Technical DocumentNote

G23-UMTS PROTOCOL STACK

SPECIFYING SERVICE ACCESS POINTS

Document Number:	06-03-22-DUO-0002
Version:	0.4
Status:	Draft
Approval Authority:	
Creation Date:	2001-Aug-08
Last changed:	2015-Mar-08 by SIJ
File Name:	8350_301_SAP_Syntax.doc

Important Notice

Texas Instruments Incorporated and/or its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products, software and services at any time and to discontinue any product, software or service without notice. Customers should obtain the latest relevant information during product design and before placing orders and should verify that such information is current and complete.

All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment. TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI products, software and/or services. To minimize the risks associated with customer products and applications, customers should provide adequate design, testing and operating safeguards.

Any access to and/or use of TI software described in this document is subject to Customers entering into formal license agreements and payment of associated license fees. TI software may solely be used and/or copied subject to and strictly in accordance with all the terms of such license agreements.

Customer acknowledges and agrees that TI products and/or software may be based on or implement industry recognized standards and that certain third parties may claim intellectual property rights therein. The supply of products and/or the licensing of software does not convey a license from TI to any third party intellectual property rights and TI expressly disclaims liability for infringement of third party intellectual property rights.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products, software or services are used.

Information published by TI regarding third–party products, software or services does not constitute a license from TI to use such products, software or services or a warranty, endorsement thereof or statement regarding their availability. Use of such information, products, software or services may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Change History

Date	Changed by	Approved by	Version	Status	Notes
2001-Aug-08	LOE		0.1	Being Processed	1
2001-Aug-31	LOE		0.2	Submitted	
2001-Nov-20	CSH		0.3	Submitted	
2003-May-07	XGUTTEFE		0.4	Draft	
2003-Sept-16	SIJ		0.5	Being Processed	2

Notes:

1. Initial version
2. Content update to the status of CC label UMTS_tools_2003616 + new doc number 06-03-22-DUO-0002

Table of Contents

List of Figures and Tables

List of References

[ISO 9000:2000] International Organization for Standardization. Quality management systems - Fundamentals and vocabulary. December 2000

1 Introduction

This document contains a description of the way service access points are specified for entities in the TI protocol stacks. The definition of a service access point is based on the layered protocol stack model. A service access point identifies the services provided by an entity to other entities placed at a higher level in the protocol stack, either in a higher layer or with hierarchically higher rank within the same layer.

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 TI terms, a service access point is defined through a Microsoft Word document. The document contains a description of the service access point in a specific format, which can be processed by the TI tool chain, resulting in output for the various TI tools (test tools, tracing tools, programming tools etc.). This format is fairly simple and changes to primitives etc are easily done. If the end-result is seen from the viewpoint of a programmer working on implementation of a protocol stack, the end-result corresponds to a set of includable source files containing the definition of the SAP as C declarations. In addition to this the original SAP document also serves as documentation.

One of the great benefits of using the SAP concept is that the resulting code will be structured according to the code standard. This way consistency is ensured in declarations as names of valid flags, counters etc. will have a consistent format throughout the code. This way it will also be easier to read read code written by different developers, as the code standard will be kept. Furthermore there will be no name clashes since newer SAP documents can use PREFIX on entity level or alias name on element level. All in all this SAP concept is a single source concept, which allows for both code and documentation in one. That is, it is possible to maintain the documentation and the code at the same time and at the same time ensure consistency in the code.

The aim of this document is to explain how to define service access points. First this document will describe how the service access point must be structured and how the different elements can be combined.

Finally this document will describe the mapping from the SAP into the resulting C code and describe some of the features in more detail. This will also include a total overview of all the legal and illegal combinations of elements, columns etc. This document does in some cases illustrate some points by use of C examples. Such examples will, however, be subject to changes in case of alterations to the TI coding standard.

2 Structure of the Specification

As it must be possible for the TI tool chain to process documents containing SAP specifications, the structure of the document is standardised. If the structure of the document is incorrectly implemented, the tool chain will not recognise the document as a SAP specification during translation. The document structure must have front page, table of contents, document control sections and introduction, which is provided in the TI SAP document template. In addition to this the SAP specification has up to four "active" main sections:

Constants - declares all global constants used in primitives, parameters or functions

Primitives - declares all primitives used in the SAP

Functions - declares all functions provided by the SAP

Parameters - declares all parameters included in primitives, functions or other parameters

The sections must be sequential in the document (e.g. [2,3,4,5] or [2.1, 2.2, 2.3, 2.4]). It is not possible to use subsections in the constants section. The primitives' section, functions section and the parameters section can have subsections, one for each primitive, function or parameter. A maximum of three sublevels are allowed, but it is not recommended to use more than one additional level (e.g. 3.2 or 4.20). Header styles are used to indicate the headline for each section/subsection.

If no functions are declared for a SAP the functions section can be left out. All other sections must be present, but empty sections are allowed if no declarations are to be made. An exception is the primitive section, which must contain at least one primitive.

It is possible to insert multi-line comments in the document by use of the standard C syntax of "/*" and "*/" for begin and end of the commented section. The "/*" and "*/" must be the first characters on a line. The tool chain simply skips the text between them. This can also be used to insert things, which is not directly a part of the SAP such as for instance a MSC.

2.1 Declarations

Each of the active sections described above corresponds to a declaration. The declaration specifies either a set of constants, primitives, parameters or functions. As part of the syntax of a declaration, a number of keywords must be used to separate the content into parts. The parts that make up a section containing a declaration are:

- **Description**
- **Pragma**
- **Definition**
- **Elements**
- **Values**
- **History**.

Some of these parts are mandatory while others are optional but if present they have to be in the order presented above. In addition to this not all of the parts above are allowed in all sections, as it would not make any sense. For a quick and complete overview of this see Table 1, which lists the possible combinations. However most of the parts can be used in the majority of the sections.

● Mandatory ⦿ Optional ○ Illegal	Description	Pragma	Definition	Elements	Values	History
Constants	●	⦿	●	○	○	●
Primitives	●	○	●	●	○	●
Functions	●	○	●	●	○	●
Parameters	●	○	●	⦿	⦿	●

Table 1 - Allowed parts for sections of SAP specification.

For instance, as can be seen in Table 1, Pragma is only allowed in the Constants section while Values can only be used for Parameters. Please notice that Description and History are mandatory in all sections as these make up the documentation part of the SAP. Therefore it is very important to make a thorough description for each primitive, parameter etc. This also goes for the History part where it will be possible to keep track of changes to the specific section.
In order to separate the parts from each other keywords for each part are put into the section containing the declaration. This is depicted in Figure 1.

