SAE The Engineering Society For Advancing Mobility Land Sea Air and Space®
INTERNATIONAL
400 Commonwealth Drive, Warrendale, PA 15096-0001

SURFACE VEHICLE STANDARD

SAE J2540

ISSUED JUL2002

Issued 2002-07

Messages for Handling Strings and Look-Up Tables in ATIS Standards

Foreword—This SAE Standard outlines the methodology and messages used to convey textual strings of messages to consumers in ATIS and other areas. It outlines how concepts originally developed for SAE J2369 can be expanded to cover the various types of ITIS messages used in US deployments in a consistent way and meet a number of goals. Among these goals is overcoming the historical problems of obsolete tables without requiring the mutual agreement and updating of all deployed users. It provides implementation details for deployments wishing to use tables in ways ranging from non-changing ("static") embodiments, to highly flexible and changing table structures ("dynamic"), and provides a single uniform formatting standard for all (regardless of specific content). In the related documents are encodings of selected national level tables used in various deployments.

See also:

SAE J2540-1 – RDS Phrase Lists
SAE J2540-2 – ITIS Phrase Lists
SAE J2540-3 – National Names Phrase Lists

TABLE OF CONTENTS

TO PLACE A DOCUMENT ORDER:
Tel: 877-606-7323 (inside USA and Canada)
Tel: 724-776-4970 (outside USA)
Fax: 724-776-0790
Email: custsvc@sae.org
SAE WEB ADDRESS: http://www.sae.org

1. Scope—This SAE Standard defines methods and messages to efficiently translate sequences of text and other types of data into and out of indexed values and look-up tables for effective transmission.

This document defines:

a. Methods and Data Elements for handling indexes and strings in ATIS applications and message sets
b. Message Sets to support the delivery and translations of tables used in such strings
c. Tables of Nationally standardized strings for use in ATIS message descriptions

And examples of each in illustrative portions.

While developed for ATIS use, the methods defined in this document are useful for any textual strings in any Telematics applications found both in Intelligent Vehicles and elsewhere.

2. References

2.1 Applicable Publications—The following publications form a part of this specification to the extent specified herein. Unless otherwise indicated, the latest version of SAE publications shall apply.

2.1.1 SAE PUBLICATIONS—Available from SAE, 400 Commonwealth Drive, Warrendale, PA 15096-0001.

SAE J2313—On-Board Land Vehicle Mayday Reporting Interface, May 2000
SAE J2353—Data Dictionary for Advanced Traveler Information Systems (ATIS), October 1999
SAE J2354—Message Sets for Advanced Traveler Information Systems (ATIS), October 2000
SAEJ2369—Standards for ATIS Message Sets Delivered Over Reduced Bandwidth Media, November 2000
SAEJ2374—Information Report based on Location Reference Message Specification, Rev. B (MDI), May22, 1997
SAE J2540-1—RDS Phrase Lists
SAE J2540-2—ITIS Phrase Lists
SAE J2540-3—National Names Phrase Lists

2.1.2 ANSI PUBLICATION—Available from ANSI, 25 West 43rd Street, New York, NY 10036-8002.

ANSI X3.4—American National Standard Code for Information Interchange

2.1.3 CEN PUBLICATIONS—Available from CEN, 36 rue de Stassart, B-1050 Brussels, Belgium.

Traffic and Traveller[1] Information (TTI) – TTI Messages via Traffic Message Coding- Part I: Coding protocol for Radio Data Systems – Traffic Message Channel (RDS-TMS) using ALERT-C, ISO/DIS 14819-1, August 1999
Traffic and Traveller Information (TTI) – TTI Messages via Traffic Message Coding- Part 2: Event and Information codes for Traffic Message Channel (TMC), PrENV 12313-2, Version 1.0, June 1996

2.1.4 GATS PUBLICATIONS—Available from the GATS forum or from CEN TC278.

GATS Message Set; Main Part Main 2034.pdf 05.08.99
GATS Message Set; Annex 2 - Basic Information Elements A2_2033.pdf 05.08.99

2.1.5 ISO PUBLICATION—Available from ANSI, 25 West 43rd Street, New York, NY 10036-8002.

ISO prEN 14819-1 (1999)—

1. European spelling of Traveler

2.2 Related Publications—The following publications are provided for information purposes only and are not a required part of this document.

2.2.1 SAE PUBLICATIONS—Available from SAE, 400 Commonwealth drive, Warrendale, PA 15096-0001.

SAE J2353—Data Dictionary for Advanced Traveler Information Systems (ATIS), October 1999
SAEJ2374—Information Report based on Location Reference Message Specification, Rev. B (MDI), May 22, 1997

2.2.2 FHWA PUBLICATIONS—Available from Lee Simmons, FHWA, HVH-1 Room 3400, 400 7th Street SW, Washington, D.C. 20590.

National ITS Architecture, Federal Highway Administration, U.S. Department of Transportation, Version 2.0, September 1998

2.2.3 ITE PUBLICATIONS—Available from ITE, 525 School Street SW, Suite 410, Washington, DC 20024.

Message Sets for External Traffic Management Center Communication (MS/ETMC2), TM 2.01, Rev. 2.0, February 1, 2001

2.2.4 W3C PUBLICATIONS—Available from W3C, http://www.w3.org.

"Hypertext Transfer Protocol -- HTTP/1.1," W3C - June 1999, RFC 2616, available at http://www.w3.org/Protocols/rfc2616/rfc2616.txt
"HTML 4.0 Specification," W3C - December 1997, revised 24 April 1998 available at http://www.w3.org/TR/1998/REC-html40-19980424
HTML 4.0 Guidelines for Mobile Access W3C Note - 15 March 1999 available at http://www.w3.org/TR/NOTE-html40-mobile
Character Model for the World Wide Web, World Wide Web Consortium Working Draft 29-November-1999 available at http://www.w3.org/TR/1999/WD-charmod-19991129/.

3. Definitions—The following terms and concepts are used in the body of this document.

3.1 Alert-C—A traffic message system using *phrase - index* encodings operating over RDS and established in 1990, see ISO prEN 14819-1 (1999) for current information. Along with a more advanced system called Alert-Plus (and typically a pay for service system) this system has been deployed in selected major cites in Europe. Typically under local government sponsorship. A number of early US deployments by Enterprise States have used derivative versions of the phrases established in this work for trial US deployments. A number of commercial vendors support derivative forms of it, both on RDS and on other media as well.

3.2 Compression—The process of replacing a sequence of simple text elements (or other strings) with a small index value for transmission is considered to be compression in this document.

3.3 Dense Table—A table where the index values increase by one each time without a break in the ordering. The index value can be implicitly computed and therefore need not be sent. Dense tables are preferred, and are easier to automatically generate than sparse tables.

3.4 Dynamic Table—A table which changes its contents periodically as a function of the textual information which is being transmitted and which attempts to reflect to achieve the best compression in the media. As contrasted with static tables which only change periodically. Strategies affecting the frequency of updating vary with implementation, media, and ISP operators' choices. Dynamic tables convey their changes over the same media as the messages using them. This is an essential aspect of their operation.

3.5 End Using Device—A term used in this document to refer to the device which received the message and strings that are a subject of this document. The capabilities of such a device may vary widely from one user to another. The device is presumed to be mobile, but need not be. The end using device gets its data from the issuing ISP over the host media.

3.6 Exception—A token which causes some change in the flow of control in processing the stream. In this document, exception tokens precede an index value and are used to select which table (and how) to expand from.

3.7 Expansion—The process of replacing an index value found in a string with the corresponding string or text value which it represents.

3.8 Expansion Code—The code which precedes an index value (also called an exception token). This code selects which of up to four tables is to be used in expanding the string back into text and provides some control flags for how the expansion is to occur.

3.9 Extending a Table—The act of adding additional index entries to the end of an existing table. All tables can be extended. By this means new entries are added as needed while preserving the older index values. Periodically a *resort* may occur which reassigns the mapping of index values. The revision bits allow the end user to determine if a table has been extended and/or resorted.

3.10 Flags—One or more bit values contained in some part of a message which effect the way a string is processed in converting it back into text. In the table message there are flags dealing with how the table itself, and how any included tables, are to be handled.

3.11 Flat Table—A table whose entries do not contain indexes to any other tables, i.e., a table which is entirely self contained. A flat table may contain nested table entries to itself.

3.12 GATS—Global Automotive Telematics Standard. A mobile phone system based on the GSM (Global System for Mobile Communication) phone system deployed in Germany and other areas of Europe. The GATS systems provides a message set which offers many of the same services in US ITS-ATIS. GATS is being advanced by the CEN TC278 process as a standard.

3.13 Group 8A—The original message type used in RDS to carry TMC messages in a complete 37-bil message. Now such messages can be carried in any valid spare group by means of a soft assignment process called ODA.

3.14 Host Medium—The host media are the transmission media that was used to transport the ATIS messages containing the string from the ISP to the end-using device (e.g., wireless data links, SubCarriers, digital phones, etc.). Often such media have a limited amount of bandwidth and it is for this reason that the string handling methods of this document were developed.

3.15 Included Table Flags—Flags providing information about tables used in the entries of this table, found in the table header.

3.16 ISP—Information Service Provider. In the context of this document the issuing source of data formatted into ATIS messages containing strings handled in accordance with this document. No presumption is made in this document regarding the business model or ownership of the ISP. The term ISP (as opposed to TMC) is often used to imply a private “for profit” venture. No such term is implied here.

3.17 ITIS—International Traveler Information System A term coined to indicate the area of common phrases for describing incident events.

3.18 Local Replacement—A term for a table entry which "overlaps" another entry in another table with the same index value. This allow the creation of local tables which can redefine a previous entry. See also "overlapping" and 5.3 of the text.

3.19 Local Table Flags—Flags providing information about the tables found in the table header.

3.20 Local Table Number—A value assigned to any table used by the ISP in its messages. Used to relate specific strings to this table (and any underlying tables).

3.21 LRMS—Location Referencing Message Set A body of work outlining a number of "profiles" used in the description of affected areas, typically roadway segments. The preferred methodology for use in ATIS is the ISP → Vehicle profile (SAE J1746) which provides a means to send Lat-Long values between the ISP and the vehicle. A highly compressed form of this is the GRID profile which is suitable for limited bandwidth applications. Indexing values, which typically require prior agreement between users, are called Geometry profiles in this body of work. The LMRS is expected to become an SAE Standard in the future and will reflect all the profiles then used, including indexes.

3.22 Modified ASCII—This is a derivative of the ASCII character set specified in detail in SAE J2369. It uses the upper encoding values (beyond 127) to replace the most commonly occurring double letter pairs with a single value, effectively using all 8 bits of the octet and providing a compression of up to 25%. The upper range is also used to represent the exception callout token used to select an index table. See SAE J2369 for complete details.

3.23 Nested Table—A table whose contents (entries) are made up of strings consisting of simple text and indexes to other tables. This is the most typical type of table (as opposed to a "flat" table).

3.24 Nested Table Entry—A specific table entry, which is made up (in part) of other table entries from the same table. This type of recursion typically occurs when there is a need for a collection of related phrases.

3.25 ODA—Open Data Applications, an improvement to the original RDS formatting which allows an application to register and use any unused message frame in a local RDS system. Developed to manage the use of the original 64 possible messages. In the US market one can reasonably expect broadcasting of traffic over RDS to use either ODA and the tables in this annex combined with index aspects of the LRMS system (as opposed to say Alert-Plus methods) or to use only Group 8A messages. ODA provides a way to send incidents in one channel and supporting table description data in another.

3.26 Overlapping—Tables may overlap each other in entries. This occurs when a table (typically a local table) is mapped into an index space of another table. Because the search order of table is deterministic, this effectively replaces any table entry further in the search order with the local entry. This is often used to provide a local expression for an entry. Refer to 5.3 for additional information.

