Service Access Point

SAP Example

Department:	Aalborg Wireless Center
Creation Date:	7 September, 2001
Last Modified:	15 September, 2003 by Kenneth Skou Pedersen
ID and Version:	8434.405.01.009
Status:	Accepted

0 Document Control

0.1 Document History

ID	Author	Date	Status
8434.405.01.001	KKS	7 September, 2001	Being Processed
8434.405.01.002	KKS	11 September, 2001	Being Processed
8434.405.01.003	CSH	17 September, 2001	Being Processed

8434.405.01.004	CSH	17 September, 2001	Being Processed
8434.405.01.005	CSH	19 September, 2001	Being Processed
8434.405.01.006	CSH	25 September, 2001	Being Processed
8434.405.01.007	KSP	7 March, 2003	Submitted
8434.405.01.008	KSP	4 August, 2003	Submitted
8434.405.01.009	KSP	15 September, 2003	Accepted

0.2 References, Abbreviations, Terms

Table of Contents

1 Introduction

The purpose of this Service Access Point (SAP) document is to illustrate the different possibilities of the SAP syntax and how these can be constructed. For a more detailed description of the provided features please refer to the description of the SAP syntax in document [TI 8350.301]. If one is not familiar with the SAP syntax it is recommended that the SAP syntax document be read before reading this example SAP.

This document does not exhaustively cover all the possibilities in the SAP syntax but tries to cover the most important aspects ranging from simple declarations to more complicated features such as for instance links to other SAP documents (To support this an include example SAP document has also been created [TI 8434.404]). The examples used in this document will be based on the fictive entities SLA and MAS, where SLA is the slave (the service provider) and MAS is the master entity (the service user). Both of these example SAP documents are fictive and the declarations are for illustration purposes only. However all the declarations are legal and hence the documents are compilable. As all the primitives, functions and parameters are fictive the description of them and their purpose is fairly short. In real SAP documents it is important to make a more thorough and detailed description.

As the SAPs besides declaring primitives and types in the C header files it also serve as documentation it is very important to make thorough descriptions of the different primitives, functions etc. This also goes for the introduction to the document. An example of how such an introduction could be constructed is given in the following:

---------- Start of example ----------

Example of an introduction for a SAP

This document holds the EXAMPLE SAP specification. The specification covers the interface between the MAS entity and the SLA entity. Please refer to the Dual Mode System Design [TI 8010.944HLS] for further description of the system design and interfaces.

This document has the following sections:

Section 2 contains a specification of constants used on the interface, which are not specific to the value of a particular parameter. Such constants are typically size fields, specifying array or element sizes.

Section 3 contains the top-level description of the primitives defined for the interface. For each primitive its functionality is described and a element list is given. For any primitive using complex structures or parameters with identified values, references will be given to subsections of section 5

Section 4 contains the top-level description of the functions defined in the interface. For each function a description and parameter list is given. For all used parameters references will be given to subsections of section 5.

Section 5 contains the specifications of the types used in the primitives or functions of the interface. It also contains specifications of parameters with predefined values or value ranges. Cross-references to subtypes may be used within this section for complex type declarations.

Section 6 contains a set of scenarios illustrating the use of the primitives defined by the SAP through message sequence charts.

Types which are common or used for several SAPs, are specified in the include SAP [TI 8434.304 EX_INC] and links to these files will be given.

The document is primarily based on the specification [3G 25.331], [3G 31.102], [3G 24.007] and [3G

24.008].

Please refer to [TI 8350.301 SAP Syntax] for further information on the SAP syntax.

---------- End of example ----------

Please note the following:

- The description of the sections above should only be present if the section is included in the SAP
- Some of the SAPs have a section in the introduction that describes specific handling of e.g. Common Configuration Stores. Naturally this should be kept
- The paragraphs mentioning the function definition and MSC section should only be included when present in the SAP. Please note that the section numbering then changes! By using cross-references when referring to other sections this can be handled.
- In some of the SAP documents specific paragraphs like the one included below from CUMAC SAP appears. It would be a good idea to keep them and add it to the paragraph with references to specifications.