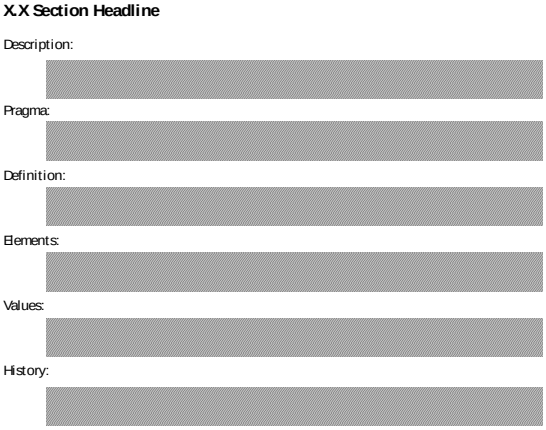


Figure 1 - Use of keywords in section of SAP document.

Please keep in mind that Figure 1 is for demonstration purposes only as it would not be legal to make such a section according to Table 1.Each of the parts mentioned above have their own purpose, which will be described in detail in the following. For each of these parts it will be described how it applies to for instance Constants, Primitives, Function and Parameters.

The actual declaration in the SAP syntax is based on tables. That is, all of the parts described above are each constructed by the use of tables. These tables can consist of several different rows and columns depending on the nature of the section, which they are a part of. Each column has a heading, which is a reserved word recognized by the tool chain and the rows are used for user defined entries. An example of a heading used, could for instance be one used for the name of a primitive (**Short Name**) or a column used for comments (**Comment**). As for the parts described above the columns allowed in a section depends on the contexts. For a complete overview of all the possibilities of combining these columns please see section **Error! Reference source not found.**.

When defining the section headers and tables in the SAP document it is important to keep some simple rules. Naturally the template designed for creating SAP documents (TI2000) should be used as well as the order and combination of the individual section must comply with the rules listed above. In addition to this there some practical rules which must be followed: As the headings of the columns are treated as reserved words in the tool chain, it is not legal to have for instance a cell in the table containing only one of these reserved words (For instance it is not legal to have a **Long Name** cell containing only the word name as it is considered a reserved word). Another thing worth noticing is that it is not allowed to have spaces in front of a declaration in a cell as the interpreter of the cell sometimes only uses the first letter and therefore would read a space instead of the correct letter. In addition to this it is not legal to manually insert line breaks in table cells. By keeping these simple rules the output of the SAP compilation should be supported by the entire tool chain. For an example of such an SAP document see [TI 8434.405].

2.1.1 Description

The description is a textual explanation of the purpose and meaning of the content of the section. It should contain enough information to clearly outline why the content of the section is included in the SAP and how it is to be used (and if necessary, how it is not to be used). This section is very important, as the quality of the descriptions for the sections will largely determine the overall quality of the documentation for the SAP. As this SAP concept is single source it is very important to be thorough when making these descriptions, as this will be the only documentation of the primitives etc. The description has the same format for all declarations such as for instance primitives, parameters or functions. The description is purely informational and is directed at the readers of the SAP document. It is not used by the TI tool chain and does not appear in any source output from the processed document.

2.1.2 Pragmas

This part is only legal in the Constants section and is used to modify the behaviour of the TI tool chain. Pragmas are contained in a table, and an example could look like this:

Pragma:

Name	Value	Comment
PREFIX	ABC	Prefix parameters/elements with "ABC"
COMPATIBILITY_DEFINES	YES	Generate compatibility defines

The name of the pragma is contained in the N**ame** column. In addition there are two more columns. The first is a **Value** column, which contains the value for the desired Pragma. The second is a C**omment** column, which can be used to explain the consequences of the pragma for the SAP.

Currently the following pragmas are supported:

PREFIX

COMPATIBILITY_DEFINES

ALWAYS_ENUM_IN_VAL_FILE

ENABLE_GROUP.

CAPITALIZE_TYPENAMES

Each of these will be explained in the following sections.

2.1.2.1 PREFIX

The pragma **PREFIX** allows all constants, elements and types generated from the SAP document to be automatically prefixed with a letter combination contained in the V**alue** column. The letter combination should follow the TI coding standard, which currently states that SAP content should be prefixed by SAP identifier for the entity to which it belongs (e.g. "RRC"). A special value (**NONE**) can be used to indicate that no prefixing is to be done for the content of the SAP. **Prefixing never applies to primitive names or function names.**

2.1.2.2 COMPATIBILITY_DEFINES

The pragma **COMPATIBILITY_DEFINES** makes the tool chain generate C pre-processor directives, redefining legacy style declarations to the current standard. The values can be **YES** and **NO** indicating whether to generate them or not.

The combination of **PREFIX** = **NONE** and **COMPATIBILITY_DEFINES** = **YES** is undefined and hence useless, as no prefixing means that all names used in the SAP will remain as is.

2.1.2.3 ALWAYS_ENUM_IN_VAL_FILE

The pragma **ALWAYS_ENUM_IN_VAL_FILE** will make the tool generate an enum for each U8, S8, U16, S16, U32 and S32 type. Each enum containing the constant associated with the corresponding type. The values can be **YES** or **NO**.

If pragma **ALWAYS_ENUM_IN_VAL_FILE** have a value different from **YES** or is not present, then #define will be generated for such constants.

2.1.2.4 ENABLE_GROUP

The pragma **ENABLE_GROUP** is used to enable groups. Groups are used for supporting more than one coding standard. The values can be **YES** or **NO**.

If pragma **ENABLE_GROUP** has a value different from **YES** or is not present, then G**roup** columns are ignored. If pragma **ENABLE_GROUP** have the value **YES**, then **Group** columns are mandatory when applicable (see individual table description), and the group cell must contain a value, which may be the special value **none** in which case no entry in the output file is generated for that row. The special group "**none"** is not allowed for types or constants used by a type having another group value than **none**.

When using **Group** columns the output h-files are named according to the group names. That is, the original SAP name does not affect which output files a type is generated in. The group name is used for prefix generation as well (pragma **PREFIX** is ignored if **Group** columns are present). Using **Groups** causes the output to be slightly altered. This is illustrated in section 2.1.11.

2.1.2.5 CAPITALIZE_TYPENAMES

This pragma is used to indicate whether the generated type names will be capitalized or not. That is for instance whether the generated type for an element called my_u8 would be T_my_u8 or T_MY_U8.

The values can be **YES** or **NO**

2.1.3 Definition

The Definition section contains a table defining some aspects of the part of the SAP described by the section. This section is mandatory for all the different parts The content of the definition depends on the kind of declaration contained in the section. In the following sections the content for respectively Constants, Primitives, Function and parameters will be explained. Please remember that the names used for instance for the **Short Name** is case sensitive.