3.27 RBDS—See RDS.

3.28 RDS—Radio Data System, also called Radio Broadcast Data System (RBDS) in the US. A modulation and message framing standard for sending low speed (1187.5 BPS) data over the sub carrier area of commercial FM broadcast standards. It was developed the in mid-80's and used to send a variety of information packets. It allows a fraction of this capacity (~37 BPS) to carry traffic messages at the rate of about one per second. This portion, called Traffic Message Channel (TMC), employed a two part ATIS message wherein one part was an index table of location references and the other part was an index into a table textual description of the event. The precise table values used have varied significantly over the past 15 years. The location numbering has invariably been redeveloped by each deployment. The phrases, while also varying to some degree, have been stable and are being advanced in the CEN TC204 process. They are commonly called Alert C and Alert+. In the parlance of this document, the LRMS format used would be called a an index type of profile, while the format used would be considered static one. A recommendation for using this media and the TMC format while keeping within the goals of this document is provided in Section 8.

3.29 Registration Table Number—A national assigned and maintained number (by the data registry process) used to identify any table of strings. There is a prearranged range for nationally reused tables, and also one for local tables. There is a range of numbers assigned to dynamic tables, which are never registered.

3.30 Resorting a Table—Tables may be extended by adding new values at the end. This process provides a means to add, but not to remove, entries. Periodically entries may be *re-sorted* as well. At this time entries may be removed or added. Resorting may change the mapping of index value to entries. Typically resorting results in alpha ordering; however this is not a requirement (and is typically of more value to the ISP in determining strings than to the end user re-expanding them). The revision bits allow the end user to determine if a table has been extended and/or resorted.

3.31 Revision Bits—An 8 bit version control value used to detect extending or re-sorting a table. The lower 4 bits (providing 16 values) are incremented whenever a table is extended. The upper four bits are incremented (and the lower four are set to zero) whenever a table is resorted.

3.32 Search Order—The order in which tables are processed to expand an index value. The search order is critical for cases where overlapping is used to insure that the proper table entries are used. The search order is implied by the order in which tables are added to the header sections, with the search following the order in which they appear.

3.33 Simple Text—Text found in a string which is composed of basic characters. A string is made up of a sequence of one or more table index entries (typically preceded by an expansion code for each table) and surrounded by simple text as needed. Simple text can be viewed as the “glue” of sentences which did not get compressed into a table entry. Note that if the character set being used is modified ASCII, then the simple text may include the various two-letter compressions in it. Also called “raw text” at times.

3.34 Sparse Table—A table where the index values are discontinuous and therefore must be explicitly sent with the table entries. Sparse tables allow the preservation of historical encoding rules at the expense of being able to automate the creation and maintenance of the tables and result in a larger size when transferred.

3.35 Static Table—A table which does not change or only changes its contents very rarely. As contrasted with dynamic tables which may change frequently while in use. Static tables may convey their changes over the same media as the messages using them, but many media do not (or cannot) provide this ability. In many deployments it is expected that static tables which cannot be sent over the media will be obtainable by other means. For example, the tables used in an RDS-TMC system might be obtainable over an Internet site. Often static table are re-expanded into forms other than text (such as a prerecorded voice). The prior agreement that the index value representing the expansion does change is vital to ensuring that the alternative media will continue to work.

3.36 Strings—A sequence of (multi-sized) data values, which can be converted to and from simple text. In order to do this the underlying character and/or string tables’ sets must be understood. The specific rules and resulting messages to achieve this form the primary content of this document. In this document the term *string* is used to refer to the encoded source material while the pre and post coded source is referred to as *text*. Strings are almost always made up a sequence of simple text and of one or more table entries, although this is not a requirement.

3.37 Table—A table consists of two parts: *the table entries* and the *table header*. In devices receiving and processing ATIS messages, tables are used to replace index values with strings in the process of converting strings back into text. This reverses the compression process that was performed at the ISP before the string was transmitted over the host media.

3.38 Table Entries—The actual string values related to an index value of a table. The table entries follow the table header in the construction of a table.

3.39 Table Header—A multi-octet section at the start of a table which provides information on the registration of the table, the table content (other tables), the character set used and the number and length of entries.

3.40 Text—A sequence of one or more octet long data values which, taken together, expresses a language concept such as a *name* or a *sentence*. Every occurrence of text must be interpreted with respect to the underlying encoding of the character set used to express it. In "English" this is typically the ASCII character set. In other (often non-Latin) languages this may not be true and the visible expression of each octet may vary. In this document the term *text* is used to refer to the pre-encoded source material or the post-decoded source material, while the term *string* is used to refer to the text when it is encoded. Text is almost always in a single character set when displayed although this is not a requirement (consider mixed English, Japanese Kanji and Kana in the same sentence).

3.41 TMC—Traffic Message Channel. An RDS term for the message format originally developed to occupy a Group 8a-type RDS message, employing a 37-bit long message. This was formatted with 16-bits being used for the LRMS portion and 11 bits being used for the index into a string table. Current TMC message formats being advanced in CEN extend this in various ways to overcome its limits. Consult CEN prEN 14819-1 for further structural details. The term TMC is at times also used to refer to the information-issuing source (more properly called the ISP in this document). In this case the acronym means Traffic Management Center.

3.42 Token—The smallest "atomic" element parsed up in processing a steam of input which may be operated on by the algorithms handling the stream.

4. Overview—This section provides a summary of the design goals of the document and a summary of what design goals were not implemented. Various features not supported in this document but found elsewhere in the ITS world deployments using strings indexes are discussed. How these needs are accommodated in US ATIS are presented.

4.1 Goals of the Document—Replacing strings of text (or other data objects) with a short index value representing the string has a history of use longer than ITS itself. The earliest systems for traffic event reporting ever devised made liberal use of this technique due to limitations in transmission bandwidth. Even when bandwidth was not a primary concern, implementers have found the use of index values in place of the complete string to be a helpful technique. Over the years a number of such systems have been deployed. In practice most such implementations suffer from a lack of extendibility, poor or nonexistent methods of updating, a diversity of implementation details which make re-use hard, and a total lack of standardization of the content contained in the actual tables. SAE ATIS itself has used table methods in existing standards (notably SAE J2369 and J2354), extending the techniques to make maintenance aspects of the tables more automated. The goal of this document is to support a rich diversity of user needs while overcoming these previous shortcomings with a single unified method of handling such strings.

There are a number of design goals that this document seeks to meet. Some of these are quite distinct from each other. Among these are the following key points:

a. Support the use of tables in systems ranging from very dynamic to very static. That is, both systems which frequently and automatically update and issue new tables to those systems which only use string tables in a fixed and never changing static nature.
b. Develop common methods for table use across such systems so that the same end receiver software can be used with a variety of media types and supported message data content. That is, regardless of the source of the ATIS messages, the same application software should be reusable to operate on it. This is a basic design goal of all SAE standards in this area, not only because it allows deployed end user systems to adapt to changes but because it also allows ISPs to evolve without obsolescence in the installed base.
c. Develop a world-class standard that fully supports both English and non-English users with multiple language support. In other words, the string and table methods must be suitable for use in any language, any character set, and allow re-expansion of the transmitted string in other languages than

the one used for original transmission (as well as other media types). This is vital for successful consumer products in world markets.

d. Support "custom" expansion where the text may vary within a common language ("Road" versus "Rd") or where no text at all used (for example, reverting an index into digital speech codes).
e. Overcome previous shortcomings that "table only" phrase systems have encountered. Prior experience has shown that systems that rely *only* on pre-determined phrases are cumbersome and never quite fit the real life operational needs. Attempts to merge the phrase lists and the exceptions at the operator interface have produced complex encoding rules that also have shortcomings. It is a goal of this document to allow a completely free mixing of pre-established phrases and free text. And to place the burden of reducing this to string encodings with the construction of the messages themselves (and hence be automatic) rather than involve human intervention.
f. Provide a common methodology and message set for sending tables over media that can support it. That is, by establishing a means to distribute such tables, the automatic updating and maintenance of them will be transparent to most user applications. This is a key requirement for sustainable deployment.
g. Provide a common methodology for extending and updating the entries in tables. That is, by establishing a means to maintain tables at the sender's end, systems can evolve (both in complexity and in content richness) without obsolescing the installed base of user equipment. This is a key requirement for sustainable deployment.
h. Provide support for tables of varying length and varying content meeting both local and national needs for standardized phrases. That is, the methodology must support both short and long tables without undue overhead, as well as national tables of traffic phrases and also local variants of such phrases – all in a consistent and interoperable way.
i. Provide tables meeting all of the above features and goals by an automatic means which removes the requirement for any overt cooperation between the table issuer and the table user. That is, the previous need for mutually agreed upon tables is eliminated allowing a wider and open market for equipment makers at both ends.
j. Finally, provide the utmost compatibility with selected installation legacy systems such as RDS where a reasonable number of end user equipment devices now exist. That is, almost every feature of these systems has been incorporated into this document; therefore deployment of these systems and the phrases they are now using can continue with only slight changes to become compatible with the path of US ATIS standards.

These goals are met by the concept of associating a compressed string with a local table number assignment (an octet value) which in turn maps to one or more actual tables that provides the expansion values and property needs to return the string to a simple text form. Systems using simple static legacy systems can provide the additional table information in a separate mapping table message which, once checked by the receiver, can in general be ignored. More complex dynamic systems will use the abilities provided in this document to implement tables changing on an "as needed" basis obtainable by the end user. The flag values associated with each table affect the processing to expand the resulting string.

The *string*, is a concept used in this document to refer to the resulting compressed set of octets, that becomes a complex data element in the terminology of the ITS registry which is uniformly applied in ATIS messages throughout SAE standards. Raw, or simple text can be viewed as being a simplistic case within the rules of a string. Single index look-up systems can be viewed as being a string composed of only one index value. In all cases a common processing recovery scheme can be implemented at the receiving end able to take advantage of any compression present (or of the lack of any compression).[1] Interoperability over a wide range of implementations is thus insured in this area.

1. A further goal of this document is to allow implementers who do not wish to perform any compression or table look up methods to avoid the issue and place complete strings of simple text into the same fields. As will be seen in the latter sections, this is accommodated. This operating mode is not of much practical value in wireless applications, but in wireline distribution can be useful from the simplicity in which systems can initially be deployed. Such systems can move to more elegant compression when conditions warrant at a subsequent date. Because of the extendibility of the tables in this document, such a change will not effect compliant receivers already deployed.

4.2 Limits of This Approach—The technical approach taken by this document accommodates a wide range of flexibility in use. Prior text has discussed a variety of these issues. This section considers some alternative approaches in string handling which are not supported by the document and provides both the rationale and alternatives to accommodate these practices within the document.

4.2.1 EXPANDED STRING VARIABLES—From the earliest operational days, it has been found that the most interesting aspects of an event were never quite sufficiently contained in any predetermined list.[1] This has led to the combining of pre-determined phrases with insertion variables to create more flexibility. For example, the descriptive phrase "Delay due to $X on roadway" where $X represents a string to be supplied at the time of use.[2] The inserted string might be chosen from a list such as [*breakdown, overturned truck, animal, wild dog, pink dancing elephants*, etc.] or from free text as conditions warranted. Examples of this problem have led to endless committee humor and debate concerning the merits of whatever the unusual description proposed by the pundit was.

Technically this problem has been solved most often by creating textual phrases with an embedded insertion point like the example above (as an alternative, other systems will either use the list or default to a total "simple text" textual description field). This is a scheme very similar to the use of "shell variables" found in Unix, DOS and other popular computing systems. Typically, the variable to be supplied is appended somewhere later in the message (resulting in a variable size message as well). Sometimes the inserted value comes from a defined list, at other times free text is allowed.[3] Usually there is a finite short limit to the number of inserted variables - which to some degree still restricts the descriptive string.

There is no analogous concept in this document, although it provides this feature in other ways. Because string indexes and simple text can be fully mixed and interchanged in any way that the message creator desires, the previous case can be viewed as a very limited subset of the features provided by this document. This is not to suggest that the use of predetermined strings is not useful and encouraged in the design of operators' consoles. This type of CAD assistance results in more consistent phraseology, a reduction in spelling errors,[4] and an increase in operator productivity.

Another variation of the shell variable concept has to do with *numerical values* added to the descriptive phrase. An example would be "… expect 10 minute delay" where the value of the delay would be provided in a subsequent variable as per the above discussion. In this document, the pre-defined number variable table provides the same functionality. This allows an integer range of 0 to 65535 and the addition of the ordinal post value (*st, nd, rd, th* in local conventions). Some systems allow words for logarithmic values, that is accommodated in this document by use of a local table entry (although many such values are also presented in the first 255 entries of the National Names table). Again, this document allows any ordering of the combination of words and indexes.