  "Snip from CUMAC SAP: The 3GPP MAC specification, [3G 25.321], defines primitives for the UMAC entity. These primitives are defined in an abstract manner that does not specify or constrain implementations. The primitives defined in the present document deviates from the ones in the 3GPP specifications, although the general principle is maintained."

2 Constants

Description:

This section contains declarations of pragmas and constant values. The pragmas are used to modify the behaviour of the TI tool chain. In this case the parameters and elements are prefixed with EX and enumerations are created in the .val file instead of #defines. The constants are used in the remaining part of the document. As can be seen in the Definition table this SAP contains a constant without a predefined value. This is because it is a linked constant from an include SAP and hence it would not make sense to assign a value to it in this SAP.

Pragma:

Name	Value	Comment
PREFIX	EXAMPLE	Functions and parameters of this document will be prefixed with EX. Primitives are never prefixed automatically. This has to be done manually.
ALWAYS_ENUM_IN_VAL_FILE	YES	Generates enums in the .val file instead of #defines
ENABLE_GROUP	NO	Do not enable h-file grouping
COMPATIBILITY_DEFINES	NO	Compatible to the old #defines

Definition:

Name	Value	Comment	Link
SLA_SIZE_MIN	2	Example of a constant used to control the minimum number of elements of an structure (value 2)	
SLA_SIZE_MAX	0x0010	Example of a constant used to control the maximum number of elements of an structure defined in hex (value 16)	
MAX_SIZE	5	Constant used to determine upper boundary of a dynamic array (value 5)	
SLA_SIZE_DEFAULT	2	Example of a constant (value 2)	
LINK_CONSTANT		Example of link to constants. (value will be taken from include SAP)	8434_404_01_EX_INC.doc - LINK_CONSTANT

History:

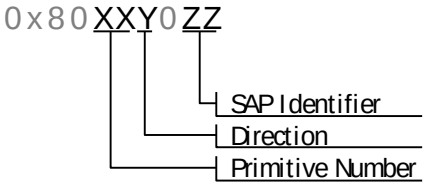
20-February-01	KKS	Initial
27-August-01	KKS	Pragmas and negative linked constant added
18-September-01	CSH	Added link to constant
19-September-01	CSH	SLA_SIZE_DEFAULT changed to 2 (used in array definition)
19-September-01	CSH	Removed COMPATIBILITY_DEFINES
19-Marts-03	KSP	Added MAX_SIZE

3 Primitives

A service access point completely defines the interface to be used to gain access to a set of services provided by an entity. The definition of the interface can be based on a set of primitives, a set of function calls or both, depending on the nature of the services provided by the entity. In this case the primitives are exchanged between a master entity and a slave entity.

For a primitive the **Short Name** is the primitive name and four general types exists. The four postfixes for theses types are: requests (REQ), confirms (CNF), indications (IND) and responses (RES), which is part of the 7-layered ISO model. The primitive ID (primitive number) is a unique identifier of the primitive. As these are not generated automatically it is up to the developer to keep track of these.

The **ID** is a 32-bit unsigned integer[1] and should be specified using the hexadecimal format (although any ANSI C syntax number format is valid). In older SAP documents, 16-bit primitive IDs may still be seen, but these are no longer to be used. The value of the primitive ID must follow a set of guidelines, currently as shown below:



The SAP identifier is a unique ID for the SAP described by the SAP document. In order to keep track of these unique IDs it is recommended to keep a document stating which SAP IDs are used and what they are used for. This way the same SAP ID will never be used twice.

Furthermore the **Direction indicates** the direction of the primitive, which can be either master-to-slave (REQ and RES) or slave-to-master (IND or CNF). For primitives of type request or response, the direction should have a value of 0. For primitives of type confirm or indication the direction should have a value of 4. The primitive number is the number of the primitive within the SAP and should start at 00 for the first primitive in the document. Please note that this direction-bit makes it possible to have primitives within the same SAP, which share the same SAP identifier and primitive number but have different direction bits.