2.1.3.1 Definition for Constants

For constants the definition serves the purpose of naming the constants and associating them with constant integer values. This corresponds to defining constants using the "#define" pre-processor directive in C. For constants the tabular definition contains three columns (**Name**, **Value** and **Comment**). Each entry in the tabular definition defines a global constant, assigns it a value and explains its purpose. An example could be:

Definition:

Name	Value	Comment
FIRST_CONSTANT	1	This is a constant assigned the value 1
SECOND_CONSTANT	-2	This is a constant assigned the value -2
THIRD_CONSTANT	0x10	This is a constant assigned the value 16

In this case three constants are declared. **Name** must follow the TI coding standard valid for the project to which the SAP belongs. The **Value** can be a positive or negative number, and can be assigned in decimal, hexadecimal or octal format using standard ANSI C-syntax. A binary format (e.g. 0b10101010) is also supported by the tool chain, with the binary value seen as right aligned (e.g. 0b1111 equals 0x0F). The **Comment** can be any text, which explains the purpose or usage of the constant. A comment field is not allowed to hold more than 256 characters.

In order to import constants from another SAP document, a **Link** column can be added to the definition table. The **Link** column links the parameter definition to a declaration in another document. The **Short name** from the definition in the external declaration must be included in the link specification as seen in the table below. Hyperlinks should be used if this is at all possible, and the path to the document linked to must be relative and not dependent on local mappings of Clearcase views, drives or similar things. An example of such a linked constant could be:

Definition:

Name	Value	Link	Comment
LOCAL_CONSTANT	1		Local constant
IMPORTED_CONSTANT		external.doc – name	Local constant with imported value

For constants imported using the **Link** column it is not possible to assign a value in the **Value** column, since the value will be taken from the external source. For a linked constant it is possible to have a empty **Name** column if no further local copy of the constant is to be used. The constant will then be exported to the file without being renamed via PREFIX. To support generation of other code standard interface header files a group column can be added. An example of such a column could be:

Definition:

Name	Value	Comment	Group
LOCAL_CONSTANT	1	Local constant	Pub_L1
IMPORTED_CONSTANT		Constant with imported value	Pub_Std

For more general information on groups, see section 2.1.2.4.

2.1.3.2 Definition for Primitives

For primitives the definition associates a primitive tag with an integer primitive identifier (**ID**), which must be unique within the system. This corresponds to defining a global constant using the "#define" pre-processor directive in C, linking the primitive name to the value of the **ID**.

Additionally, a primitive is always seen as a user defined structural type, since it is created for the purpose of passing data from one entity to another. The effect is the same as that of the definition of a parameter of content type **STRUCT** (see section 2.1.3.4). Consequently, the definition also corresponds to a type declaration in C, where a new structural type is declared with a body as defined by the "elements" part of the section (see section 2.1.4).

In order to establish the direction of the primitive on the SAP, and the entities involved in the use of it, the definition also specifies the sending and receiving entities.

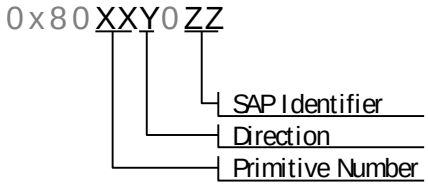
The name of the primitive is contained in the tabular definition under the heading **Short Name**, the global primitive identifier numerical value is contained under the heading **ID** and the direction is specified under the heading **Direction**. An example could be:

Definition:

Short Name	ID	Direction
ENTA_SOMETHING_REQ	0x08004001	ENTB -> ENTA

The S**hort Name** must follow the TI coding standard. A primitive may for instance be a request (_REQ), a confirm (_CNF), an indication (_IND) or a response (_RES).

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.

For an example of such a document see the UMTS document numbering scheme for UMTS (for an example see [TI SAP_NUMBERING_SCHEME]).

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.

The **Direction** field shows from which entity to which entity the primitive is sent. The direction is shown using an arrow in the form "->" or "<-". The field is informational only, and the direction of the arrow is not important. If more than one entity sends/receives the primitive on the SAP, additional direction specifications can be present in the cell. For primitive definitions the section headline should be the same as the S**hort Name**, since it makes it considerably easier to locate elements in the document by use of for instance the table of contents. **Group** columns are also supported in this part. For more general information on groups, see section 2.1.2.4.

2.1.3.3 Definition for Functions

The handling of functions 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. An example of the definition for a function on the SAP could be:

Definition:

Short Name	ID	Direction
extern T_RETURN_TYPE * ent1_function(T_ARGUMENT_TYPE * arg)	InlineC	ENT2 ->ENT1

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 (see section 2.1.3.2). 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 dec-

[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.

laration without interpretation, and it is the responsibility of the designer of the SAP to find the correct type names. The type names can be found from the elements contained in the declaration of the function (see section 2.1.4.3), combined with knowledge about the code generated by the tool chain based on elements and declarations (see section 2.1.5 and section 2.1.6) and knowledge about the TI coding standard. If one is not familiar with the conversion from SAP to source files one can simply have a look in the generated source files in order to find the desired types.

2.1.3.4 Definition for Parameters

The definition table for a parameter declaration associates a content type (see section 2.1.5) with the name of the parameter. The tabular definition for a parameter contains the content type of the parameter declared, the **Short Name** associated with the parameter and a **Comment** explaining the parameter contents. An example could be:

Definition:

Type	Short Name	Comment
U16	e_one	Element one parameter declaration

Or (notice the additional **C-Name** column):

Definition:

Type	Short Name	Comment	C-Name
U16	e_one	Element one parameter declaration	parameter_name

The **Short Name** of the parameter is also the name of a user-defined type (see section 2.1.7). Unless a non-empty **C-Name** cell is present in which case that is used (see Table 2). For more information on **C-Name,** see section 2.1.8.

It is possible to have multiple entries in the table, each with the same content type but with a new short name and comment. This allows the definition of several user defined types in one go, each unique declaration with identical content.

The **Comment** is outputted into the h-file and is used as a hint in some of the output formats from the TI tool chain.

The content type serves the purpose of identifying the content of the parameter as either a basic type (**U8**, **S8**, **U16**, **S16**, **U32** or **S32**), as an enumeration (**ENUM**) or as a complex type (**STRUCT**, **UNION**). Six legacy basic types are also supported (**UBYTE**, **BYTE**, **USHORT**, **SHORT**, **ULONG** and **LONG**). These are no longer to be used, but may still be found in older SAP documents.

If a basic type is used, the definition only defines the reference point between the declaration and the elements in other declarations using it. The content of the definition has no impact on the end product generated by the tool chain. If the content is a basic type the declaration can have no elements part and the elements part must not be present in the declaration.

