Bellcore

Bell Communications Research

BELLCORE PRACTICE
BR 252-573-305
ISSUE 7, DECEMBER 1996
CSAS RELEASE 8.5

TCM Translation Administration

TCM Translation Administration

Contents

List of Figures

List of Tables

1. General

NOTE — For CSAS documents, any reference to "ZRxxxx" should be changed to "VMxxxx". Also, any reference to "RMxxxx" should be changed to "MMxxxx".

1.1 Document References

This document is one of a series of user manuals that describes the Bellcore CSAS Communications Module (TCM). The list of documents follows:.

Number	Title
252-573-260	TCM Online Message Directory
252-573-301	TCM Overview
252-573-302	TCM Route Administration (RA)
252-573-303	TCM Message Administration (MA)
252-573-304	TCM Network Administration (NA)
252-573-305	TCM Translation Administration (TA)
252-573-383	TCM Cron User Manual
252-551-703	CSAS Interface Data Catalog
252-551-791	Planning Transition Guide for a SOP to CSAS Interface
252-541-230	TQS User Manual

1.2 Introduction To Translation Administration

Translation Administration (TA) is a component of TCM. Translation Administration defines a translation rule language, provides compiling and loading facilities and an executor. These functions provide an environment and a set of tools enabling translation of data sent to the *local* TCM from any *external* system *as well as* from the local TCM to any external system.

Before and after the translation, data is converted from/to FCIF (Flexible Computer Interface Format, the form of non-TCM system data), to/from the set of TCM system data names that is understood by CSAS application subsystems. Because the translation rules are application specific, all examples will be in generic terms.

The format of FCIF data is data name-data value pairs which are grouped into a hierarchy of aggregates. (Refer to Section 1.3 for a definition of an "aggregate".) The data names received by a TCM from an external non-TCM system are usually, but not necessarily, from the set of CSAS system data names. The data values received should be those expected by the CSAS System. Similarly, the data names and data pairs sent by the

"Home" TCM to an external non-TCM system or to an external TCM system in a different IMS Region are those known to the TCM module.

If the data names and/or data values received by a TCM are not known to the receiving CSAS system, the user can specify the translation rules needed to change the input. Likewise, if the data sent by a TCM are not the desired form for the receiving system, the user can specify translation rules to be performed while TCM is processing the message.

The TCM Parser and Mapper (TPAM) Translator applies translation rules to the TCM/ TPAM input. TCM does not validate the input data, it just processes the data (messages).

1.3 TA Translation Tools

The TA translation tools allow data name changes and data value changes within aggregates. They do not allow changes to an "aggregate" name, i.e., at the level of aggregates. An input "aggregate" to TPAM is defined as data grouped by name-value pairs, in hierarchical order (nested), and preceded by an aggregate name. The aggregate names are a known group of names. (Please refer to Section 3.) The term "*tool*" is used to indicate a general capability, and "*rule*" is used to indicate a specific instance of a tool. The currently available translation tools are:

1. Data value replacement and field name change;
2. Data value change via table look-up;
3. Deletion of field name(s);
4. Manipulation of data strings;
5. Conditional execution of translation rules;
6. Conditional execution of a block of translation rules;
7. Previewing of a lower level aggregate
8. Comments.

The translation tools provided are not a replacement for a re-formatter. However, the translation tools are needed for the following reasons:

1. The translation tools include the Bellcore CSAS Table Management System (TTS), a facility that may not be present in external non-TCM systems. Any re-formatting that requires table look-up may be done as a last step through translation.
2. Similarly, if a CSAS system application is interfacing with more than one version of a re-formatter, the differences can be accommodated via translation rules.
3. The need for small changes may arise from time to time. If these small changes are CSAS system related, the external non-TCM system user may find it faster and easier

to put a translation rule in the TCM machine than to negotiate a change to a reformatter.

4. Variations in the data caused by regional differences can be performed by translation rules. In a company with more than one TCM, the translation tools can be used to manage simple differences between TCM machines rather than having separate reformatters for each machine.
5. Simple release differencing from TCM to TCM can be done using the translation rules.

2. Translation Administration Process

Translation Administration (TA) defines a translation rule language, provides compiling and loading facilities and an executor. Batch run VMMPM02 compiles the translation rules; and the TCM Parser and Mapper (TPAM) Translation Controller dynamically calls or invokes the loader and executor as part of the Translator portion of TPAM. This process is depicted in Figure 2-1.

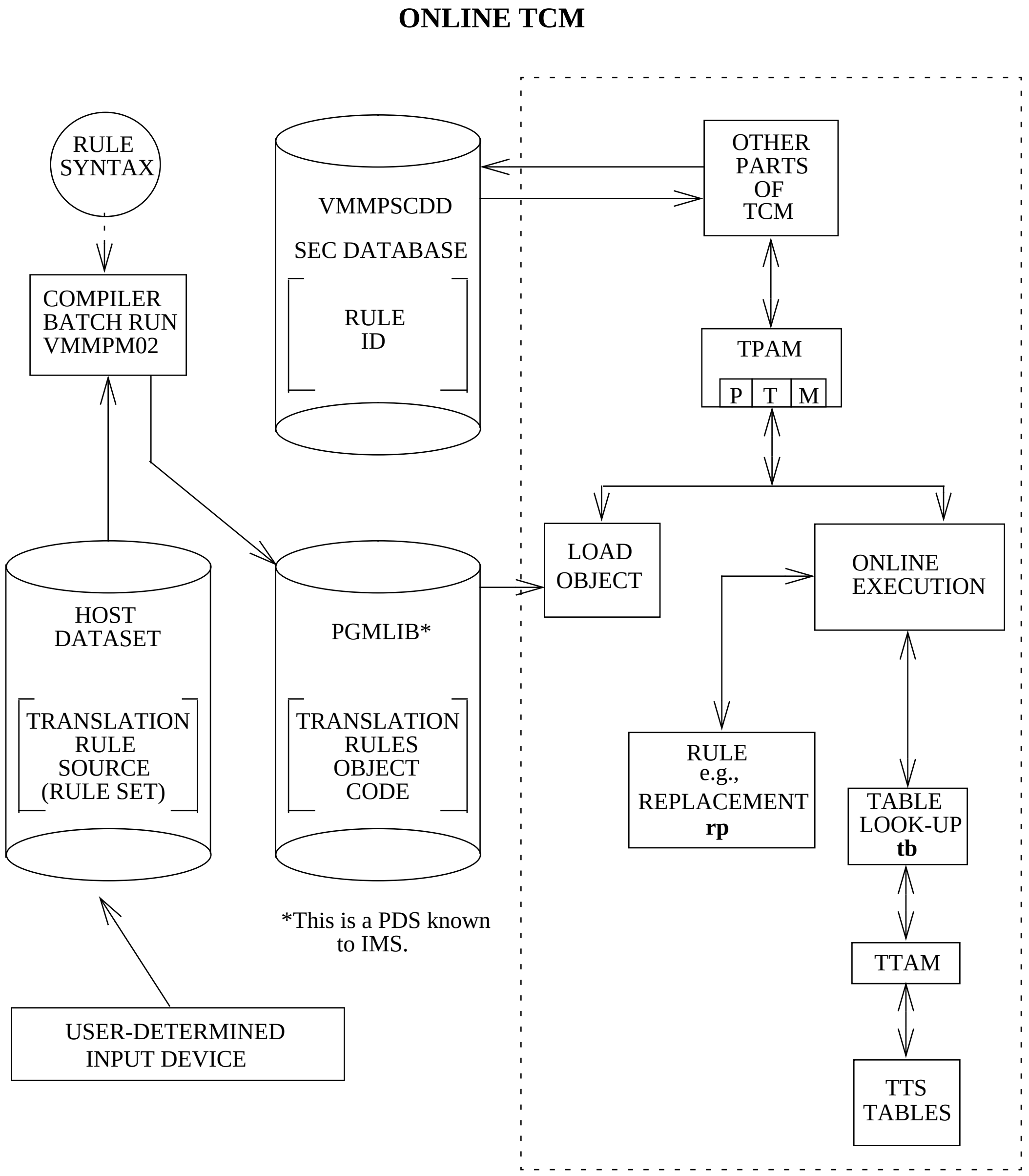
ONLINE TCM
RULE SYNTAX
COMPILER BATCH RUN VMMPM02
VMMPSCDD
SEC DATABASE
RULE ID
OTHER PARTS OF TCM
TPAM
P
T
M
LOAD OBJECT
ONLINE EXECUTION
HOST DATASET
TRANSLATION RULE SOURCE (RULE SET)
PGMLIB*
TRANSLATION RULES OBJECT CODE
RULE e.g., REPLACEMENT rp
TABLE LOOK-UP tb
TTAM
TTS TABLES
*This is a PDS known to IMS.
USER-DETERMINED INPUT DEVICE

Figure 2-1. Translation Administration Process

2.1 Rule Set Identification

2.1.1 Rule Set ID For Rule Sets

Each translation rule set is applied to a pre-defined “category” of messages. The rule set must be given an identification that can be related to the message categories. The name of the rule set, e.g., the *rule set id*, categories, and the association of rule IDs with categories must satisfy the following criteria:

1. The user needs to associate individual rule sets by their application.
2. Rule sets for system to system release translation must be associated with system release numbers.
3. Because a rule set will generally have a version associated with it, a means is needed to identify rule set versions.
4. A version of a rule set may span more than one system release. A rule set for an external non-TCM system application may be used for many CSAS system releases. A rule set for release translation from the “Home” TCM system to an external TCM system may also span releases.

2.1.1.1 Rule Set ID

The rule set creator selects the rule set id. The rule set id *must* be unique within the local TCM. The version number may be included as part of the rule set id. The maximum size of the rule set id is eight (8) alphanumerics. The rule id *must* be stored in the appropriate Path segment of the Network (SEC) Database before online execution of the translation rule set (refer to the TCM Network Administration (NA) User Manual, BR 252-573-304).

The creator of the rule ID may choose any meaningful name. For example, assume that the user would like to assign a rule ID to the combination of Release 6.3 of System A, Release 4.2 of System B, and Version 1. The user might construct the rule ID as follows:

System A	6.3
System B	4.2
Version	1
Rule ID	A63B4201

2.1.1.2 Matching a Message to a Rule Set ID

Messages are mapped to the rule set id by *path*. A path is uniquely identified by three fields:

1. The *external* SEC (as described below);
2. The “PATHID”;
3. The “SCTYPE” (scenario type);

As each message arrives at the local TCM, the first IMS segment starts with a *ROUTCTL section. The *ROUTCTL section contains the necessary 3 fields of the path which provides the key to the optional rule set id. For messages going to an external SEC, the external SEC is identified by the RSYS (receiving system). For messages arriving from an external SEC, the external SEC is identified by TSYS (transmitting system). RSYS is also known as the “target SEC” and TSYS is also known as the “source SEC”.

Each combination of computer system, IMS copy, and product line (SPL) in the network has a unique System Entity Code (SEC) to identify the message origin or message destination.

The local SEC matches *my SEC* as defined in the local copy of the TTS table “TCM USER CONTROL”. The non-matching SEC is the external SEC. This TTS table must be complete before any messages are sent.

The external SEC to the local SEC is further qualified by the “PATHID” in order to allow variations including: the rule set id, the MFD and the receiving transaction. The rule set id and the MFD permit TCM to handle specific data in a very generic manner. Since the path controls what data can go over the path, different paths may be required between a given pair of SECs.

The scenario type indicates whether the data is application data or an acknowledgement to the application data. Certain scenario types also indicate the presence of application headers (*C3/*C0).

The combination of these 3 fields (external SEC, PATHID, SCTYPE) also point to the path segment of the Network (SEC) Database, where the required MFD and the optional rule set id are stored.

2.1.2 Resolution of Rule IDs by Scenario

Each message handled by TCM has a “Scenario Type” in the SCTYPE field in the message header. The Scenario Type (SCTYPE) along with the “remote SEC” (RSYS or TSYS) and PATHID determines the routing of a Class 1 Message, and supplies the final piece of data needed to determine the desired rule ID in the SEC Database. Refer to the TCM Routing Administration (RA) User Manual, BR 252-573-302 for a description of Class 1 Messages. The possible scenarios are summarized as follows:

SCTYPE	SCENARIO DESCRIPTION
S	T/T(S) - TCM to TCM (sending) The TCM has received a message from its local TCM (the sending system) for transmittal to a remote TCM (the target or receiving system).
R	T/T(R) -TCM to TCM (receiving) The TCM has received a message from a remote TCM (the sending system) for transmittal to its local TCM (the target system).
A	T/NT - TCM to non-TCM The TCM has received a message from its local TCM (the sending system) for transmittal to a remote non-TCM (the target system).
Z	NT/T - non-TCM to TCM The TCM has received a message from a remote non-TCM (the sending system) for transmittal to its local TCM (the target system).
I	INTRA-SEC Within the same IMS control region.

The local TCM is the one that resides in the same IMS control region as the application.

The user can enter the TSYS or RSYS, PATHID, and SCTYPE and display the rule ID to be executed on the MMPNET screen. For a description of the operation of the MMPNET screen, refer to the NA User Manual, BR 252-573-304.

