



# MULTOS Standard C API

MAO-DOC-TEC-016 v1.02

This Agreement is made between MAOSCO Limited, a company incorporated under the laws of England (registered number 3290642) whose registered office is at St Andrews House, Kelvin Close, The Links, Brichwood, Warrington, WA3 7PB, England (MAOSCO), a subsidiary of Bamboo Holdings, a Cayman Islands company, dba StepNexus (StepNexus) and the Licensee.

IT IS AGREED as follows:

## 1. GRANT AND EXTENT OF LICENCE

(1) In consideration of the mutual covenants and undertakings contained in this Agreement and subject to the terms of this Agreement, MAOSCO grants to the Licensee and each of its Affiliates (for so long as it remains an Affiliate of the Licensee), and the Licensee accepts from MAOSCO, a non-exclusive, world-wide licence to use the Licensed Rights and the MULTOS Developer Documents for the sole purpose of developing, using, exploiting, improving and maintaining one or more MULTOS Applications and tools for the development of MULTOS Applications. For the avoidance of doubt, neither the Licensee nor any of its Affiliates shall be entitled, under this Agreement, to use the Licensed Rights and the MULTOS Developer Documents (a) to develop MULTOS Implementations or (b) for any purpose other than as specifically authorised under this subclause.

(2) The rights granted by MAOSCO to Affiliates of the Licensee under subclause (1) shall only apply to those Affiliates of the Licensee whose name and principal place of business have been notified in writing, physically or electronically, to MAOSCO (each an "Authorised Affiliate"). The Licensee undertakes to ensure that each Authorised Affiliate is aware of the extent of the Licensee's rights and the Licensee's obligations under this Agreement and shall procure that each Authorised Affiliate observes those rights and obligations. The Licensee shall assume responsibility for any breach by an Authorised Affiliate of the terms of this Agreement and any such breach shall be deemed to be a breach by the Licensee.

(3) Other than the Licensed Rights, any Intellectual Property subsisting in any MULTOS Application developed by the Licensee or any of its Affiliates pursuant to subclause (1) shall, as between MAOSCO and the Licensee, vest in the Licensee ("Licensee MULTOS Intellectual Property"). The parties acknowledge that the MULTOS Developer Documents are intended to be an industry standard. The Licensee covenants not to sue, and shall procure that its Affiliates shall not sue, StepNexus, MAOSCO (or any of its sublicensees) with respect to the use, for any purpose, of the Licensee MULTOS Intellectual Property in relation to any MULTOS Application or MULTOS Implementation to the extent that the Licensee MULTOS Intellectual Property is comprised or incorporated in the MULTOS Developer Documents.

## 2. CRYPTOGRAPHIC CERTIFICATION

(1) The Licensee acknowledges that:

(a) the MULTOS Card issuer will be required to obtain MULTOS Application load and MULTOS Application delete certificates from MAOSCO and to pay to MAOSCO a fee (to be determined by MAOSCO) in respect of each loading onto, and each deletion from, a MULTOS Card of a MULTOS Application although the MULTOS Applications will not be required to be submitted to MAOSCO;

(b) the cryptographic certification process enables the MULTOS Card issuer to control which MULTOS Applications are loaded onto its MULTOS Cards;

(c) the cryptographic certificate may also determine whether MULTOS Applications shall be able to call certain cryptographic functions from the relevant MULTOS Implementation;

(d) the cryptographic certification process is not a quality control process for MULTOS Applications and that it will remain the MULTOS Card issuer's responsibility to ensure that (i) MULTOS Applications meet its requirements and quality standards and (ii) any territorial specific commercial cryptographic licensing requirements have been fulfilled;

(e) legislation in certain jurisdictions restricts the use of some cryptographic functions and, therefore, MULTOS Card issuers and the Licensee may be required to demonstrate to MAOSCO that they have received approval from the appropriate authorities before access to these cryptographic functions will be granted; and

(f) the MULTOS Card issuer shall contact the MULTOS security manager (details of whom are published on the MULTOS website on <http://www.MULTOS.com>), in order to make arrangements for Cryptographic Certification.

(2) MAOSCO shall have the right to subcontract the performance of Cryptographic Certification to StepNexus or an Affiliate of StepNexus or any other person.

## 3. LICENSEE'S UNDERTAKINGS

(1) The Licensee undertakes to MAOSCO that it shall:

(a) not alter or modify the whole or any part of the MULTOS Developer Documents; and

(b) not alter, obscure, remove or interfere with all or any, and shall reproduce on all authorised copies all, of the trade marks, trade names, markings or notices affixed to or contained in any part of the MULTOS Developer Documents and shall mark all authorised copies of the MULTOS Developer Documents with those notices or marks or both as may be required by law;

(c) not to disclose all or part of the MULTOS Developer Documents except to the extent expressly permitted by this Agreement.

(2) The Licensee agrees that the use of all copies of the MULTOS Developer Documents is subject to the terms of this Agreement.

(3) The Licensee shall comply with the Specification Management Procedures.

## 4. MULTOS DEVELOPER DOCUMENTS AND TECHNICAL ASSISTANCE

(1) MAOSCO undertakes to give the Licensee access via website to the MULTOS Developer Documents, with a right to download copies of the MULTOS Developer Documents, for the purposes set out in clause 1(1) as soon as reasonably practicable after MAOSCO has received electronic notification of the Licensee's acceptance of this Agreement.

(2) MAOSCO shall comply with the Specification Management Procedures.

(3) MAOSCO shall, at the reasonable request of the Licensee from time to time, use its reasonable endeavours to assist the Licensee in dealing with any technical problems which have arisen in the Licensee's use of the MULTOS Developer Documents insofar as MAOSCO is able to do so (including, but not limited to, availability of personnel). Such assistance will be provided by e-mail or website only.

(4) If the Licensee requires more assistance than can be provided under clause 4(3), the Licensee should submit a written request for assistance to MAOSCO (by letter, fax or e-mail). The Request should provide details of the nature of the assistance required. If MAOSCO agrees to accept the request MAOSCO's assistance shall be chargeable at its standard rates of charge applicable from time to time. The Licensee shall also reimburse MAOSCO all reasonable expenses incurred by MAOSCO in providing that assistance. Any request for assistance from the Licensee to MAOSCO shall be made only by the designated Licensee's Nominated Representative (or such other person as the Licensee has notified to MAOSCO in accordance with clause 10) to the MULTOS Representative at MAOSCO (details of whom are published on the MULTOS website on <http://www.MULTOS.com>).

## 5. PAYMENTS BY LICENSEE TO MAOSCO

(1) The Licensee shall pay to MAOSCO, against issue of a valid VAT invoice, the amount of any VAT chargeable on a supply by MAOSCO to the Licensee and, without limiting that payment, each sum payable by the Licensee under the terms of this Agreement is exclusive of VAT (if any) and is accordingly to be construed as a reference to that sum plus any VAT in respect of it.

## 6. WARRANTY AND INFRINGEMENT

(1) Each party warrants to the other that it has all requisite corporate power and authority to enter into this Agreement and is fully capable of performing its obligations under this Agreement on the terms provided for in this Agreement.

(2) The Licensee shall promptly inform MAOSCO of any proceedings involving the validity, or any infringement or threatened infringement, of the Licensed Rights and of any unauthorised use of the Licensed Rights or the MULTOS Developer Documents coming to its notice. MAOSCO shall take all action reasonably necessary to prevent the infringement or defend proceedings for revocation or to prevent that unauthorised use, and the Licensee shall, at MAOSCO's request and expense, render all reasonable assistance in connection with that action.

(3) Subject to clause 9, the Licensee shall indemnify MAOSCO and its Affiliates from and against all and any damages, losses, costs, expenses and other liabilities awarded against or incurred by MAOSCO and/or any of its Affiliates as a result of or in connection with any claim or action arising out of or relating to any MULTOS Applications developed by or on behalf of the Licensee including, without limitation, any product liability claim ("Claim"), except to the extent that the Claim (if it is a product liability claim) arises out of any specific product feature or functionality expressly mandated by the MULTOS Developer Documents, provided that:

(a) MAOSCO promptly notifies the Licensee in writing of any Claim of which it has notice;

(b) MAOSCO does not make any admission as to liability or agree to any settlement of or compromise any Claim without the prior written consent of the Licensee which shall not be unreasonably withheld or delayed; and

(c) the Licensee may, at its request and expense, conduct all negotiations and (to the extent legally permissible) litigation, and (to the extent legally permissible) settle all litigation, arising from any Claim and MAOSCO shall, at the Licensee's request and expense, give the Licensee all reasonable assistance, at Licensee's expense, in connection with those negotiations and litigation.