4.2.2 NESTED MULTIPLE LIST INDEXES (MULTIPLE TABLES)—Some phrase systems were developed with the operator creating the phrases in mind, rather than with the methodology of handling the resulting octet of data in mind. These systems produce many short phrases (each with some assigned numbering value) which can then be strung together to form sentences in the natural language supported. An example would be the below where the final descriptive text is selected in a cafeteria style of one item from each column. See Table 1.

---

1. One ISP has stated that this type of variation has been found to in fact be a selling point to clients, saying that they seem to enjoy the odd details of events. They feel that the diversity of free text increases the marketability of their data.
2. This example is slightly simplistic. A more typical example would be "$1 due to $2 on $3" where $3 might be roadway, street, overpass, etc., providing a range of roadway terms and $1 might be delay, slow down, etc. In this example, $1 and $3 would likely be from predetermined lists only, while $2 is likely to require free text.
3. This also often resulted in ineffective nested index value schemes, discussed further in the text.
4. Note that the technical approach of separating the phrase creation (and operator dispatch functions) from that of the phrase encoding (a message transmission handling function) has the effect that should the operator misspell a word, it would still be sent without a problem. If this is routinely performed, a dynamic system would even table-ize such an entry.

TABLE 1—TYPICAL MULTI-PART PHRASE TABLE

Table #1	Table #2	Table #3
Delay due to	breakdown	on roadway
Congestion from	overturned truck	on overpass
Slowdown due to	animal	by exit
Accident from	wild dog	near access road
Extreme caution	pink dancing elephants	

The problem with the previous is threefold. First, the index values (and typically a further index to select which column) are not efficiently encoded with respect to any other text strings, which may be present. This results in a number of short (typically under 50) table entries needing to be maintained (as well as any other strings used in the same message family). Encoding problems can occur when any sub-list gains one too many members (say a 33rd item). Avoiding this by having large index ranges (say 256) results in extra wasted bits being sent. This document tends to promote the fewest number of such tables for efficiency reasons. In use, the above could be encoded as the following: [table #1][item #x] [table #2][item #y], [table #3][item #4]. In some cases, the use of table 1,2,3 keywords is removed - but this limits the ability to construct complex phrases or to insert free text. A secondary problem which can occur with this system is that the software built to decode indexing of this sort is normally built to each specific case and little of it can be reused by ATIS implementers wishing to operate on strings from another source.[1]

It should be pointed out that to a limited degree the creation of multiple tables has been done in establishing the ITIS phrase list. In this case, the need for nationally agreed upon list of incident features which could be used in almost all of the existing ITS standards for messages outweighed the need for the most efficiently encoded possible list. In the case of ITIS some pre-assigned ranges for adding additional subtypes have been established within the 16-bit indexing range (and are maintained by the SAE as part of the registry process). This is in addition to the allowance for large local list entries.

Finally, the existing collection of such tables takes control away from the data issuers (who are likely to be combining ATIS information with others multiple sources) and prevents them from generating tables which are optimized to both the content and frequency of use. Frequency of use is a vital concept; many historically existing tables have a vast number of phrases, which are not in fact used very often. Some, when used, are used frequently. Consider the frequency of use for the term SNOW in summer versus in winter. Also, many times several phrases are most often used together (e.g., Delay due to breakdown) and it may make better compression sense to combine such entries in transmission.[2] Few of these optimizations can occur when a pre-existing or fixed family of short lists is forced to be used.

In this document, this type of encoding should be avoided, and it is preferred that a “flat” collection of all possible strings be developed and encoding into the smallest number of tables possible (and employing nested table entries rather than nested tables). Nested table entries should also be used to concatenate sub-strings commonly used together as well. If the media allows dynamic table distribution, then the resulting table should be fairly short and messages not frequently used appended to it when their periodic usage warrants. The remainder of the time, such unusual phrases should be transmitted as simple text in the final string.[3]

1. Typically, the decoding logic is a collection of nested case statements where within each case the known collection of possible indexes is handled. By this method some very efficient but complex logic can be developed. However, code rework is normally required when a list changes. By contract, string decoding in this document proceeds in a linear (as opposed to nested) fashion expanding one token at a time as the string is processed.
2. Such grouping also promotes automatic sorting and filtering to occur in client receiver devices.
3. Systems using phrase tables to assist the operator (rather than encode them for transmission) can ignore this issue as they are presumed to produce full text messages. Such full text would be compressed and converted into tables by automatic equipment downstream, typically at the point of transmission by the ISP.

If it is determined that such prior index tables are to be used then the resulting table should be contrived such that the lower bits of the index represent the pre-existing index values and the upper bits represent the preexisting table values. By this method, implementers can preserve the vast majority of the encoding previously used, and any implementers in compliance with this document can also process such strings as being fully compliant. The resulting table is likely to be a "sparse" table with unused entries wherever the prior system did not fully use the indexing range of the tables it created. The expansion of these tables into multiple languages would proceed as it did before without change, however implementers should be aware that language word order issues will exist.

4.2.3 MULTIPLE EVENT OR LOCATION REFERENCE MESSAGES—In some traffic messages using index methods (such as found in Alert Plus) there are provisions to report on an event covering multiple locations in the encoding. Typically, this is done with a range of numerically related index values. This subject is not covered by this document and in general SAE ATIS constructs messages with each event separately. In some broadcast messages (notably those of SAE J2369) an index to a single descriptive string will be used in multiple messages so as to conserve bandwidth. In such a case the affected roadway segments may all point to a common textual string for information (which in that standard is carried in a common table when reused in this way).

This document provides methods to transmit such strings both in tables (for shared use) and in-line in a specific message for which it is intended. A later chapter provides specific recommendations for existing SAE standards. It does not offer any recommendations on the LRMS methods used to refer to the affected area.

4.2.4 TIME VALUES—In some traffic messages using index methods (such as found in RDS AlertC) there are provisions for reporting the expected time or duration of an event as part of the string. This subject is not covered by this document and it is expected that such information would not be sent in the string. Existing SAE ATIS message sets offer a variety of standardized means to encode the same information.

4.2.5 BAUDOT AND OTHER CHARACTER SETS—In some traffic messages' textual fields (such as found in GATS) there are provisions using various 6-bit and 5-bit Baudot character sets[1] which encode as a sub-set of the ASCII and Latin_1 character sets (some of these provide only the upper case alphabet and number characters). These character sets are not supported in this document. The original rationale for using these codes was that a percentage saving is obtained by saving 2-bits or 3 bits out of every 8 needed by ASCII[2] or Latin_1. A similar level of savings can be obtained by using the modified ASCII outlined in this document and even more when using the string compression methods. Further, these codes (and their regional variants such as the Murray code used in the UK) do not allow a uniform mapping to world character sets. In rejecting support for these older codes, SAE is following the same decisions made by the WWW and other standards bodies concerned in providing support for world languages.

One exception to this is the provision to allow the insertion of characters into a sequence of indexes as per 5.5 Table Expansion Values. Among the table exception tokens defined in Table 3 there is a token which has an effect similar to the "shift-lock" character of Baudot. When processed in the stream, the effect of this character is that the bytes following are interpreted in the default character set of the string until the end of string character (for that character set) is found.

4.2.6 MULTIPLE LANGUAGE SUPPORT—The system of string handling proposed in this document overcomes a limitation found in SAE J2369 as it currently exists today. While the effort of producing SAE J2369 was instrumental in advancing the automatic handling of such strings, and while it did support ASCII, Latin_1 and Unicode character sets, it did not allow the use of the efficient string compression methods in any character set but ASCII and modified ASCII. This was sufficient for the US market but not for a world class standard.

---

1. Formally called the ITA2 code (International Telegraph Alphabet Number two).
2. American National Standard Code for Information Interchange (ANSI X3.4). All of which are now superseded by the work of Unicode. [Ed note: where it is referred to as International Alphabet No. 5 (IA5) - This is where the term IA5String comes from in the ASN.1 syntax].

In SAE J2369 other languages were supported but the ability to invoke and use tables was displaced when this occurred. The use of a limited set of values from 236 to 255 to be the exception token meant that the following values (commonly used in European languages) could not be transmitted:

TABLE 2—LATIN_1 CHARACTER EXCLUDED BY CURRENT SAE J2369 CODINGS

Symbol	HTML Call out	Description
Ì	&#236 &igrave	Small i, grave accent
í	&#237 &iacute	Small i, acute accent
î	&#238 &icirc	Small i, circumflex
ï	&#239 &iuml	Small i, diæresis / umlaut
ð	&#240 &eth	Small eth, Icelandic
ñ	&#241 &ntilde	Small n, tilde
ò	&#242 &ograve	Small o, grave accent
ó	&#243 &oacute	Small o, acute accent
ô	&#244 &ocirc	Small o, circumflex
õ	&#245 &otilde	Small o, tilde
ö	&#246 &ouml	Small o, diæresis / umlaut
÷	&#247 &divide	Division sign
ø	&#248 &oslash	Small o, slash
ù	&#249 &ugrave	Small u, grave accent
ú	&#250 &uacute	Small u, acute accent
û	&#251 &ucirc	Small u, circumflex
ü	&#252 &uuml	Small u, diæresis / umlaut
ý	&#253 &yacute	Small y, acute accent
þ	&#254 &thorn	Small thorn, Icelandic
ÿ	&#255 &yuml	Small y, diæresis / umlaut

This has been overcome in this document. This document provides complete support for mixing multiple languages and character sets in the string at that same time.[1] At the same time, the slightly more efficient termination and exception tokens established in SAE J2369 and useful for English language market have been preserved.

4.3 Summary—This section has provided a background of the design goals that this document seeks to meet. In the section which follows the precise operating principles of the document are outlined. The messages and data elements which make up the standard are then provided in a format used by the SAE and the ITS industry. This is a data registry format where the actual coding elements are expressed in the ASN.1 language syntax. English dialog provides any further information needed to use the resulting messages. Next, a chapter of recommendations and advice to use this document in related SAE ATIS standards and other ITS formats is provided. Finally, in companion volumes (SAE J2540-1, -2, and -3) are the results of efforts to develop National standardized phrase description lists. These phrases are expressed in the formatting and rules of this document.

1. The technical process, which is outlined in subsequent chapters, uses the “double null” token now popular in Windows, Mac, and Java programming environments to delimit Unicode strings.

5. Operating Concepts—This section outlines the use of the contents of the table messages and data element structures and provides an overview of the operational exchange used in message transactions.

5.1 Assignment of a Table in Use—Tables are assigned a local number value ranging from 1 to 255 when used. This is different from the nationally assigned and maintained 16-bit number, the table registration number. The purpose of the local values is to map whatever tables the local ISP is using into a common data space or use by the end application. Some table values are predefined and therefore the table contents they refer to are also static[1] and predefined (or at least change extremely slowly). Up to 255 tables can exist for each user, nested or independent, and they may or may not all be used by any one end user device or over any specific media. It is the responsibility of the ISP to maintain a coherent set of tables to each end user they are supporting.

In practice, only a very small subset of the possible range of tables would be expected to be found in use. It is anticipated that most ISPs will use a common set of one or two tables (made up of national phrases with local variants) tuned for the given distribution media and reuse this set with every customer. If an ISP is offering tiered services (such as a news feed and ATIS messages over the same media) then tables for each may exist or a common table may exist as best suits that ISPs needs. If an ISP is using extremely limited bandwidth media (such as RDS) then the table numbers will probably be a) singular (only one table in use) and b) transmitted periodically so as to conserve bandwidth.[2]

The key point is that the end-using device needs to always associate the local table number with any string that it is processing. Given these two values, the string and its expansion are fully defined.[3]

5.2 Properties of a Table—The table (with its local number assignment) consists of two parts. First, a header with information on its registration and how its strings are to be processed and second, the index of string expansions themselves. Both parts are at times optional, depending on the table used and the media being supported.

Most tables have at least the header content present in the collection of supported messages. The header contains the nationally assigned registry value for the table. In this section are a variety of expansion properties. Refer to Section 7 for the precise bit level definition of their use.

5.3 Nesting of Tables—The nesting of tables is a key method used to extend a national table to local use, and to use pre-existing tables to make up the content of a new table in the most space efficient way. Tables can be nested in three ways: within the table itself (referred to as *Nested Table Entries*), between tables (referred to as *Nested Tables*), and by extending a table with a new table (referred to as overlapping).