If the content type is **ENUM**, the definition corresponds to a type declaration in C, where an enumeration type is declared with enumerators as defined by the value part of the section. The declaration can have no elements in this case.

If the content type is **STRUCT** or **UNION**, the definition corresponds to a type declaration in C, where a new type is declared with a body as defined by the "elements" part of the section. The elements part of the declaration must be present.

Multiple entries in the definition table are allowed if the declaration is referenced from different declarations in the SAP under different names. In that case an entry in the definition table must be present for each element **Short Name** used for the declaration. The content type must be the same for each entry. An example of this could be two different primitives using the same basic type with a different **Short Name**. This is illustrated in the following table:

Definition:

Type	Short Name	Comment	C_Name
U16	e_one	U16 element used by a primitive	
U16	e_two	U16 element used by another primitive	

Group columns are also supported in this part. For more general information on groups, see section 2.1.2.4.

When defining parameters it is also possible to import parameter declarations from other SAP documents. This can be done by addition of a **Link** column to the definition table. An example could be:

Definition:

Type	Short Name	Link	Comment
U16	E_one	external.doc – first_name	Import number one
STRUCT	E_two	external2.doc – second_name	Import number two

The **Link** column links the parameter definition to a declaration in another document. The **Short Name** from the definition in the external declaration must be included in the link specification. Hyperlinks should be used if this is at all possible, and the path to the document linked to must be relative and not dependent on local mappings of Clearcase views, drives or similar things. As can be seen from the example the content type does not need to be the same for each entry in the definition table when linking is used (both U16 and STRUCT used here). No further specification for the defined parameter can be given, since all the properties will be inherited from the link-source.

For each entry in the definition table a new type will be created based on the imported one, without restating the contents of the type (corresponding to a type declaration in C of the form "typedef T_IMPORTED T_LOCAL;"), in this case even for parameters where the content type is a basic type. Linking is also possible to contents included in the MDF format used by the tool chain, which makes it possible to use other sources than SAP documents when importing parameter declarations. The nicest way to perform the linking in the SAP would be to make a section in the Parameter section containing all the imported parameters. Whenever a parameter is used in the SAP the reference will lead to the imported parameters section. This way it is also easier to locate the linked parameters.

When a structure or type is to be used by several SAPs the smartest approach would be to create an include SAP which contains all the parameters/elements which are to be used by several SAPs. In this include SAP it is necessary to have a section in the primitives section called "Exported Parameters". In this section all the linked parameters should be listed An example of of such a section could be:

Definition:

Short Name	ID	Direction
EXAMPLE_INCLUDE_EXPORT	0x0000	ENTA->ENTB

Elements:

Long Name	Short Name	Ref	Type
Linked u8 for testing	linked_u8	4.1	U8
Linked struct for testing links	linked_struct	4.8	STRUCT

Here the references in the **Ref** column lead to he actual declarations of the types used, and therefore only needs to be defined once. This way it is also easy to change a structure used in several SAPs without having to change every single SAP.

2.1.4 Elements

The elements part contains a tabular representation of the elements contained within a complex type declared in the definition part of the section. Elements can only be used for primitives, functions and for parameters. An additional requirement for parameters is that the defined content type is either **STRUCT** or **UNION** (see section 2.1.5.3 and 2.1.5.4). For parameters the way elements are described depends on the complex type used in the definition, structure or union. For functions the description of elements has a slightly different syntax. In the following these different possibilities will be explained for respectively Structures, Unions and Functions.

2.1.4.1 Elements in Structures

For primitives, and parameters of type **STRUCT** (see section 2.1.5.3), the element table gives the same set of possibilities. An exception to this rule is that for primitives, the elements section must be present but can be empty except for the headline row. This option must be used in case the primitive contains no elements. It is not possible to define an empty element table when declaring a parameter. An example of an Elements table could be:

Elements:

Long Name	Short Name	Ref	Type
Unsigned 8 bit integer with local value set	8_bit_local		U8
Unsigned 16 bit integer	16_bit	4.1	U16
Unsigned 32 bit external	32_bit_external		U32
Enumeration with local value set	enumeration_local		ENUM
Enumeration	enumeration	4.2	ENUM
Structure	structure	4.3	STRUCT
Union	Union	4.4	UNION

Or e.g. (notice the additional **C-Name** column)

Elements:

Long Name	Short Name	Ref	Type	C-Name
Unsigned 8 bit integer with local value set	u_8_bit_local		U8	
Unsigned 16 bit integer	u_16_bit	4.1	U16	u_16_bit_a
Unsigned 16 bit integer	u_16_bit	4.1	U16	u_16_bit_b
Unsigned 32 bit external	u_32_bit_external		U32	

The **Long Name** can be seen as a comment, where the element is given a free form name, which should make its purpose or content as clear as possible. This is also used as the definition comment if there is no explicit definition for this **Short Name** i.e. used as a hint in some of the output formats from the TI tool chain.

The **Short Name** is the actual C identifier used for the element in the output from the tool chain, unless a non-empty **C-Name** cell is present in which case that is used. The **Short Name** (or **C-Name** if present) must be used when using the SAP in the implemented protocol stack entities.

The **Ref** column refers to the section where the parameter declaration for the element can be found. If the element does not have a parameter declaration associated with it, the reference is left empty. If there is a reference, the short name used for the element and the short name used in the definition part of the parameter declaration must be exactly the same.

The **Type** column contains the content type specifiers for the elements (see section 2.1.5). Again the content type specifier must match the one used in the definition of the parameter declaration.

In order to modify elements to become pointers or arrays, an additional **ctrl** column must be added to the element table (see section 2.1.6.1 and section 2.1.6.3). Array modification of elements of content type **UNION** is not allowed. An example could be:

Elements:

Long Name	Short Name	Ctrl	Ref	Type
Pointer to 8 bit integer	element_1	PTR	4.1	U8
Fixed array of 8 bit integers	element_2	[5]	4.2	U8
Variable array of 8 bit integers	element_3	[VAR_MIN..VAR_MAX]	4.3	U8
Code non-transparent dynamic array of 8 bit integers	element_4	PTR[3..VAR_MAX]	4.4	U8
Code transparent dynamic array of 8 bit integers	element_5	DYN[1..VAR_MAX]	4.5	U8

It should be noted, that in most cases the use of pointers implies the use of shared stores between the entities using the interface defined by the SAP. The description of a shared store is given through the parameter reference in the element table, but this does not cause allocation of such a store to occur. Any memory referenced by a pointer included in an element table will have to be allocated and handled independently of the SAP.