3.1 EXAMPLE_INTEGER_REQ

Description:

> This message is an example of a simple request primitive. The primitive contains three unsigned and one signed integer. In each case the size of the integer is given by its type (e.g. a U8 corresponds to an unsigned 8-bit integer with a possible value range from 0-255 and a S8 to a signed 8-bit integer with a possible value range from –128 to 127). The actual declaration of the elements can be found simply by following the cross-reference given in the **Ref** column. In this case the only purpose of the parameter is to illustrate the use of them. As will be seen in the following section any types can be combined in a primitive. Please notice that the order of the elements is not insignificant as fill bytes are generated when the SAP is compiled due to alignment. The order provided below would require 12 bytes using 4 alignment bytes while the optimized order (U8, S8, U16, U32) only would require 8 bytes using no alignment bytes.

[1] This is the case at the moment, but as the header format for primitives in the TI protocol stack framework is being revised, this may change.

Definition:

Short Name	ID	Direction
EXAMPLE_INTEGER_REQ	0x80000000	MAS->SLA

Elements:

Long Name	Short Name	CTRL	Ref	Type
8-bit unsigned integer example	unsigned_8_bit		5.1	U8
16-bit unsigned integer example	unsigned_16_bit		5.3	U16
32-bit unsigned integer example	unsigned_32_bit		5.5	U32
8-bit signed integer example	signed_8_bit		5.2	S8

History:

20-February-01	KKS	Initial
17-September-01	CSH	Changed to use 32-bit operation codes.

3.2 EXAMPLE_ENUMERATION_IND

Description:

This message is an example of a request primitive containing an enumeration. In this case the purpose of the parameter is to illustrate the use of an enumerations.

Definition:

Short Name	ID	Direction
EXAMPLE_ENUMERATION_IND	0x80004000	MAS->SLA

Elements:

Long Name	Short Name	CTRL	Ref	Type
Enumeration example	enumeration		5.7	ENUM

History:

1-September-01	KKS	Initial
17-September-01	CSH	Changed to use 32-bit operation codes.

3.3 EXAMPLE_EMPTY_CNF

Description:

This is an example of an empty confirm primitive. When the SAP is compiled the compiler automatically generates a dummy parameter of type U8, as the type generated in the header file cannot be empty. However this dummy element is not to be used.

Definition:

Short Name	ID	Direction
EXAMPLE_EMPTY_CNF	0x80014000	SLA->MAS

Elements:

Long Name	Short Name	CTRL	Ref	Type

History:

20-February-01	KKS	Initial
17-September-01	CSH	Changed to use 32-bit operation codes.

3.4 EXAMPLE_STRUCT_IND

Description:

This is an example of an indication primitive containing a struct. For information on the content of the struct follow the cross-reference to the actual declaration.

Definition:

Short Name	ID	Direction
EXAMPLE_STRUCT_IND	0x80024000	SLA->MAS

Elements:

Long Name	Short Name	Ref	Type
Struct example	struct_element_1	5.8	STRUCT

History:

20-February-01	KKS	Initial
17-September-01	CSH	Changed to use 32-bit operation codes.

3.5 EXAMPLE_UNION_IND

Description:

This is an example of an indication primitive containing two unions as elements. When the SAP is compiled, an element (a union controller) is inserted for each union to indicate which of the possible structures in the union to use. The elements will in this example be named ctrl_union_element_1 and ctrl_union_element_2. By following the cross-references in the **Ref** column the valid values for the union can be found in the **TAG id** column in the Elements part of 5.13 and 5.15.

Furthermore by following the cross-reference 5.15 in the **Ref** column it can be seen that the second union is also an example of a definition included from an external include SAP document.

Definition:

Short Name	ID	Direction
EXAMPLE_UNION_IND	0x80034000	SLA->MAS

Elements:

Long Name	Short Name	Ref	Type
Simple union example	union_element_1	5.13	UNION
Complex Union example	union_element_2	5.15	UNION

History:

20-February-01	KKS	Initial
17-September-01	CSH	Changed to use 32-bit operation codes.