(4) NO REPRESENTATION, WARRANTY OR CONDITION, EXPRESS OR IMPLIED, STATUTORY OR OTHERWISE, AS TO CONDITION, QUALITY, PERFORMANCE OR FITNESS FOR PURPOSE IS GIVEN OR ASSUMED BY MAOSCO IN RESPECT OF THE MULTOS DEVELOPER DOCUMENTS AND ALL SUCH REPRESENTATIONS, WARRANTIES AND CONDITIONS ARE EXCLUDED SAVE TO THE EXTENT THAT SUCH EXCLUSION IS PROHIBITED BY LAW.

#### 7. TERMINATION

(1) MAOSCO shall have the right, without prejudice to its other rights or remedies, to terminate this Agreement immediately by written notice to the Licensee if the Licensee:

- (a) shall have failed to pay, in accordance with the terms of this Agreement, any sum due to MAOSCO under the terms of this Agreement and that sum remains unpaid for 30 days after receiving written notice from MAOSCO that it has not been paid;
- (b) is in material breach of any of its obligations under this Agreement and either that breach is incapable of remedy or the Licensee shall have failed to remedy that breach within 30 days after receiving written notice from MAOSCO requiring it to remedy that breach;
- (c) challenges the validity of, or MAOSCO's rights in, any of the Licensed Rights provided that compliance by the Licensee of its obligations under clause 6(2) shall not constitute such a challenge; or
- (d) becomes insolvent or an order is made or a resolution passed for its liquidation, administration, winding-up or dissolution (otherwise than for the purposes of a solvent amalgamation or reconstruction) or an administrative or other receiver, manager, liquidator, administrator, trustee or similar officer is appointed over all or any substantial part of its assets or it enters into or proposes any composition or arrangement with its creditors generally or anything analogous to the foregoing occurs in any applicable jurisdiction.

(2) The Licensee shall have the right to terminate this Agreement immediately by written notice to MAOSCO.

(3) Any termination of this Agreement shall not affect any accrued rights or liabilities of either party nor shall it affect the coming into force or the continuance in force of any provision of this Agreement which is intended to come into force or continue in force on or after that termination.

#### 8. CONSEQUENCES OF TERMINATION

(1) On termination of this Agreement the Licensee shall, subject to subclause (2), immediately cease all use of the Licensed Rights and the MULTOS Developer Documents.

(2) Within 30 days after the termination of this Agreement the Licensee shall return to MAOSCO or, at MAOSCO's direction, destroy) all documents and materials (whether in tangible or electronic form) supplied by MAOSCO to the Licensee pursuant to this Agreement and/or which contain or disclose any information contained in or relating to the MULTOS Developer Documents) and all copies then in the possession or under the control of the Licensee and shall use its reasonable endeavours to return to MAOSCO any documents and other materials supplied by the Licensee to a third party.

(3) The obligations of the Licensee under clause 1(3) shall survive the termination of this Agreement for whatever reason.

#### 9. LIABILITY

(1) The aggregate liability of MAOSCO and its Affiliates to the Licensee under this Agreement, whether arising under any indemnity or from negligence, breach of contract or otherwise, shall not exceed in aggregate £10,000.

(2) The aggregate liability of the Licensee to MAOSCO and its Affiliates under this Agreement, whether arising under this indemnity or from negligence, breach of contract or otherwise, shall not exceed in aggregate £10,000.

(3) Each party and its Affiliates does not limit its liability for death or personal injury arising from it negligence or that of its employees, agents or sub-contractors.

(4) Each party and its Affiliates shall not be liable to the other party for any indirect or consequential loss or damages including, without limitation, loss of business or profits, whether arising under any indemnity or from negligence, breach of contract or otherwise.

(5) MAOSCO and its Affiliates shall not be liable to the Licensee for any loss or damages, howsoever arising, in relation to any Cryptographic Technology that is not owned by MAOSCO or StepNexus including, without limitation, ECC, RSA, AES and DES encryption technology.

(6) The limitations of liability contained in this clause shall not affect MAOSCO's right to be paid by the Licensee, and the obligations of the Licensee to pay to MAOSCO, the amounts payable to MAOSCO under this Agreement when due.

#### 10. NOTICES

Any notice or other document to be served under this Agreement may be delivered or sent by prepaid first class registered airmail or facsimile process to the party to be served as follows:

(a) to MAOSCO at St Andrews House, Kelvin Close, The Links, Birchwood, Warrington, WA3 7PB;  
fax no: +44 (0) 1925 882051; marked for the attention of the Company Secretary,

(b) to Licensee at its address and fax number and marked for the attention of the Licensee's Nominated Representative, all identified in the Developer's Licence Agreement electronic registration form, or at any other address or to any other facsimile number or addressee as it may have notified to the other in accordance with this clause.

#### 11. ASSIGNMENT

(1) The Licensee shall not assign, sub-license, transfer, mortgage, charge or part with any of its rights, or sub-contract any of its duties or obligations, under this Agreement provided that the Licensee shall be entitled to sub-license (subject to the terms of this Agreement):

(a) the Licensed Rights and the MULTOS Developer Documents to a contractor, for such contractor to develop MULTOS Applications for the Licensee under the control of the Licensee and for no other purpose; and

(b) the Licensed Rights to customers of the Licensee as end users of the Licensee's MULTOS Applications to the extent that the Licensed Rights subsist in those MULTOS Applications.

(2) Nothing in this Agreement shall preclude the Licensee from entering into any collaborative arrangements for the development and/or exploitation of MULTOS Applications with any person who has entered into an agreement with MAOSCO on substantially the same terms as this Agreement.

#### 12. GENERAL

(1) A waiver (whether express or implied) by one of the parties of any of the provisions of this Agreement or of any breach of or default by the other party in performing any of those provisions shall not constitute a continuing waiver and that waiver shall not prevent the waiving party from subsequently enforcing any of the provisions of this Agreement not waived or from acting on any subsequent breach of or default by the other party under any of the provisions of this Agreement.

(2) Except in relation to the Schedules, any amendment, waiver or variation of this Agreement shall not be binding on the parties unless set out in writing, expressed to amend this Agreement and signed by or on behalf of each of the parties.

(3) The invalidity, illegality or unenforceability of any of the provisions of this Agreement shall not affect the validity, legality and enforceability of the remaining provisions of this Agreement.

(4) This Agreement and the documents referred to in it contain the whole agreement between the parties relating to the subject matter of this Agreement and supersede all previous agreements between the parties relating to that subject matter. In entering into this Agreement, neither party may rely on any representation, warranty, collateral contract or other assurance (except those set out in this Agreement and the documents referred to in it, and any confidentiality agreement in relation to the MULTOS Developer Documents) made by or on behalf of the other party before the signing of this Agreement and each party waives all rights and remedies which, but for this subclause, might otherwise be available to it in respect of any such representation, warranty, collateral contract or other assurance provided that nothing in this subclause shall limit or exclude any liability for wilful misrepresentation or fraud.

(5) This Agreement is governed by and shall be construed in accordance with English law. Each party submits to the exclusive jurisdiction of the English courts for all purposes relating to this Agreement.

(6) This Agreement will come into effect when MAOSCO has received electronic notification from the Licensee that the Licensee has accepted the terms of this Agreement

#### 13. INTERPRETATION

In this Agreement:

"Affiliate" means, in relation to any person, any other person that, directly or indirectly through one or more intermediaries, controls, or is controlled by, or is under common control with, such person. The term "control" (including, with its correlative meanings, "controlled by" and "under common control with") means possession, directly or indirectly, of power to direct or cause the direction of management or policies (whether through ownership of securities or partnership or other ownership interests, by contract or otherwise), and provided that if any person:

- (a) beneficially owns voting securities of another person conferring more than 50 per cent. of the votes eligible to be cast in the election of directors or any similar matter; or
- (b) has the right (by contract or otherwise) to appoint or remove directors (or members of a governing body having functions similar to a board of directors) representing more than 50 per cent. of the votes exercisable by all the directors (or persons having similar functions) of another person, then the first person is conclusively deemed to control the other person for the purposes of this definition.

"Cryptographic Certification" means the process described in clause 2(1).

"Cryptographic Technology" means those mechanisms that are functionally designed (a) to preserve the confidentiality, authenticity or integrity of digitally stored and/or transmitted data or (b) to authenticate information as part of a security service through the use of algorithms that involve secret parameters, commonly known as keys.

"IC Card" means an integrated circuit card.

"Intellectual Property" means patents, patent applications, patented and unpatented inventions, design rights, copyrights (including, without limitation, rights in computer software), topography rights, know-how and trade secret rights, and all other intellectual property rights and the rights or forms of protection of a similar nature or having equivalent or similar effect to any of these rights which may subsist anywhere in the world (whether or not any of these rights is registered and including, without limitation, applications for registration of, and rights to apply for, any such rights).

"Licensed Rights" means all Intellectual Property in and to the MULTOS Developer Documents.

"MULTOS Application" means a program written to run, in a compiled, interpretative, executable or any other form, on or interface with, any MULTOS Implementation or which processes data to be used on or in conjunction with any MULTOS Implementation.