In order to modify the presence of elements to become optional, and additional **Pres** (short for presence) column must be added to the element table (see section 2.1.6.2). An example could be:

Elements:

Long Name	Short Name	Pres	Ref	Type
Mandatory element	element_1	Mandatory	4.40	U8
Optional element	element_2	Optional	4.41	U8

The two modifications (addition of **Ctrl** and **Pres** columns) can be combined.

2.1.4.2 Elements in Unions

For parameters of content type **UNION**, the set of possibilities is a subset of the possibilities for structures (see section 2.1.4.1). An example could be:

Elements:

Tag ID	Long Name	Short Name	Ref	Type
IS_8_BIT_LOCAL	Unsigned 8 bit integer	8_bit_local		U8
IS_16_BITS	Unsigned 16 bit integer	16_bit	4.1	U16
IS_ENUMERATION_LOCAL	Enumeration with local value set	enumeration_local		ENUM
IS_ENUMERATION	Enumeration	enumeration	4.2	ENUM
IS_STRUCTURE	Structure	structure	4.3	STRUCT

This would correspond to a union, where the element chosen could be one of the elements contained in the elements table. A **Tag ID** column is included in the element table for unions, which is used to name the instance of the union actually used, and is included along with the union in a control parameter automatically added in the generated C output by the tool chain (see section 2.1.5.4). The **Tag ID** can be either a string (e.g. “ABC”), a number (e.g. “12”) or a string equal to a number (e.g. “ABC = 12”). The usual style is to name the instances using a string. As for structures, the elements can be local if this is legal for the content type or declared as a parameter.

The main difference is in what cannot be done when the elements of a union are declared. A union cannot contain elements of **UNION** content type. If a union must be a part of another union, it must be contained in a structure declaration. Modifications of elements into variable size arrays are also not legal, although pointers can be used (see section 2.1.6.3). It is not possible to make elements in a union optional.

2.1.4.3 Elements for Functions

For functions the notation is similar to the one used for structures, although the use of arrays and optional parameters is not allowed and would not make any sense. An example could be:

Elements:

Long name	Short name	Ctrl	Ref	Type
Input: Argument type for the call	argument_type	PTR	5.1	U8
Output: Return type for the call	return_type	PTR	5.2	STRUCT

The elements part of the declaration of a function used on the SAP serves mainly the purpose of referring to the parameter declarations used by the function prototype given in the definition (see section 2.1.3.3). The references in the **Ref** column are necessary, since parameters declared but not referenced from another declaration of a primitive, function or parameter are not included in the output from the TI tool chain. The elements table should specify all the parameters given in the function prototype from the definition. If a function has no input or output arguments, no entry should exist in the elements table for the "empty" argument type. If no arguments are used at all, the elements table should be left out.

2.1.4.4 Values

This part can only be used with parameter declarations and contains values that must be associated with the parameter or its elements. The values may define legal ranges or identified constant values that are specific to the parameter or to its elements (e.g. the values to be used with a cause parameter). The main purpose of the range specifications is to allow range checking in the test tools used within TI.

A value part can only be included in the declaration of a parameter, if the definition indicates a basic type or a content type of **ENUM**, or if the elements part of the declaration contains elements of basic type or content type **ENUM** without references to other parameter declarations. If the values are to be associated with a basic type parameter or content type **ENUM**, where the declaration contains no elements part, an example could be:

Values:

Value	C-macro	Comment
0	E_ONE_VALUE_ZERO	Element one value zero
4	E_ONE_VALUE_FOUR	Element one value one
0..4		Element one range

The use of tags for the values in the c-macro column corresponds to the use of an enumeration associated with the parameter or element in C. If the values are to be associated with elements in a parameter of a user-defined type, it is necessary to specify the element name in the value part. An example could be:

Values:

Name	Value	Comment
e_two	-5-5	The element has a range from –5 to 5
e_three	0-4096	The element has a range from 0 to 4096

The **Name** used is the **Short Name** of the element given in the element table. Please note if no range is defined in the "Values" section the minimum and maximum for the given type is generated implicitly.

Group columns are also supported in this part. For more general information on groups, see section 2.1.2.4. When using groups in this part please note that if the **Name** cell contains a value then the **Group** cell must contain one too. Such a **Group** value specifies the **Group** for the whole user defined type with the name in the **Name** cell. Thus not possible to specify a different **Group** for the individual values associated with a **Name**. If the **Name** cell is empty the **Group** cell must be empty too.

2.1.4.5 History

The history part contains the history of the individual section it is a part of. The history is simply a list of entries, each consisting of date, initials and a comment for each change to the section. As the description, the history is a mandatory part of all sections, since it contributes to the documentation included in the SAP. Therefore it is very important to maintain the History part as will make it possible to keep track of changes to the specific section.

2.1.5 Content Types

The TI tool chain allows for the use of several different predefined C content types when declaring the content of primitives, parameters and functions. These types vary from simple integer to more complicated types such as enums, structs and unions and will be described in the following section. The content type used when declaring the content of primitives, parameters and functions must be one of the sets described in the following sections. In addition to this the following sections will also describe the mapping from SAP declarations into the generated C code. This will be done by some small C code examples.

2.1.5.1 Basic Types

The basic types that can be used for parameters and elements are **U8**, **U16**, **U32**, **S8**, **S16** and **S32**. The prefix letter indicates whether the type is signed or unsigned, and the number the size of the type in bits. No floating-point basic types are defined. The TI tool chain also supports sets of legacy basic types (**UBYTE**, **BYTE**, **USHORT**, **SHORT**, **ULONG**, **LONG**). These are no longer to be used, but may still be found in older SAP documents.

A parameter of this content type cannot have elements, but can have values associated with it. An element of this content type can refer to a declaration of the same content type, or can have values associated with it.

When used as the content type of an element in a structure (see section 2.1.5.3) or a union (see section 2.1.5.4), the result in C will be:

```
U8 short_name;                  /* Comment */
S8 short_name;                  /* Comment */
U16 short_name;                 /* Comment */
S16 short_name;                 /* Comment */
U32 short_name;                 /* Comment */
S32 short_name;                 /* Comment */
```

where short name is the one given for the element.

2.1.5.2 Enumerations

In order to declare an enumeration the **ENUM** content type must be used in the definition of the parameter. In that case, the content type must also be used for the element referring to the parameter declaration (in the declaration of a primitive or another parameter).