3.6 EXAMPLE_OPTIONAL_RES

Description:

This is an example of a response primitive containing an optional element. The optional/mandatory information of an element is set in the **Pres** column (Pres is the abbreviation for presence). By default, elements are mandatory and a **Pres** column is not needed if all elements are mandatory. However when the presence column is present it should be used for mandatory elements as well as can be seen in the table below. For optional elements a U8 element (valid flag) is generated when the SAP is compiled, which will determine whether the optional element is valid or not. The element in this example will be named v_optional_element.

Definition:

Short Name	ID	Direction
EXAMPLE_OPTIONAL_RES	0x80010000	MAS->SLA

Elements:

Long Name	Short Name	Pres	CTRL	Ref	Type
Mandatory element	unsigned_8_bit	Mandatory		5.1	U8
Optional S16 element	signed_16_bit	Optional		5.4	S16
Mandatory S32 element	signed_32_bit	Mandatory		5.6	S32
Optional element example	optional_element	Optional		5.3	U16

History:

20-February-01	KKS	Initial
17-September-01	CSH	Changed to use 32-bit operation codes.

3.7 EXAMPLE_POINTER_RES

Description:

This is an example of a response primitive, containing a pointer element. The first element is a pointer to an unsigned 8-bit element and the second element is a pointer to a dynamic size array of unsigned 8-bit elements.

Definition:

Short Name	ID	Direction
EXAMPLE_POINTER_RES	0x80020000	MAS->SLA

Elements:

Long Name	Short Name	CTRL	Ref	Type
Dynamic array of 8-bit integers	pointer_array	PTR[SLA_SIZE_MIN.. SLA_SIZE_MAX]	5.1	U8

History:

20-February-01	KKS	Initial
17-September-01	CSH	Changed to use 32-bit operation codes.

3.8 EXAMPLE_ARRAY_RES

Description:

This is an example of a response primitive containing fixed and variable sized elements. The size of the fixed/dynamic sized elements is controlled by the **CTRL** column. If an element must have a fixed size it is given as for the fixed_array_element. If an element must have a dynamic size it is given as for the dynamic_array_element. For a dynamic sized element an element (counter) is generated when the SAP is compiled, which is used to indicate the number of elements used. The counter element will have a c_ prefix and will in this example be named c_dynamic_array_element, and its type will depend on the value of SLA_SIZE_MAX. In this example the array is of the type U8, but other types such as for instance STRUCT is also valid for creating an array of structs (but not UNION, see syntax document).

Definition:

Short Name	ID	Direction
EXAMPLE_ARRAY_RES	0x80030000	MAS->SLA

Elements:

Long Name	Short Name	CTRL	Ref	Type
Fixed size array example	fixed_array_element	[SLA_SIZE_DEFAULT]	5.14	U8
Dynamic size array example of structs	dynamic_array_element	[SLA_SIZE_MIN.. SLA_SIZE_MAX]	5.14	U8

History:

20-February-01	KKS	Initial
17-September-01	CSH	Changed to use 32-bit operation codes.

3.9 EXAMPLE_PTR_STRUCT_IND

Description:

This is an example of an indication primitive containing a pointer to a structure. As can be seen in the **CTRL** column the PTR keyword is used to indicate that it is a pointer. When PTR is used the type name will be prefixed with ptr_, which in this case will result in the type name to be ptr_struct_element_1.

Definition:

Short Name	ID	Direction
EXAMPLE_PTR_STRUCT_IND	0x80044000	SLA->MAS

Elements:

Long Name	Short Name	CTRL	Ref	Type
Struct example	struct_element_1	PTR	5.8	STRUCT

History:

25-September-01	CSH	Initial

3.10 EXAMPLE_LINKED_STRUCT_REQ

Description:

This is an example of a request primitive containing a linked struct from the example include SAP. Note that the cross-reference leads to section 5.15 where all elements imported from other SAP documents are present.

Definition:

Short Name	ID	Direction
EXAMPLE_LINKED_STRUCT_REQ	0x80040000	MAS -> SLA

Elements:

Long Name	Short Name	CTRL	Ref	Type
Linked struct example	linked_struct		5.15	STRUCT

History:

16-Marts-03 KSP Initial

3.11 EXAMPLE_POINTER_STRUCT_REQ

Description:

This is an example of a request primitive containing some different elements to illustrate that it is possible to combine any desired types in the Elements part. Furthermore this primitive also uses optional elements, dynamic arrays and a pointer to a linked dynamic array of structs.