"MULTOS Card" means an IC Card which incorporates a MULTOS Implementation.

"MULTOS Developer Documents" means the documents listed in Schedule 1, as amended by MAOSCO from time to time.

"MULTOS Implementation" means an operating system (including the silicon platform) which (a) supports multiple applications, (b) conforms, or is intended to conform, to the MULTOS Specification and (c) uses Cryptographic Certification.

"MULTOS Specification" means the specifications, licensed by MAOSCO, defining the functional and security requirements and the application programming interface for multi-application architectures, as amended by MAOSCO from time to time.

"StepNexus" means Bamboo Holdings, a Cayman Islands company, dba StepNexus.

"Specification Management Procedures" means the procedures set out in Schedule 2 or any other procedures which MAOSCO may notify the Licensee in writing from time to time.

"Update" means all and any updates or new releases or new versions of the MULTOS Developer Documents.

"VAT" means any value added tax chargeable by virtue of any enactment in a territory introduced by reason of the Sixth Council Directive 77/388/EC or its predecessors or similar subsequent legislation or any similar tax.

## SCHEDULE 1

### MULTOS Developer Documents

MULTOS Developers Guide (MDG)  
MULTOS Developers Reference Manual (MDRM)  
Guide to Loading and Deleting MULTOS Applications (GLDA)  
Guide to Generating Application Load Units (GALU)  
MULTOS Implementation Report (MIR)  
Developer Card: User Guidelines (DCUG)  
MULTOS Standard C API (CAPI)

Plus other documents made available from time to time by MAOSCO.

## SCHEDULE 2

### Specification Management Procedures

1. MAOSCO shall, as soon as practicable after they become available, make available to the Licensee all and any Updates which are issued generally by MAOSCO to other persons who have been licensed by MAOSCO to use the MULTOS Developer Documents.
2. The Licensee shall ensure that within 30 days after access to any Updates is permitted by MAOSCO to the Licensee, all documents or materials (whether in tangible or electronic form) supplied by MAOSCO pursuant to this Agreement which contain or reference that part of the MULTOS Developer Documents which has been substituted or amended by any Update supplied by MAOSCO are returned to MAOSCO (or, at MAOSCO's direction, destroyed).
3. The details of the person responsible on behalf of MAOSCO for the Specification Management Procedures as set out in this Schedule are published on the MULTOS website on <http://www.MULTOS.com>.



## ***Copyright***

© Copyright 2005 – 2006 MAOSCO Limited. This document contains confidential and proprietary information. No part of this document may be reproduced, published or disclosed in whole or part, by any means: mechanical, electronic, photocopying, recording or otherwise without the prior written permission of MAOSCO Limited.

## ***Trademarks***

MULTOS is a trademark of MAOSCO Limited.

All other trademarks, trade names or company names referenced herein are used for identification only and are the property of their respective owners

## ***Published by***

MAOSCO Limited  
St. Andrews House  
Kelvin Close  
The Links  
Birchwood  
WARRINGTON  
Cheshire  
WA3 7PB  
United Kingdom  
.

## ***General Enquiries***

Email: [info@multos.com](mailto:info@multos.com)  
Tel: +44 (0) 1925 882 050  
Fax: +44 (0) 1925 882 051  
Web: <http://www.multos.com>

## ***Technical Enquiries***

Email: [dev.support@multos.com](mailto:dev.support@multos.com)  
Web: <http://www.multos.com>

## ***Document References***

All references to other available documentation is followed by the document acronym in square [ ] brackets. Details of the content of these documents can be found in the MULTOS Guide to Documentation, and the latest versions are always available from the MULTOS web site (<http://www.multos.com>).



## *Data References*

All references to MULTOS data can be cross-referenced to the MULTOS Data Dictionary.

## *Version Information*

0.1	September 2003	pre-release draft for public comment
1.00	June 2004	First Released Version
1.01	November 2005	Trademark change
1.02	November 2006	Reformatted



## Contents

1	Introduction .....	1
2	Exclusions .....	2
3	Notes .....	3
3.1	Data Types .....	3
3.2	Conventions .....	3
3.3	System Variables .....	3
4	Function Prototypes .....	5
4.1	multosBlockAdd .....	5
4.2	multosBlockAnd .....	5
4.3	multosBlockCompare .....	5
4.4	multosBlockClear .....	6
4.5	multosBlockCopy .....	6
4.6	multosBlockDecrement .....	7
4.7	multosBlockDivide .....	7
4.8	multosBlockIncrement .....	7
4.9	multosBlockInvert .....	8
4.10	multosBlockLookup .....	8
4.11	multosBlockMultiply .....	8
4.12	multosBlockOr .....	9
4.13	multosBlockShiftLeft .....	9
4.14	multosBlockShiftRight .....	9
4.15	multosBlockSubtract .....	10
4.16	multosBlockTestZero .....	10
4.17	multosBlockXor .....	10
4.18	multosCallCodelet .....	11
4.19	multosCallExtensionPrimitive .....	11
4.20	multosCardBlock .....	12
4.21	multosCheckCase .....	12
4.22	multosChecksum .....	12
4.23	multosControlAutoResetWWT .....	12
4.24	multosDelegate .....	13
4.25	multosDESECBDecipher .....	13
4.26	multosDESECBEncipher .....	13
4.27	multosExchangeData .....	14
4.28	multosExit .....	14
4.29	multosExitLa .....	14
4.30	multosExitSW .....	14
4.31	multosExitSWLa .....	14
4.32	multosGenerateAsymmetricHash .....	15
4.33	multosGenerateAsymmetricHashIV .....	15
4.34	multosGenerateDESCBCSignature .....	15
4.35	multosGenerateTripleDESCBCSignature .....	16
4.36	multosGetDelegatorAID .....	16
4.37	multosGetDIRFileRecord .....	17
4.38	multosGetFileControlInformation .....	17
4.39	multosGetManufacturerData .....	17
4.40	multosGetMemoryReliability .....	18



4.41	multosGetMultosData .....	18
4.42	multosGetRandomNumber .....	18
4.43	multosModularExponentiation .....	19
4.44	multosModularExponentiationCRT .....	19
4.45	multosModularMultiplication .....	19
4.46	multosModularReduction .....	20
4.47	multosQueryChannel .....	20
4.48	multosQueryCodelet .....	20
4.49	multosQueryPrimitive .....	21
4.50	multosResetSessionData .....	21
4.51	multosResetWWT .....	21
4.52	multosReturnFromCodelet .....	21
4.53	multosSetATRFileRecord .....	22
4.54	multosSetATRHistoricalCharacters .....	22
4.55	multosSetTransactionProtection .....	22
4.56	multosSetSelectSW .....	23
4.57	multosSHA1 .....	23
5	Function Listing by Type .....	24
5.1	Block Manipulations .....	24
5.2	Cryptographic .....	24
5.3	System .....	25
5.4	EMV .....	26
A.	Appendix A : Biometric C API .....	27
A.1	Introduction .....	27
A.2	Constants .....	27
A.3	Data Types .....	27
A.4	Data .....	28
A.5	Function Prototypes .....	28
A.5.1	bioInit .....	28
A.5.2	bioUpdate .....	29
A.5.3	bioDoFinal .....	29
A.5.4	bioResetUnblockAndSetTryLimit .....	29
A.5.5	bioGetBioType .....	29
A.5.6	biolsInitialized .....	30
A.5.7	biolsValidated .....	30
A.5.8	bioGetVersion .....	30
A.5.9	bioGetPublicTemplateData .....	30
A.5.10	bioGetTriesRemaining .....	31
A.5.11	bioReset .....	31
A.5.12	bioInitMatch .....	31
A.5.13	bioMatch .....	32



# 1 Introduction

---

The MULTOS Standard C API is intended to standardise the syntax used by C developers writing MULTOS applications. The Standard C API will be supported by MULTOS Development Tools either directly or by using C header libraries.

The Standard C API is meant to cover the needs of most C developers however, developers may still use assembler or development tool supplied C functions not included in this document.

Only functions that provide an interface to a MULTOS instruction or primitive are included. For this reason only the API of the function is described, not the function itself. If further clarification of a particular function is required, including examples, consult the MULTOS Developers Reference Manual, [MDRM].

The Appendices to this document contain industry specific C APIs developed by Application Developers and contributed to this specification. The MULTOS Consortium welcomes the submission of new C APIs to this specification from the Application Developer community.  
Paragraph



## 2 Exclusions

For the sake of clarity and simplicity only mainstream MULTOS functions are included in the Standard C API. Some less frequently used functions are excluded. APIs for the following MULTOS instructions and primitives are not included.