A parameter of this content type cannot have elements, but must have values associated with it. When used as content type in the definition of a declaration the end result in C-code will be:

```
typedef enum
{
  C_MACRO_1 = 0x00,                 /* Comment */
  C_MACRO_2 = 0x01,                 /* Comment */
  . . .
  C_MACRO_N = 0xNN                  /* Comment */
} T_SHORT_NAME;                     /* Comment */
```

Where the short name is used for the name of the type. The enumerators are taken from **c-macro** column of the value part, and therefore do not necessarily start with the value 0x00 and may have gaps. An element of this type must refer to a declaration of a parameter of the same content type, or have values associated with it. When used as a content type for an element in a structure (see section 2.1.5.3) or a union (see section 2.1.5.4), the result in C will be:

```
T_SHORT_NAME short_name;         /* Comment */
```

where short name is valid for both the element itself and the parameter definition referred to.

2.1.5.3 Structures

In order to declare a parameter containing multiple elements, the **STRUCT** content type must be used in the definition of the parameter. The type must then also be used for the element referring to the parameter declaration (in the declaration of a primitive or another parameter).

A parameter of this content type must have elements, and can have values associated with these elements. When used as a content type in the definition part of a parameter declaration, the resulting structure in C will be:

```
typedef struct
{
  U8 element_1;                 /* Comment */
  T_element_2 element_2;        /* Comment */
  T_element_3 element_3;        /* Comment */
} T_SHORT_NAME;                 /* Comment */
```

Where **Short Name** from the definition is used for the type name. For elements of basic type no user defined type is generated, irrespective of whether a reference to a declaration is given or not.

An element of this content type must refer to a declaration of the same content type. When used as a content type for an element in another structure or in a union (see section 2.1.5.4), the result in C will be:

```
T_SHORT_NAME short_name;              /* Comment */
```

where short name is valid for both the element itself and the parameter definition referred to.

2.1.5.4 Unions

In order to declare a parameter containing one of a set of elements, the **UNION** type must be used in the definition of the parameter. The type must also be used for the element referring to the declaration (in the declaration of a primitive or another parameter).

A parameter of this content type must have elements, and can have values associated with these elements. When used as the content type in the definition part of a parameter declaration, the result in C will be:

```
typedef enum
{
  TAG_ID_ELEMENT_1,
  TAG_ID_ELEMENT_2,
  TAG_ID_ELEMENT_3
} T_CTRL_SHORT_NAME;

typedef union
{
  U8 element_1;                      /* Comment */
  T_ELEMENT_2 element_2;             /* Comment */
  T_ELEMENT_3 element_3;             /* Comment */
} T_SHORT_NAME;                      /* Comment */
```

Where **Short Name** from the definition is used for the type names and as a qualifier for the enumeration. The enumeration has one enumerator for each possible element choice in the union, the name of which is taken from the **Tag ID** column (see section 2.1.4.2).

An element of this content type must refer to a declaration of the same content type. When used as a content type for an element in another structure, the result in C will be:

```
T_CTRL_SHORT_NAME ctrl_short_name;   /* Comment */
T_SHORT_NAME short_name;             /* Comment */
```

Where **Short Name** is valid for both the element itself and the parameter definition referred. When used, the "controller" named ctrl_short_name is used to indicate actual choice of element within the union, using the tag ids listed in the **Tag ID** enumeration. This also explains why an element of content type UNION cannot be used in a parameter, which is also of content type **UNION**. The tool chain would insert a controller for the illegal union along with the illegal union itself inside the generated legal union type, which means that the illegal controller and union would share the same memory.

2.1.6 Element Modifiers

The tool chain allows elements in declarations to be modified in order to achieve different constructions, such as arrays, pointers and optional parameters.

2.1.6.1 Making Elements Arrays

Arrays can only be used in the elements part of a declaration of a primitive or a parameter with the content type **STRUCT**. Arrays are possible by addition of a column to the elements table with the heading **ctrl**. For each element in the table that is

to be an array, a length specifier is included in the **ctrl** column. All elements in the array will have the same content type as declared for the element, and the same user defined type if a reference to another declaration exists for the element.

The length specifier always has the form:

[DYN|PTR][MINIMUM_ELEMENT_NUMBER[..MAXIMUM_ELEMENT_NUMBER]]

giving the possibility of specifying an array with either a fixed or a variable number of elements. The **MINIMUM_ELEMENT_NUMBER** can be an integer or the name of global constant. It must have a value larger than zero and must be smaller than the **MAXIMUM_ELEMENT_NUMBER** if a variable array is specified. The **MAXIMUM_ELEMENT_NUMBER** can be an integer or the name of a global constant, and must be larger than the minimum element number.

If the array is to have a fixed number of elements only a minimum element number is used. For fixed size arrays the array specification for an element results in the C expression:

```
T_SHORT_NAME short_name[MINIMUM_ELEMENT_NUMBER];
```

where short name is valid for both the element itself and the parameter definition referred to. For basic types, no user defined type is used, and T_SHORT_NAME is replaced by the content type specified for the element.

If the number of elements is to be variable, both a minimum and a maximum number of elements is used. For dynamic size arrays the use of the keyword **DYN**[2] or **PTR** is also possible, which allows the specification of an array where the amount of memory allocated is dynamically dependent on the number of elements. For variable size arrays (without the use of the keyword **DYN** or **PTR**) the array specification for an element results in a slightly different construction of the C generated:

```
U8 c_short_name;
T_SHORT_NAME short_name[MAXIMUM_ELEMENT_NUMBER];
```

For dynamic size arrays (where either the keyword **DYN** or **PTR** is used) the construction is similar. If the keyword **DYN** is used:

```
U8 c_short_name;
T_SHORT_NAME * short_name;
```

and if the keyword **PTR** is used:

```
U8 c_short_name;
T_SHORT_NAME * ptr_short_name;
```

In all cases the parameter c_short_name contains information about the number of elements in the variable size array. The type of c_short_name depends on the value of `MAXIMUM_ELEMENT_NUMBER`. Due to this additional parameter, variable size arrays cannot be used in declarations of parameters with content type **UNION**.

If the keywords **DYN** or **PTR** are not used, the array is actually declared of maximum size, and only the array elements [0; c_short_name - 1] contain valid data.

As can be seen from the C examples, the behaviour of **DYN** and **PTR** when specifying dynamic size arrays is very similar. In both cases the result is the generation of a pointer to the type specified, in which the elements can be accessed in the same way as for a normal array, since the C syntax will be the same. Between **DYN** and **PTR** declarations only the naming of the pointer is different. The main difference to the fixed or variable size array, is that for dynamic arrays the memory containing the data is actually not allocated at the position of the element in the element table of the declaration. Instead the pointer references the memory allocated for the array, which will have the size necessary to hold the number of entries indicated by

[2] The DYN keyword is intended to be used for future memory optimizations.

c_short_name. The difference in the use of the elements between **DYN** and **PTR** declarations lies more in the behaviour when elements are declared optional (see 2.1.6.2). Please note that it is not possible to make array of pointers.