Definition:

Short Name	ID	Direction
EXAMPLE_POINTER_STRUCT_REQ	0x80050000	MAS -> SLA

Elements:

Long Name	Short Name	Pres	CTRL	Ref	Type
Simple optional signed integer	signed_8_bit	Optional		5.2	S8
Example of dynamic array of simple structs	simple_struct	Mandatory	[1..2]	5.12	STRUCT
Linked struct example. The struct is linked from the include SAP	linked_struct	Mandatory	PTR[1..5]	5.15	STRUCT

History:

19-Marts-03 KSP Initial

3.12 EXAMPLE_OPTIONAL_U8_REQ

Description:

This is another example of a request primitive containing some different elements.

Note that the references in the **Ref** column for the U8 types are identical. By following the cross-reference to the actual definition it can be seen that it is possible to define two identical elements with different short names in the same section. In this example the type U8 is used, but this feature is also available for other types such as for instance STRUCTs

Definition:

Short Name	ID	Direction

EXAMPLE_OPTIONAL_U8_REQ	0x80060000	MAS -> SLA

Elements:

Long Name	Short Name	Pres	CTRL	Ref	Type
Struct elements with local elements	struct_element_1	Mandatory		5.8	STRUCT
Simple optional unsigned integer	unsigned_8_bit_a	Optional		5.1	U8
Simple optional unsigned integer	unsigned_8_bit	Optional		5.1	U8

History:

31-July-03 KSP Initial

3.13 EXAMPLE_ELEMENT_C_NAME_REQ

Description:

This is an example of a request primitive where the **C-Name** column is used in the Elements part. In this case the Elements part contains two identical structs with the same **Short Name** and the **C-Name** is used for the first struct only. The result of using this **C-Name** will be that the member name of the first struct contained in the primitive will be names according to the name in the **C-Name** column instead of the name in the **Short Name** column. The type the member name will however be named according to the **Short Name**. That is the first struct element in the primitive will be named new_struct_name instead of struct_element_2 and the type of the member will be named according to the **Short Name**, namely T_struct_element_2. As no **C-Name** is given for the second struct both the member name and type name of the struct will be named according to the **Short Name,** namely struct_element_2.

Definition:

Short Name	ID	Direction
EXAMPLE_ELEMENT_C_NAME_REQ	0x80070000	MAS -> SLA

Elements:

Long Name	Short Name	Pres	CTRL	C-Name	Ref	Type
Struct element with local elements	struct_element_2	Mandatory		new_struct_name	5.9	STRUCT
Struct element with local elements	struct_element_2	Mandatory	PTR		5.9	STRUCT

History:

31-July-03 KSP Initial

3.14 EXAMPLE_REF_DEFINITION_C_NAME_REQ

Description:

This is an example of a request primitive where the **C-Name** column is used in the referenced Definition part. In this primitive the Elements part contains two structs and the **C-Name** is not used in the Elements part but only in the definition of the referenced parameter.

The result of using this combination of **C-Name** will be that the type name of the member will be named according to the name in the Definition **C-Name** column instead of using the name from the **Short Name** column. The name of the member will still be named according to the **Short Name**. Please follow the cross-reference for a more detailed description.

Definition:

Short Name	ID	Direction
EXAMPLE_REF_DEFINITION_C_NAME_REQ	0x80080000	MAS -> SLA

Elements:

Long Name	Short Name	Pres	C-Name	Ref	Type
Struct elements with local elements	struct_element_3	Mandatory		5.10	STRUCT

History:

15-September-03 KSP Initial

3.15 EXAMPLE_REF_DEF_AND_ELEMENT_C_NAME_REQ

Description:

This is an example of a request primitive where the **C-Name** column is used both in the Elements part and in the referenced Definition part. Again a single struct element is used for illustration.