Instruction or Primitive	Reason for exclusion
ADDB	handled by multosBlockAdd
ADDW	handled by multosBlockAdd
BRANCH	C programming handles this implicitly
CALL	C programming handles this implicitly
CMPB	C programming handles this implicitly
CMPN	C programming handles this implicitly
CMPBW	C programming handles this implicitly
INDEX	C programming handles this implicitly
JUMP	C programming handles this implicitly
LOAD	C programming handles this implicitly
LOADA	C programming handles this implicitly
LOADI	C programming handles this implicitly
PRIMRET	API provides primitive interface
SETB	C programming handles this implicitly
SETW	C programming handles this implicitly
STACK	C programming handles this implicitly
STORE	C programming handles this implicitly
STOREI	C programming handles this implicitly
SUBB	handled by multosBlockSubtract
SUBW	handled by multosBlockSubtract
Bit Manipulate Byte	handled by included binary and unary instructions
Bit Manipulate Word	handled by included binary and unary instructions
Get Purse Type	not required by most programmers
Load CCR	not required by most programmers
Memory Copy Fixed Length	handled by multosBlockCopy
Memory Compare Fixed Length	handled by multosBlockCompare
Store CCR	not required by most programmers



## 3 Notes

---

### 3.1 Data Types

The following Data Types are used:

Data Type	Definition
BOOL	boolean (byte)
BYTE	unsigned byte (byte)
SBYTE	signed byte (byte)
WORD	unsigned word (2 bytes)
SWORD	signed word (2 bytes)
DWORD	unsigned double word (4 bytes)
SDWORD	signed double word (4 bytes)

### 3.2 Conventions

The conventions used in this document are:

- All function names start with "multos".
- The keyword "const" is used to indicate whether the parameter must be a compile-time constant (i.e. not a value held in a variable).
- All function parameters are ordered such that inputs come first and outputs come second.

### 3.3 System Variables

Type	Name	Description
BYTE	multosProtocolFlags	Protocol flags in Public, encoded as follows: MULTOS_MASK_P3_VALID, MULTOS_MASK_LC_VALID, MULTOS_MASK_LE_VALID, MULTOS_MASK_CMD_DATA_RECEIVED
BYTE	multosProtocolType	Protocol type in Public, encoded as follows: MULTOS_PROTOCOL_T0, MULTOS_PROTOCOL_T1
BYTE	multosGetResponseCLA	Get Response CLA in Public
BYTE	multosGetResponseSW1	Get Response SW1 in Public
BYTE	multosCLA	CLA in Public
BYTE	multosINS	INS in Public
BYTE	multosP1	P1 in Public
BYTE	multosP2	P2 in Public
BYTE	multosP3	P3 in Public
WORD	multosP1P2	P1P2 in Public
WORD	multosLc	Lc in Public



WORD	multosLe	Le in Public
WORD	multosLa	La in Public
BYTE	multosSW1	SW1 in Public
BYTE	multosSW2	SW2 in Public
WORD	multosSW12	SW1 and SW2 in Public



## 4 Function Prototypes

---

### 4.1 multosBlockAdd

```
void multosBlockAdd (const BYTE blockLength, BYTE *block1, BYTE *block2, const BYTE *result)
```

The parameters are:

- const BYTE *blockLength*: size of the blocks to add.
- BYTE \**block1*: address of the first block
- BYTE \**block2*: address of the second block
- const BYTE \**result*: address of the block that will hold the result of the operation

This function adds the value found in *block1* to that found in *block2* and places the sum in the block indicated in the *result* parameter. Note that *block1*, *block2* and *result* are all considered to be of size *blockLength*.

This is an interface to the instruction ADDN.

### 4.2 multosBlockAnd

```
void multosBlockAnd (const BYTE blockLength, BYTE *block1, BYTE *block 2, const BYTE *result)
```

The parameters are:

- const BYTE *blockLength*: size of the blocks to add. Both blocks must be the same size.
- BYTE \**block1*: address of the first block
- BYTE \**block2*: address of the second block
- const BYTE \**result*: address of the block that will hold the result of the operation

This function performs a logical AND using the values found in *block1* and *block2*. The result is written the location given in the *result* parameter. Note that *block1*, *block2* and *result* are all considered to be of size *blockLength*.

This is an interface to the instruction ANDN.

### 4.3 multosBlockCompare

```
BYTE multosBlockCompare (WORD blockLength, BYTE *block1, BYTE *block2)
```

The parameters are:



- WORD *blockLength*: size of the blocks to be compared. Both blocks must be the same size.
- BYTE *\*block1*: address of the first block
- BYTE *\*block2*: address of the second block

This function compares *block1* and *block2*, returns one of the following results:

- MULTOS\_BLOCK1\_GT\_BLOCK1
- MULTOS\_BLOCK2\_GT\_BLOCK1
- MULTOS\_BLOCK1\_EQ\_BLOCK2

Note that *block1* and *block2* are considered to be of size *blockLength*.

This is an interface to the primitive Memory Compare.

## 4.4 multosBlockClear

```
void multosBlockClear (const BYTE blockLength, const BYTE
    *block)
```

The parameters are:

- const BYTE *blockLength*: size of the block to clear
- const BYTE *\*block*: address of the block to be cleared

This function sets the value of each byte of the block of size *blockLength* to zero.

This is an interface to the instruction CLEARN.

## 4.5 multosBlockCopy

```
void multosBlockCopy (WORD blockLength, BYTE *blockSource, BYTE
    *blockDest)
```

The parameters are:

- WORD *blockLength*: the size of the block to copy
- BYTE *\*blockSource*: address of the source block
- BYTE *\*blockDest*: address of the destination block

This function copies data of size *blockLength* from *blockSource* to *blockDest*.

This is an interface to the primitive Memory Copy.



## 4.6 multosBlockDecrement

```
void multosBlockDecrement (const BYTE blockLength, const BYTE *block)
```

The parameters are:

- const BYTE *blockLength*: size of the block on which to perform the operation
- const BYTE *\*block*: address of the block on which to perform the operation

This function decrements the value held in *block* by one.

This is an interface to the instruction DECN.

## 4.7 multosBlockDivide

```
void multosBlockDivide (const BYTE blockLength, BYTE *numerator,  
BYTE *denominator, BYTE *quotient, BYTE *remainder, BYTE  
*status)
```

The parameters are:

- const BYTE *blockLength*: the size of all operand and result blocks
- BYTE *\*numerator*: address of the block where the numerator is held.
- BYTE *\*denominator*: address of the block where the denominator is held.
- BYTE *\*quotient*: address of the block where the quotient is to be written.
- BYTE *\*remainder*: address of the block where the remainder is to be written.
- BYTE *\*status*: address of the block where the result status is to be written.

This function divides the *numerator* by the *denominator*. The results of the operation are written to the blocks *quotient* and *remainder*. In order to flag any special circumstances, the flag *status* is set to one of the following values: MULTOS\_DIVIDE\_BY\_ZERO, MULTOS\_DIVIDE\_QUOTIENT\_ZERO, MULTOS\_DIVIDE\_OK.

This is an interface to the primitive DivideN.

## 4.8 multosBlockIncrement

```
void multosBlockIncrement (const BYTE blockLength, const BYTE *block)
```

The parameters are:

- const BYTE *blockLength*: size of the block on which to perform the operation
- const BYTE *\*block*: address of the block on which to perform the operation

This function increments the value held in *block* by one.



This is an interface to the instruction INCN

## 4.9 *multosBlockInvert*

```
void multosBlockInvert (const BYTE blockLength, const BYTE *block)
```

The parameters are:

- const BYTE *blockLength*: size of the block on which to perform the operation
- const BYTE *\*block*: address of the block on which to perform the operation

This function logically inverts the value held in *block*.

This is an interface to the instruction NOTN

## 4.10 *multosBlockLookup*

```
BOOL multosBlockLookup (BYTE value, BYTE *block, BYTE *result)
```

The parameters are:

- BYTE *value*: the value to locate
- BYTE *\*block*: address of the array to be searched.
- BYTE *\*result*: address of the byte to which the result will be written

This function locates the first occurrence of *value* within *block*. Note that this function treats the first byte of the array as indicating the total number of bytes in the array. The function returns TRUE if *value* is found and *result* is the offset within the array where *value* is first found.

This is an interface to the primitive Lookup.

## 4.11 *multosBlockMultiply*

```
void multosBlockMultiply (const BYTE blockLength, BYTE *block1, BYTE *block2, BYTE *result)
```

The parameters are:

- const BYTE *blockLength*: the size of the operands
- BYTE *\*block1*: address of the first byte of block1
- BYTE *\*block2*: address of the first byte of block2
- BYTE *\*result*: address of the first byte of result



This function multiplies the value held in *block1* by that held in *block2* and writes the result to the block result of size *blockLength* + *blockLength*.

This is an interface to the primitive MultiplyN.