2.1.6.2 Making Elements Optional

Elements contained in the declaration of primitives or parameters of content type **STRUCT** can be made optional by the addition of a column with the heading **Pres** to the elements table. For each element, a presence specifier is then added in the **Pres** column. The presence specifier is either **OPTIONAL** or **MANDATORY**. For mandatory elements, this has no consequence for the C construction, but for optional elements the result is:

```
U8 v_short_name;
T_SHORT_NAME short_name;
```

A valid flag v_short_name is added to the construction. The value of the parameter v_short_name indicates whether the content of the element is valid or not. The value can be 0 for not valid or 1 for valid. When the keyword **PTR** is used in a **Ctrl** column for an optional element, either for a dynamic size array (see section 2.1.6.1) or for a pointer (see section 2.1.6.3), no valid flag is added. Instead the element is not present if the value of the pointer is NULL.

2.1.6.3 Making Elements Pointers

Pointers can be used as elements in declarations of primitives, functions and parameters of content type **STRUCT** or **UNION**. Pointers are made possible by addition of a column to the table in the elements part of the declaration with the heading **Ctrl**. For each element in the table that should be a pointer, the keyword **PTR** is added in the **Ctrl** column, causing the C construction generated to become:

```
T_SHORT_NAME * ptr_short_name;
```

Where short name is valid for both the element itself and the parameter definition referred. If the element is of content type **UNION** (see section 2.1.5.4), the C construction becomes:

```
T_CTRL_SHORT_NAME ctrl_short_name;   /* Comment */
T_SHORT_NAME * ptr_short_name;       /* Comment */
```

Allowing identification of the element chosen in the union pointed to. Pointers are primarily used in order to share stores between the entities using a SAP, passing only the reference to the memory where the store is to be found between them. The stores will however need to be allocated independently of the SAP. When using the PTR keyword it is very important to be aware of the consequences. One extremely important point is that the entire store pointed to is traced out. This may result in insufficient bandwidth when testing and therefore should be used with care.

2.1.7 User Defined Types

A user-defined type in the context of a SAP specification corresponds to the declaration of a parameter of content type **STRUCT**, **UNION** or **ENUM** (see section 2.1.5). Within the SAP specification, the name of the user-defined type is the short name given in the definition part of the declaration (see section 2.1.3.4).

Using an instance of the user defined type corresponds to adding an element in the declaration of a primitive or another parameter. The element must refer to the section where the user-defined type is declared. This reference can also be to an external declaration, e.g. in another SAP specification (see section 2.1.3.4).

Whenever a reference is made, the short name of the element must be the same as the short name used in the declaration of the parameter (the user defined type). This means that in the context of the SAP specification, the name of an element and the name of its type is the same, if the type is user defined (declared and referenced as a parameter).

As such a construction is illegal in C, the tool chain will generate a C-style user defined type based on the declaration of the parameter. The name of the user-defined type will be based on the short name used in the declaration, following the TI coding standard (currently "T_X", where X is the short name used in the declaration).

2.1.8 Description of C-Name

In SAP documents it is possible to determine the resulting code name of the defined types by the use of the column **C-Name**. This feature is called C-Name because it allows the developer to determine the type name used in the resulting C code. This feature can for instance be used to generate another name for the parameter. That is, the defined type when the **C-Name** is non-empty will not result in a type named after the **Short Name** but named after the **C-Name** in the resulting code file. This way it is possible to use the same user defined type in more than one primitive without having identical names. For more detailed information on whether the **Short Name** or **C-Name** is used, see section 2.1.8.

2.1.9 "Definition C-Name" versus "Element C-Name"

The approach should be as follows:

If 2 elements are basically different, c-names should be used in **definitions**. This is the most common case.

If 2 elements are basically the same, c-names should be used in **elements**, the main effect is that the tools see only one type i.e. the 2 elements will have the same comment in the trace.

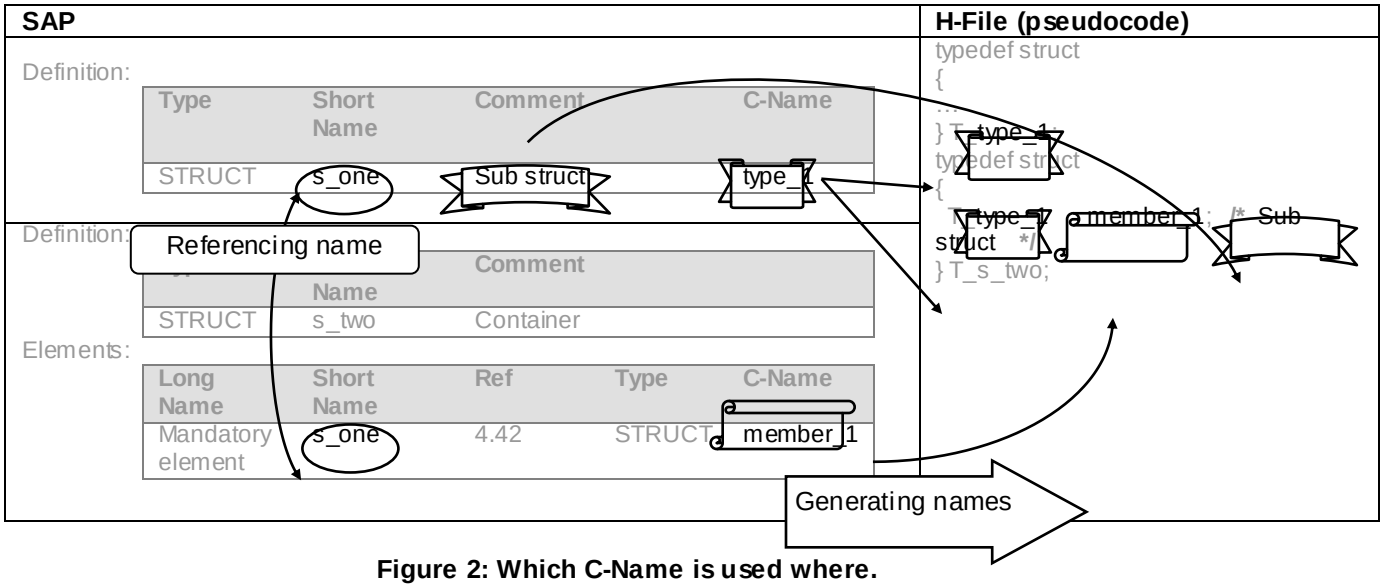


Figure 2: Which C-Name is used where.