The result of using this combination of **C-Name** will be that the type name of the member will be named according to the name in the Definition **C-Name** column instead of using the name from the **Short Name** column. The name of the member will be named according to the **C-Name** from the element table. Please follow the cross-reference for a more detailed description.

Definition:

Short Name	ID	Direction
EXAMPLE_REF_DEF_AND_ELEMENT_C_NAME_REQ	0x80090000	MAS -> SLA

Elements:

Long Name	Short Name	Pres	C-Name	Ref	Type
Struct elements with local elements	struct_element_4	Optional	new_struct_name	5.11	STRUCT

History:

15-September-03 KSP Initial

4 Functions

4.1 Sla_example_function_call

Description:

This is an example of how to define a function interface. The function has an unsigned 8-bit input parameter and returns an unsigned 32-bit parameter. The handling of functions in the SAPs is special since it relies on inline specification of the function prototypes using C syntactical notation. In contrast to the other sections in the SAP the actual content of the definition is not really checked by the TI tool chain. What identifies this as a special definition to the tool chain is the use of the keyword **InlineC** in the **ID** column of the table. The **Direction** column identifies the entities involved just as for primitives. The **Short Name** column is used to specify the C function prototype for the SAP function.

There is only a minimum check in the tool chain of the syntax for the prototype. The function prototype must have brackets around the argument list, and there must be a space before the function name. Apart from this, it is treated as an inline declaration without interpretation, and it is the responsibility of the designer of the SAP to find the correct type names The actual functionality has to be handled else where by the developer. An example of the definition for a function, which takes a single U8 argument of type unsigned_8_bit and returns a U32 type, could be:

Definition:

Short Name	ID	Direction
extern U32 example_function_call(U8 unsigned_8_bit)	InlineC	MAS->SLA

Elements:

Long Name	Short Name	CTRL	Ref	Type
Input: Unsigned 8-bit parameter	unsigned_8_bit		5.1	U8
Output: Unsigned 32-bit parameter	unsigned_32_bit		5.5	U32

History:

29-August-01 KKS Initial

4.2 Function_returning_u16

Description:

This is an example of a simple function, which again takes a single U8 as argument and returns a U16 of type T_EX_ unsigned_16_bit. This type name cannot be found directly in the SAP but requires a little knowledge about how the type names are constructed. If one is not familiar with the naming of the types one can easily find the desired type name by looking in the generated header files. In this case the type name is constructed by the type prefix T_ plus the SAP prefix EX_ and finally the short name of the U16. Concatenating these three, results in the type name T_EX_ unsigned_16_bit.

Definition:

Short Name	ID	Direction
extern T_EX_ unsigned_16_bit example_function_returning_u16(U8 unsigned_8_bit)	InlineC	MAS->SLA

Elements:

Long Name	Short Name	CTRL	Ref	Type

Input: Unsigned 8-bit parameter	unsigned_8_bit	5.1	U8
Output: U16 of type T_EX_ unsigned_16_bit	unsigned_16_bit	5.3	U16

History:

31-July-03 KSP Initial

5 Parameters

This section contains declarations of parameter types, parameter values and parameter ranges.

5.1 Unsigned Integer, 8-bits (U8)

Description:

This is the definition of the parameter 8-bit unsigned integer with the possible values given. Note that it is possible to declare several identical types in the definition, simple by adding another row to the Definition part. In the Values part the style for defining the range used is “0x00 .. 0x10”. For backward compatibility reasons the old style “0x00-0x10” is also supported. Please notice that it is possible to have multiple ranges, which can even overlap each other or exceed the enums defined as shown below. See also in section 5.5 for another example of how to use ranges.

Definition:

Type	Short Name	Comment
U8	unsigned_8_bit	Description of the type
U8	unsigned_8_bit_a	Description of the type
U8	pointer_array	Description of the type

Values:

Value	C-Macro	Comment
0x00 .. 0x10		Range of parameter
0x00	VALUE0	Example of value definition
0x01	VALUE1	Example of value definition
0x02	VALUE2	Example of value definition

History:

20-February-01 KKS Initial

5.2 Signed Integer, 8-bits (S8)

Description:

This is the definition of the parameter 8-bit signed integer. As opposed to the section above no values or ranges are given.