The combination of the TSYS or RSYS, PATHID, and SCTYPE determines which rule ID will be fetched from the SEC Database, and, therefore, which rule set will be executed. These combinations are illustrated in Figure 2-2 for TCM to TCM (scenarios “S” and “R”); and in Figure 2-3 for TCM to non-TCM (scenario “A”) and for a non-TCM to TCM (scenario “Z”).

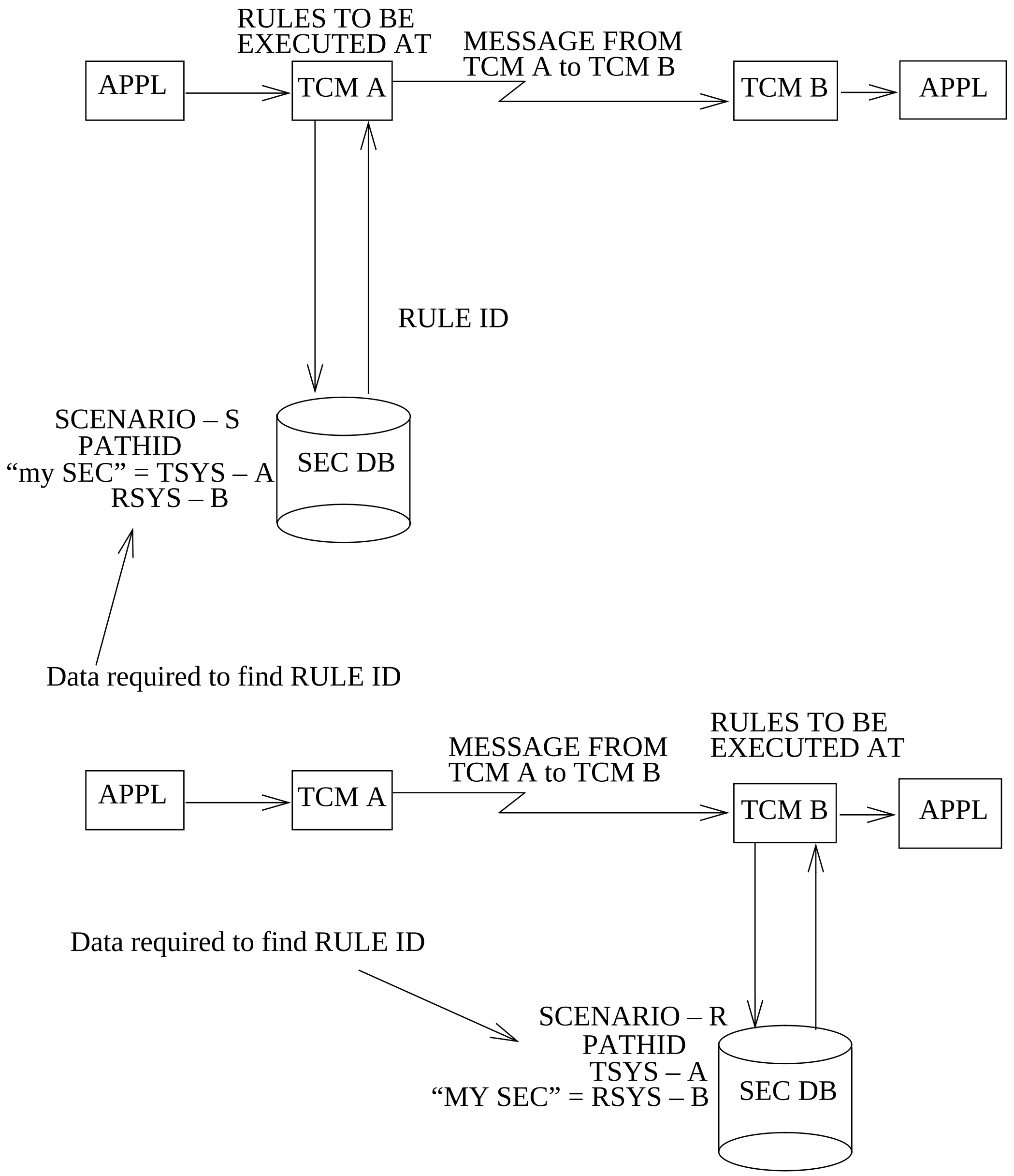

Figure 2-2. Data Required to Determine Rule ID for Scenarios S and R

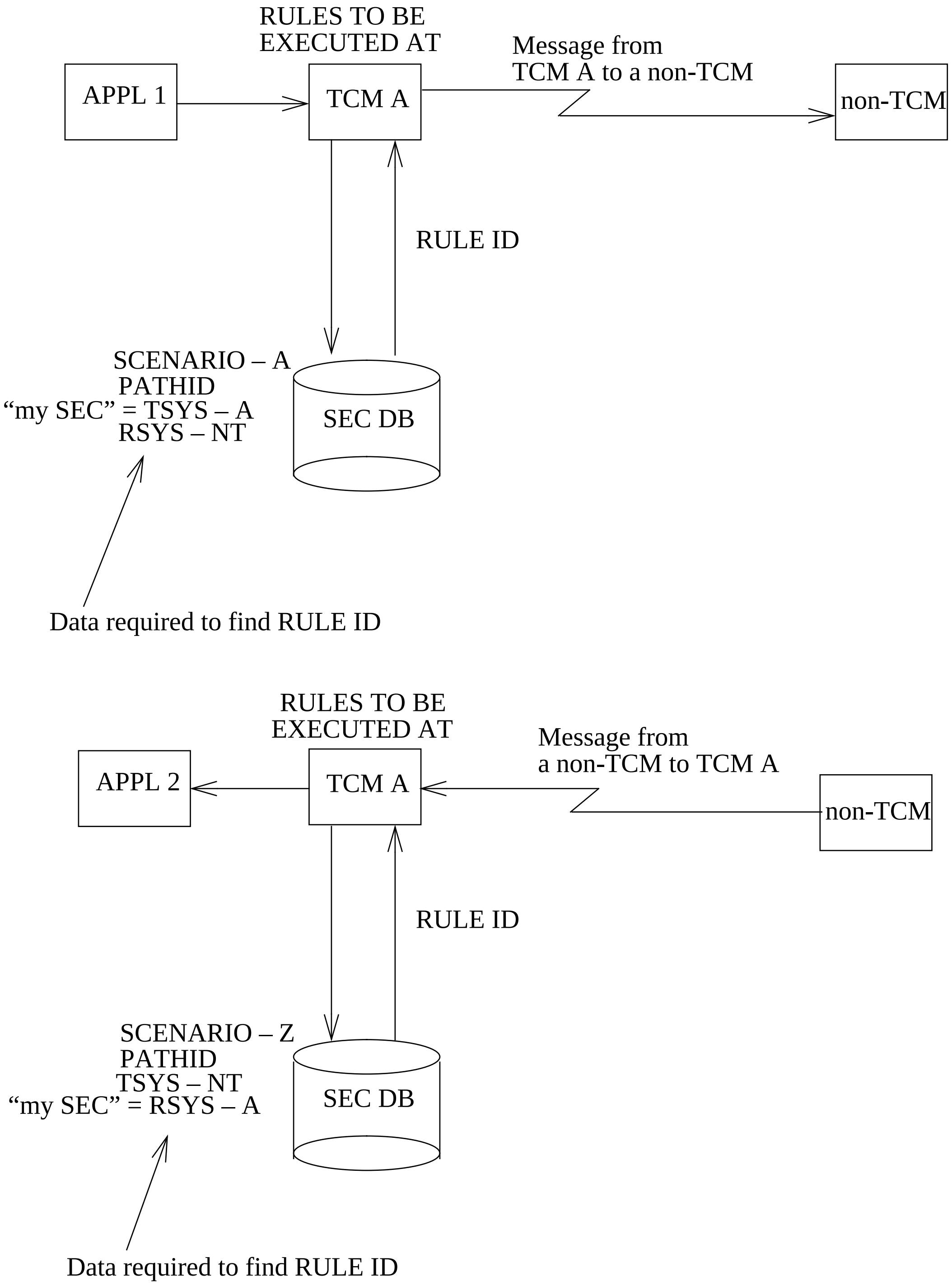

Figure 2-3. Data Required to Determine Rule ID for Scenarios A and Z

Note that the three pieces of data determining the rule ID selected from the SEC Data Base did not include “my SEC”. For example, in Figure 2-2 for the “S” type scenario, the TCM designated as “my SEC” is the Transmitting System (TSYS). The Receiving System (RSYS) and the PATHID to be used are known from the message header. TCM A uses the combination RSYS, PATHID, and SCTYPE to locate the rule ID of the rule set to be executed.

2.1.3 Rule Sets for Different Combinations of TSYS or RSYS, PATHID and SCTYPE

One SEC record exists in the SEC Database for each SEC in the network. Only one of these SECs is the “my SEC”. Each SEC record has one SEC segment (the “root” segment) and potentially many subordinate (“child” segments) referred to as Path segments. The Path segments reflect the various combinations of paths to and from “my SEC” and other SECs in the network.

In particular, the Path segments are used to store each combination of PATHID and Scenario Type (SCTYPE). Each Path segment also has storage space for the rule ID associated with the PATHID and SCTYPE. Note that PATHID can have many different values. So for each SEC, there may be multiple rule sets for each path direction, one per PATHID.

For example, for scenarios “S” and “R”, potentially two rule sets may be applied to a message:

1. One rule set may apply when the message is sent by the sending TCM (TSYS), and
2. One rule set may apply when the message is received by the receiving TCM (RSYS).

For more information on the SEC Database structure refer to the TCM Overview, BR 252-573-301. Since each Path segment can contain a rule ID, and each rule ID has a corresponding rule set, it is obvious that a large number of rule sets are possible. Refer to Figure 2-4 for the representation of path combinations which have associated rule IDs, and, therefore, rule sets.

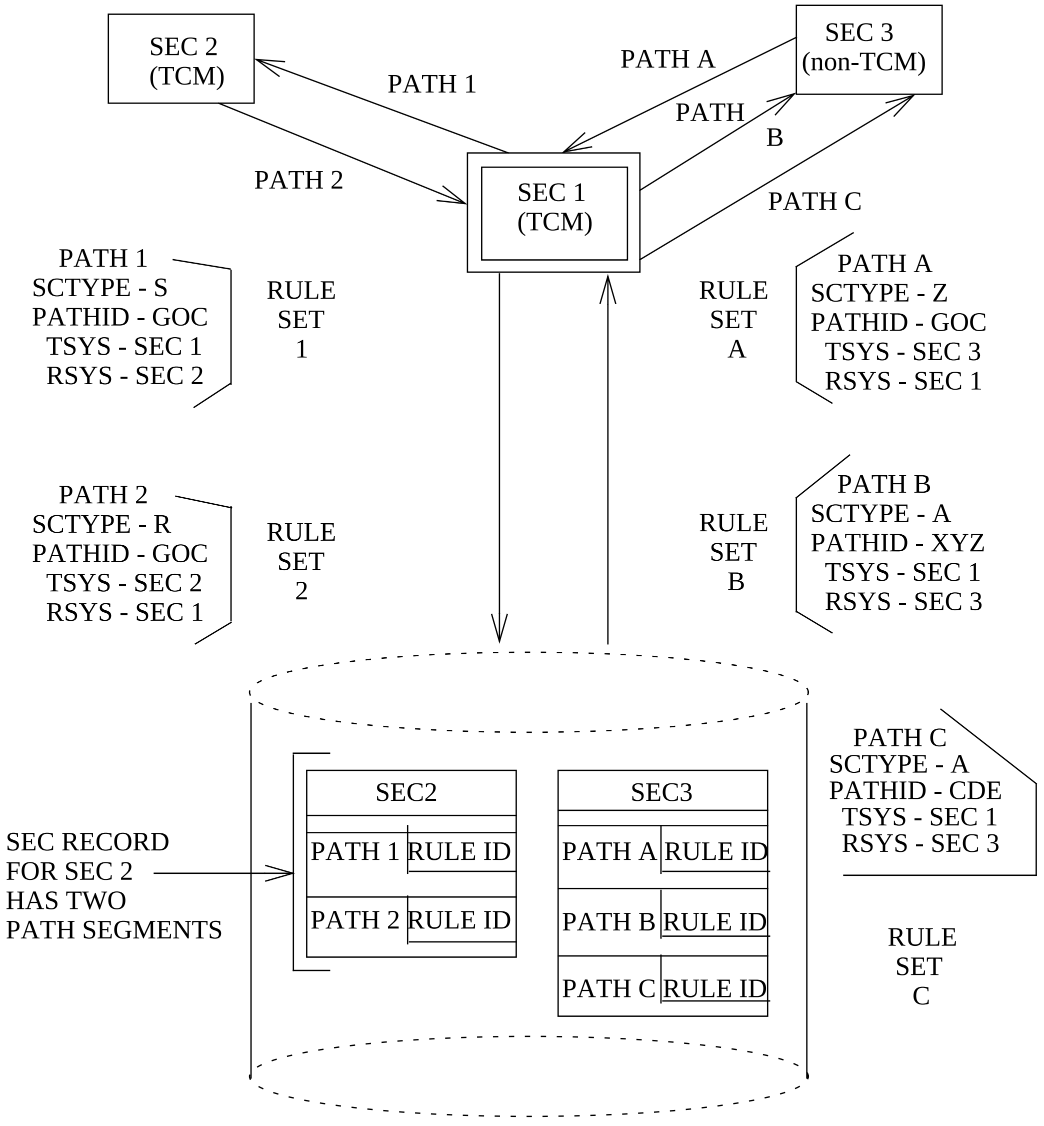

Figure 2-4. Sample Path Combinations for Rule Sets

2.1.4 TCM System to TCM System Rule Set Release Protocol

Two TCM systems may be on different release levels. In general, the TCM system at the higher release has the responsibility of doing release translation for the TCM system at the lower release. That is, a lower release was issued before the higher existed, and, therefore, cannot know the higher release.