#### 4.12 multosBlockOr

```
void multosBlockOr (const BYTE blockLength, BYTE *block1, BYTE
*block2, BYTE *result)
```

The parameters are:

- const BYTE *blockLength*: the size of all operand and result blocks
- BYTE \**block1*: address of block1
- BYTE \**block2*: address of block2
- BYTE \**result*: address of the first byte of block where the result is to be written.

This function performs a logical OR operation using the values in *block1* and *block2* as operands. The output of the operation is written to *result*.

This is an interface to the instruction ORN.

#### 4.13 multosBlockShiftLeft

```
void multosBlockShiftLeft (const BYTE blockLength, const BYTE
numShiftBits, BYTE *blockSource, BYTE *blockDest)
```

The parameters are:

- const BYTE *blockLength*: size of the block to shift
- const BYTE *numShiftBits*: number of bits to shift
- BYTE \**blockSource*: address of the source block
- BYTE \**blockDest*: address of the destination block

This function does left shifts the value found in *blockSource*, *numShiftBits* times and writes the result to *blockDest*.

This is an interface to the primitive Shift Left.

#### 4.14 multosBlockShiftRight

```
void multosBlockShiftRight (const BYTE blockLength, const BYTE
numShiftBits, BYTE *blockSource, BYTE *blockDest)
```

The parameters are:

const BYTE *blockLength*: size of the block to shift



const BYTE *numShiftBits*: number of bits to shift  
BYTE *\*blockSource*: source block  
BYTE *\*blockDest*: destination block

This function does right shifts the value found in *blockSource*, *numShiftBits* times and writes the result to *blockDest*.

This is an interface to the primitive Shift Right.

## 4.15 multosBlockSubtract

```
void multosBlockAdd (const BYTE blockLength, BYTE *block1, BYTE  
*block2, const BYTE *result)
```

The parameters are:

- const BYTE *blockLength*: size of the blocks to subtract. Both blocks must be the same size.
- BYTE *\*block1*: address of the first block
- BYTE *\*block2*: address of the second block
- const BYTE *\*result*: address of the block that will hold the result of the operation

This function subtracts the value found in *block1* to that found in *block2* and places the difference in the block indicated in *result*.

This is an interface to the instruction SUBN.

## 4.16 multosBlockTestZero

```
void multosBlockTestZero (const BYTE blockLength, const BYTE  
*block, BOOL *isZero)
```

The parameters are:

const BYTE *blockLength*: size of the block to test  
const BYTE *\*block*: the address of the block to test  
BOOL *\*isZero*: flag indicating if all bytes are zero

This function tests each byte in block has a value of zero. The flag *isZero* is set to TRUE if all bytes are zero, otherwise it is set to FALSE.

This is an interface to the instruction TESTN.

## 4.17 multosBlockXor

```
void multosBlockXor (const BYTE blockLength, BYTE *block1, BYTE  
*block2, BYTE *result)
```



The parameters are:

- const BYTE *blockLength*: the size of all operand and result blocks
- BYTE \**block1*: address of block1
- BYTE \**block2*: address of block2
- BYTE \**result*: address of the first byte of block where the result is to be written.

This function performs a logical XOR operation using the values in *block1* and *block2* as operands. The output of the operation is written to *result*.

This is an interface to the instruction XORN.

#### 4.18 multosCallCodelet

```
void multosCallCodelet (WORD codeletID, WORD entryAddress)
```

The parameters are:

- WORD *codeletID*: the unique identifier of the codelet being called
- WORD *entryAddress*: codelet entry point

This function invokes a codelet and executes the code starting at the entry address.

This is an interface to the primitive Call Codelet.

#### 4.19 multosCallExtensionPrimitive

```
void multosCallExtensionPrimitive (const BYTE extensionNum,  
const BYTE primTypeLo, const BYTE primTypeHi, const BYTE  
paramByte)
```

The parameters are:

- const BYTE *extensionNum*: number indicating the MULTOS implementor
- const BYTE *primTypeLo*: least significant byte of the proprietary primitive identifier as allocated by the implementor
- const BYTE *primTypeHi*: most significant byte of the proprietary primitive identifier as allocated by the implementor
- const BYTE *paramByte*: byte that may be used to pass a parameter to the proprietary primitive

This function calls a proprietary extension primitive.

This is an interface to the primitive Call Extension.



## 4.20 multosCardBlock

```
void multosCardBlock (const BYTE offsetMAChi, const BYTE  
offsetMACLo, BOOL *cardBlockedOk)
```

The parameters are:

- const BYTE *offsetMAChi*: most significant byte of the address of the CBMAC in public memory
- const BYTE *offsetMACLo*: least significant byte of the address of the CBMAC in public memory
- BOOL \**cardBlockedOk*: set to TRUE if the block is successful

This function evaluates a card block MAC and blocks the MULTOS card if the MAC is verified.

This is an interface to the primitive Card Block.

## 4.21 multosCheckCase

```
BOOL multosCheckCase (BYTE isoCase)
```

The parameter is a single byte indicating the expected ISO case of the incoming command.

This function (a) instructs the operating system as to how to interpret the APDU received and (b) checks the ISO case of the received command. It returns TRUE if the case is recognised.

This is an interface to the primitive Check Case.

## 4.22 multosChecksum

```
DWORD multosChecksum (WORD blockLength, BYTE *block)
```

The parameters are:

- WORD *blockLength*: the length of the block to use as input to the checksum algorithm
- BYTE \**block*: the address of the first byte of the input block

The function generates a four-byte checksum of block and returns the four-byte value.

This is an interface to the primitive Checksum.

## 4.23 multosControlAutoResetWWT

```
void multosControlAutoResetWWT (BOOL disable)
```

The parameter is Boolean value set to TRUE if the auto reset WWT request function is to be disabled.

This function enables or disables the automatic generation of WWT extension messages.



This is an interface to the primitive Control Auto Reset WWT.

#### 4.24 *multosDelegate*

```
void multosDelegate (BYTE *aid)
```

The parameter is a pointer to the AID of the application that will be delegated to.

This function delegates execution to an application with and AID of aid.

This is an interface to the primitive Delegate.

#### 4.25 *multosDESECBDecipher*

```
void multosDESECBDecipher (BYTE *cipherText, BYTE *plainText,  
BYTE key[8])
```

The parameters are:

- BYTE *\*cipherText*: address of the ciphertext input
- BYTE *\*plaintext*: address to which to write the plaintext
- BYTE *key[8]*: the 8-byte DES key

This function performs a DES ECB decipher on eight bytes of *cipherText* using key into eight bytes of *plainText*.

This is an interface to the primitive DES ECB Decipher.

#### 4.26 *multosDESECBEncipher*

```
void multosDESECBEncipher (BYTE *plainText, BYTE *cipherText,  
BYTE key[8])
```

The parameters are:

- BYTE *\*plaintext*: address of the plaintext input
- BYTE *\*cipherText*: address to which to write the ciphertext
- BYTE *key[8]*: the 8-byte DES key

This function performs a DES ECB encipher on eight bytes of *plainText* using key into eight bytes of *cipherText*.

This is an interface to the primitive DES ECB Encipher.



## 4.27 multosExchangeData

```
void multosExchangeData (BYTE channelId, BYTE *data)
```

The parameters are:

- BYTE *channelID*: identifier of the channel to which the data should be sent
- BYTE *\*data*: address of the data to send

This function exchanges data through channel with ID *channelID*.

This is an interface to the primitive Exchange Data.

## 4.28 multosExit

```
void multosExit (void)
```

This function exits application.

This is an interface to the instruction SYSTEM.

## 4.29 multosExitLa

```
void multosExitLa (const BYTE la)
```

The parameter is a single byte value indicating the actual length of response data.

This function exits application setting *La* to the value given as *la*.

This is an interface to the instruction SYSTEM.

## 4.30 multosExitSW

```
void multosExitSW (const WORD sw)
```

The parameter is a word value indicating the value of the status word.

This function exits application with status word of *sw*.

This is an interface to the instruction SYSTEM.

## 4.31 multosExitSWLa

```
void multosExitSWLa (const WORD sw, const BYTE la)
```



The parameters are:

- const WORD *sw*: a word value indicating the value of the status word
- const BYTE *la*: a single byte value indicating the actual length of response data.

This function exits application with an SW of *sw* and an La of *la*.

This is an interface to the instruction SYSTEM.

### 4.32 multosGenerateAsymmetricHash

```
void multosGenerateAsymmetricHash (WORD plainTextLength, BYTE  
*plainText, BYTE hash[16])
```

The parameters are:

- WORD *plainTextLength*: the length of the input data
- BYTE *\*plainText*: the address of the input data
- BYTE *hash[16]*: array that holds the resulting hash digest

This function generates a 16 byte hash from *plainText*.

This is an interface to the primitive Generate Asymmetric Hash, where the default IV is used.