Definition:

Type	Short Name	Comment
S8	signed_8_bit	Description of the type
S8	second_signed_8_bit	Description of the type

History:

29-August-01 KKS Initial

5.3 Unsigned Integer, 16-bit (U16)

Description:

This is the definition of two 16-bit unsigned integer parameters. No values are given.

Definition:

Type	Short Name	Comment
U16	unsigned_16_bit	Description of the type
U16	optional_element	Description of the type

History:

20-February-01 KKS Initial

5.4 Signed Integer, 16-bit (S16)

Description:

This is the definition of two 16-bit signed integer parameters. No values are given.

Definition:

Type	Short Name	Comment
S16	signed_16_bit	Description of the type

History:

15-September-03 KSP Initial

5.5 Unsigned Integer, 32-bit (U32)

Description:

This is the definition of the parameter 32-bit unsigned integer. In the Value section an example of double value definition is given. In the Values part the style for defining the range used is “0x00 .. 0xFF”. For backward compatibility reasons the old style “0x00-0xFF” is also supported.

Definition:

Type	Short Name	Comment
U32	unsigned_32_bit	Description of the type

Values:

Value	C-Macro	Comment
0x00..0xFF	CONFIG_ID_MEASUREMENT	Range to be used in connection with measurement configurations.
0x00 .. 0xFD	CONFIG_ID_BCH_CONFIG	Range to be used in connection with CPHY_BCH_CONFIG_REQ.
0xFE	CONFIG_ID_NETWORK_SCAN	Fixed configuration ID used in conjunction with network scans.
0xFF	CONFIG_ID_ACTIVATION	Fixed configuration ID used by PHY for CPHY_BCH_DATA_IND.

History:

20-February-01 KKS Initial

15-September-03 KSP Values added.

5.6 Signed Integer, 32-bit (S32)

Description:

This is the definition of the parameter 32-bit signed integer. No values are given.

Definition:

Type	Short Name	Comment
S32	signed_32_bit	Description of the type

History:

15-September-03 KSP Initial

5.7 Enumeration

Description:

This is the definition of an enumeration parameter with the possible values given.

Definition:

Type	Short Name	Comment
ENUM	enumeration	Description of the type

Values:

Value	C-Macro	Comment
0x00	ENUM_VALUE_0	Example of value definition
0x01	ENUM_VALUE_1	Example of value definition
0x02	ENUM_VALUE_2	Example of value definition

History:

1-September-01 KKS Initial

5.8 Struct element 1

Description:

This is an example of the definition of a STRUCT. This struct is made up of 3 elements. A 32-bit unsigned integer defined in 5.4 and two local unsigned integers with the values given in the value table below. In the Values part the style for defining the range used is “0x00 .. 0xAA”. For backward compatibility reasons the old style “0x00-0xAA” is also supported.

Definition:

Type	Short Name	Comment
STRUCT	struct_element_1	Container

Elements:

Long Name	Short Name	CTRL	Ref	Type
Unsigned long	unsigned_32_bit		5.5	U32
Local unsigned char	local_unsigned_char			U8
Local unsigned int	local_unsigned_int			U16

Values:

Name	Value	C-Macro	Comment
local_unsigned_char	0x00 .. 0xAA		Range of parameter
	0x00	CHAR_VALUE0	Comment to avoid warning

	0x01	CHAR_VALUE1	Comment to avoid warning
local_unsigned_int	0x0000 .. 0xBEAF		Range of parameter
	0x00	INT_VALUE0	Comment to avoid warning
	0x01	INT_VALUE1	Comment to avoid warning

History:

20-February-01 KKS Initial

5.9 Struct element 2

Description:

This is an example of the definition of a STRUCT. This STRUCT is made up of a single element namely a 32-bit unsigned integer defined in 5.4. Note that the **C-Name** column is present in the definition part of the parameter but it is not used. The use of the **C-Name** column in the definition part of parameters is demonstrated in the element struct_element_3 in section 5.10

Definition:

Type	Short Name	Comment	C-Name
STRUCT	struct_element_2	Container	

Elements:

Long Name	Short Name	CTRL	Ref	Type
Unsigned long	unsigned_32_bit		5.5	U32

History:

31-July-03 KSP Initial

5.10 Struct element 3

Description:

This is an example of the use of **C-Name.** In this example the **C-Name** is only used in the referenced Definition and not in the Element section of the primitive using the parameter. This struct is made up of a single element namely a 32-bit unsigned integer defined in 5.4.