Whenever the releases of the two TCM systems are the same, any translations will be to accommodate differences in the usage of the TCM system, i.e., regional differences.

If two TCM systems are at different levels, the lower release TCM system still has the responsibility to accommodate the regional differences.

The higher release TCM system accommodates release translation as well as regional differences it normally would accommodate.

At times, release translation rule sets will not change across releases. Because “version” is independent of release, the same version can be used for more than one CSAS system release.

2.2 Translation Rule Source Code

Translation Administration assumes that the source for the set of translation rules, e.g., the rule set, is stored in a Partitioned Data Set (PDS). There are no restrictions upon the user in determining the method chosen to develop this data set.

2.3 Compilation

The compilation process translates the translation rule source from a data set into a set of assembler control blocks readable by the executor. The compiler is a required pre-processor to the TPAM Translator.

The compiler input is a simple sequential file, such as a TSO file, and the output is the object code in a PDS known to IMS (Information Management System), e.g., PGMLIB. The user can choose any of the data sets in the IMS Bring-up deck, or can create a new data set for rules and include the new data set in the Bring-up deck.

Batch run VMMPM02, described in BR 252-573-511, is used to compile the translation rules.

Usually, the name of the PDS member is the name of the set of translation rules, e.g., the rule ID, although this can be overridden in the JCL. Before execution of any translation, this rule set name must be entered in the Network (SEC) Database (VMMPSCDD). The object code for the compiled rule set is in binary control blocks readable by the executor.

2.4 TCM Parser And Mapper (TPAM)

The TCM Parser and Mapper (TPAM) has three main parts: the Parser, the Translator, and the Mapper. The Parser and Mapper are always used, but the Translator is only used when a translation of a rule set is to be performed.

The functions of the three parts are described in the following paragraphs. For more detailed information on the Parser and Mapper parts of TPAM, please refer to the TCM Route Administration User Manual, BR 252-573-302.

2.4.1 TPAM Parser

The TPAM Parser logic controls all input message text processing. The three steps involved in the input message processing and logging are:

- Retrieval of the input message text.
- Conversion of the input message text to an internal data format called Hierarchical Map Data Area (HMDA) used for processing in TPAM.

Note that the Parser's operation is dependent on the type of data format it receives as input. If the input is in the FCIF format, the Parser can generate the HMDA format directly from the input. If the input is in a Map Data Area (MDA) format or Data Section (DSECT) format, the Parser must use that input in combination with a special Message Format Descriptor (MFD) to generate the HMDA format.

The MFD is provided by the CSAS system application, to describe the aggregates into which the MDA's data fields are to be grouped. The MFD for the Parser (MFD-IN) must be stored in a PDS known to IMS for each application. The Parser obtains the name of the MFD from the Path segment of the SEC Database.

2.4.2 TPAM Translator and Mapper

The TPAM Translator and Mapper logic control all output message text processing. The three steps involved are:

- Translation of the message text, as specified by the user.
- Conversion of the message text to an appropriate output data format (MDA, DSECT or FCIF).

2.4.2.1 TPAM Translator

The TPAM Translation Controller controls the translation function. The purpose of the translation is to assign values to data names, change data names, delete data names, and reassign previously assigned data values. The translation rules specified in Translation Administration (TA) determine the translations performed on the message text.

The translation is initiated only if the name of a set of rules, e.g., the rule ID, is found in the Path segment of the SEC Database. The name identifies the rule set that is stored in a Partitioned Data Set (PDS) known to IMS, e.g., PGMLIB. The Rules Access Manager (RAM) part of the TPAM Translator loads the rule set into core. The TPAM Translation Controller/Executor executes each rule one at a time. This process is explained in more detail in Sections 2.5 and 2.6.

The execution of a rule is based on the rule type and its associated operands or logic expressions. The existing rule types are:

- Replacement (**rp**)
- Table Look-up (**tb**)
- Deletion (**dl**)
- Manipulation of Data Strings (**str**)
- Conditional Execution (**test** and **if**,**end**)
- Preview (**spv** and **epv**)
- Comment Statements (these are in addition to the rule types)

All the translation rule types and comment statements are explained in detail in Section 4.

2.4.2.2 TPAM Mapper

The TPAM Mapper converts the message text from HMDA to an appropriate output format. The Mapper's action is controlled by the Scenario Type (SCTYPE), and by whether or not output has been received from the TPAM Parser/Translator.

If the Mapper output is to be transmitted to a remote TCM or non-TCM (SCTYPE = S or A), the conversion is to the FCIF format. The Mapper can generate the FCIF format directly from the HMDA input.

If the Mapper output is to be transmitted to the local CSAS system application (SCTYPE = R or Z), the conversion is to the MDA or DSECT format. In this case, the Mapper must use the HMDA input in combination with a special Message Format Descriptor (MFD) to generate the MDA or DSECT format. The MFD is needed to tell the Mapper exactly what fields are to be extracted from the HMDA and passed to the CSAS system application. The MFD for the Mapper (MFD-OUT) must be stored in a PDS known to IMS for each application. The Mapper obtains the name of the MFD from the Path segment of the SEC Database.

If input was received from the TPAM Translator, the Mapper has an additional function to perform. The output of the Translator consists of a "supplementary" aggregate HMDA. This data area contains all the data translations that were performed by the Translator. The Mapper must select data from the "supplementary" HMDA before it uses the remaining original data from the HMDA that was created by the TPAM Parser.

Figure 2-5 is a representation of the message flows through TPAM. Note that the dotted lines for Translation indicate that it is an optional function.

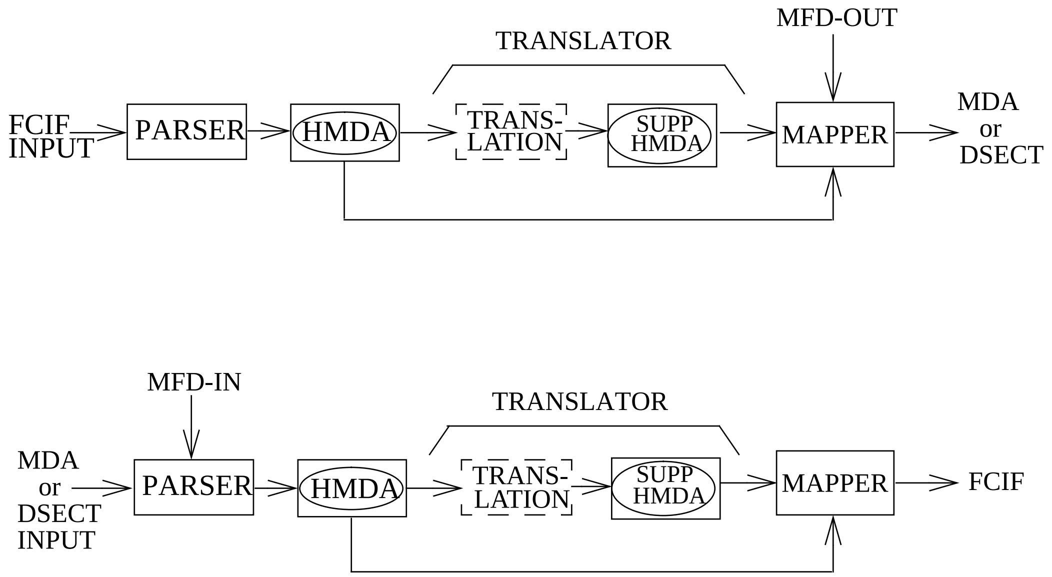

Figure 2-5. TCM/TPAM Message Flows

2.5 Loading

The load procedure is performed by the Rules Access Manager (RAM) part of the TPAM Translator as a result of a call from the TPAM Translation Controller. RAM fetches all the object code for a given rule set and loads it into core contiguously. Since all addresses are offsets, the object is relocatable. The load procedure returns a pointer to the start of the executable load module of the given rule set to the TPAM Translation Controller. A detailed representation of the load and rule execution process are presented in Figure 2-6.

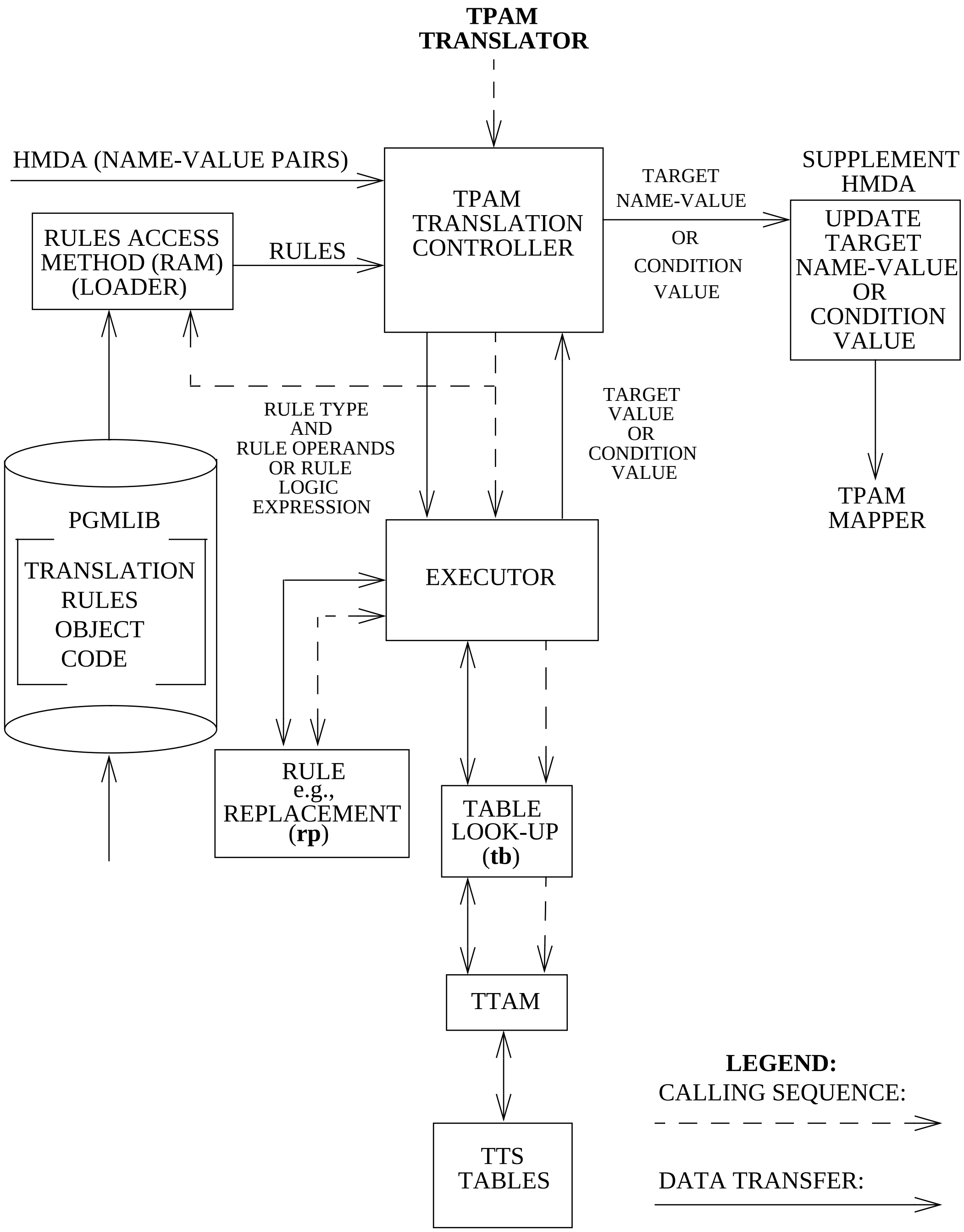

Figure 2-6. TPAM Load and Rule Execution Process

2.6 Execution

The TPAM Translator calls the TPAM Translation Controller to have the translation rules executed. After loading (explained above), the Translation Controller examines the TPAM input - the HMDA formatted data - and applies the translation rules in a sequence described in the following paragraphs. Note that the load procedure returned a pointer to the start of the executable load module for the desired rule set.

The HMDA input is organized by “aggregates” previously defined as data grouped by name-value pairs, in hierarchical order (nested), and preceded by an aggregate name.

The translation rule set is also organized by aggregates (refer to Section 3). However, an aggregate name in a rule set can only occur once, whereas, the aggregate name in the TPAM input can occur more than once. The aggregates in the rule set define the translations for all occurrences of the comparable aggregates in the TPAM input.

The TPAM Translation Controller/Executor applies the rules while traversing through the input aggregates in a top-down, left-to-right fashion until the TPAM input data is exhausted.