Nested table entries are table entries in which the expansion of the string results in another index into the SAME table. This results in one of the four included table entries being listed as the same table. This is different from an overlapping a table, which is discussed as follows. Within the index string of the table would be found another general string following the same formatting as all strings (an expansion code selecting the table to use (in this case the same table) and how to expand it, followed by the index itself).

---

1. A further issue with static tables is that some are never resorted, that is the assignment of an index value (one made) would never change. This allows data issues and users who have made alternative mappings (say a re-expansion into an audio recording) reply on such values never changing. If an entry in such a tabel is ever made obsolete, then it remains forever as an unused entry in the table (the table may be dense or sparse as suits the needs of the users).
2. ATIS messages often provide a means to send the table in use within the beginning of the message which is then applied to the various strings found in the message. It is recommended that this practice be extended to all the messages in the ITS community so that they can also use sac string concepts. For a 37-bit RDS-TMC of the “Classic Alert C” sense, the local table number would be found in another message (in the ODA section it would likely call out a predetermined national table which is not available to download over this media.)
3. Presuming of course that the associated local table number can be mapped to the supporting tables and that the indexes they refer to are available to the device.

Nested tables are table entries in which the expansion of the string results in another index into ANOTHER table. The selected table will be one of the four included table entries in the table header section. The size of the table, its character sets, and its local number will be found in the header section. Typically this will be a national table. Conceptually such tables could nest up to the limit of available table numbers. Practically, it is recommended that such nesting be limited to no more than two deep (that is, no table may include other tables, when they themselves are made up of other tables). The sole exception to this rule shall be those tables which are national in scope, static in nature and therefore not subject to downloading. Under these conditions such a table may be implemented with as much nesting as the end equipment maker sees fit to use. Note that this allowance has no effect on the messages as transmitted. Within the index string of the table would be found another general string following the same formatting as all strings (an expansion code selecting the table to use (in this case another table) and how to expand it, followed by the index itself).

A final way of nesting a table is by *overlapping*. In this method the new table (typically a local table) is mapped into one or more of the (used or unused) index spaces of a prior existing (typically national) table. Overlapping is particularly useful for those occasions when the resulting string is limited to containing ONLY index values from a single table. In cases where a more general string can be used, the four exception flags (which allow pointing to four tables with independent ranges of index and character sets and nesting, etc.) are more flexible and should be used. The issuer of the table must ensure that the used index space does not cause a conflict with the preexisting space.[1] The overlapping method is also used to replace a table entry with a variant entry in another table. Because overlapping can be used at any time tables it is a requirement that tables SHALL be searched in the order in which they are entered in the table header during re-expansion.

The entity that issues the table is responsible for ensuring that the table is coherent (that is that every index entry value correctly and successfully can be expanded). It is not acceptable to issue a table which has values that cannot be expanded (such entries should be set to zero if they are not removed) or that indefinitely loops (one or more index entries that point to themselves and never resolve). Note that tables may be created not only by the ISP but by the end equipment manufacturers. This is because the equipment maker may find it useful to control the expansion of national tables and their storage within his device.[2]

Nested tables loaded into the end-user device may also end up being “flattened” in that device if the implementers so decide. This is an implementation issue beyond this document. Observe that this means that a multiple number of ways to organize tables can be chosen as needed yet still result in the same final textual strings being displayed.

The most typical example is expected to be an ISP using a single local table (mapped to #128) which is used to augment a supporting national table. In this case common and frequently used local terms would be in the local table while unusual terms would appear as simple text in the final strings.

5.4 Supported Vocabularies and Alphabets—All tables support the four basic character sets of Modified ASCII, ASCII, Latin_1, and Unicode. Every table is implemented using one and only one of these character sets.[3] The character set is used to determine how text is displayed and exception tokens are detected in processing the string to call out which table an index comes from. There are some uses of strings where the exception token is not required (because the format of that message indicates that only an index will be present rather than general strings). The character set which is in use is determined by looking at the suitable flag information in that table.

---

1. The order in which end using devices look up table indexes is always defined by this document to occur in the order in which the tables are listed in the header. Conceptually if the table flags indicated that no overlapping occurs, then search order should not matter - however implementers are cautioned not to reply on this.
2. Consider a display in which due to limits of space, the phrase “... on Freeway” was reduced to “...on Fwy.” Also, internal highly nested table entries can save considerable memory space in the device.
3. Note that a string can always be “promoted” in that Modified ASCII can become ASCII, ASCII can become Latin_1 etc., as the end using device requires. This can occur in the end device as needed or the table itself can be translated. This is beyond the realm of this document.

The character sets of Modified ASCII and ASCII use the delimiter of a single octet of Hex #00 to signify the end of a string. They use the delimiters of values from Hex #EC to #FF to select which of four possible expansion tables are to be used and how the expansion is to occur (the two lower bits are used for controlling first letter capitalization and adding a space after the string). The index value itself (occupying one or two bytes) follows directly thereafter. A complete summary of the values of the Modified ASCII character set (including the two letter compression values) can be found in SAE J2369.

The character sets of Latin_1, and Unicode use the delimiter of a double octet of Hex #00, #00 to signify the end of a string. An exception table is a two byte field where the first field is a Hex #00 and the second field is as per the previous.[1] The index value itself (occupying one or two bytes) follows directly thereafter.

5.5 Table Expansion Values—Tables are selected in the string by the use of the table expansion token preceding them. The lower four bits of this value also select how the expansion of the resulting string is to take place. These rules are completely defined in SAE J2369, Section 5.2.

In SAE J2369 the original concept was that only four tables would ever exist, and two of these were predefined and static in nature (the National Table and the Imputed Numbers Table). This is expanded in this document to allow up to four tables to be used at once (up to 255 of which may co-exist) and any combination of static and dynamic or nesting to be contrived as the issuing ISP sees fit.

Reprinting Table 7 from that document and reflecting the previous changes, the exception tokens defined in this document are as follows. Note that this also provides backward compatibility with SAE J2369 when the subject table used is defined as local table #1.

TABLE 3—TYPES OF TABLES AND THEIR EXCEPTION TOKENS

Table Value Range	Type	Use
0xEC ~ 0xEF	First Table	Any Valid Use
0xF0 ~ 0xF3	Second Table	Any Valid Use
0xF4 ~ 0xF7	Third Table	Any Valid Use
0xF8 ~ 0xFB	Fourth Table	Any Valid Use
0xFC ~ 0xFE	Reserved	
0xFF	Toggle Mode	Toggle stream tokens to sequence of native Character set mode if an index mode (as per about-flags). Toggle stream of tokens to sequence of indexes (into first table) if a string (as per about-flags). Toggle reverts when Termination token is found in the string.

Each of the table invoking exception values allows the lower two bits to control the way the string is expanded. Reprinting two tables from SAE J2369, here are explanations and examples of the expansions allowed.

TABLE 4—EXAMPLE OF TABLE CONTROL EXPANSION BITS (FROM SAE J2369 TABLE 8)

Table Bits	Meaning	Example 1	Example 2	Example 3
00	No Space, No Cap	<main>	<montgomery>	<elSid>
01	Add Space, No Cap	<main >	<montgomery >	<elSid >
10	No Space, Make Cap	<Main>	<Montgomery>	<ElSid>
11	Add Space, Make Cap	<Main >	<Montgomery >	<ElSid >

1. Note that this is similar to the way various menu items and other resources are stored in Windows, Macintosh, and Java programming environments. Programmers may wish to view this process in the same light as the handling of various 2-byte UCS 8 and UCS 16 character sets and make use of existing industry code for handling such strings.

In the numerical table, the first character capitalization rule for letter glyphs is changed to indicate the addition of the of the ordinal post character. Note that this should be done to meet local language conventions.[1]

TABLE 5—EXAMPLE OF NUMERIC TABLE CONTROL EXPANSION BITS (FROM SAE J2369 TABLE 9)

Table Bits	Example Returned Values				
00	<1>	<111>	<2>	<3>	<123>
01	<1 >	<111 >	<2 >	<3 >	<123 >
10	<1st>	<111th>	<2nd>	<3rd>	<123th>
11	<1st >	<111th>	<2nd >	<3rd >	<123th >

The toggle mode exception token of Table 3 does not allow the same type of expansion control. This mode is provided as a simple means to "slip in" either a *sequence of indexes* or a *sequence of characters* (as would form a word) into the string. Into which mode the stream changes can be determined by examining the values of the About-Flags within the Local-Table-Flags. The native character set then in use can be determined by examining the Included-Table-Flags in this same part of the message. Unlike the other exception tokens which are then followed by a single index value, this token is "sticky" and it remains in effect until canceled. Cancellation occurs by the presence of either a Termination token in the native character set (typically a one or two byte value all set to zeros), or by an index Termination (all zero of the length defined by the index length).

5.6 Simple Text within Strings—A common occurrence when constructing descriptive phrases is that some type of highly specific noun is required which the commonly reused phrases do not contain. This requirement is easily handled by means of constructing the phrases required and inserting a sequence of text as needed. Two examples illustrate this in practice. Consider the prior incident description example in 4.2.2 with the phrase: "Delay due to wild dog on roadway" to be sent.

If the string has been determined to be one of mostly indexes with some character values than the phrases "Delay due to " and " on roadway" are likely to exist in tables while "wild dog" is not. This would be encoded as follows <index#> <0xFF> <wild dog> <0x00> <index#> <0x00>. Note that once the Toggle Mode token (0xFF) is used, then a sequence of characters can be inserted without any additional expansion tokens. This example is perhaps the most typical use of such tables as it provides a good mix of simple phrases expanded with plain English when the operational need arises. In messages when the descriptive portion can be of variable length, this format choice is preferred.

If the string has been determined to be one of mostly characters with some index values than the dual of the above holds true. In this case the any sub-string in the phrase which exists in a table available to the end user may be substituted. For example, if " on roadway" was a table entry, the stream of simple text could be replaced by <Delay due to wild dog> <0xEC> <index#> <0x00> as a valid encoding. Alternatively, this could be encoded using the Toggle Mode token as <Delay due to wild dog> <0xFF> <index#> <0x00>. This has the inherent limitation that the index are taken from the first tables, while the prior example allows selecting from all four tables as well as the expansion bit use. It has the inherent benefit that if more than one index is to be placed in the string, no precursor exception token is required for the subsequent indexes after the first one.[2] This method also has the effect of limiting the use of the Latin_1 character "ÿ" (value 0xFF) under some circumstances.

1. In the American English this would be [*st, nd, rd, th,*...] but this varies considerably with other languages. For example, in North American Spanish a greater variety of ordinal terms are used as well as the more generic Masculine Ordinal symbol used in Europe. In other languages and cultures this can result in completely different glyphs being used. The precise capitalization, hyphenation, ^{super} or $_{sub}$ scripting, or other distinguishing markings used is NOT defined by this document and may be limited by the ability of the display device as well as the language conventions.
2. If this is in fact the normal mode of operation, (e.g., typically strings are composed of sequences of indexes and on occasion free text is to be added) then the use of the "just-indexes" option in the About-Flags will eliminate the need for perquisite exception tokens and should be used.

Note that in either of these cases the decoder algorithms used by the receiving device remain the same. In the above examples, one byte characters and indexes have been presumed (such as ASCII, Latin_1 and Modified ASCII use). Observed that multi-byte Unicode characters or longer index would simply result in a termination character of 0x0000 being used but would otherwise be the same.

5.7 Registering a Table within ITS—Every table has a registration number, a 16-bit value which uniquely identifies it. Some registration numbers are pre-assigned for national use and interoperability reasons. Others are locally defined and should be registered with the SAE as part of its registry authority in the ITS wide data registration process. Still others are dynamic in nature and are never registered with any authority other than the issuing ISP. Predefined ranges allow the separation of these types of tables. This information can be found in the header of each table. With the exception of the national tables, (which are typically static and may be not available for download over all media) the use of a table registration number is limited to determining that the loaded contents in the end-user device are in fact the correct table. More frequently, tables may have become obsolete due to modification of its contents. The methods supporting this are described as follows.