### 4.33 multosGenerateAsymmetricHashIV

```
void multosGenerateAsymmetricHashIV (BYTE initialValue[16], WORD  
plainTextlength, BYTE *plainText, BYTE hash[16])
```

The parameters are:

- BYTE *initialValue[16]*: the IV to use for the asymmetric hash algorithm
- WORD *plainTextLength*: the length of the input data
- BYTE *\*plainText*: the address of the input data
- BYTE *hash[16]*: array that holds the resulting hash digest

This function generates a 16 byte hash from the value held at *plainText* using IV *initialValue*.

This is an interface to the primitive Generate Asymmetric Hash.

### 4.34 multosGenerateDESCBCSignature

```
void multosGenerateDESCBCSignature (WORD plainTextLength, BYTE  
*plainText, BYTE initialValue[8], BYTE key[8], BYTE  
signature[8])
```

The parameters are:



- WORD *plainTextLength*: the length of the input data
- BYTE *\*plaintext*: the address of the input data
- BYTE *initialValue[8]*: initial value to use in DES CBC algorithm
- BYTE *key[8]*: DES key to use
- BYTE *signature[8]*: array to hold 8-byte signature

This function generates an eight byte DES CBC Signature over *plainText* using *initialValue* and *key* and writes the resulting value to *signature*.

This is an interface to the primitive Generate DES CBC Signature.

### 4.35 *multosGenerateTripleDESCBCSignature*

```
void multosGenerateTripleDESCBCSignature (WORD plainTextLength,  
BYTE *plaintext, BYTE initialValue[8], BYTE keys[16], BYTE  
signature[8])
```

The parameters are:

- WORD *plainTextLength*: the length of the input data
- BYTE *\*plaintext*: the address of the input data
- BYTE *initialValue[8]*: initial value to use in Triple DES CBC algorithm
- BYTE *key[16]*: DES keys to use
- BYTE *signature[8]*: array to hold 8-byte signature

This function generates an eight byte DES CBC Signature over *plainText* using *initialValue* and *key* and writes the resulting value to *signature*.

This is an interface to the primitive Generate Triple DES CBC Signature.

### 4.36 *multosGetDelegatorAID*

```
void multosGetDelegatorAID (const BYTE aidLength, BYTE *aid,  
BOOL *notADelegate)
```

The parameters are:

- const BYTE *aidLength*: length of the AID to fetch
- BYTE *\*aid*: address where delegator AID is written
- BOOL *\*notADelegate*: Boolean value indicating if the application has been delegated to

This function stores the AID of the delegating application into *aid* and sets the flag *notADelegate* to TRUE if the application has not been delegated to.

This is an interface to the primitive Get Delegator AID.



### 4.37 multosGetDIRFileRecord

```
void multosGetDIRFileRecord (const BYTE dirRecordLength, BYTE
recordNum, BYTE *numCopied, BYTE *output, BOOL *noRecordFound)
```

The parameters are:

- const BYTE *dirRecordLength*: the number of bytes of the DIR record to copy
- BYTE *recordNum*: DIR record number
- BYTE \**numCopied*: actual number of bytes copied
- BYTE \**output*: address where result is written
- BOOL \**noRecordFound*: Boolean value

This function retrieves the record *recordNum* from the DIR File and copies it to *output*. It also indicates the actual number of bytes copied in *numCopied* as well as setting the flag *noRecordFound* to TRUE if there is no record *recordNum* in the DIR File.

This is an interface to the primitive Get DIR File Record.

### 4.38 multosGetFileControlInformation

```
void multosGetFileControlInformation (const BYTE
fciRecordLength, BYTE recordNum, BYTE *numCopied, BYTE *output,
BOOL *noRecordFound)
```

The parameters are:

- const BYTE *fciRecordLength*: the number of bytes of the FCI record to copy
- BYTE *recordNum*: FCI record number
- BYTE \**numCopied*: actual number of bytes copied
- BYTE \**output*: address where result is written
- BOOL \**noRecordFound*: Boolean value

This function retrieves the record *recordNum* from the FCI file and copies it to *output*. It also indicates the actual number of bytes copied in *numCopied* as well as setting the flag *noRecordFound* to TRUE if there is no record *recordNum* in the FCI file.

This is an interface to the primitive Get File Control Information.

### 4.39 multosGetManufacturerData

```
void multosGetManufacturerData (const BYTE numDataBytes, BYTE
*numCopied, BYTE *output)
```

The parameters are:



- const BYTE *numDataBytes*: the number of bytes to be copied
- BYTE *\*numCopied*: actual number of bytes copied
- BYTE *\*output*: address where result is written

This function retrieves the Manufacturer Data from MULTOS chip and writes the result to *output*. It also indicates the actual number of bytes copied in *numCopied*.

This is an interface to the primitive Get Manufacturer Data.

## 4.40 *multosGetMemoryReliability*

```
BYTE multosGetMemoryReliability (void)
```

This function returns the status of the current reliability of the non-volatile memory as follows:

- MULTOS\_MEMORY\_RELIABLE
- MULTOS\_MEMORY\_MARGINAL
- MULTOS\_MEMORY\_UNRELIABLE

This is an interface to the primitive Get Memory Reliability.

## 4.41 *multosGetMultosData*

```
void multosGetMultosData (const BYTE numDataBytes, BYTE  
*numCopied, BYTE *output)
```

The parameters are:

- const BYTE *numDataBytes*: the number of bytes to be copied
- BYTE *\*numCopied*: actual number of bytes copied
- BYTE *\*output*: address where result is written

This function retrieves the MULTOS Data from MULTOS chip and writes it to *output*. It also indicates the actual number of bytes copied in *numCopied*.

This is an interface to the primitive Get MULTOS Data.

## 4.42 *multosGetRandomNumber*

```
void multosGetRandomNumber (BYTE result[8])
```

The parameter is an eight byte array where the random number will be written.

This function generates an 8-byte random number and writes that value to *result[8]*.



This is an interface to the primitive Get Random Number.

#### 4.43 multosModularExponentiation

```
void multosModularExponentiation (WORD exponentLength, WORD  
modulusLength, BYTE *exponent, BYTE *modulus, BYTE *input, BYTE  
*output)
```

The parameters are:

- WORD *exponentLength*: the length of the exponent used
- WORD *modulusLength*: the length of the modulus
- BYTE \**exponent*: address of the exponent
- BYTE \**modulus*: address of the modulus
- BYTE \**input*: address of the input value
- BYTE \**output*: address of where to write the result of the operation

This function performs a modular exponentiation. Note that the values held at *modulus*, *input* and *output* are all considered to be of size *modulusLength*.

This is an interface to the primitive Modular Exponentiation.

#### 4.44 multosModularExponentiationCRT

```
void multosModularExponentiationCRT (WORD dpdqLength, BYTE  
*dpdq, BYTE *pqu, BYTE *input, BYTE *output)
```

The parameters are:

- WORD *dpdqLength*: length of dpdq
- BYTE \**dpdq*: address of dpdq
- BYTE \**pqu*: address of pqu
- BYTE \**input*: address of input
- BYTE \**output*: address of output

This function performs a modular exponentiation using Chinese Remainder Theorem.

This is an interface to the primitive Modular Exponentiation CRT.

#### 4.45 multosModularMultiplication

```
void multosModularMultiplication (WORD modulusLength, BYTE  
*modulus, BYTE *block1, BYTE *block2)
```

The parameters are:



- WORD *modulusLength*: the length of the modulus used
- BYTE *\*modulus*: address of the modulus
- BYTE *\*block1*: address of the first operand
- BYTE *\*block2*: address of the second operand

This function performs a modular multiplication. The result of the operation is written to *block1*.

This is an interface to the primitive Modular Multiplication.

## 4.46 *multosModularReduction*

```
void multosModularReduction (WORD operandLength, WORD  
modulusLength, BYTE *operand, BYTE *modulus)
```

The parameters are:

- WORD *operandLength*: the length of the operand
- WORD *modulusLength*: length of the modulus
- BYTE *\*operand*: address of the operand
- BYTE *\*modulus*: address of the modulus

This function performs a modular reduction. The result overwrites the operand.

This is an interface to the primitive Modular Reduction.

## 4.47 *multosQueryChannel*

```
BOOL multosQueryChannel (BYTE channelID)
```

The parameter is a byte value indicating the channel to query.

This function verifies the existence of a specific channel with ID *channelID* and returns TRUE if channel supported.

This is an interface to the primitive Query Channel.

## 4.48 *multosQueryCodelet*

```
BOOL multosQueryCodelet (WORD codeletID)
```

The parameter is a byte value indicating the codelet to query.

This function verifies the existence of a specific codelet with ID *codeletID*. Returns TRUE if codelet supported.

This is an interface to the primitive Query Codelet.