As each input aggregate is encountered, the rules for that aggregate are executed in sequence. The values returned by each rule is put into an “aggregate supplement” for the current input aggregate. The aggregate supplement is the modified data that results from the translation indicated by the application of the translation rules in an aggregate. There is an aggregate supplement created for each aggregate in the TPAM input that used at least one translation rule.

The rules for a particular aggregate are executed only when a corresponding aggregate is found in the TPAM input.

The modified data (aggregate supplement) is appended to the TPAM input for that aggregate. Data required for the TPAM output, e.g., the Mapper, is fetched from the modified data, and if it is not found there, the data is fetched from the input to the TPAM Translator (HMDA). Thus, the name-value pairs whose input name and value are identical to its output name and value do not need a translation rule.

For example, assume the structure of a sample translation rule set is:

```
NAME
   translation rule(s)

ADDRESS
   translation rule(s)

CITY
   translation rule(s)
```

And the structure of a sample input HMDA in aggregate form is:

NAME
name-value pair(s)

ADDRESS
name-value pair(s)

CITY
name-value pair(s)

ADDRESS
name-value pair(s)

ADDRESS
name-value pair(s)

CITY
name-value pair(s)

The TPAM Translation Controller examines the HMDA input and determines the order of execution for the aggregates. The highest order aggregate with corresponding translation rules of the same aggregate level is processed first. In this case, the highest level is NAME. The analysis by the Controller results in the following order for the example:

Input Aggregate	*Aggregate Execution Order*
NAME	Aggregate 1 name-value pair(s)
ADDRESS	Aggregate 2 name-value pair(s)
CITY	Aggregate 3 name-value pair(s)
ADDRESS	Aggregate 4 name-value pair(s)
ADDRESS	Aggregate 5 name-value pair(s)
CITY	Aggregate 6 name-value pair(s)

Figure 2-7 is a representation of this order of translation.

The circle from the lowest level of aggregate - Aggregate 3 - around the highest level of aggregate - Aggregate 1 - indicates that during the translation process, the TPAM Translation Controller/Executor will go back to higher aggregate levels looking for values for an operand (please refer to Section 4 for a description of the translation rule operands). For example, if an operand is a name that cannot be found in Aggregate 3, then the TPAM Tanslation Controller/Executor will go back to Aggregate 2 looking for it. If the name still cannot be found in Aggregate 2, then the Controller will go back to Aggregate 1. This search for operand starts from the aggregate currently being executed and then moves backward to higher level aggregates that are directly connected. The search stops when the operand is found or it reaches the highest level aggregate. **NOTE** that the search will **NOT go to any aggregates that are at the same or higher level of the aggregate being executed, if they are not directly connected.**

In the previous example, the TPAM Controller will not look for the name in Aggregates 4, 5 and 6 because they are not in the same circle as Aggregate 3. In addition, the search for the operand does not usually go forward to lower level aggregates that are directly below the aggregate being executed, unless the rule containing the operand is within A Preview block. (For more detail, please refer to Section 4 - *Description of Translation Tools*).

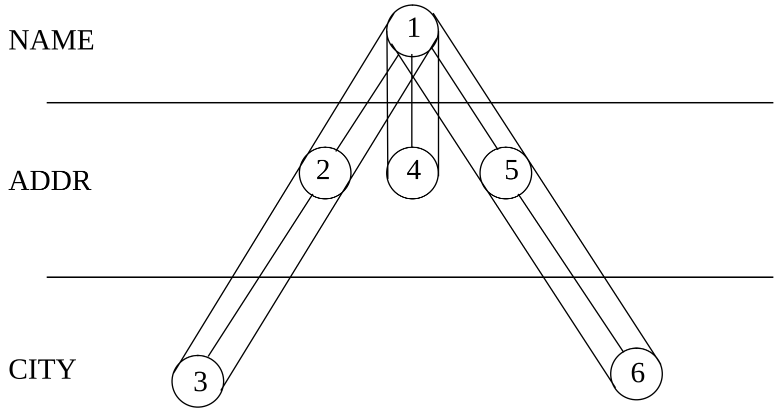

Figure 2-7. Representation of Order of Translation for Sample Input

The translation is performed as follows:

1. The translation rules for NAME are applied to Aggregate 1. The result of each application of a translation rule is a name\(mivalue pair which is stored in the aggregate supplement associated with the input aggregate being translated, i.e., Aggregate 1.
2. The translation rules for ADDRESS are applied to Aggregate 2. Results are stored in the aggregate supplement for Aggregate 2.
3. The translation rules for CITY are applied to Aggregate 3. Results are stored in the aggregate supplement for Aggregate 3.
4. The translation rules for ADDRESS are applied to Aggregate 4. Results are stored in the aggregate supplement for Aggregate 4.
5. The translation rules for ADDRESS are applied to Aggregate 5. Results are stored in the aggregate supplement for Aggregate 5.
6. The translation rules for CITY are applied to Aggregate 6. Results are stored in the aggregate supplement for Aggregate 6.

This may be summarized as follows:

NAME		
1 aggregate aggr supp		
ADDRESS		
2 aggregate aggr supp	4 aggregate aggr supp	5 aggregate aggr supp
CITY		
3 aggregate aggr supp		6 aggregate aggr supp

Rules for a particular aggregate, e.g., NAME, are executed only when a corresponding aggregate is found in the TPAM input.

2.6.1 Translation Process for Replacement

TCM uses Replacement for assigning a numeric or alphanumeric value to a field name, or for changing a field name. When the TPAM Translation Controller/Executor encounters an “**rp**” operator (Replacement Operator) in the rules (please refer to Section 4), it either associates the literal with the name to the left of the “**rp**” adding the new field and value to the input data, or associates the name with a value. The value can be from this input aggregate or from a higher aggregate, but not from a lower level aggregate.

For example, if the translation rules for the NAME and ADDRESS aggregates for Figure 2-1 were:

```
NAME      ag
                 ORD       rp     ‘LIT’;
ADDRESS   ag
                 TARGET    rp     VALUE;
                 ADDR      rp     WORK;
```

And the HMDA name-value pair input was:

```
NAME>
                        VALUE=C005;

ADDRESS<
                         NAME=C001;
>
>
```

The results of the translation would be:

Translation Rules			Resulting Aggregate	Supplement
NAME	**ag;**			
ORD	**rp**	‘LIT’;	ORD	LIT
ADDRESS	**ag**		TARGET	C005
TARGET	**rp**	VALUE;		
ADDR	**rp**	WORK;		

Note that ORD is not in the original input, but is added to the input. ORD is the result of the replacement of a literal which is defined by the replacement statement.

For the next aggregate level (ADDRESS), the TPAM Translation Controller/Executor looked up the chain (refer to Figure 2-7) to Aggregate 1 for VALUE and took the first VALUE it found, e.g., C005. If VALUE had *not* been in the HMDA input, TARGET would not be added. The translation rule ADDR **rp** WORK is ignored because WORK value is **not** in the input data stream. That is, ADDR is not added to the input because WORK is not present. Thus, no data is defined for ADDR.

2.6.2 Translation Process for Table Look-up

TCM uses the CSAS Table System (TTS) for the Table Look-up capability. When the TPAM Translation Controller/Executor encounters a **tb** operator (Table Operator) in the rules (refer to Section 4), it calls TTS via the TTS Access Manager (TTAM). The call returns a field rather than a record.

TTAM is given the table name, the table key (optional), the table record key (optional), and a field (column) identifier. TTAM calls TTS to get the descriptor and then the record. Using the descriptor, TTAM extracts the requested field. The field identifier must be an identifier used in the descriptor.

Note that currently only DSECT and PASSBACK Tables are supported.

2.6.3 Translation Process for Deletion

TCM uses Deletion for removing data fields that do not make sense for the destination application. These deletions are useful when messages are sent to a non-TCM application since this application lacks a Message Format Descriptor (MFD) to filter the data fields. The data fields can be present as input or created by translation.

When the TPAM Translation Controller/Executor encounters a **dl** operator (Deletion Operator) in the rules (refer to Section 4), it deletes the data field(s) from the processing aggregate. In other words, the data field does not have to be searched for in the input data stream - it is just marked as deleted.

An example of this translation rule is:

dl TEXO,RKN;

This would cause data fields TEXO and RKN to be deleted from the input data stream.

2.6.4 Translation Process for Manipulation of Data Strings

TCM uses this rule to perform basic string manipulation functions. The set of basic string manipulations are as follows:

- Concatenation for Multiple Data Fields
- Substring Selection from a Given Data Field
- Substring Insertion into a Given Data Field
- Substring Deletion from a Given Data Field.

When the TPAM Translation Controller/Executor encounters the **str** operator (String Manipulation Operator) in the rules (refer to Section 4), it examines the first field in the operand field to determine which of the string manipulation functions is required. Depending upon the keyword specified in the first operand field, the following translation process occurs:

1. Concatenation for Multiple Data Fields

   The first operand field contains the keyword, \fB.CON.\fR , followed by the data to be concatenated. A maximum of nine data strings can be concatenated. The concatenated data is associated with the name in the Target field.

   An example of this translation rule is:

   HOMEADD **str** **.CON.**,'12','456';

   This would result in the string 12456 being associated with the Target name, HOMEADD. That is, the resulting aggregate supplement is

   HOMEADD 12456

2. Substring Selection from a Given Data Field

   The first operand field contains the keyword, **.SEL.** , followed by the data string from which the data is to be selected, the start position of the selection, and the length of the selection. The first character of the string has position “1”.

An example of this translation rule is:

CNONM **str** **.SEL.**,TEMPVAR,'1','8';

The string is being selected from TEMPVAR. Within the string represented by TEMPVAR, the selection starts at character position “1” and continues for eight character positions. The resulting string is associated with the Target field, CNONM.

3. Substring Insertion into a Given Data Field

   The first operand field contains the keyword, **.INS.** , followed by the data string into which the insertion is to be made, the position at which the insertion is to be made, and the data string to be inserted. If the data string into which the insertion is to be made is not specified, e.g., if the field is “,,”, the translation process assumes that the string is the Target field. Also, the position at which the insertion is to be made is defined such that the position between the first and second character is position “1”.

   An example of this translation rule is:

   HOMEADD **str** **.INS.**,,'14',' ';

   Because the data string into which the insertion is to be made is specified as the “null” string, the translation assumes that the string is HOMEADD, the Target field. The position at which the insertion is to start is character 14, and the character to be inserted is a blank. The resulting string is associated with the Target field, HOMEADD.

4. Substring Deletion from a Given Data Field

   The first operand field contains the keyword, \fB.REM.\fR, followed by the position at which the removal is to be made, and the length of the string to be removed.

   An example of this translation rule is:

   HOMEADD **str** **.REM.**,'15','1';

The translation executes the removal on the Target field, HOMEADD. The removal starts at character position “15” and has a length of one character. The resulting string is associated with the Target field, HOMEADD.

2.6.5 Translation Process for Conditional Execution

There are two types of conditional execution. They are represented by the **test** operator (Conditional-Test-of-a-Rule Operator), and the combination of the **if** and the **end** operators (Conditional-Test-of-a-Block-of-Rules Operators).

The conditional test of a rule is provided to compare two data fields in the EBCDIC collating sequence, chronological order, or arithmetic order in setting a condition.

The conditional test of a block of rules is provided to conditionally execute a block of translation rules based on the logic value of the set condition(s) or their boolean expression.

Conditional translation rule blocks can be nested up to ten (10) levels within the same aggregate level.

When the TPAM Translation Controller/Executor encounters the **test** operator or the **if** and **end** operators in the rules (refer to Section 4), it examines the Logical Expression.

The translation process will evaluate the Logical Expression in the following order:

1. Left to right per statement
2. Comparisons are performed before logical operations
3. The order of evaluation of the logical operators is:

    .NOT.

    .AND.

    .OR.

    The precedence of the logical operators is:

    .OR.

    .AND.

    .NOT.

4. The precedence of the logical operators can be altered by use of parentheses. Parentheses are not required unless the order of execution is being changed.

In a conditional execution of a block of translation rules, the translation rules between the **if** rule and the **end** rule are only executed if the Logic Expression has a logic value of True in the **if** rule. The **end** rule terminates the execution of the block of rules.

2.6.6 Translation Process for Preview

TCM uses Preview to look forward down the aggregate path to a lower level aggregate and to obtain data from the lower level aggregate for use at the higher level aggregate. When the TPAM Translation Controller/Executor encounters the **SPV** operator in the rules (refer to Section 4), it temporarily suspends processing of the present aggregate and goes into a lower level aggregate. It looks at the full path listing from the current aggregate in which the SPV rule is encountered, to the source aggregate from which data is to be obtained, as well as flags indicating whether or not each of the aggregates on the path between them is required or optional. Preview rules are in the form of a block (Preview Block) and are applied to the first occurrence of the source aggregate that meets the selection criteria. When an **EPV**operator is encountered, processing will either return to the higher level aggregate or try to find another occurrence of that lower level aggregate, based on the continue flag set by the user (i.e, special variable !CONT).

Presence Flags concatenated to the front of an aggregate Name are either an asterisk ("*") to indicate that the aggregate is required on the Path to the source aggregate or a question mark ("?") to indicate that the aggregate is optional.