5.8 Adding to a Table's Entries—The most common event over a table's lifetime is the adding of new entries to it. New table entries are added by appending them to the end of the table and changing the index start and stop values in the header to reflect this. In addition, the "changed" flag is set and the least significant bit of the revision value is incremented by one. No table entries may be removed,[1] and no indexes of existing entries are changed. The new entries may use nesting to the same table or to any other table previously specified in the table header. This process allows the table to grow but does not cause the obsolesce any of its current contents.

5.9 Table Resorting—Periodically a table may *re-sort*. That is, the contents may be re-ordered in any fashion desired. At this time there may be changes including removal of some strings and additional of others, changing of the tables which it nests with, and the character set which it uses. Usually the changes made are much less dramatic. The reordering of a table is frequently in alphabetical order but need not be so.[2] Table reordering also changes the revision value and change bits. The revision value shall be incremented to the next modulo-16 value when a re-sorting occurs.

Every time the revision value crosses the modulo-16 value a resorting may have occurred. The receiving device should therefore check for this. Changes of less than 16 in value are always table additions, and the receiving device can normally obtain just these new entries from the ISP (by either broadcasting the additions in a 1-way media or requesting the additions in a 2-way media).

The 8-bit value of the revision bit allows 256 changes to occur before overlap. Consider a dynamic table changing every hour. This provides a rollover every 10.6 days. For a static table multiple years may occur before rollover.

Note also that some tables are expressly forbidden to resort (by the value in the About-Flags). Such tables shall NEVER reassign an index value once used. If it is determined that an index will not be used in such a table, then the value remains and is set to the null token. In that case of this type of table, changes in the revision bit over the modulo-16 value shall have no additional meaning other than that stated in 5.7.

1. Removal of entries can only occur in a re-sorting. Unused strings must remain in the table until that time.
2. Implementers at the ISP end who are detecting the table strings to extract from a simple text string may find such alpha ordering useful in string sorting. No similar action occurs in the receiver device where string expansion is occurring and the order is immaterial. It is recommended that ISP implementers consider creating a master table for string extractions where strings from all tables being used are merged.

5.10 Considerations for End User Equipment—Regardless of the above provision, changes in an issued table cause effort to be required in the end-user device. The device may use the messages of this document to obtain the new table information. In a worst case scenario, the device may need to obtain the new tables from another media, perhaps requiring the manual intervention of the operator to do so.

More typically, two messages support the downloading of tables, and are described more fully in the sections that follow. The *request_table* message and the *request_partial_table* message allow the device to obtain the new table if the media and ISP support this message. Another message, *translate_index*, can also be used in 2-way transactions to ask the issuing ISP to replace the requested index with its simple string equivalent. A further variation, *translate_string*, will produce a complete translation of a general string into the basic character set being used, removing all indexed strings. The support for these messages will vary based on both media in use and ISP features.

In media which are one-way and broadcast in nature, the ISP must issue tables (in the *table_message* and *partial_table_message*) to provide the needed tables to the end user when a change is made. In this mode of operation there is always a percentage of user equipment that has come on-line during such a table change and is without the proper table until it is transmitted to it. It is the responsibility of the issuing ISP to control the timing and issuing of such messages over their network to reduce the occurrence of this.

If a table is unavailable, presumably a short-term event, then the expansion of the message should not occur, as the results are unpredictable.

5.11 Changes to Existing Standards—Implementing this document with its fuller support for indexes, strings and multiple languages will require some slight changes in existing SAE standards. The most notable are the addition of the new messages to distributing tables and perform table translation services. ISPs using indexes are required to also transmit support tables in all but the most bandwidth-reduced cases. ISPs are strongly urged to implement the table translation functions as well.

In a number of existing SAE messages found in SAE J2354 the addition of a local table number must be added in the header of the message itself. In addition the supporting table message must also be sent. In a number of existing SAE messages found in SAE J2369 handling of strings and the precise format of tables are changed by improvements in this document. Strings found anywhere in the ATIS message set can be expressed in the formats of this document, and are urged to do so. In both cases the changes are minor and explicitly noted in Section 8.

Note that non-compressed strings are a special case of compressed strings in this document. Again, this was adopted to allow the ISP to use as little or as much string compression as they saw fit. Because non-compressed strings can be handled by the same processing software in the receiver as strings this document, it is recommended that implementers adopt a use of common code to process both.

6. Messages—The messages of this document support the distribution of tables to end-using devices. They are intended to be implemented by any host media which can support them in cooperation with other messages which support the bulk of the ATIS content (see SAE J2354 and J2369). Some media will not be able to support table distribution (and are limited therefore to the static use of such tables). Other media may provide table updating by alternative means of downloading than the method used for ATIS message delivery. Still others will embrace the dynamic use of tables and will frequently change and update the issued table set. Not all media will implement all messages. Specially, the service of translating a string or an index is not likely to be provided by every ISP.

The messages defined here build upon the definitions found in prior messages. That is, data frames are defined the first time they are used in a message and then subsequently reused in messages that follow. The definitive name of the message or data element in question is given by the provided ASN.1 code definition and not by the title of the containing section.

Implementers should be aware of any tagging environment employed by the host media. In general, ATIS messages, while specified in the ASN.1 syntax, do NOT use ASN.1 methods of tagging.

6.1 The Table Message—The table message is the message used to convey the complete data of a table to the end-using device. It contains both the table header and the table entries and is defined as follows. ATIS applications in conformance with this and other SAE standards will require this table to successfully decode the strings found in them.

This message may be broken up into packets as required by the host media, or some of the other messages may be used to send the table. In some very limited cases (RDS-TMC being the only one known at the time of writing) the message may never actually be sent and the equivalent message may be artificially constructed by the network link layer for the use of the upper applications layers. Regardless, the end-using device must be able to access this information in order to successfully decode the strings which use it.

The message may be sent in response to a Table Request message (which requested the entire table) or issued periodically as the ISP sees fit (more typically in a broadcast media).

The format of the message is as follows:

```
The-table ::= SEQUENCE {

      header      Table-Header,     -- the 26 byte header
      body        Table-Body        -- the actual strings
      }
```

where the component parts listed previously are made up of the definitions that follow. These definitions are uniformly used throughout the remainder of the document.

```
Table-Header ::= SEQUENCE {
      registration      Table Registration Value,-- a 16 bit unique value
      flags             Local-Table-Flags, -- a 16 bit value
      start-index       Index-Start,       -- a 16 bit value
      stop-index        Index-Stop,        -- a 16 bit value
      count             Index-Count,       -- a 16 bit value
      type              Entry-Type,        -- three 8 bit values
      this-table        Table-Entry,       -- a 24 bit value
      table-1           Table-Entry,       -- a 24 bit value
      table-2           Table-Entry,       -- a 24 bit value
      table-3           Table-Entry,       -- a 24 bit value
      table-4           Table-Entry,       -- a 24 bit value
      crc               CRC-16             -- a 16 bit value
      }


 Table-Body ::= CHOICE {
      dense       Dense-Table       CHOICE {
                        txt   {           -- textual entries
                              SEQUENCE OF SEQUENCE {
                              string      SAE-string, -- Octet string,
                              token       Termination
                              },
                        bin   {           -- binary entries
                              SEQUENCE OF SEQUENCE{
                              len         Word-Count,
                              entry       OCTET STRING,
                              token       Termination
                              }
                        }
      sparse      Sparse-Table      CHOICE {
                        txt   {           -- textual entries
                              SEQUENCE OF SEQUENCE{
                              index       Index-Value,
                              string      SAE-string, -- Octet string,
                              token       Termination
                              },
                        bin   {           -- binary entries
                              SEQUENCE OF SEQUENCE{
                              index       Index-Value,
                              len         Word-Count,
                              entry       OCTET STRING,
                              token       Termination
                              }
                        }
      }
```

The determination of sparse and dense tables types can be made be examining the Local Table Flags (and its internal member of type Included Local Flags). The determination of text and bin table entries can be made be examining the value of Entry type (the values “txt” and 0x000 implied text while all others imply binary table entries).

Note that in sparse type tables where the index of each entry must be provided, the length in bits is determined by the flags in the USING table (and not by any ASN.1 tagging).

```
Index-Value ::= CHOICE {
      index-8     INTEGER (SIZE(8)),
      index-11    INTEGER (SIZE(12)), -- an 11B index is still a 12B element
      index-12    INTEGER (SIZE(12)),
      index-16    INTEGER (SIZE(16))
      }
```

Note that the string termination token varies with what character set is being used, this is found in the flags of the table (and not by any ASN.1 tagging).

```
Termination ::= CHOICE {
      single      INTEGER (SIZE(8))       {0},
      double      INTEGER (SIZE(16))      {0}
      }
```

Where the sizes of termination and index values is determined by the flags in the using table. It is recommended that tables be limited to under 32K in total size except in the case of binary table contents. Tables with binary content SHALL always use the double termination value above.

6.2 The Partial Table Message—The Partial Table message is used for sending the table header, the table body, or a portion of the table body. When a table is extended, this message provides a convenient way to transmit the new body contents. When a table is based entirely on a known standard set of string entries, this message provides a convenient way to send just the header confirming that fact.

The message may be sent in response to a Table Request message (which may request the portion of the table of interest) or issued periodically as the ISP sees fit (more typically in a broadcast media after a table has been extended).

The format of the message is as follows:

```
Partial-Table ::= CHOICE {
      head  Table-Header,
      body  Table-Body,
      part  Partial-Body      SEQUENCE {
                  start-index Index-Start,   -- a 16 bit value
                  stop-index  Index-Stop,    -- a 16 bit value
                  count       Index-Count,   -- a 16 bit value
                  body        Table-Body
            }
      }
```

The reply will be based on the requested information. The start and stop indexes and count value for a partial body will reflect the values being transmitted (as opposed to the whole table).

Upon receipt of any message containing a header, the end-using device should compare it with any previous copy and note if this table or the underlying table revisions have been changed. If so, the end using device should request the supporting table(s) if it does not already have them (and of the same revision).

Note that in general a new underlying tables should result in a reloading process, but this is not the case with national static tables where the end using device may have loaded internally a variety of ways to construct the underlying tables of a national table. In this case, the revision flags are sufficient to determine if the loaded table has been revised. If the national static table has not been revised, then differences between the underlying table of the ISPs header and that of the loaded table in the end user device can be safely ignored.

6.3 The Table Request Message—The Table Request message is issued by the end-using device (on 2-way media) to request from the ISP all or part of a needed table. This message may be issued when a device becomes aware that its tables are no longer up to date. This can occur for a variety of reasons besides the ISP issuing a new table. The device may require a first time initialization. The device may be newly arrived in a regional operating area and be in need of local tables. The operating settings of the device may have changed, prompting the need of such tables. The message will result in a *The-Table* or a *Partial-Table* message being issued by the ISP.

The format of the message is as follows:

```
Table-Request ::= SEQUENCE {
      what        Request-Type,     -- a 16 bit string of items
      who         Local-Number,     -- an 8 bit value
      start-index Index-Start,      -- a 16 bit value
      stop-index  Index-Stop        -- a 16 bit value
      }
```

The details of how to set the *what* value to obtain different results are provided in the DE section of the document. When the *start* and *stop* values are not needed by the type of request (such as requesting the full table), their values should be set to zero. The values shall be ignored by the receiving device in this case.

6.4 The Index Translate Message—The Index Translate message is used to send to the end using device the expansions (translations) of a requested set of (one or more) tables and indexes. The returned value may or may not be flattened. The message is sent in response to an Index Translate Request message.

The format of the message is as follows:

```
Index-Translate ::= SEQUENCE OF SEQUENCE {
      Flags       Request-Type,     -- a 16 bit string of items
      result      SAE-String
      }
```

The character set of the returned string is provided by the flags. The last octet or two octets of the string will be a termination character as determined by the character set in use. If one or more of the requested strings could not be translated then the result field for that string will be the termination character. In a message with multiple strings to translate, the order of the replies is in the same order as the request.

6.5 The Index Translate Request Message—The Index Translate Request message is used to send to the ISP one or more indexes and their associated tables for translation into strings and then down to either the next set of tables or to simple text in the requested character set. The message results in an Index-Translate message containing the translations.