#### 4.49 **multosQueryPrimitive**

```
void multosQueryPrimitive (const BYTE setNum, const BYTE  
primitiveNum, BOOL *primitiveSupported)
```

The parameters are:

- const BYTE *setNum*: the set to which the primitive belongs
- const BYTE *primitiveNum*: the number of the primitive within its set
- BOOL *\*primitiveSupported*: Boolean flag

This function verifies the existence of a specific primitive with number *primitiveNum* within set *setNum*. The flag *primitiveSupported* is set to TRUE if the primitive is supported, otherwise it is set to FALSE.

This is an interface to the primitive Query Primitive.

#### 4.50 **multosResetSessionData**

```
void multosResetSessionData (void)
```

This function allows a shell application to reset the session data of all other applications on the MULTOS card.

This is an interface to the primitive Reset Session Data.

#### 4.51 **multosResetWWT**

```
void multosResetWWT (void)
```

This function sends a WWT extension request.

This is an interface to the primitive Reset WWT.

#### 4.52 **multosReturnFromCodelet**

```
void multosReturnFromCodelet (const BYTE numBytesIn, const BYTE  
numBytesOut)
```

The parameters are:

- const BYTE *numBytesIn*: the number of stack bytes that were passed to the codelet
- const BYTE *numBytesOut*: the number of stack bytes returned by the codelet.

This function returns from the currently executing codelet and ensures that *numBytesIn* are removed from the stack and *numBytesOut* replace them.



This is an interface to the primitive Return From Codelet.

## 4.53 *multosSetATRFileRecord*

**BYTE multosSetATRFileRecord** (BYTE length, BYTE \*record)

The parameters are:

- BYTE *length*: the length of the record
- BYTE \**record*: the address of the value to write to the ATR file

This function writes a record into the ATR File returns number of bytes written.

This is an interface to the primitive Set ATR File Record.

## 4.54 *multosSetATRHistoricalCharacters*

**BYTE multosSetATRHistoricalCharacters** (BYTE length, BYTE \*input)

The parameters are:

- BYTE *length*: the length of the input
- BYTE \**input*: the address of the value to write to the ATR historical characters.

This function writes input data to the historical characters of the card's ATR and returns number of bytes written.

This is an interface to the primitive Set ATR Historical Characters.

## 4.55 *multosSetTransactionProtection*

**void multosSetTransactionProtection** (const BYTE options)

The parameter is a byte value specifying what option to use with transaction protection.. The only valid options are the following:

- MULTOS\_TP\_OFF\_AND\_DISCARD
- MULTOS\_TP\_OFF\_AND\_COMMIT
- MULTOS\_TP\_ON\_AND\_DISCARD
- MULTOS\_TP\_ON\_AND\_COMMIT

This is an interface to the primitive Set Transaction Protection.



## 4.56 multosSetSelectSW

```
void multosSetSelectSW (const BYTE sw1, const BYTE sw2)
```

The parameters are:

- const BYTE *sw1*: the value to be written to the most significant byte of the status word
- const BYTE *sw2*: the value to be written to the least significant byte of the status word

This function sets the 2-byte status word that will be returned by MULTOS when the application is next selected.

This is an interface to the primitive Set SelectSW.

## 4.57 multosSHA1

```
void multosSHA1 (WORD messageLength, BYTE *message, BYTE *hash)
```

The parameters are:

- WORD *messageLength*: length of the message to submit to the SHA-1 algorithm
- BYTE \**message*: address of the message
- BYTE \**hash*: address where to write the 20-byte digest

This function uses the value found in *message* of size *messageLength* as input to the SHA-1 hashing algorithm. The resulting 20-byte digest is written to *hash*.

This is an interface to the primitive SHA-1.



## 5 Function Listing by Type

---

### 5.1 Block Manipulations

```
void multosBlockAdd (const BYTE blockLength, BYTE *block1, BYTE
*block2, const BYTE *result)
void multosBlockAnd (const BYTE blockLength, BYTE *block1, BYTE
*block2, const BYTE *result)
BYTE multosBlockCompare (WORD blockLength, BYTE *block1, BYTE
*block2)
void multosBlockClear (const BYTE blockLength, const BYTE
*block)
void multosBlockCopy (WORD blockLength, BYTE *blockSource, BYTE
*blockDest)
void multosBlockDecrement (const BYTE blockLength, const BYTE
*block)
void multosBlockDivide (const BYTE blockLength, BYTE *numerator,
BYTE *denominator, BYTE *quotient, BYTE *remainder, BYTE
*status)
void multosBlockIncrement (const BYTE blockLength, const BYTE
*block)
void multosBlockInvert (const BYTE blockLength, const BYTE
*block)
BOOL multosBlockLookup (BYTE value, BYTE *block, BYTE *result)
void multosBlockMultiply (const BYTE blockLength, BYTE *block1,
BYTE *block2, BYTE *result)
void multosBlockOr (const BYTE blockLength, BYTE *block1, BYTE
*block2, const BYTE *result)
void multosBlockShiftLeft (const BYTE blockLength, const BYTE
numShiftBits, BYTE *blockSource, BYTE *blockDest)
void multosBlockShiftRight (const BYTE blockLength, const BYTE
numShiftBits, BYTE *blockSource, BYTE *blockDest)
void multosBlockSubtract (const BYTE blockLength, BYTE *block1,
BYTE *block2, const BYTE *result)
void multosBlockTestZero (const BYTE blockLength, const BYTE
*block, BOOL *isZero)
void multosBlockXor (const BYTE blockLength, BYTE *block1, BYTE
*block2, BYTE *result)
```

### 5.2 Cryptographic

```
DWORD multosChecksum (WORD blockLength, BYTE *block)
void multosDESECBDecipher (BYTE *cipherText, BYTE *plainText,
BYTE key[8])
void multosDESECBEncipher (BYTE *plainText, BYTE *cipherText,
BYTE key[8])
```

```

void multosGenerateAsymmetricHash (WORD plainTextLength, BYTE
*plainText, BYTE hash[16])
void multosGenerateAsymmetricHashIV (BYTE initialValue[16], WORD
plainTextLength, BYTE *plainText, BYTE hash[16])
void multosGenerateDESCBCSignature (WORD plainTextLength, BYTE
*plainText, BYTE initialValue[8], BYTE key[8], BYTE
signature[8])
void multosGenerateTripleDESCBCSignature (WORD plainTextLength,
BYTE *plainText, BYTE initialValue[8], BYTE keys[16], BYTE
signature[8])
void multosGetRandomNumber (BYTE result[8])
void multosModularExponentiation (WORD exponentLength, WORD
modulusLength, BYTE *exponent, BYTE *modulus, BYTE *input, BYTE
*output)
void multosModularExponentiationCRT (WORD dpdqLength, BYTE
*dpdq, BYTE *pqu, BYTE *input, BYTE *output)
void multosModularMultiplication (WORD modulusLength, BYTE
*modulus, BYTE *block1, BYTE *block2)
void multosModularReduction (WORD operandLength, WORD
modulusLength, BYTE *operand, BYTE *modulus)
void multosSHA1 (WORD messageLength, BYTE *message, BYTE *hash)

```

### 5.3 System

```

void multosCallCodelet (WORD codeletID, WORD entryAddress)
void multosCallExtensionPrimitive (const BYTE extensionNum,
const BYTE primTypeLo, const BYTE primTypeHi, const BYTE
paramByte)
BOOL multosCheckCase (BYTE isoCase)
void multosControlAutoResetWWT (BOOL disable)
void multosDelegate (BYTE *aid)
void multosExchangeData (BYTE channelID, BYTE *data)
void multosExit (void)
void multosExitLa (const BYTE la)
void multosExitSW (const WORD sw)
void multosExitSWLa (const WORD sw, const BYTE la)
void multosGetDelegatorAID (const WORD aidLength, BYTE *aid,
BOOL *notADelegate)
void multosGetDIRFileRecord (const BYTE dirRecordLength, BYTE
recordNum, BYTE *numCopied, BYTE *output, BOOL *noRecordFound)
void multosGetFileControlInformation (const BYTE
fciRecordLength, BYTE recordNum, BYTE *numCopied, BYTE *output,
BOOL *noRecordFound)
void multosGetManufacturerData (const BYTE numDataBytes, BYTE
*numCopied, BYTE *output)
BYTE multosGetMemoryReliability (void)
void multosGetMultosData (const BYTE numDataBytes, BYTE
*numCopied, BYTE *output)
BOOL multosQueryChannel (BYTE channelID)
BOOL multosQueryCodelet (WORD codeletID)

```



```
void multosQueryPrimitive (const BYTE setNum, const BYTE  
primitiveNum, BOOL *primitiveSupported)  
void multosResetSessionData (void)  
void multosResetWWT (void)  
void multosReturnFromCodelet (const BYTE numBytesIn, const BYTE  
numBytesOut)  
BYTE multosSetATRFileRecord (BYTE length, BYTE *record)  
BYTE multosSetATRHistoricalCharacters (BYTE length, BYTE *input)  
void multosSetTransactionProtection (const BYTE options)
```

## 5.4 EMV

```
void multosSetSelectSW (const BYTE sw1, const BYTE sw2)  
void multosCardBlock (const BYTE offsetMACHi, const BYTE  
offsetMACLo, BOOL *cardBlockedOk)
```



## A. Appendix A : Biometric C API

### A.1 Introduction

This appendix documents the MULTOS Biometric C API. This API has been chosen to be compatible with the Java Card biometric API and to simplify the porting of existing biometric Java Card applets to MULTOS.