Note: if no **C-Name** is present then the **Short Name** is used for that generation.

Presence of C-Name	Used for the type name of the member	Used for member name
C-Name present in both element table and referenced definition table.	**C-Name** from referenced definition table	**C-Name** from element table
C-Name only present in element table.	**Short Name**	**C-name** from element table
C-Name only present in referenced definition table.	**C-Name** from referenced definition table	**Short Name**
C-Name not present in element table neither in referenced definition table.	**Short Name**	**Short Name**

Table 2: Which C-Name is used where. This schema is valid also when the referenced definition

table is external (imported definition table).

2.1.10 Interactions

The following is an attempt at providing an overview of the interactions between the different sections, parts, content types and element modifiers used in the SAP document. The interactions are indicated in a table (see Table 3).

● Used/Supported
◉ Used/Supported with limitations
○ Not Used/Supported

		Active Section																
		Constants				Primitives				Functions				Parameters				
		Description	Pragma	Definition	History	Description	Definition	Elements	History	Description	Definition	Elements	History	Description	Definition	Elements	Values	History
Free Text		●	○	○	○	●	○	○	○	●	○	○	○	●	○	○	○	○
Date, Initials, Comment		○	○	○	●	○	○	○	●	○	○	○	●	○	○	○	○	●
C-Macro		○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○
Type																		
	U8	○	○	○	○	○	○	●	○	○	○	●	○	○	●	●	○	○
	S8	○	○	○	○	○	○	●	○	○	○	●	○	○	●	●	○	○
	U16	○	○	○	○	○	○	●	○	○	○	●	○	○	●	●	○	○
	S16	○	○	○	○	○	○	●	○	○	○	●	○	○	●	●	○	○
	U32	○	○	○	○	○	○	●	○	○	○	●	○	○	●	●	○	○
	S32	○	○	○	○	○	○	●	○	○	○	●	○	○	●	●	○	○
	ENUM	○	○	○	○	○	○	●	○	○	○	●	○	○	●	●	○	○
	STRUCT	○	○	○	○	○	○	●	○	○	○	●	○	○	●	●	○	○
	UNION	○	○	○	○	○	○	●	○	○	○	●	○	○	●	◉	○	○
Comment		○	●	●	○	○	●	○	○	○	●	○	○	○	●	○	●	○
Ctrl																		
	[MIN]	○	○	○	○	○	○	◉	○	○	○	◉	○	○	○	◉	○	○
	[MIN..MAX]	○	○	○	○	○	○	◉	○	○	○	◉	○	○	○	◉	○	○
	PTR[MIN..MAX]	○	○	○	○	○	○	◉	○	○	○	◉	○	○	○	◉	○	○
	PTR	○	○	○	○	○	○	●	○	○	○	●	○	○	○	●	○	○
Direction		○	○	○	○	○	●	○	○	○	●	○	○	○	○	○	○	○
ID	*Value*	○	○	○	○	○	●	○	○	○	●	○	○	○	○	○	○	○
	InlineC	○	○	○	○	○	○	○	○	○	●	○	○	○	○	○	○	○
Link		○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○
Long Name		○	○	○	○	○	○	●	○	○	○	●	○	○	○	●	○	○
Name		○	●	●	○	○	○	○	○	○	○	○	○	○	○	○	●	○
Pres[3]																		
	Mandatory	○	○	○	○	○	○	●	○	○	○	○	○	○	○	●	○	○
	Optional	○	○	○	○	○	○	◉	○	○	○	○	○	○	○	◉	○	○
Ref		○	○	○	○	○	○	●	○	○	○	●	○	○	○	●	○	○
Short Name		○	○	○	○	○	●	●	○	○	●	●	○	○	●	●	○	○
Value		○	●	●	○	○	○	○	○	○	○	○	○	○	○	○	●	○
Group		○	○	●	○	○	●	○	○	○	●	○	○	○	●	○	◉	○
C-Name		○	○	○	○	○	○	●	○	○	○	○	○	○	●	●	○	○

Table 3 - Support for table columns, content types, keywords and element modifiers.

[3] Only first letter of cell context is significant and case is ignored, i.e. M == Mandatory

2.1.11 Output consequences of using the Group column

When using **Group** columns the output h-files are named according to the group names. That is, the original SAP name does not affect which output files a type is generated in. The group name is used for prefix generation as well (pragma **PREFIX** is ignored if **Group** columns are present).

Apart from the names of the output files there are some small differences in the code output. The generated names are modified e.g.: for struct member names underscores in the **Short Name** (or **C-Name**) is replaced with next character upper-case. An example of this could be:

test_set → testSet

The automatically generated "v_" prefix is replaced with a "Valid" suffix. An example of this could be:

v_test_set → test_set_Valid

The automatically generated "c_" prefix is replaced with a "Counter" suffix. An example of this could be:

c_test_set → test_set_Valid

In addition to this arrays are also changed with an "Array" suffix. An example of this could be:

test_set → test_set_Array

3 Typical Problems

The following is a list of typical problems encountered when working with SAP specifications:

- **A declared parameter is syntactically correct but does not result in any output from the tool chain**

  Two things can cause this. The parameter is of a basic content type and does not have a values part. Alternatively the parameter declaration is not referred to from any elements in other declarations of primitives, functions or parameters.

- **When the SAP document is run through the tool chain an error file is generated, but the error is not specified**

  This problem is typically caused by a missing history part for one or more declarations. The history part must be present for each section containing a declaration.

- **Some of the content in the document does not appear in the output from the tool chain, and there are no errors or warnings**

  This can be caused by document content which is not recognised by the tool chain in the context where it is used. In general unrecognised content is ignored by the tool chain, and will not necessarily cause errors or warnings to be generated.

- **Problem in translation of a table in part of the SAP**

  The use of new-line inside table cells is not allowed. If necessary, use CTRL-TAB instead. The first character in a cell is not allowed to be a space (blank character).

- **Comment fields are causing problems**

  A comment field is not allowed to hold more than 256 characters.

4 Examples

An example SAP specification, using the various constructions described in this document, can be found in the document document [TI 8434.404] and [TI 8434.405].

Appendices

A. Acronyms

DS-WCDMA Direct Sequence/Spread Wideband Code Division Multiple Access

B. Glossary

International Mobile Telecommunication 2000 (IMT-2000/ITU-2000) Formerly referred to as FPLMTS (Future Public Land-Mobile Telephone System), this is the ITU's specification/family of standards for 3G. This initiative provides a global infrastructure through both satellite and terrestrial systems, for fixed and mobile phone users. The family of standards is a framework comprising a mix/blend of systems providing global roaming. <URL: http://www.imt-2000.org/>