The format of the message is as follows:

```
Index-Translate-Request ::= SEQUENCE {
      how   Request-Type,           -- a 16 bit string of items
      count INTEGER (1..255,)       -- how many items follow
      list  SEQUENCE OF SEQUENCE{
                  table       Local-Number,
                  index       Index-16     -- always a 16 bit value
                  }
      }
```

The index sent is always a 16 bit value, regardless of the length of the original index. In a message with multiple indexes to translate, the count field will contain the number of strings. The order of the replies in the Translate message will be the same as the order used in the list.

6.6 The String Translate Message—The String Translate message is similar to the Index Translate message except that a complete string (being made up of one or more indexes) may be translated. The String Translate message is the reply to the String Translate Request message, but it is otherwise identical to the Index Translate message.

The format of the message is as follows:

```
String-Translate ::= SEQUENCE OF SEQUENCE {
      Flags       Request-Type,
      result      SAE-String
      }
```

6.7 The String Translate Request Message—The String Translate Request message is used to send to the ISP one or more complete strings and their associated tables for translation into strings and then down to either the next set of tables or to simple text in the requested character set. The message results in an String-Translate message containing the translations.

The format of the message is as follows:

```
String-Translate-Request ::= SEQUENCE {
      how   Request-Type,
      count INTEGER (1..255),             -- how many items
      list  SEQUENCE OF SEQUENCE{
                  table       Local-Number,  -- the top most table
                  string      SAE-String     -- the complex string
                  }
      }
```

The string value can be any valid string built upon the tables issued by the ISP. There is no guarantee that if the end-using device were to make up a string not issued by the ISP (but nonetheless valid), the ISP would be able to translate it. In a message with multiple strings to translate, the count field will contain the number of strings. The order of the replies in the String Translate message will be in the same order as the element in the list of the request.

7. Data Elements—The section provides normative definitions and instructions on the use of data elements that are used in the messages of the preceding section. It also provides further definition of the critical *SAE-String* value which is found in other messages and which is expanded back into simple text within end using devices.

7.1 Date Element: Table-Registration-Value—The Table Registration Value is a 16-bit value which is uniquely assigned by the SAE to any registered tables. This value is published in the ITS wide registry and adheres to the following rules.

It is defined as follows:

```
Table-Registration-Value      ::=   INTEGER (SIZE(16))
```

The following ranges are reserved for assignment:

TABLE 6—TABLE REGISTRATION VALUE RANGES

0			Not allowed
1	..	255	Reserved for National & International Static Tables
256	..	8095	Reserved for Regional Use Static Tables
8096	..	65279	Reserved for Issuing ISP Use
65280	..	65534	Reserved for Receiving Device Internal Use
65535			Not allowed

Values of 8095 and less will be registered as part of the ITS registry. Values between 8096 and 65279 would only be registered should the issuing ISP desire it and may conflict with other ISPs.[1] Values above 65280 will not be registered.[2]

The following specific values are assigned at this time:

TABLE 7—RESERVED TABLE REGISTRATION VALUES

Value 0x10	The National words Table, in ASCII
Value 0x11	The National words Table of the “first 255”
Value 0x12	The Modified ASCII Table (an Imputed Table)
Value 0x13	The Latin_1 Table (an Imputed Table)
Value 0x14	The National Numbers Table (an Imputed Table)
Value 0x20	The National RDS Phrase List (all phrases)
Value 0x21	The National RDS Phrase List (US linked to CEN)
Value 0x22	The National RDS Phrase List (CEN only)
Value 0x23	The National RDS Extended Phrase List
Value 0x30	The National ITIS Phrase List

Implementers are urged to consult the ITS registry for the most up to date values of table assignment and their contents.

Some of these tables are “imputed” which is to say that they content can be deterministically developed by an algorithm rather than requiring an outside authority to define them. Implementors MAY wish to construct such tables on the fly, if at all.

The modified ASCII table is an optional imputed table provided for completeness. It provides an expansion from the assigned one octet values back to one and two character ASCII values. It is never transmitted.

The Latin_1 table is an optional table which may be implemented by some end using devices to expand the various diacritical marks found in the ISO Latin_1 character set into characters suitable for display of less featured devices. An example might be a device with an alpha numeric display which is forced to move an inflexion mark to another character position when displayed (e.g., “é” would become “è”).

1. We presumed that only one ISP will be issuing data to the end using device at a time, or that the device will be able to differentiate between multiple ISPs at once. This process allows ISP to cooperate, but does not require them to do so.
2. For reasons of internal optimization, an end-using device may implement tables of its own devising. An example might be a flattened table if memory were not an object, or – more likely - a highly nested table to maximally compress a known static table at the express of recursive expansions. This range table value is set aside for such use.

7.2 Date Element: Local-Table-Flags—The Local-Table-Flags is a 16-bit value made up of two set of flags. It is defined as follows:

```
Local-Table-Flags :== SEQUENCE {
      about       About-Flags,
      use         Included-Table-Flags
      }
```

It is used to convey information about the type of table including how its strings are used and parsed by others, if it is nested within itself or other strings, a recently changed flag, and the bit size (and hence length) of the indexes to follow. See the two definitions for About Flags and Included Flags for addition information. Note that the Included Flags value is also used in the Table Entry element. In the Local Table Flags case it refers to the way a string uses the table indexes. In the Table entry it refers to the way a table itself is formatted.

7.3 Date Element: About-Flags—The About -Flags is an 8-bit value made up of various bit patterns as defined as follows:

```
About-Flags :==   BIT STRING (SIZE(8))

      -- definition provided below, composed of terms
      -- logically and-ed together to form final word

      -- If the table is downloadable over this media
      -- found in bits 'xx000000'B
      not-downloadable        About-Flags ::= '00000000'B
      downloadable-by-bcast   About-Flags ::= '10000000'B
      downloadable-by-request About-Flags ::= '11000000'B

      -- If the table is dynamic (in this media)
      -- (applies only to dynamic table)
      -- found in bits '00x00000'B
      is-dynamic        About-Flags ::= '00100000'B
      is-static         About-Flags ::= '00000000'B

      -- If the table is new or recently changed
      -- found in bits '000x0000'B
      recent-change     About-Flags ::= '00010000'B

      -- If the elements of this table are NEVER re-ordered
      -- That is, regardless of revision bit changes the actual
      -- value and entries will never be reassigned
      -- found in bits '000x000'B
      never-ReOrder     About-Flags ::= '00001000'B
      can-ReOrder       About-Flags ::= '00000000'B

      -- These bits: '00000x00'B are reserved
```

```
-- How the table is used by strings (allowed contents)
-- found in bits '000000xx'B
full-string       About-Flags ::= '00000000'B
      -- the string is full of character data as the default
      -- any index will be preceded by an exception token, and
      -- such indexs can appear anywhere in the string
just-1-index      About-Flags ::= '00000001'B
      -- the string in use is then just a one token value
      -- and the entire meaning is contained in that entry
      -- primitive systems (such as classic RDS) would use this
just-indexes      About-Flags ::= '00000010'B
      -- the string is composed of a sequence of index values
      -- and simple text is the exception (rather than the rule)
      -- such text would be preceded by an exception token
index-then-string About-Flags ::= '00000011'B
      -- the first token is an index, the remainder of the string
      -- then follows the format as per the heading values
      -- note that if just a simple string with no indexing
      -- is required, but tables are to be used, select the
      -- full-string option above.
```

It is used in the table header to convey information about the type of table including how its strings are used and if this table is obtainable over the media. Note that these bits define *how* the string may use the table. The variants can select from a complex string (raw simple text and indexes preceded by exception token) to "just one or more indexes" here. Note that any unlimited sequence of indexes must still be terminated with a Hex 0x00 just as a string must (e.g., the rules for shifting into and out of index and tables outlined in 5.5 still apply). Strings composed of a single index or message which have a fixed a *priori* known length do not need this (this is typically defined as part of the using message format).

7.4 Date Element: Index-Start—The Index-Start is a 16-bit value used to define the starting index of the table in which it appears.

Most tables start with a value of "1" (the index of zero being reserved); however some tables of a type called "overlapping" begin at a point past where the table they are overlapping ended. In this case, the value is reflective of the point where the first overlapping index is to be placed.

Note that a 16-bit value is always used, even when the table type uses an index of less bits (8, 11 or 12-bit wide indexes). If for some reason the starting index of the table is not known, the value shall be set to zero.

```
Index-Start ::= Index-16   -- same as: INTEGER (SIZE(16))
```

7.5 Date Element: Index-Stop—The Index-Stop is a 16-bit value used to define the last used index of the table in which it appears.

Note that a 16-bit value is always used, even when the table type uses an index of less bits (8, 11 or 12-bit wide indexes). If for some reason the last index of the table is not known, the value shall be set to zero.

```
Index-Stop  ::= Index-16   -- same as: INTEGER (SIZE(16))
```

7.6 Date Element: Index-Count—The Index-Count is a 16-bit value used to define the total number of entries present in the table in which it appears. Note that this is not deterministic from the start and stop index values, as "sparse" tables are discontinuous in the way they assign indexes.

Note that a 16-bit value is always used, even when the table type uses an index of less bits (8, 11 or 12 bit wide indexes). If for some reason the total number of indexes in the table is not known, the value shall be set to zero.

```
Index-Count ::= Index-16   -- same as: INTEGER (SIZE(16))
```

7.7 Date Element: Index-8—The Index 8 element is defined an 8-bit unsigned value [0 to 255], used as a table index:

```
Index-8 ::=  INTEGER (0..255)
```

The value of zero is valid as an entry, however there is never a valid index entry zero.

Registry Note: The definition of this data element is shared with SAE J2369

7.8 Date Element: Index-11—The Index 11 element is defined as a 12-bit unsigned value [0 to 2047], used as a table index. The most significant bit is always zero. In some media (notably RDS TMC) this data element is transmitted as an 11-bit field when used in a string. In other media, and when handled as a part of these message, it is handled as a 12-bit field to preserve octet-nibble alignment.

```
Index-11 ::=  INTEGER (0..2047)
```

The value of zero is valid as an entry, however there is never a valid index entry zero.

7.9 Date Element: Index-12—The Index 12 element is defined as a 12-bit unsigned value [0 to 4095], used as a table index:

```
Index-12 ::=  INTEGER (0..4095)
```

The value of zero is valid as an entry; however there is never a valid index entry zero.

Registry Note: The definition of this data element is shared with SAE J2369

7.10 Date Element: Index-16—The Index 16 element is defined as a 16-bit unsigned value [0 to 65535], used as a table index:

```
Index-16 ::=  INTEGER (0..65535)
```

The value of zero is valid as an entry, however there is never a valid index entry zero.

Registry Note: The definition of this data element is shared with SAE J2369

7.11 Date Element: Table-Entry—The Table-Entry is a 24-bit value composed of three elements (each defined below) which provides information about the subject table. In a table header there are always five table entries. The first entry refers to that table, while the others refer to up to four other tables, which may be found in the string entries of that table's body section. If fewer than four tables are used, the first entries are filled out and the extra one(s) are set to a value of all zeros. The first entry (the one for the table itself) is always filled out. The Table-Entry is defined as follows:

```
Table-Entry ::= SEQUENCE {
      num          Local-Number,
      flags        Included-Table-Flags,
      rev          Revision
      }
```

In cases where the value of "num" is set to 255 (an unused entry) the value of the flags and revision codes are indeterminate and shall be ignored.

7.12 Date Element: Local-Number—The Local-Number is an 8-bit value assigned by the issuing ISP and used to refer to the table within the context of a set of messages. Multiple strings using multiple tables may exist in the collection of messages issued by the ISP and the use of these tables must remain consistent (with other tables) within this context.

```
Local-Number       ::=   INTEGER (SIZE(8))
```

The values as follows are reserved from use. In addition, values between hex 0x12 and 0x30 are reserved.

All other numbers may be assigned by the issuing ISP. It is recommended that the ISP begin issuing its own tables at the value of 128 and increase them by one (129, 130,…)

The following reserved values are explicitly assigned to tables as noted. The end-using device may presume that these tables are in always in use[1] and, if a table header is present, that their use is in accordance with national Static tables. Data implementers are still urged to provide table messages in support of these lists when the media allows. Data receivers are encouraged to pre-build these tables into their products (implemented in any fashion that suits their memory and resource needs). Note that some static table also have index values assigned which will never be changed by resorting (the never-ReOrder flag). Examples of suitably formulated table headers for these values can be found in the specific standard for each list.