### A.2 Constants

	Description
BIO_VERSION_LENGTH	The length of the Biometric API version string.
BIO_MAX_PUBLIC_TEMPLATE_LENGTH	The maximum length of the public template.
BIO_MINIMUM_SUCCESSFUL_MATCH_SCORE	The minimum successful template match score.
BIO_MATCH_NEEDS_MORE_DATA	The match score that indicates that more data is required to complete the match process.

### A.3 Data Types

```
enum BIO_TYPE
{ /* Facial feature recognition (visage). */
  FACIAL_FEATURE,
  /* Pattern is a voice sample (specific or unspecific speech).*/
  VOICE_PRINT,
  /* Fingerprint identification (any finger). */
  FINGERPRINT,
  /* Pattern is a scan of the eye's iris. */
  IRIS_SCAN,
  /* Pattern is an infrared scan of blood vessels of the retina of
  the eye. */
  RETINA_SCAN,
  /* Hand geometry ID is based on overall geometry/shape of the
  hand. */
  HAND_GEOMETRY,
  /* Written signature dynamics ID (behavioral). */
  SIGNATURE,
  /* Keystrokes dynamics (behavioral). */
  KEYSTROKES,
  /* Lip movement (behavioral). */
  LIP_MOVEMENT,
  /* Thermal face image. */
  THERMAL_FACE,
  /* Thermal hand image. */
  THERMAL_HAND,
  /* Gait (behavioral). */
  GAIT_STYLE,
```



```
/* Body odor. */
BODY_ODOR,
/* Pattern is a DNA sample for matching. */
DNA_SCAN,
/* Ear geometry ID is based on overall geometry/shape a ear. */
EAR_GEOMETRY,
/* Finger geometry ID is based on overall geometry/shape of a
finger. */
FINGER_GEOMETRY,
/* Palm gemoetry ID is based on overall geometry/shape of palm.
*/
PALM_GEOMETRY,
/* Pattern is an infrared scan of the vein pattern in a face,
wrist or hand. */
VEIN_PATTERN,
/* General password (a PIN is a special case of the password).
*/
PASSWORD
};
```

```
typedef BYTE BIO_VERSION[BIO_VERSION_LENGTH];
An array that holds the Biometric API version string.
```

```
typedef BYTE
BIO_PUBLIC_TEMPLATE[BIO_MAX_PUBLIC_TEMPLATE_LENGTH];
An array that holds the public template.
```

```
struct BIO_TEMPLATE
{
    // Contents implementation-specific
};
A structure that contains the biometric template.
```

## A.4 Data

The application must define one Static data structure of type `BIO_TEMPLATE` for each biometric template that it wishes to use.

## A.5 Function Prototypes

### A.5.1 `bioInit`

```
void bioInit (struct BIO_TEMPLATE *refTemplate, BYTE
*dataBuffer, WORD dataLength)
```

The parameters are:

- `struct BIO_TEMPLATE *refTemplate`: pointer to the reference template data memory
- `BYTE *dataBuffer`: pointer to buffer holding data to be enrolled
- `WORD dataLength`: length of data in data buffer



This function initialises the enrolment of a reference template.

### A.5.2 bioUpdate

```
void bioUpdate (struct BIO_TEMPLATE *refTemplate, BYTE  
*dataBuffer, WORD dataLength)
```

The parameters are:

- struct BIO\_TEMPLATE \*refTemplate: pointer to the reference template data memory
- BYTE \*dataBuffer: pointer to buffer holding data to be enrolled
- WORD dataLength: length of data in data buffer

This function continues the enrolment of a reference template. This function should only be used if all the data required for the initialisation is not available in one data buffer.

### A.5.3 bioDoFinal

```
void bioDoFinal (struct BIO_TEMPLATE *refTemplate,  
BYTE newTryLimit)
```

The parameters are:

- struct BIO\_TEMPLATE \*refTemplate: pointer to the reference template data memory
- BYTE newTryLimit: the number of tries allowed before the reference is blocked

This function finalises the enrolment of a reference template. Final action of enrolment is to designate a reference template as being complete and ready for use (marks the reference as initialised, set the try limit and resets the try counter). This function may also include some error checking prior to the validation of reference template as ready for use.

### A.5.4 bioResetUnblockAndSetTryLimit

```
void bioResetUnblockAndSetTryLimit (struct BIO_TEMPLATE  
*refTemplate, BYTE newTryLimit)
```

The parameters are:

- struct BIO\_TEMPLATE \*refTemplate: pointer to the reference template data memory
- BYTE newTryLimit: the number of tries allowed before the reference is blocked

This function resets the global validated flag, updates the try limit value and resets the try counter to the try limit value.

### A.5.5 bioGetBioType

```
BYTE bioGetBioType (void)
```



This function returns the biometric type. Valid types are described in `BIO_TYPE`.

## A.5.6 `bioIsInitialized`

```
BOOL bioIsInitialized (struct BIO_TEMPLATE *refTemplate)
```

The parameter is:

- `struct BIO_TEMPLATE *refTemplate`: pointer to the reference template data memory

This function returns the initialisation status of the reference template. This is independent of whether or not the match process has been initialised (see `bioInitMatch`).

## A.5.7 `bioIsValidated`

```
BOOL bioIsValidated (struct BIO_TEMPLATE *refTemplate)
```

The parameter is:

- `struct BIO_TEMPLATE *refTemplate`: pointer to the reference template data memory

This function returns TRUE if the template has been successfully checked since the last card reset or last call to `bioReset()`.

## A.5.8 `bioGetVersion`

```
BYTE bioGetVersion (BIO_VERSION bioVersion)
```

The parameter is:

- `BIO_VERSION bioVersion`: pointer to the array in which the version ID will be stored

This function gets the matching algorithm version or ID and returns the number of bytes written in the version data buffer.

## A.5.9 `bioGetPublicTemplateData`

```
WORD bioGetPublicTemplateData (struct BIO_TEMPLATE *refTemplate,  
BYTE *dataBuffer, WORD dataLength)
```

The parameters are:

- `struct BIO_TEMPLATE *refTemplate`: pointer to the reference template data memory
- `BYTE *dataBuffer`: pointer to the destination area
- `WORD dataLength`: the number of bytes to copy



This function gets the public part of the reference template. It copies all or a piece of the reference public data to the destination area.

#### **A.5.10 bioGetTriesRemaining**

**BYTE bioGetTriesRemaining** (struct BIO\_TEMPLATE \*refTemplate)

The parameters are:

- struct BIO\_TEMPLATE \*refTemplate: pointer to the reference template data memory

This function returns the number of times remaining that an incorrect candidate template can be presented before the reference template is blocked.

#### **A.5.11 bioReset**

**void bioReset** (struct BIO\_TEMPLATE \*refTemplate)

The parameters are:

- struct BIO\_TEMPLATE \*refTemplate: pointer to the reference template data memory

This function resets the reference validated flag.

#### **A.5.12 bioInitMatch**

**WORD bioInitMatch** (struct BIO\_TEMPLATE \*refTemplate, BYTE \*dataBuffer, WORD dataLength)

The parameters are:

- struct BIO\_TEMPLATE \*refTemplate: pointer to the reference template data memory
- BYTE \*dataBuffer: pointer to (a part of) the candidate template
- WORD dataLength: length of the candidate data to be used

This function initialises or re-initialises a biometric matching session. The exact return score value is implementation-dependant and can be used, for example, to code a confidence rate. The returns score can fall into one of the following bands:

0...(BIO\_MINIMUM\_SUCCESSFUL\_MATCH\_SCORE-1): the match has failed.

BIO\_MINIMUM\_SUCCESSFUL\_MATCH\_SCORE or more: the match has succeeded.

BIO\_MATCH\_NEEDS\_MORE\_DATA: the match process requires more data.

If a matching session is in progress, a call to `bioInitMatch()` makes the current session to end in the failed state and starts a new matching session.



## A.5.13 bioMatch

**SWORD bioMatch** (struct BIO\_TEMPLATE \*refTemplate, BYTE \*dataBuffer, WORD dataLength)

The parameters are:

- struct BIO\_TEMPLATE \*refTemplate: pointer to the reference template data memory
- BYTE \*dataBuffer: pointer to (a part of) the candidate template
- WORD dataLength: length of the candidate data to be used

This function continues the biometric matching session. Refer to `bioInitMatch()` for further details.

----- End of Document -----