Note that the **C-Name** column is present in the definition part of the parameter and not in the Elements part of the primitive using the parameter. This results in the type name of the member to be named according to the name in the **C-Name** column instead of using the name from the **Short Name** column. That is, the type name in the header file will be named T_EX_struct_c_name instead of T_EX_struct_element_3. The name of the member will however still be named according to the **Short Name**.

Definition:

Type	Short Name	Comment	C-Name
STRUCT	struct_element_3	Container	struct_c_name

Elements:

Long Name	Short Name	CTRL	Ref	Type
Unsigned long	unsigned_32_bit		5.5	U32

History:

15-September-03 KSP Initial

5.11 Struct element 4

Description:

This is an example of the use of **C-Name** in both the Elements section of the primitive and in the referenced Definition section of the parameter.

The result of using this combination of **C-Name** will be that the type name of the member will be named according to the name in the Definition **C-Name** column instead of using the name from the **Short Name** column. That is, the type name in the header file will be named T_EX_struct_c_name instead of T_EX_struct_element_4. The name of the member will be named according to the **C-Name** from the element table. That is, the member name in the header file will be named T_EX_new_struct_name instead of T_EX_struct_element_4.

Definition:

Type	Short Name	Comment	C-Name
STRUCT	struct_element_4	Container	struct_c_name

Elements:

Long Name	Short Name	CTRL	Ref	Type
Unsigned long	unsigned_32_bit		5.5	U32

History:

15-September-03 KSP Initial

5.12 Simple struct

Description:

This is an example of the definition of a STRUCT. This STRUCT is made up of 4 elements. Two 8-bit unsigned integer defined in 5.1 and an enumeration and an union defined in section 5.6 and in section 5.13 respectively.

Definition:

Type	Short Name	Comment
STRUCT	simple_struct	Container

Elements:

Long Name	Short Name	CTRL	Ref	Type
Simple signed integer	signed_8_bit		5.2	U8
Example of using same definition for a type with another short name	second_signed_8_bit		5.2	U8
Example of using the same short name in different structures.	union_element_1		5.13	UNION
Example of using an ENUM	enumeration		5.7	ENUM

History:

19-Marts-03 KSP Initial

5.13 Union element 1

Description:

This is an example of the definition of a simple UNION called union_element_1, which contains two integers.

Definition:

Type	Short Name	Comment
UNION	union_element_1	Container

Elements:

Tag ID	Long Name	Short Name	Ref	Type
unsigned_long_example	32-bit unsigned integer	unsigned_32_bit	5.5	U32
unsigned_char_example	8-bit unsigned integer	unsigned_8_bit	5.1	U8

History:

20-February-01 KKS Initial

5.14 Array Element

Description:

This is an example of the definition of two U8 used for arrays of fixed and dynamic size.

Definition:

Type	Short Name	Comment
U8	fixed_array_element	Container
U8	dynamic_array_element	Container

History:

20-February-01 KKS Initial

5.15 Imported elements

Description:

These elements are imported from the include SAP listed in the **Link** column. The actual definition in these include documents can be found by following the hyperlink in the **Link** column.

Definition:

Type	Short Name	Comment	Link
UNION	union_element_2	This field can be empty	8434_404_01_EX_INC.doc - union_element_2
STRUCT	linked_struct	Linked struct from include SAP	8434_404_01_EX_INC.doc - linked_struct

History:

1-Spetember-01	KKS	Initial
17-September-01	CSH	Changed link to “8434_405_01_EX_INC.doc”
16-Marts-03	KSP	Added element linked_struct