TABLE 8—ASSIGNED LOCAL TABLE NUMBERS

Value 0x00	The subject strings contain NO index values
Value 0x10	The National words Table, in ASCII
Value 0x11	The National words Table of the "first 255"
Value 0x12	The Modified ASCII Table (an Imputed Table)
Value 0x13	The Latin_1 Table (an Imputed Table)
Value 0x14	The National Numbers Table (an Imputed Table)
Value 0x20	The National RDS Phrase List (all phrases)
Value 0x21	The National RDS Phrase List (US linked to CEN)
Value 0x22	The National RDS Phrase List (CEN only)
Value 0x23	The National RDS Extended Phrase List
Value 0x30	The National ITIS Phrase List
Value 0xFF	This table entry is not used

1. This does not imply that the issued string from any ISP uses them, this is an ISP decision.

In addition the following table values are reserved to maintain backward compatibility with other SAE standards which may make use of them. As a general recommendation, new implementations should use list from table 8 rather than from table 9.

TABLE 9—RESERVED TABLE NUMBERING ASSIGNMENTS

Table Name	Use in SAE J2369	Assigned Value
Table of Tables	Required	0x00
National Strings Table	Required, may not be sent	0x01
National Numbers Table	Required, is created not sent	0x02
Grid table	Required, first entries defined	0x03
Model Tables	Required, first entries are defined	0x04
Incident Modeling Table	Required if not all in line	0x05
Incident Numbering Table	Required if not all in line	0x06
Forward Referencing	Optional use	0x07
Local Strings Table	Required if any local text is used	0x08
Alternative Names Table	Optional use	0x09
Temp Strings Table	Required, if not all in-line	0x0A
Textual Incident Encoding Table	Optional use	0x0B
Private Text Strings	Optional use	0x0C
RDS Incident Encoding Table	Optional use	0x0D
Resource Availability Table	Optional use	0x0E
Transit Adherence Encoding	Optional use	0x0F
Transit Schedule Encoding	Optional use	0x10
Weather Table	Optional use	0x11

Registry Note: The definition of this data element is shared with SAE J2369

7.13 Date Element: Included-Table-Flags—The Included Table Flags is an 8-bit value which is used to provide information about the structure, character set, and token size of a tables entries. This information is needed to successfully decode the table body entries.

Note that is this information about the table of strings NOT the string in which and index appears that refers to that table (that is encoded in the table header in the about flags element of the local flags).

Observe that an 8-bit index found in a given table string entry could refer to a 16-bit index in its native table. Alternatively, a 16 bit index may appear in a string and be used in an 8-bit table. Extension or truncation of the index found in a string shall be applied as needed to meet the encoding of the included table.[1]

It is defined as follows:

1. It is the responsibility of the issuing ISP not to attempt to use an index in a string which cannot be truncated (e.g., a 16-bit index in a string for a table with only 8, 11, or 12 bit indexes). It is legal to use the lower set of indexes from a table in a string with table encoding of a lesser length (e.g., in a string using 8 bit indexes, an index can still map to the first 255 entries of a table with >255 entries.) Of course this would preclude using any entry from the table higher than the index allowed in the carrying string.

```
Included-Table-Flags    :==   BIT STRING (SIZE(8))
      {
      -- definition provided below, composed of terms
      -- logically and-ed together to form final word

      -- How the table is constructed
      -- found in bits 'xx000000'B
      dense-no-Overlap  Local-Table-Flags ::= '00000000'B
      dense-Overlap     Local-Table-Flags ::= '01000000'B
      sparse-no-Overlap Local-Table-Flags ::= '10000000'B
      sparse-Overlap    Local-Table-Flags ::= '11000000'B

      -- The nesting used in the table
      -- found in bits '00xx0000'B
      self-nests  Local-Table-Flags ::= '00100000'B
            -- points to self in indexes of this table
      other-nests Local-Table-Flags ::= '00010000'B
            -- points to other tables in indexes of this table

      -- The size of the index bit length
      -- found in bits '0000xx00'B
      idx8        Local-Table-Flags ::= '00000000'B
      idx11       Local-Table-Flags ::= '00000100'B
      idx12       Local-Table-Flags ::= '00001000'B
      idx16       Local-Table-Flags ::= '00001100'B

      -- The character set used in the string
      -- found in bits '000000xx'B
      ascii       Local-Table-Flags ::= '00000000'B
            -- 8 bit tokens and terminations in strings
            -- 8 bit table selection exception token
      mod-ascii   Local-Table-Flags ::= '00000001'B
            -- 8 bit tokens and terminations in strings
            -- 8 bit table selection exception token
      latin-1     Local-Table-Flags ::= '00000010'B
            -- 16 bit tokens and terminations in strings
            -- 16 bit table selection exception token
      unicode     Local-Table-Flags ::= '00000011'B
            -- 16 bit tokens and terminations in strings
            -- 16 bit table selection exception token
      -- binary files shall use the unicode keyword
            -- 16 bit tokens and terminations in string entries
            -- al table entries preceded by a word count value
            -- table selection exception token not used
```

7.14 Date Element: Request-Type—The Request Type is a 16-bit value which is used to select what type of table information or translation is desired or to encoded what type of reply is being provided

It is defined as follows:

```
Request-Type      ::=   BIT STRING (SIZE(16))
      {
      -- definition provided below, composed of terms
      -- logically and-ed together to form final word

      -- The flattening requested or returned
      no-flat     Local-Table-Flags ::= '00000000'B -- do not flatten
      all-flat    Local-Table-Flags ::= '01000000'B -- flatten to a string
      just-local  Local-Table-Flags ::= '10000000'B -- remove 'local' items
      first-flat  Local-Table-Flags ::= '11000000'B -- reduce only top level

      -- The success of the request involved
      none  Local-Table-Flags ::= '00000000'B  -- use for requests as well
      some  Local-Table-Flags ::= '00000100'B  -- one or more items in
                                               -- a multi item request were
                                               -- not able to be processed
      all   Local-Table-Flags ::= '00001100'B  -- all item processed ok

      -- The size involved
      whole       Local-Table-Flags ::= '00000000'B
      complete    Local-Table-Flags ::= '00000100'B
      partial     Local-Table-Flags ::= '00001000'B
      single      Local-Table-Flags ::= '00001100'B

      -- The character set used in the string
      -- this may be the 'preferred' or requested char set
      -- found in bits '000000xx'B
      ascii       Local-Table-Flags ::= '00000000'B
            -- 8 bit tokens and terminations in strings
            -- 8 bit table selection exception token
      mod-ascii   Local-Table-Flags ::= '00000001'B
            -- 8 bit tokens and terminations in strings
            -- 8 bit table selection exception token
      latin-1     Local-Table-Flags ::= '00000010'B
            -- 16 bit tokens and terminations in strings
            -- 16 bit table selection exception token
      unicode     Local-Table-Flags ::= '00000011'B
            -- 16 bit tokens and terminations in strings
            -- 16 bit table selection exception token
      -- binary files shall use the unicode keyword
            -- 16 bit tokens and terminations in string entries
            -- al table entries preceded by a word count value
            -- table selection exception token not used
```

7.15 Date Element: Revision—The revision element is an 8-bit value which is used to reflect updates to tables and is defined as:

```
Revision    ::=   INTEGER (0..255)
```

It is incremented by one every time the object to which it refers (typically a table), is updated. Updating is of an extending nature rather than a re-numbering nature (e.g., if more items are added to an existing table rather than the items in a table being removed, and/or re-ordered and hence re-numbered). If the object is re-numbered, the value shall be incremented such that the fourth bit will change value (increased to the next modulo 16 value) and the lower bits will be set to zero. If the value is 0xFF, an increment of one or to the next modulo value would produce a new value of 0x00.

The effect of this value encoding is to allow up to 15 incremental updates to occur and be reflected in the revision data element. Changes in the revision value beyond every 16th, or as needed, implies (but does not guarantee) that a re-ordering of the objects may have occurred. Therefore, in a receiver device using a table with a revision which has incremented by a value less than modulo 16, the device can continue to use its current table with confidence that none of the entries are outdated and can seek to recover the new table or simply add the new entries to its current table. By contrast, a renumbering implies that the receiver may no longer use its current table with confidence and must obtain the new table.

Registry Note: The definition of this data element is shared with SAE J2369

7.16 Date Element: Word-Count—The Word-Count is a simple count of the octets which follow for the binary table entry. It does not include the 16 bit end of entry delimiter in the count. It is used to determine the end of the entry and is required because the termination character (which is sufficient in the textual cases) may occur in the contents of the binary entry itself.

It is defined as:

```
Word-Count  ::= INTEGER (SIZE(16))
```

7.17 Date Element: Entry-Type—The Entry-Type is three octet value used to denote the type of table entry contents. At a gross level, all tables are either textual or binary in nature. All entries in a table are of the same type.

Textual tables may nest and build up complex expressions based upon the rules of this document. The entries in the table header provide definitive information regarding the resulting table structures.

Binary tables are also free to be nested, however no such rules are provided by this document. The meaning of the five table entry values in the case of a table with binary table entries is not defined by this document. The rest of the table header SHALL remain as defined by this document.

In general, while textual tables are often made up of a local table nested to a national table, binary tables are not nested and their table-entry values are set to zero.

It is defined as:

```
Entry=-Type ::= OCTET STRING (SIZE(3))
```

The values of 0x00,00,00 or of “TXT”, “Txt:”, “txt” or any other possible combination of capitalization, is taken to imply that the table entries will be made up of one of the four types of character set data. Any other values SHALL be taken to imply binary entry contents. The recommended value SHALL be all zeros (0x00,00,00) for textual entries.

It is the responsibility of the data issuer and data user to agree upon the structure, the format, and the use for all binary entries.

Many popular file extensions exist and the three octet encoding can be used to reflect such contents. For example a Entry-Type of “WAV” might represent a popular sound file format. Other derivatives such as “Wav” and “wav” might also represent that same files format, but this document does not assume that. It is recommend that all entry types use lower case ASCII when possible to minimize this problem.

Experimental table entries content, and formats which do not follow various industry ad hoc standards are recommend to start with a first octet value other than the letter and number sets (A-Z , a-z, and 0-9).

Entry-Type values beginning with the first octet value equal to 0x2B (expressed as the plus character “+” in ASCII) are reserved for future use.

7.18 Date Element: CRC-16—The CRC-16 is a simple checksum generated as described as follows. It is used, in conjunction with the table registration number and its revision value, to determine if the table has changed. It is not intended to be used in place of adequate error detection and correction functions that the lower layers of the host transport media are presumed to provide.

It is defined as:

```
CRC-16      ::= INTEGER (SIZE(16))
```

The CRC shall be computed over every element bit of the outgoing message starting with the initial byte and continuing up to the last data element. It shall skip over the 2 octet field where the CRC resides. The final 16-bit CRC value calculated shall then be inserted in the location specified in the message. The generating polynomial to be used will be the “CRC-CCITT” polynomial comprised of:

$$G = x16 + x12 + x5 + 1$$

The CRC shall be initialized to a value of all ones before the computation of the CRC. The final format order of the message shall be:

$$S_1\ S_2\ S_3\ S_4 \ldots S_N\ \ C_1\ C_2\ \ S_{N+1}\ S_{N+2}\ S_{N+3} \ldots S_M$$

Where S_1… represents the stream of octets which form the message. C_1 and C_2 represent the computed value of the CRC, with C_1 being the most significant octet. A non-normative code example of how to implement a CRC generator in software using a common table look-up method can be found in SAE J2313.

Registry Note: The definition of this data element is shared with SAE J2313

7.19 Date Element: SAE-String—The data element SAE-String is a (typically) multi-bit sequence which is composed of three types of elements: raw text in the selected character set, exception tokens to select tables, and table index values. The specific ordering of these will vary to provide unlimited string contents. A dual perspective regarding this is that the SAE_String is composed of a sequence of simple text interspersed with indexes to other text, or alternatively, that it is composed of a sequence of indexes interspersed with simple text. Either viewpoint is equally valid.