For example:

CKLCWL **SPV** ?CCR,*CIRSEG;

The path list can have a maximum of eight (8) aggregate names excluding the current or the source aggregates, which are not listed in the aggregate name path.

2.6.7 Translation Process for Comment Statements

The comment statements are passed over in the processing.

3. Description Of Translation Rule Aggregate

An input "aggregate" to TPAM is defined as an aggregate name followed by data (grouped by name-value pairs). Input aggregates are usually nested in hierarchical order. A translation rule likewise has aggregate rule set(s) which have an *aggregate delimiter* followed by one or more translation *rules*.

The first statement of a translation rule set *must* be an aggregate delimiter.

3.1 SYNTAX Of Aggregate Delimiter

The aggregate delimiter has the following syntax:

Aggregate Name **ag**;

The aggregate delimiter *must* precede the translation rules for a particular aggregate. During TPAM translation, each input "aggregate" is matched to the aggregate delimiter (if any). Also, within the translation rule set, one set of aggregate rules must be separated (delimited) from other sets, hence the name of aggregate delimiter. The delimiter is the input aggregate name and the operator **ag**. An input message's aggregate name can be used only once in an aggregate delimiter, which must be on a line by itself. The aggregate name and the operator **ag** are ended with a semicolon (;).

Aggregate Name:

Defines the beginning of the aggregate in the rule set. The contents of the aggregate rule set may be changed independently of the application. Refer to the Message Administration User Manual, BR 252-573-303 for information on valid aggregate names.

Operator is: **ag**

3.2 Existence Of Aggregate

An aggregate rule set can either exist in the translation rule set, or not. If the aggregate exists, the aggregate name must be the first statement, followed by at least one or more translation rule(s). The aggregate name and its associated translation rule(s) and comments is the rule set for that aggregate.

Other rule sets for other aggregates can follow, but each aggregate name can only appear once in any translation rule set. A rule set must specify at least one aggregate rule set. Also, an aggregate rule set *must* contain at least one rule.

4. Description Of Translation Tools

4.1 Overview

The following are the translation tools that are grouped by aggregates.

1. Replacement
2. Table Look-up
3. Deletion
4. Manipulation of Data Strings
5. Conditional Execution
6. Preview
7. Comment Statements.

The syntax of the tools can be generalized as follows:

Target field	Operator or	Operand field;
Target Condition field	Operator or	Logic expression field;
Rule-Block Label field	Operator or	Logic expression field;
* Comment Text;		

A specific use of a tool is a rule, and the group of rules for a particular application is a translation rule set. Each rule set is defined by a user specified translation rule name, e.g., the rule ID. This identification must exist in the TCM SEC Database (ZMMPSCDD) before the rule set is used. Refer to Section 2.2 for a description of rule set identification.

4.2 Coding Rules

Each translation rule and comment should exist as a separate statement and be ended with a statement delimiter, a semicolon (;). There is no maximum statement length, where statement is defined as the data from the Aggregate Name, Target field, Target Condition field, Rule-Block Label field, or asterisk (*) to the delimiter (;). Although there is no concept of lines, it is necessary that the rules be coded statement by statement with each new statement starting in Column 1.

The translation rules (the statements) are coded on 80 column record, but only the first 72 columns can be used. The Target field, Target Condition field, or Rule-Block Label field; Operator; and first Operand field must be separated by at least one blank. The operands in the Operand field must be delimited by commas and have no embedded blanks. However, there can be embedded blanks between the single quotes of a literal.

If a statement uses more than 72 columns, the statement is continued on the next consecutive card image. No continuation character is used. The continuation of the statement must begin in Column 1. The continuation of a literal must begin in Column 16. The statement is ended with the semicolon (;). All trailing blanks on a card are ignored.

For comments having more than 69 characters (the allowable length for a single comment), the comment is continued on the following card image as a new comment. Each comment must begin with an asterisk (*) in Column 1 and end with a semicolon (;).

Two single quotes together ('') are treated as a legitimate literal with zero length. However, a single quote between two single quotes of the literal is not permitted, e.g., ''' is invalid. Two commas together (,,) are treated as a legitimate *empty* operand, i.e., is not present. The data can be entered in upper and/or lower case.

4.3 Replacement

4.3.1 Syntax of Rule

Replacement is the simplest tool and has the following syntax:

Target field **rp** Operand field;

The replacement tool assigns the value of the Operand field to the Target field.

Target field valid entry is:

1 to 8 alphanumerics specifying a field name

Operator is:

rp

Operand field valid entries are:

1. 1 to 8 alphanumerics specifying a field name. The value of the named field is placed in the Target field, or
2. 1 to n alphanumerics enclosed by single quotes (literal), e.g., 'SYMBOL'. The literal is placed in the Target field.

4.3.2 Examples

Several examples of the use of the replacement tool are:

```
T1       rp  'VALUE';
TARGET   rp  SOURCE1;
ORD      rp  C005;
T2       rp  '';
```

Each use of the replacement tool is a replacement rule.

4.4 Preview

Preview allows the user to look forward down the aggregate path to a lower level aggregate and to obtain data from the lower level aggregate for use at the higher level aggregate. It temporarily suspends processing in the aggregate the user is in and puts them into a lower lever aggregate. Preview rules are in the form of a block (Preview Block) and are applied to the first occurrence, and optionally subsequent occurrences, of the source aggregate from which data is to be obtained that meets the selection criteria.

The Preview Block is a group of statements that begin with the name of the aggregate that is being Previewed (source aggregate) in the first position, a space, the operator "SPV" (Start Preview), a space and a listing of the full path from the current aggregate to the source aggregate (unless there is no path as in the case of the immediately lower aggregate), separated by commas, together with flags indicating whether or not each of the aggregates on the path is required or optional.

These flags (Presence Flags), concatenated to the front of an aggregate Name, are either an asterisk ("*") to indicate that the aggregate is required on the path to the source aggregate or a question mark ("?") to indicate that the aggregate is optional. The Presence Flag/aggregate name concatenation is followed by a comma (,) if it is to be followed by another Presence Flag/aggregate name concatenation. The final Presence Flag/aggregate name concatenation is followed by a semicolon (;) to end the rule. The path list can have a maximum of eight (8) aggregate names excluding the current or the source aggregates. The current and the source aggregates are not named in the path list, therefore, no path list exists if the aggregate to be Previewed is the immediately lower aggregate.

The Preview Block ends with the name of the aggregate that was Previewed in the first position, a space and the operator "EPV" (End PreView). The rule ends with a semicolon (;).

The presence of a Continue Flag is implied and the default value for the flag is "N". A value of "N" will process only one occurrence of the source aggregate. The user may override this value by establishing an additional rule within the rule set which replaces the value of the continue flag with a "Y" thus producing multiple iterations through more than one source aggregate. In other words, setting the Continue Flag to 'Y' within a Preview Block allows the block to be re-executed for many lower level (i.e., source) aggregates. The rule creator should reset the Continue Flag to 'N' within the Preview Block as soon as the desired data is obtained to avoid needlessly processing additional lower level aggregates (i.e., for performance reasons).

4.4.1 Syntax of Rule

The Preview Block has the following syntax:

Source Aggregate **SPV** Optional Path Listing;
Source Aggregate **EPV**;

Example of a Preview Rule:

```
ORDR AG;
!TESTLGS RP 'N';
CIRSEG SPV ?CCR;
!CONT RP 'Y';
!CKTID45 STR .SEL.,CKTID,'4','3';
$BLK1 IF !CKTID45,.EQ.,'PLN';
!TESTLGS RP 'Y';
!CONT RP 'N';
$BLK1 END;
CIRSEG EPV;
$BLK2 IF !TESTLGS,.EQ.,'Y';
TRO RP 'R01';
$BLK2 END;
```

The example above will loop through all CIRSEG aggregates under the ORDR aggregate (a CCR aggregate may or may not exist between them) looking for the characters 'PLN' in positions 4 through 6 of the CKTID field of the lower level aggregate. Lower level aggregates will be processed until the characters are found, at which time the Continue Flag is set to 'N' to stop looking through additional aggregates. Once the SPV Block is exited, the TESTLGS flag is tested. It was set to 'Y' within the SPV Block if the characters 'PLN' were found. Thus, the TRO field in the ORDR aggregate is set to R01 if 'PLN' was found in any lower level aggregate.

NOTE — Only variables beginning with an exclamation point (!) can be used as targets for rules within a Preview Block. These are called Universal Variables (see Section 5.03). These variables are 'global' in that once their values are set, they are accessible across aggregates at different hierarchical levels, They are used to pass data (e.g.,

!TESTLGS) from a lower level aggregate to a higher aggregate during Preview processing.

4.5 Table Look - Up

4.5.1 Syntax of Rule

Table Look-Up is a simple table look-up and has the following syntax:

Target field **tb** Operand field;

The Table Look-up tool assigns the table value determined by the Operand field to the Target field.

Target field valid entry is:

1 to 8 alphanumerics specifying a field name

Operator is:

tb

Operand field is actually four fields each delimited by commas. The valid entries for the four fields, correctly ordered, are:

TNAME: The Table Name (required field)

1. 1 to 8 alphanumerics specifying the name of an input field which contains the Table Name as a data value, or
2. A valid TTS Table Name enclosed by single quotes (literal), e.g. 'TCM SCHEDULE'. The field is a maximum of 18 characters including quotes since 16 characters is the maximum length of a TTS Table Name.

TKEY: The Table Key (optional field)

1. 1 to 8 alphanumerics specifying the name of an input field which contains the Table Key as a data value, or
2. A valid TTS Table Key enclosed by single quotes (literal). The field is a maximum of 17 characters including quotes since 15 characters is the maximum length of a TTS Table Key, or
3. The value zero (0) or null.

SKEY: The TTS Table Record Key (optional field)

1. 1 to 8 alphanumerics specifying the name of an input field which contains the TTS Table Record key as a data value, or

2. 1 to n alphanumerics enclosed by single quotes (literal). The field length is defined in the TTS Table Descriptor, or
3. Null.

FID: A TTS Table field within the TTS record (required field).

1. 1 to 8 alphanumerics specifying the name of an input field which contains a Table Record field name as a data value, or
2. A valid TTS Table field enclosed by single quotes (literal), e.g. 'TRANCD1'. The field is a maximum of 10 characters including quotes.

   The FID is the field whose value is to be assigned to the Target field.

4.5.2 Examples

Several examples of the use of the Table Look-up tool are:

```
ODOC  tb  'DOC TABLE',,C017,'DOC';
CDR   tb  'DR TABLE',I004,CLO,'DR';
CSR   tb  'DR TABLE',,,'SR';
```

Each use of the Table Look-up tool is a table look-up rule.

TCM uses the CSAS Table System (TTS) for the Table Look-up capability. When the TPAM Translation module encounters a **tb** operator (Table Operator) in the rules, it calls TTS via the TTS Access Manager (TTAM). The call returns a field rather than a record, which is the usual action.

TTAM is given the table name, the table key (optional), the table record key (optional), and a field identifier. TTAM calls TTS to get the descriptor and then the record. Using the descriptor, TTAM extracts the requested field. The field identifier must be an identifier used in the descriptor.

4.5.3 Procedure for Building TTS Tables

The procedures for building TTS Tables are described in BR 252-551-700.

4.5.4 Value of Targets For The Table Look-Up Rule

The value of a target will depend on the existence of certain fields in the table command structure.

- If the TNAME or FID is not valid, then the target will not be created, and a TCM Translation error will be generated.
- If the TNAME and FID are valid:
  - — If the TKEY and SKEY are valid or not required, then the Target will be created and assigned the value of the FID. (The TKEY and SKEY fields are only required, if they are used by the specified TTS table).
  - — If either the TKEY or SKEY is invalid or not specified when it is required, then the Target will be created with a value of NULL.

4.6 Deletion

4.6.1 Syntax of Rule

Deletion has the following syntax:

Target field **dl** Operand field;

The deletion tool removes the value(s) specified in the Operand field.

Target field valid entry is:

The *only* valid entry is a blank

Operator is:

dl

Operand field valid entries are:

1. 1 to 8 alphanumerics specifying a field name. The value is deleted, or
2. 1 to n alphanumerics enclosed by single quotes (literal), e.g.. 'SYMBOL'. The literal is deleted.

4.6.2 Example

An example of the use of the deletion tool is:

```
Col.1       Col.10      Col.16
|           |           |
|           |dl         |TEXO,RKN;
```

This would cause data fields TEXO and RKN to be deleted from the output data stream.

4.7 Manipulation Of Data Strings

4.7.1 Syntax of Rule

Manipulation of data strings has the following syntax:

Target field **str** Operand field;

The manipulation of data strings tool performs various basic string manipulations depending upon the function specified in the first operand field.

Target field valid entry is:

1 to 8 alphanumerics specifying a field name

Operator is:

str

Operand field valid entries are dependent upon the keyword in the *first* operand field.

CASE 1: If the first field contains **.CON.**, there are up to nine fields delimited by commas. The valid entries, correctly ordered, are:

.CON.: The Data String Concatenation keyword (required field)

data string 1: A string to be concatenated (required field)
data string 2: A string to be concatenated (required field)
data string 3: A string to be concatenated (optional field)
. .
. .
. .
data string 9: A string to be concatenated (optional field)

1. 1 to 8 alphanumerics specifying a field name, or
2. 1 to n alphanumerics enclosed by single quotes (literal), e.g., '123'. The literal is concatenated.

