As a result for a delete operation the following output value is stored in the output variable.

Value	Description
'00'h	Delete executed successfully
'03'h	Operation not allowed
'01'h, '02'h, '04'h...'FF'h	Reserved for Future Use



### **WIG WML Example of Delete Operation**

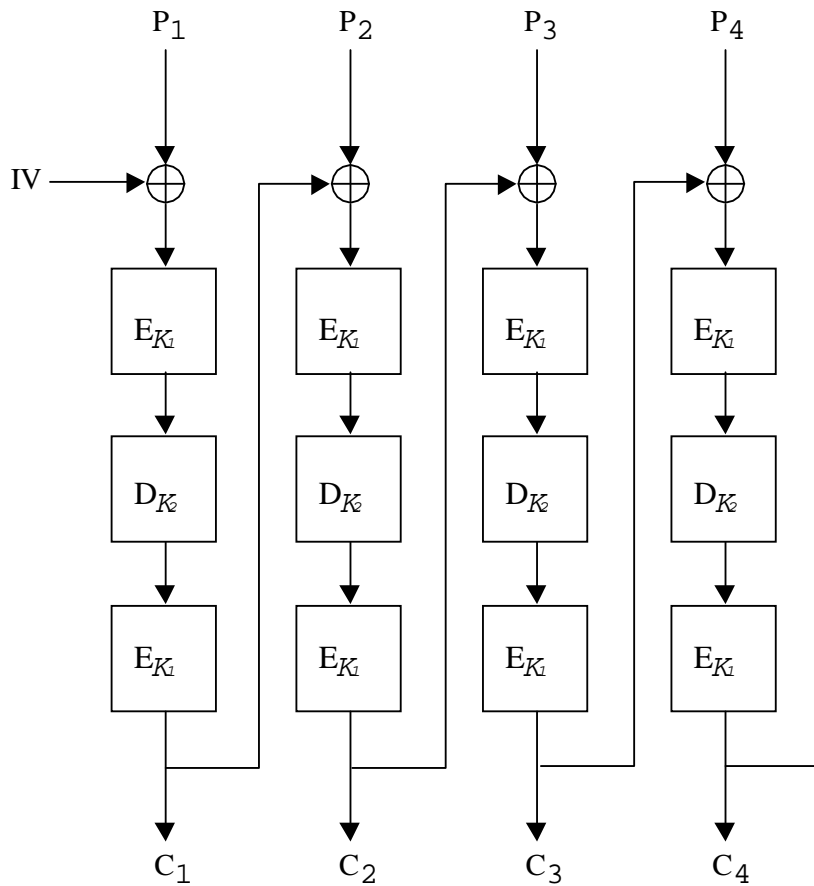
The following deletes the tags 'E0'h and 'E1'h from PAD.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE wml PUBLIC "-//SmartTrust//DTD WIG-WML 4.0//EN"
"http://www.smarttrust.com/DTD/WIG-WML4.0.dtd">
<wml>
<card>
<p>
Delete E0 and E1
<!-- Delete storage -->
<plugin name="*PAD" params="D\xE0" destvar="Value1"/>
<plugin name="*PAD" params="D\xE1" destvar="Value2"/>
$(Value1) - $(Value2)
</p>
</card>
</wml>
```

## Appendix A: Triple DES modes

### A.1 Triple encryption (TDEA\_ENCR)

The figure below illustrates DES triple encryption in outer CBC mode using two keys in encrypt-decrypt-encrypt (EDE) operation.

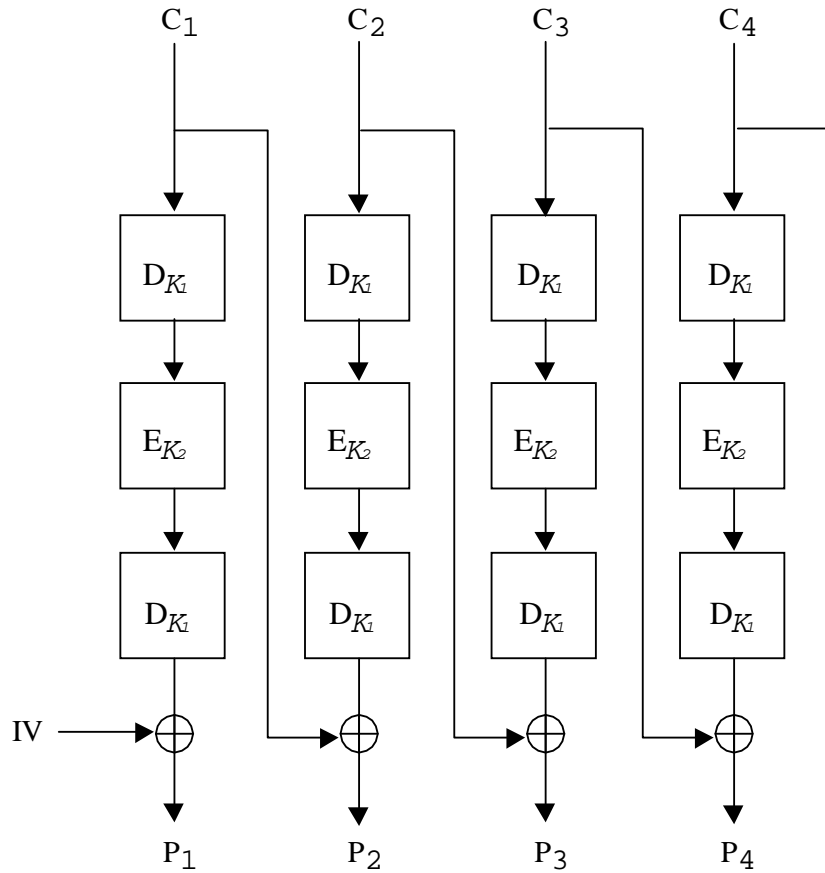


ECB operation is achieved by removing the ciphertext feedback and the IV.

See [DEA] for details regarding the DES algorithm and [MODES] for details regarding the CBC and ECB cipher mode.

### A.2 Triple decryption (TDEA\_DECR)

The figure below illustrates DES triple decryption in outer CBC mode using two keys in decrypt-encrypt-decrypt (DED) operation.



ECB operation is achieved by removing the ciphertext feedback and the IV.

See [DEA] for details regarding the DES algorithm and [MODES] for details regarding the CBC and ECB cipher mode.