The interpretation of the “raw bytes / bits” found in the string is the subject of this document. By selecting various “dialects” of string (by way of the setting of flags in the table header) the need for the presence of exception tokens in the string before the index, or for the use of interspersed simple text can be removed, leaving only index values in the string.[1] This is determined by the setting of the flags controlling the table which the string is referenced to.

Strings are most often used as data elements in other messages that are not a part of this document. When a string is used in a message it must be possible to associate it with a local table to interpret it by. This is the responsibility of the message structure definition (and at times the media conventions) in use. Strings are used in the table entries of the message of this document.[1]

A limit of 1000 bytes or bits is used in the definition as follows. The actual limiting length will be determined by the message which uses the string, and no such limit is created by this definition. Often the resulting length will be expressible as an octet value, but this need not be so. Any padding bits added at the end of an octet based instance of the string shall be set to zeros.

```
SAE-String  ::=   OCTET STRING (0..1000)  -- Decoded as per SAE J2540

SAE-String  ::=   BIT STRING  (0..1000)  -- Decoded as per SAE J2540
```

8. Using this Document in Other Messages—This section provides guidelines for using this document in the string and phrases found in other standards and formats. Consult the various actual list standards for further information about using a specific list in a specific message set context.

8.1 General Comments for Use—Once a table is established and sent to the end user (typically identified in the start of a message exchange session), then the use of strings adhering to the conditions of this document can begin. Strings appear throughout ATIS messages to convey a wide variety of information.

The string is a (typically) multi-bit sequence which is composed of three types of elements: raw text in the selected character set, exception tokens to select tables, and table index values. The specific allowed ordering of these can found by examining the flag words in the table header. Note that this provides for index strings to be used in any message as follows:

a. Not at all (e.g., just simple text to be used in the message with a final termination token)
b. As a single index (e.g., one value will be found in the subject message element of the length selected and no termination token is required). All text MUST be selected from prior entries in the selected table.
c. As a multiple sequence of indexes (e.g., after an index is received the next token is also an index with no intervening raw text or exception token). No simple text may be present, except as noted. The string is terminated by either being of a known length or (more normally) by a termination character of Hex 0. Simple text may be inserted by using the toggle mode exception. This is often the best choice for describing traffic incidents (which are composed of reoccurring phrases with occasional odd words inserted).
d. Any combination of indexes and simple text is allowed. This is the most flexible option but does require an exception token before each index and a termination token at the end of the string. This is the preferred option for all messages sending strings to use when conditions allow.
e. As an initial index, followed by any combination of text and indexes. By this variation the need for an initial exception token is eliminated with some saving, yet the ability to send any possible combination of text is preserved. A termination token is still always required.

---

1. This type of setting reverts the string contents to being just an index value and preserves backward compatibility with existing *index only* systems such as RDS-TMC. At the other extreme, if no indexes are used, then the string may be composed of only simple text as found in the wireline standards of ITS.
1. In this case the local table value is provided in the table header.

Each of these modes has an appropriate time for its proper use. Sometimes the media forces restrictions in this regard as well (for example RDS-TMC). Sometimes the format from the data issuer will change modes based on the content then available to be sent. The using message set or transport media may contain other restrictions, effectively making requiring or allowing only the use of a sub-set of the allowed operations of this document. This can effect the range of expressions allowed to be encoded (for example preventing the use of free text). In any of these modes the key concept is the coupling of all strings to its relevant local table. Given the local table (and the information in the table header) the complete decoding of a string can be obtained by the receiver device in a uniform manor regardless of other restrictions at the encoding end.

8.2 Connecting Strings Across Multiple Message Fields—In some extremely limited bandwidth media it may be necessary to have the lower levels of the receiver software in the end user device reconstruct the message from compromises made in transmission into formats again meeting the requirement of the ATIS message set. This is not to be viewed as a shortcoming with respect to any media that performs this, however the reconstruction is needed if the resulting message set is to be compatible with this document and other standards of the SAE. By maintaining such compatibility, software developed by one firm (e.g., receiver data recovery SW), can successfully pass ATIS messages to software developed by another firm (e.g., SW using ATIS for routing or display) without overt coordination. It is expected that case-by-case definitions of this type of mapping will be developed in the future as ATIS message are profiled into specific media types.

A simple example of this is the auxiliary information contained in an 8-bit index message field found in the some RDS-TMC systems. In order to extend the messages which could be carried in the primary 11-bit index field (of the 37 bit message) a second message was sent which contained (among other items) an 8-bit field indexed into a set of phrases to be appended to the primary phrase. Examples being “your parking ticket covers the return ride” or “due to holiday traffic.” In terms of this document, these can be considered two cases of single index only fields. A receiver device, desiring to reconstruct an ATIS message could extract these two fields and append them as follows. One possible process would be to simply take the 11-bit index, add the proper exception token before and after it, and append the extension word. The result would be a fully qualified string (presumably referring to the two tables which this layer of software would also send to the next layer) which any ATIS receiver meeting this document could decode and use.

8.3 Examples of Use in SAE J2354—In general, SAE J2354 messages which employ strings (or the optional use of any strings) should have inserted in a new first byte of the message content the local table being used. The remainder of the messages would be unchanged. The strings themselves and the messages themselves would not require changing.

The message set was developed with the concept of existing with other messages in mind. Hence the adding of support for the table delivery messages is not a problem.

The table request messages described in this document should also be supported, but is possible that they can be merged with the existing request function messages. This needs to be studied further.

An example of how this would work in practice is shown as follows with the EventInformation message used to deliver a variety of current and reoccurring incident information. The changes to the message are shown in bold type. The strings, now meeting this document, are shown in Bold Italics.

```
EventInformation ::= SEQUENCE {
      table                          Local-Number,-- the table we are using
      eventLocation     LocationReference,
      event-DescriptionTypeEvent     OCTET STRING (SIZE(8)),
      subType                        EventSubType OPTIONAL,
      event-Description              OCTET STRING (SIZE(0..255)) OPTIONAL,
      event-LanesBlockedOrClosedCount INTEGER (0..255) OPTIONAL,

      laneClosedList                 SEQUENCE OF INTEGER(0..32) OPTIONAL,
      laneConfigurationList          SEQUENCE OF INTEGER(0..32) OPTIONAL,
      event-LanesDirectionOfTravel   OCTET STRING (SIZE(8)) OPTIONAL,
      event-TimeLineStart            DateTimePair OPTIONAL,
      event-TimeLineEstimatedDuration INTEGER  (0..4294967296) OPTIONAL,
      event-TimeLineScheduledEnd     DateTimePair OPTIONAL,
      timeList                SEQUENCE OF IA5String (SIZE(9)) OPTIONAL,
      dayList                 SEQUENCE OF IA5String (SIZE(10)) OPTIONAL,
      event-TimeLineScheduleDaysOfWeek    ATIS-DayOfWeek
}
```

8.4 Examples of Use in SAE J2369—The structure of SAE J2369 is primarily to allow sets of reports to be effectively broadcast to end-using devices. While SAE J2354 is a 2-way transactions format in nature, SAE J2369 is a broadcast (typically one-way) type of message set. It makes extensive use of tables in the original design and contains a number of the concepts outlined in this document. With a more complete and uniform set of features offered by this document, a number of the optional settings in SAE J2369 become superfluous. In addition provisions for 8, 11, 12, and 16-bit index are made.

It is recommended that the tables, which SAE J2369 defined, should be redefined to be in exact harmony with the table document. Refer to Table Nine for table number assignments which are backward compatible with SAE J2369. It is recommended that new implementation use the lists found in Table 8.

As with SAE J2354, it is necessary to send the local table number. There are already provisions in the SAE J2369 messages to do this in the “message context” portion of suitable messages. All messages in the “message payload” which follow the context refer to this local table number. For example, in the incident flow message the “event table” provides this value:

TABLE 10—INCIDENT FLOW CONTEXT TABLE ENTRIES (FROM TABLE 24 OF SAE J2369)

Grid Index	1 byte	/ the Grid used
Issuing Time	2 bytes	/ the time for which offset models are issues
Local Name Table	1 byte	/ the local segment table used
Model Table	1 byte	/ the shared model table used by these MSG’s
Event Table	1 byte	/ the event stream for text & codes
Private Event Table	1 byte	/ the event stream for additional text & codes

The defined data element *the_string* does not change as a function of this document. Any decoder meeting this document can also decode strings from SAE J2369 and would be fully backward compatible.

Note that the table provides in the SAE J2369 standard general textual phrases that can be converted to this document or left in that standard as the implementers of SAE J2369 desires. It is recommended that this document be used for new work.

An example of the key “incident flow message“ implemented using the string method of this document would be exactly the same as the standard currently specifies. E.g., a link value (in a variety of GRID formats) would be followed by an option sequence of information (flow rates etc.) and an optional sequence of RDS phrases, indexed only strings, or general strings.

Note that there will be some minor redundancy in encoding the type of strings between the local table flag and the flag found in the message context area.[1] In general, it is recommended that new work encode the context areas as a general string and then vary the actual string encoding by control of the local flags of the table as described by this document.

PREPARED BY THE SAE ADVANCED TRAVELERS INFORMATION SYSTEM COMMITTEE

1. In the SAE J2369 body of work, some of the encoding concepts of this document were attached to a “header” of each message rather than to a table for the strings as is done in this document.

Rationale—Not applicable.

Relationship of SAE Standard to ISO Standard—Not applicable.

Application—This SAE Standard defines methods and messages to efficiently translate sequences of text and other types of data into and out of indexed values and look-up tables for effective transmission.

This document defines:

a. Methods and Data Elements for handling indexes and strings in ATIS applications and message sets
b. Message Sets to support the delivery and translations of tables used in such strings
c. Tables of Nationally standardized strings for use in ATIS message descriptions

And examples of each in illustrative portions.

While developed for ATIS use, the methods defined in this document are useful for any textual strings in any Telematics applications found both in Intelligent Vehicles and elsewhere.

Reference Section

SAE J2313—On-Board Land Vehicle Mayday Reporting Interface, May 2000

SAE J2353—Data Dictionary for Advanced Traveler Information Systems (ATIS), October 1999

SAE J2354—Message Sets for Advanced Traveler Information Systems (ATIS), October 2000

SAEJ2369—Standards for ATIS Message Sets Delivered Over Reduced Bandwidth Media, November 2000

SAEJ2374—Information Report based on Location Reference Message Specification, Rev. B (MDI), May22, 1997

SAE J2350-1—RDS Phrase Lists, Rev. 1.0, SAE staff will Insert Publication Date

SAE J2350-2—ITIS Phrase Lists, Rev. 1.0, SAE staff will Insert Publication Date

SAE J2350-3—National Names Phrase Lists, Rev. 1.0, SAE staff will Insert Publication Date

ANSI X3.4—American National Standard Code for Information Interchange

Traffic and Traveller[1] Information (TTI) – TTI Messages via Traffic Message Coding- Part I: Coding protocol for Radio Data Systems – Traffic Message Channel (RDS-TMS) using ALERT-C, ISO/DIS 14819-1, August 1999

Traffic and Traveller Information (TTI) – TTI Messages via Traffic Message Coding- Part 2: Event and Information codes for Traffic Message Channel (TMC), PrENV 12313-2, Version 1.0, June 1996

GATS Message Set; Main Part Main 2034.pdf 05.08.99

GATS Message Set; Annex 2 - Basic Information Elements A2_2033.pdf 05.08.99

1. European spelling of Traveler

National ITS Architecture, Federal Highway Administration, U.S. Department of Transportation, Version 2.0, September 1998

Message Sets for External Traffic Management Center Communication (MS/ETMC2), TM 2.01, Rev. 2.0, February 1, 2001

“Hypertext Transfer Protocol -- HTTP/1.1,” W3C - June 1999, RFC 2616, available at http://www.w3.org/Protocols/rfc2616/rfc2616.txt

“HTML 4.0 Specification,” W3C - December 1997, revised 24 April 1998 available at http://www.w3.org/TR/1998/REC-html40-19980424

HTML 4.0 Guidelines for Mobile Access W3C Note - 15 March 1999 available at http://www.w3.org/TR/NOTE-html40-mobile

Character Model for the World Wide Web, World Wide Web Consortium Working Draft 29-November-1999 available at http://www.w3.org/TR/1999/WD-charmod-19991129/

Developed by the SAE Advanced Travelers Information System Committee