CASE 2: If the first field contains, **.SEL.**, there are four fields delimited by commas. The valid entries, correctly ordered, are:

.SEL. : The Data Substring Selection keyword (required field)

dsrsel: The data string from which a substring is to be selected (required field)

1. 1 to 8 alphanumerics specifying the name of an input field, or,
2. 1 to n alphanumerics enclosed by single quotes (literal).

srtpos: The starting position of the selection (required field)

1. 1 to 8 alphanumerics specifying the name of an input field. The first character of the string is character position "1", or
2. 1 to n numerics representing a positive integer enclosed by single quotes (literal).

lngsel: The length of the selection (required field)

1. 1 to 8 alphanumerics specifying the name of an input field, or
2. 1 to n numerics enclosed by single quotes (literal).

CASE 3: If the first field contains, **.INS.**, there are four fields delimited by commas. The valid entries, correctly ordered, are:

.INS. : The Data Substring Insertion keyword (required field)

dsrins: The data string into which a substring is to be inserted. If two commas („) are specified, the program assumes that the data string is the Target field (optional field).

1. 1 to 8 alphanumerics specifying the name of an input field, or
2. 1 to n alphanumerics enclosed by single quotes (literal).

inspos: The position at which the insertion is to be made. This position is defined such that the position between the first and second character is position "1" (required field).

1. 1 to 8 alphanumerics specifying the name of an input field, or
2. 1 to n numerics representing a positive integer enclosed by single quotes (literal).

strins: The data string to be inserted (required field).

1. 1 to 8 alphanumerics specifying the name of an input field, or
2. 1 to n alphanumerics enclosed by single quotes (literal).

CASE 4: If the first field contains, **.REM.**, there are three fields delimited by commas. The valid entries, correctly ordered, are:

.REM. : The Data Substring Deletion keyword (required field) *rempos:* The position at which the removal is to be made (required field).

A positive integer enclosed by single quotes (literal).

lngstr: The length of the string to be removed (required field).

1. 1 to 8 alphanumerics specifying the name of an input field, or
2. 1 to n numerics representing a positive integer enclosed by single quotes (literal).

4.7.2 Examples

Examples of each CASE are presented in Section 2.6.4.

4.8 Conditional Executions

4.8.1 Syntax of Rule

Conditional executions of transaction rules may be expressed by use of two different types of syntax.

(1) Target Condition field **test** Logic Expression;

(2) Rule-Block Label field **if** Logic Expression;

.

Rule-Block Label field **end**;

The first conditional execution tool (1) is used to compare two data fields for EBCDIC order, or for chronological order, or for arithmetic order which sets the *target condition* field.

The second conditional execution tool (2) is used to conditionally execute a block of translation rules based on the logic value of the set condition(s) or their boolean expression. Conditional translation rule blocks can be nested within the same aggregate level.

Both conditional execution tools share a number of *test keywords* as follows:

Table 4-1. Test Keyword Table

TEST KEYWORD	*USED FOR*
	ALPHANUMERIC COMPARISONS
.LE.	Alphanumeric "Less Than or Equal" Comparison
.EQ.	Alphanumeric "Equal" comparison
.NE.	Alphanumeric "Not Equal" comparison
.GT.	Alphanumeric "Greater Than" comparison
.GE.	Alphanumeric "Greater Than or Equal" comparison
.LT.	Alphanumeric "Less Than" comparison
	CHRONOLOGICAL COMPARISON IN CALENDAR DATES
%EQ.	Chronological "Equal" comparison in Calendar Dates
%NE.	Chronological "Not Equal" comparison in Calendar Dates
%GT.	Chronological "Greater Than" comparison in Calendar Dates
%GE.	Chronological "Greater Than or Equal" comparison in Calendar Dates
%LT.	Chronological "Less Than" comparison in Calendar Dates
%LE.	Chronological "Less Than or Equal" comparison in Calendar Dates
	CHRONOLOGICAL COMPARISON IN JULIAN DATES
&EQ.	Chronological "Equal" comparison in Julian Dates
&NE	Chronological "Not Equal" comparison in Julian Dates
&GT.	Chronological "Greater Than" comparison in Julian Dates
&GE.	Chronological "Greater Than or Equal" comparison in Julian Dates
&LT.	Chronological "Less Than" comparison in Julian Dates
&LE.	Chronological "Less Than or Equal" comparison in Julian Dates
	INTEGER NUMBER COMPARISONS
#EQ.	Numeric "Equal" comparison
#NE.	Numeric "Not Equal" comparison
#GT.	Numeric "Greater Than" comparison
#GE.	Numeric "Greater Than or Equal" comparison
#LT.	Numeric "Less Than" comparison
#LE.	Numeric "Less Than or Equal" comparison
	LOGICAL OPERATIONS
.AND.	Logical "AND"
.OR.	Logical "OR"
.NOT.	Logical "NOT"

NOTE — Note that the logical functions for the manipulation of binary bits are defined as follows:

Logical "AND"

1100
0110 AND

____

0100 Result

Logical "OR"

1100
0110 OR

____

1110 Result

Logical "NOT"

01 NOT

__

10 Result

4.8.2 Conditional Test of a Rule

The first condition execution tool (1) has the syntax:

Target Condition field **test** Logic Expression;

The **test** execution tool assigns the logical result of the Logical Expression to the Target Condition field. The logical expression is evaluated for being true or false. When it is true, then the target condition field is assigned "T", otherwise it is assigned a "F". The target condition field value can be used in later testing. Target fields are never mapped out.

Target Condition field valid entry is:

Eight alphanumerics specifying the Target Condition Name. The name *must* start with "@".

Operator
is:

test

Logic Expression has a complex structure as follows:

Logic Expression can be expanded to indicate a comparison operation or Test Expression.

Test Expression has the syntax:

Data Variable 1,Test Keyword,Data Variable 2

where valid entries are:

Data Variables 1 and 2:

1. 1 to 8 alphanumerics specifying a field name where the first character should be an alpha. This is also referred to as an Extended Binary Coded Decimal Interchange Code (EBCDIC) string.
2. 1 to n alphanumerics enclosed by single quotes (literal or EBCDIC string). The maximum length is 1500 characters (bytes).
3. Dates in Calendar Date form (MMDDYY, YYMMDD or CCYYMMDD). There are two system defined names, *%DATE* and the *%CCDATE*, that returns the Calendar Date in the form MMDDYY and CCYYMMDD, respectively.
4. Dates in Julian Date form (YYDDD or CCYYDDD). There are two system defined names, *&DATE*, and the *&CCDATE*, that returns the Julian Date. The returned date has the form YYDDD and CCYYDDD, respectively.
5. An eight alphanumeric character name used for Integers. The first character *must* be "#". Only unsigned integers are allowed.

Test Keyword:

See preceding Table 4-1.

The comparison result depends upon the type of data variables. They can reflect the alphabetical order for EBCDIC strings, the chronological order for the dates, or the numeric order for the integers.

4.8.3 Examples of Conditional Test of a Rule

Examples of this form of the tool are as follows:

1. In these Alphanumeric comparisons, assume that ISC contains the value 0253, BLANKS contains the value blanks, and CRO contains the value 05.

```
@ISC      test     ISC,.NE.,BLANKS;
@CRO      test     CRO,.EQ.,'1';
```

In the first comparison, the logic result is "T" for True, and this value is associated with the Target Condition Name @ISC.

In the second comparison, the logic result is "F" for False, and this value is associated with the Target Condition Name @CRO.

2. In these Chronological comparisons, assume that *%DATE* has a value 020395, *&DATE* has a value 95034..

   ```
   @CDATE       test      020195,.%EQ.,%DATE;
   @CRO         test      95215,.&EQ.,&DATE;
   ```

   In both cases, the translation puts an "F" for False, and associated with the Target Condition Names @CDATE and @JDATE.

   In addition, assume %CCDATE has a value of 19961210 and CCDATE has a value of 1996345...

   ```
   @LCDATE       test      19961210, %EQ., %CCDATE;
   @LJDATE       test      2000015, &EQ., &CCDATE;
   ```

   In the first case, the translation puts a "T" for True and associates it with the Target Condition Name @LCDATE. In the second case, the translation puts an "F" for False and associates it with the Target Condition Name @LJDATE.

3. In this Integer Number comparison, assume that #TEMP contains the value 7.

   @TEMP **test** #TEMP,.#LT.,'9';

   The logic result is "T" for True, and this value is associated with the Target Condition Name @TEMP.

As previously defined,

Logic Expression is Test Expression

and

Test Expression is Data-Var1,Test Keyword,Data-Var2

Logic Expression can also be defined via the following production rules:

Logic Expression is TERM

or

TERM,.OR.,Logic Expression

TERM is FACTOR

or

FACTOR,.AND.,TERM

FACTOR is Target Condition Name

or

.NOT.,FACTOR

or

Test Expression

This implies that the structure of Logic Expression can become as complex as the user wants. It also implies the precedence with which a complex logic structure is evaluated.

The translation process evaluates a conditional translation rule in the following order:

1. Left to right per statement
2. Comparisons are performed before logical operations
3. The precedence of the logical operators is:

   .OR.
   .AND.
   .NOT.

The order of evaluation is:

   .NOT.
   .AND.
   .OR.

If the user wants to force the evaluation of an .OR. before an .AND., the precedence order can be altered by use of parentheses around the comparisons.

For example, in the complex structure:

@VALUE **test** (RCKTID,.NE.,''),.AND.,((RFTM,.EQ.,'3'),.OR.,(RFMT,.EQ.,'S'));

The comparisons within the parentheses will be evaluated from left to right. Then, because there are double parentheses around the expressions containing the logic operator .OR., that boolean expression will be evaluated before the logical .AND. is performed.

The parentheses are not required unless the order of execution is being changed.

4.8.4 Conditional Test of a Block of Rules

The second condition execution tool (2) has the syntax:

Rule-Block Label field **if** Logic Expression;
.
Rule-Block Label field **end**;

The Conditional-Test-of-a-Block-of-Rules tool provides a mechanism for conditional execution of a block of translation rules.

Rule-Block Label field valid entry is:

> An eight alphanumeric character name used to label the beginning and end of the conditional rule block. The first character *must* be *$*.

Operators are:

if	Begins the conditional block
end	Ends the conditional block

Logic Expression

> Refer to the description under Conditional Test of a Rule.

For this conditional execution, the translation rules between the **if** rule and the **end** rule (the block of rules) are only executed if Logic Expression has a logic value of True in the **if** rule. The **end** rule terminates the execution of the block of rules.

The **if** and **end** rules must always be used as a pair, and they must both have the same Rule-Block Label. Further, the blocks of rules can only be defined within the aggregate levels. The **if** blocks can be nested up to 10 levels.

The execution precedence described for the Conditional-Test-of-a-Rule tool is also valid for the Conditional-Test-of-a-Block-of-Rules tool.

4.8.5 Example of Conditional Test of a Block of Rules

Assume that ISC contains the value blanks, BLANKS contains the value blanks, and CRO contains the value ABC.

```
$BLK1      if      (ISC,.EQ.,BLANKS),.AND.,(CRO,.NE.,BLANKS);
ORELORD    rp      CRO;
$BLK1      end;
```

The **if** rule is evaluated from left to right. ISC and BLANKS both have a value of blanks so the first Test Expression is True (value 1).

CRO has a value ABC and BLANKS has a value blanks so the test for "Not Equal" is True (value 1) and the second Test Expression is True (value 1).

The logical .AND. is performed:

```
 1
 1 AND
___
 1 Result
```

The Logic Expression is True (value 1) in the **if** rule so the block of translation rules is executed. ORELORD is assigned the value of CRO which is ABC. The conditional block execution is terminated with the **end** rule.

4.9 Comment Statements

4.9.1 Syntax of Tool

The comment statement has the following syntax:

```
* comment text;
```

The comment statement tool is indicated by an asterisk (*) in Column 1.

comment text valid entry is:

1 to 69 alphanumerics excluding semicolons

If a comment text exceeds 69 alphanumerics, it is continued on the next consecutive card as a separate comment statement. Note that the asterisk (*) must be present in Column 1 of each comment statement whether the comment is complete on one card or continued on the next card. *Every* comment statement *must* end with a semicolon (;).

4.9.2 Examples

Several examples of the use of the comment statement tool are:

```
* THIS IS THE SOACRULE SET              ;
* THE ONLY REQUIRED AGGREGATE IS        ;
* THE NAME AGGREGATE                    ;
```

4.10 Sample Rule Applications For An Aggregate

A translation rule at a particular level, e.g., NAME, applies to all translations in the input aggregate at that level. For example, if the translation rules in the rule set are

```
NAME    ag;
MSR     rp     'SOR';
ORD     rp     C005;
```

and the TPAM input is

```
*CSAS
  NAMEE<
     C005=D5;
      .
      .
      .
     CITYY<...>>%
```

Translation Administration examines the TPAM input, and applies the translation rules to translate the value of the Target field, ORD, from a value of C005 to a value of D5. That is, the results of the translations are

ORD is assigned the value D5 in the TPAM output, *and*

MSR is replaced by (assigned the value) SOR.

4.11 Summary Of Syntax Rules

The syntax rules for the aggregate delimiter, the translation rules, and the comments can be summarized as follows:

AGGREGATE DELIMITER

Aggregate Name	**AG**;	

TRANSLATION RULES

Target Name	**RP**	Operand;
Target Name	**TB**	Table Name,[Table Key],[Table Record Key],FID;
Target Name	**DL**	FID1[,FID2[,...[,FID10]];
Target Name	**STR**	**.CON.,**D-Str1,D-Str2,[D-Str3,...,D-Str9];
Target Name	**STR**	**SEL.,**dsrsel,srtpos,lngsel;
Target Name	**STR**	**.INS.,**[dsrins],inspos,strins;
Target Name	**STR**	**.REM.,**rempos,lngstr;
Target Cond Name	**TEST**	Logic Expression;
Rule-Blk Lbl Name	**IF**	Logic Expression;
Rule-Blk Lbl Name	**END**;	
Aggregate Name	**SPV**	Path Agg1[,Path Agg2[,...[Path Agg3]]];
Aggregate Name	**EPV**;	

COMMENTS

* Comment Text;

The brackets ([ ]) indicate optional fields. The values permitted in the fields can be summarized as follows:

1. All Target Names, except the Target Name with the Deletion rule (**dl**) and Universal Variable Names, must be data names of 1 to 8 characters. The **dl** has a blank for the Target Name. Universal Variable Names are an exclamation point (!) followed by 1 to 7 characters.
2. The Target Condition Name is 8 alphanumerics and starts with "@".
3. The Rule-Block Label Name is 8 alphanumerics and starts with "$".
4. Current operators are **rp, tb, dl, str, test, SPV, EPV** and **if, end**.
5. Operands must be:
   a. A data name (1 to 8 characters) in the input message, or

   b. A Temporary Target Name determined before reaching the current rule from the aggregate for which the rule is executing or from a hierarchically superior aggregate, or

   c. A literal (text and/or blanks enclosed in single quotes), or

   d. A Universal Variable Name (an exclamation point followed by 1 to 7 characters) determined before reaching the current rule.

6. Logic Expression must be a valid construction as presented in Para.4.07.

7. Comment Text can be any alphanumeric except semicolon (;).

8. Aggregate names for the AG, SPV and EPV rules must be valid aggregate names in the interface message for which the rule is being processed.

5. Translation Rule Processing

The administrator writing a translation rule set should follow certain guidelines so that he/she can obtain the desired results when translating the TPAM input. These guidelines are presented in the following paragraphs.

5.1 Input And Translation Rule Order

A. The translation rules should be ordered according to the hierarchy of the data with the highest aggregate, e.g., NAME, first. Within each aggregate, the rules are executed in the order in which they appear. The rules are applied hierarchically until the TPAM input data is exhausted. Thus, for example, the translation rule aggregate for CKLCWL is applied for each CKLCWL occurrence in the input.

Because of this ordered execution, the user writing the translation rule set should be aware that the order in which the data-value pairs are presented in the input affects the output.

B. At the aggregate level, e.g., NAME, the translation process is driven by the TPAM input. When an input aggregate is processed, the TPAM Translation/Executor processes the rules of the corresponding aggregate sequentially. Thus, the user *cannot* use the rules to create an aggregate occurrence.

C. If every input name-value pair has the same input name and value as its output name and value, the data does not need a translation rule.

D. An operand can be a symbol, a literal, or one of the special keywords. A symbol references a target field name in the TPAM input, or a Target Name whose value has been assigned before reaching the symbol.

Therefore, an operand may reference a data field in the Target's aggregate or in an aggregate that includes the Target's aggregate, but it may never reference a field in an aggregate included in the Target's aggregate. Thus, a data field can come from an aggregate higher in the hierarchy path, but *not* from one lower in the hierarchy path.

E. During execution, operand values are taken from the data in the current aggregate or in its aggregate supplement, or from the data in an aggregate or supplement that preceded the current aggregate in the *current* hierarchy path.

5.2 Temporary Target

A *temporary target* is a target that is referenced in an operand of a rule, but is not mapped into the TPAM output. Such targets can be used as temporary variables. They have all the attributes of a target destined for output. The MFD controls if the target is/is not included in the output.

NOTE — If the receiving system is a non-TCM system, no mechanism may be available to filter out the fields. Since the MFD specifies the fields sent to TCM by the local application, all fields will be included in the FCIF message to the external system, unless the rule is written to delete one or more specific fields.

5.3 UNIVERSAL VARIABLE

A *universal variable* is a "global" variable that can be created from an aggregate being processed and can be referenced from any aggregate in process, across hierarchical boundaries. It can be used in either the target field or operand field of any type of rule. Universal variables begin with the character "!" followed by 1 to 7 alphanumeric characters (except !CONT which is a reserved word used for the Continue Flag). Universal variables may be used with any aggregate, including aggregates on different Paths. TCM does not associate them with any specific aggregate nor does it store them in any specific aggregate's "aggregate supplement" during the Translation process. These variables are reset at the start of the translation of each individual message.

5.4 Missing Operands

If the translation rule refers to an input source field that is not present in the input, the following occurs:

1. For *Replacement*, the replacement does not occur, i.e., the Target Name and value are not created.
2. For *Table Look-up*, the table look-up does not occur.
3. For *Deletion*, the deletion of the missing field(s) does not occur. Fields which are present are deleted.
4. For *String Manipulation*, the string manipulation does not occur.
5. For *IF*, the result is false. Also see section 7.07.

5.5 Redundant Rules

Missing operands often result because possibly two or more fields in the input are *mutually exclusive*. That is, one field or the other will occur in the input, but not both. Or, missing operands result because a set of fields may occur in the input that have precedence. In both cases, a rule for a Target field would be specified for each input. If the input fields have precedence, the field with the highest precedence should be last.

For example,

Mutually Exclusive			*Precedence*		
T1	**rp**	'VALUE'	T1	**rp**	'VALUE'
T1	**rp**	S1	T1	**rp**	S1
T1	**rp**	S2	T1	**rp**	S2
T1	**rp**	S3	T1	**rp**	S3

In the mutually exclusive case, only one of S1, S2, or S3 will occur. The rule will not be executed for the operands that do not occur. The order of the rules is inconsequential.

In the precedence case, S3 is of higher precedence than S2, and S2 is of higher precedence than S1. If S3 does *not* occur, the last rule will *not* be executed and T1 will be assigned (from a prior rule) S2. If none of the operands occur, T1 is assigned the value, 'VALUE'. The order of the rules is *significant*.

The rule sequences in the two cases are identical. The result of the translation is determined by the input to TPAM.

6. Sample Translation

The following pages present, as an example, the translation rules for the CSAS/SOAC Interface.

Pages 6-2 and 6-3 present the translation rule set, SOACRULE, on the "SOURCE LISTING REPORT".

Pages 6-4 and 6-5 present the generated intermediate assembler code containing an equivalent set of data constants for each translation rule. This assembler code is passed to the system assembler.

Page 6-6 presents the result of the assembly of the intermediate version of the equivalent translation rule data constants into a temporary object module.

Page 6-7 presents the result of invoking the linkage editor to generate an executable load module containing the translation rules.

```
NOTICE: NOT FOR DISCLOSURE OUTSIDE THIS COMPANY OR BELL COMMUNICATIONS RESEARCH WITHOUT WRITTEN PERMISSION.
 08/18/92 15:20:50                 BELLCORE CSAS SYSTEM /  T C M                PAGE       1
 RUN BOOK(S): 252-573-511                TPAM TRAN RULES GENERATOR                   RUN ID   : VMMPM02
 USER DOC(S):                      CARRULE  SOURCE LISTING REPORT                REPORT ID: VMMPM02A
 BELL COMMUNICATIONS RESEARCH, INC.                                           RELEASE  : 16.0.2
 ------------------------------------------------------------------------------------------------------------
 NUM    RUNTIME NUM    STATEMENT
 0001             AGG1 AG;
 0002    0001     FLD101 RP Z;
 0003    0002     AGG10 SPV *AGG2,?AGG3,*AGG4,*AGG5,?AGG6,?AGG7,*AGG8,*AGG9;
 0004    0003     $BLK1  IF FLD1001,.EQ.,'A';
 0005    0004     $BLK2  IF FLD1002,.EQ.,'AB';
 0006    0005     $BLK3  IF FLD1003,.EQ.,'ABC';
 0007    0006     $BLK4  IF FLD1004,.EQ.,'ABCDEFG';
 0008    0007     $BLK5  IF FLD1005,.EQ.,'ABCDE';
 0009    0008     $BLK6  IF FLD1006,.EQ.,'ABCDEFGHIJ';
 0010    0009     $BLK7  IF FLD1007,.EQ.,'ABCDEFGH';
 0011    0010     $BLK8  IF FLD1008,.EQ.,'ABCDEFGHIJKL';
 0012    0011     $BLK9  IF FLD1009,.EQ.,'ABCDEFGHIJKLMNOPQRSTUVWXYZABCD';
 0013    0012     $BLK10 IF FLD1010,.EQ.,'ABCDEFGHIJ';
 0014    0013     !TEMP STR .SEL.,FLD101,'1','1';
 0015    0014     !FLD101 RP 'Z';
 0016    0015     !FLD102 RP 'YZ';
 0017    0016     !FLD103 RP 'XYZ';
 0018    0017     !FLD104 RP 'WXYZ';
 0019    0018     !FLD105 RP 'VWXYZ';
 0020    0019     !FLD106 RP 'UVWXYZ';
 0021    0020     !FLD107 RP 'TUVWXYZ';
 0022    0021     @ISC TEST FLD1009,.NE.,BLANKS;
 0023    0022     !FLD108 RP 'STUVWXYZ';
 0024    0023     !FLD109 RP 'RSTUVWXYZ';
 0025    0024     !TEMP2 STR .SEL.,FLD1010,'2','2';
 0026    0025     $BLK10 END;
 0027    0026     $BLK9  END;
 0028    0027     $BLK8  END;
 0029    0028     $BLK7  END;
 0030    0029     $BLK6  END;
 0031    0030     $BLK5  END;
 0032    0031     $BLK4  END;
 0033    0032     $BLK3  END;
 0034    0033     $BLK2  END;
 0035    0034     $BLK1  END;
 0036    0035     AGG10 EPV;
 0037    0036     $LAST IF !FLD102,.EQ.,'YZ';
 0038    0037     FLD101 RP !FLD101;
 0039    0038     FLD102 RP !FLD102;
 0040    0039     FLD103 RP !FLD103;
 0041    0040     FLD104 RP !FLD104;
 0042    0041     FLD105 RP !FLD105;
 0043    0042     FLD106 RP !FLD106;
```

```
NOTICE: NOT FOR DISCLOSURE OUTSIDE THIS COMPANY OR BELL COMMUNICATIONS RESEARCH WITHOUT WRITTEN PERMISSION.
 08/18/92 15:20:50                BELLCORE CSAS SYSTEM /  T C M             PAGE      2
 RUN BOOK(S): 252-573-511              TPAM TRAN RULES GENERATOR                 RUN ID   : VMMPM02
 USER DOC(S):                     CARRULE  SOURCE LISTING REPORT              REPORT ID: VMMPM02A
 BELL COMMUNICATIONS RESEARCH, INC.                                        RELEASE  : 16.0.2
 --------------------------------------------------------------------------------------------------------
 NUM    RUNTIME NUM    STATEMENT
 0044     0043     FLD107 RP !FLD107;
 0045     0044     FLD108 RP !FLD108;
 0046     0045     FLD109 RP !FLD109;
 0047     0046     $LAST END;
 0048     0047     $BLK IF FUNCTIND,.EQ.,'BBBBBBB';
 0049     0048     FUNCTIND RP UNCTINDX;
 0050     0049     $BLK END;
 0051            AGG2 AG;
 0052     0001     @TSTLAST TEST FLD201,.NE.,BLANKS;
 0053     0002     ECD RP 'ABCDEF';
 !!!!!!!! COMPILATION SUCCEEDED !!!!!!!!

                              END OF REPORT
```

```
                         EXTERNAL SYMBOL DICTIONARY       PAGE   1
SYMBOL   TYPE  ID  ADDR  LENGTH  LD ID FLAGS                          ASM H V 02 15.20 08/18/92
CARRULE   SD 0001 000000 00061C      00
```

```
                                    PAGE   2
 LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                        ASM H V 02 15.20 08/18/92
000000                         1 CARRULE  CSECT
000000 061C0002C3C1D9D9          2  DC X'061C0002C3C1D9D9E4D3C540F9F2F2F3F1F1F5F2F0F4F905CF0031C1C7C7F140'
000020 4040400036000101          3  DC X'4040400036000101C6D3C4F1F0F1404000000001E2E90097000809C1C7C7F1F0'
000040 4040400460002100          4  DC X'404040046000210051000 5E25CC1C7C7F2005B0005E26FC1C7C7F300650005E2'
000060 5CC1C7C7F4006F00          5  DC X'5CC1C7C7F4006F0005E25CC1C7C7F500790005E26FC1C7C7F600830005E26FC1'
000080 C7C7F7008D0005E2          6  DC X'C7C7F7008D0005E25CC1C7C7F800000005E25CC1C7C7F900C000030B5BC2D3D2'
0000A0 F14040400442001E          7  DC X'F14040400442001E00B40007E2C6D3C4F1F0F0F100BA0001D3C100000001D611'
0000C0 00EA00030B5BC2D3          8  DC X'00EA00030B5BC2D3D2F24040400435001C00DD0007E2C6D3C4F1F0F0F200E400'
0000E0 02D3C1C200000001          9  DC X'02D3C1C200000001D611011500030B5BC2D3D2F34040400428001A01070007E2'
000100 C6D3C4F1F0F0F301         10  DC X'C6D3C4F1F0F0F3010F0003D3C1C2C300000001D611014400030B5BC2D3D2F440'
000120 4040041B00180132         11  DC X'4040041B001801320007E2C6D3C4F1F0F0F4013E0007D3C1C2C3C4C5C6C70000'
000140 0001D61101710003         12  DC X'0001D611017100030B5BC2D3D2F5404040040E001601610007E2C6D3C4F1F0F0'
000160 F5016B0005D3C1C2         13  DC X'F5016B0005D3C1C2C3C4C500000001D61101A300030B5BC2D3D2F64040400401'
000180 0014018E0007E2C6         14  DC X'0014018E0007E2C6D3C4F1F0F0F6019D000AD3C1C2C3C4C5C6C7C8C9D1000000'
0001A0 01D61101D300030B         15  DC X'01D61101D300030B5BC2D3D2F740404003F4001201C00007E2C6D3C4F1F0F0F7'
0001C0 01CD0008D3C1C2C3         16  DC X'01CD0008D3C1C2C3C4C5C6C7C800000001D611020700030B5BC2D3D2F8404040'
0001E0 03E7001001F00007         17  DC X'03E7001001F00007E2C6D3C4F1F0F0F80201000CD3C1C2C3C4C5C6C7C8C9D1D2'
000200 D300000001D61102         18  DC X'D300000001D611024D00030B5BC2D3D2F940404003DA000E02240007E2C6D3C4'
000220 F1F0F0F90247001E         19  DC X'F1F0F0F90247001ED3C1C2C3C4C5C6C7C8C9D1D2D3D4D5D6D7D8D9E2E3E4E5E6'
000240 E7E8E9C1C2C3C400         20  DC X'E7E8E9C1C2C3C400000001D611027F00030B5BC2D3D2F1F0404003CD000C026A'
000260 0007E2C6D3C4F1F0         21  DC X'0007E2C6D3C4F1F0F1F00279000AD3C1C2C3C4C5C6C7C8C9D100000001D61102'
000280 A90004045AE3C5D4         22  DC X'A90004045AE3C5D4D740404002920001D602029D0006E2C6D3C4F1F0F102A300'
0002A0 01D3F100000001D3         23  DC X'01D3F100000001D3F102BC0001015AC6D3C4F1F0F14000000001D3E902D00001'
0002C0 015AC6D3C4F1F0F2         24  DC X'015AC6D3C4F1F0F24000000002D3E8E902E50001015AC6D3C4F1F0F340000000'
0002E0 03D3E7E8E902FB00         25  DC X'03D3E7E8E902FB0001015AC6D3C4F1F0F44000000004D3E6E7E8E90312000101'
000300 5AC6D3C4F1F0F540         26  DC X'5AC6D3C4F1F0F54000000005D3E5E6E7E8E9032A0001015AC6D3C4F1F0F64000'
000320 000006D3E4E5E6E7         27  DC X'000006D3E4E5E6E7E8E903430001015AC6D3C4F1F0F74000000007D3E3E4E5E6'
000340 E7E8E9036D000305         28  DC X'E7E8E9036D0003057CC9E2C340404040035C0007E2C6D3C4F1F0F0F903670006'
000360 E2C2D3C1D5D2E200         29  DC X'E2C2D3C1D5D2E200000001D61203870001015AC6D3C4F1F0F84000000008D3E2'
000380 E3E4E5E6E7E8E903         30  DC X'E3E4E5E6E7E8E903A20001015AC6D3C4F1F0F94000000009D3D9E2E3E4E5E6E7'
0003A0 E8E903CD0004045A         31  DC X'E8E903CD0004045AE3C5D4D7F2404003B50001D60203C10007E2C6D3C4F1F0F1'
0003C0 F003C70001D3F200         32  DC X'F003C70001D3F200000001D3F203DA0000075BC2D3D2F1F0404003E70000075B'
0003E0 C2D3D2F940404003         33  DC X'C2D3D2F940404003F40000075BC2D3D2F8404040040100000 75BC2D3D2F74040'
000400 40040E0000075BC2         34  DC X'40040E0000075BC2D3D2F6404040041B0000075BC2D3D2F5404040042800000 7'
000420 5BC2D3D2F4404040         35  DC X'5BC2D3D2F44040400435000 0075BC2D3D2F340404004420000075BC2D3D2F240'
000440 4040044F0000075B         36  DC X'4040044F0000075BC2D3D2F14040400460000000AC1C7C7F1F040404000970020'
000460 048A00030B5BD3C1         37  DC X'048A00030B5BD3C1E2E3404040056B0009047D0007E25AC6D3C4F1F0F2048400'
000480 02D3E8E900000001         38  DC X'02D3E8E900000001D61104A3000101C6D3C4F1F0F1404000000007E25AC6D3C4'
0004A0 F1F0F104BC000101         39  DC X'F1F0F104BC000101C6D3C4F1F0F2404000000007E25AC6D3C4F1F0F204D50001'
0004C0 01C6D3C4F1F0F340         40  DC X'01C6D3C4F1F0F3404000000007E25AC6D3C4F1F0F304EE000101C6D3C4F1F0F4'
0004E0 404000000007E25A         41  DC X'404000000007E25AC6D3C4F1F0F40507000101C6D3C4F1F0F5404000000007E2'
000500 5AC6D3C4F1F0F505         42  DC X'5AC6D3C4F1F0F50520000101C6D3C4F1F0F6404000000007E25AC6D3C4F1F0F6'
000520 0539000101C6D3C4         43  DC X'0539000101C6D3C4F1F0F7404000000007E25AC6D3C4F1F0F70552000101C6D3'
000540 C4F1F0F840400000         44  DC X'C4F1F0F8404000000007E25AC6D3C4F1F0F8056B000101C6D3C4F1F0F9404000'
000560 000007E25AC6D3C4         45  DC X'000007E25AC6D3C4F1F0F905780000075BD3C1E2E340404005A800030B5BC2D3'
000580 D24040404005C200         46  DC X'D24040404005C20001059600 08E2C6E4D5C3E3C9D5C405A20007D3C2C2C2C2C2'
0005A0 C2C200000001D611         47  DC X'C2C200000001D61105C2000101C6E4D5C3E3C9D5C400000008E2E4D5C3E3C9D5'
0005C0 C4E700000000075B         48  DC X'C4E700000000075BC2D3D24040404000000002C1C7C7F240404040 0604000305'
0005E0 7CE3E2E3D3C1E2E3         49  DC X'7CE3E2E3D3C1E2E305F30006E2C6D3C4F2F0F105FE0006E2C2D3C1D5D2E20000'
000600 0001D61200000001         50  DC X'0001D6120000000101C5C3C44040404040000000006D3C1C2C3C4C5C6'
                                51        END
```

```
                         DIAGNOSTIC CROSS REFERENCE AND ASSEMBLER SUMMARY                PAGE    3
                                                       ASM H V 02 15.20 08/18/92
   NO STATEMENTS FLAGGED IN THIS ASSEMBLY
 OVERRIDING PARAMETERS-  NODECK,OBJECT,ESD,RLD,LIST,BATCH,XREF(SHORT)
 OPTIONS FOR THIS ASSEMBLY
  NODECK, OBJECT, LIST, XREF(SHORT), NORENT, NOTEST, BATCH, ALIGN, ESD, RLD, NOTERM, NODBCS,
  LINECOUNT(55), FLAG(0), SYSPARM()
  NO OVERRIDING DD NAMES
   51 CARDS FROM SYSIN       0 CARDS FROM SYSLIB
   69 LINES OUTPUT          30 CARDS OUTPUT
```

```
MVS/DFP VERSION 3 RELEASE 3 LINKAGE EDITOR    15:20:56  TUE  AUG 18, 1992
JOB CLIFF     STEP JOBSTEP2   PROCEDURE STEP03
INVOCATION PARAMETERS - RENT,XREF,,LIST
ACTUAL SIZE=(317440,79872)
OUTPUT DATA SET CSASI16.F0002.PGMLIB IS ON VOLUME SMN581

                          CROSS REFERENCE TABLE

 CONTROL SECTION                ENTRY

  NAME   ORIGIN  LENGTH          NAME  LOCATION    NAME  LOCATION    NAME  LOCATION    NAME  LOCATION
 CARRULE     00    61C

ENTRY ADDRESS      00

TOTAL LENGTH      620
** CARRULE  REPLACED AND HAS AMODE 24
** LOAD MODULE HAS RMODE 24
** AUTHORIZATION CODE IS        0.
**MODULE HAS BEEN MARKED REENTERABLE, AND REUSABLE.
'br\}
```

7. Translation Set - Up Procedures

Various functions must be performed inside and outside of Translation Administration (TA) and preliminary to execution so that the translation can be performed. These functions are as follows:

1. The rule set name (rule **ID**) must be stored in the Path segment in the SEC Database. The rule ID identifies the translation rules for a particular application, CSAS system release, and over a specific path and direction (scenario),
2. The translation rules must be written, compiled using batch run VMMPM02, and stored in a PDS known to IMS, e.g., PGMLIB.
3. Data must be stored in the appropriate TTS Table(s) if the Table Look - up translation is used.

Appendix A: TPAM Translation Tips

This section discusses some common questions and problems that occur during the creation of rule sets.

A.1 Coding “ELSE” Logic

PROBLEM: How to code an "ELSE" for false logic when the rule's language does not directly provide an ELSE for **if/end** logic?

SOLUTION: There are two choices: either code the "ELSE" indirectly or redesign the logic. If you elect the former choice, here is the suggested thought process:

1. Use a temporary variable (such as RESULT) and assume by an **rp** that RESULT will be FALSE.
2. Do the **if** logic testing for your logical condition being TRUE.
    a. If the logical condition is TRUE, set RESULT to TRUE.
    b. Do any other needed rules for the TRUE condition within this **if/ end** block.
3. Complete the *if*/**end** block with the normal **end**.
4. Start another **if/end** testing for RESULT being false.

    An example follows:

```
RESULT      rp        'F';
$BLK1       if        STATE,.EQ.,'CA';
RESULT      rp        'T';
*                     true logic goes here;
$BLK1       end;
$BLK2       if        RESULT,.EQ.,'F';
*                     false logic goes here;
$BLK2       end;
```

A.2 Handling Missing FIDS

PROBLEM: How to test for a missing FID when the **if** rule is not executed when the FID is missing?

SOLUTION: Testing for missing FIDs requires an extension of the above ELSE technique. We know if the FID is present, then that FID will always be equal to itself. Likewise, if the desired FID is absent, then the **if** rule will not be executed; rather, the TPAM translation continues with the next rule (if there is any) within the rule set aggregate following the corresponding **end** statement. Assume that we may or may not have an optional FID called MAYBE and that MISSING is our temporary variable in the following example:

```
MISSING   rp     'T';
$BLK3     if     MAYBE,.EQ.,MAYBE;
*                logic case when FID called MAYBE is not missing;
MISSING   rp     'F';
$BLK3     end;
$BLK4     if     MISSING,.EQ.,'T';
*                logic when FID is missing goes here - such as below;
MAYBE     rp     'DEFAULT VALUE';
$BLK4     end;
```

A.3 FCIF Delimiters

PROBLEM: While FCIF syntax needs escape characters (\) for its delimiters (**<=;>%\), are FCIF delimiters a concern within the rule?

SOLUTION: No, you do **NOT** account for FCIF delimiters at translation time when the rule set load module is used.

Explanation: Parsing of the FCIF messages into the internal format during translation time has stripped the FCIF escape characters from the message. Conversely, when TPAM maps an output FCIF message, TPAM restores the FCIF escape characters as needed.

Example: We want E=MC**2 as the value for FID named FORMULA. Please note that both = and * are FCIF delimiters. Internal translation format and the MDA and DSECT formats have the value as E=MC**2 for a length of seven characters. The FCIF equivalent for this FID is "FORMULA=E\\=MC**2;". The rule to test the formula being equal to the above value is:

```
$ALBERT    if       FORMULA,.EQ.,'E=MC*
                    *2';
*                   true logic goes here;
$ALBERT    end;
```

A.4 Identically Named Fields In An Aggregate

PROBLEM: What is the handling of identically named fields within an aggregate?

For example, ORDRR<DUP=ABC;DUP=XYZ;>

SOLUTION: Translation only handles the *first* occurrence of fields with the same name in the aggregate occurrence. In this example, translation can *only* access the field with the value of ABC. Even if the field with the value of ABC were to be deleted, the field with the value of XYZ would remain. Therefore, it is recommended that duplicate named fields *not* be used.