ADEPT for PBX Integration Boards

User's Guide

May 2006

05-2520-001

COPYRIGHT NOTICE

Publication Date: May, 2006

Document Number: 05-2520-001

Intel Converged Communications, Inc.
1515 Route 10
Parsippany NJ 07054

For **Technical Support**, visit the Intel Telecom Support Resources website at:
http://developer.intel.com/design/telecom/support

For **Products and Services Information**, visit the Intel Telecom Products website at:
http://www.intel.com/design/network/products/telecom

For **Sales Offices** and other contact information, visit the Buy Telecom Products page at:
http://www.intel.com/buy/networking/telecom.htm

Revision History

This revision history summarizes the changes made in each published version of this document.

Document No.	Publication Date	Description of Revisions
05-2520-001	May 2006	Initial version of document.

Table of Contents

1 INTRODUCTION

1.1 Purpose

This document describes a tool called the ANI/DNIS-Enabled Parsing Tool (ADEPT). ADEPT is a set of software modules that functions as a portable, generic, Calling Party Identification (CPID), Automatic Number Identification (ANI) and Dialed Number Identification Service (DNIS) display parser. By using ADEPT, a user can update display parsing rules that are specific to a site or an application, very easily.

Previous display parsers for the PBX Integration Boards could not be modified without rebuilding the firmware. With the ADEPT implementation, the CPID parsing is no longer controlled in the firmware. Instead, it is controlled in the d42 library with appropriate rules changes. The advantage of ADEPT is that a customer can change the display parser just by changing the rules file. The rules file is implemented as a simple text file.

Both the Intel® Dialogic® D/82 JCT-U and D/42 JCT-U boards support the ADEPT functionality.

1.2 Files Changed/Added

The files that were changed or added to the Intel Dialogic System Release for ADEPT implementation are listed below:

File name	**Added/Changed**	**Location**
Hicom.adt	Added	<*installation folder*>\dialogic\cfg
Lucent.adt	Added	<*installation folder*>\dialogic\cfg
Mitel.adt	Added	<*installation folder*>\dialogic\cfg
Nec.adt	Added	<*installation folder*>\dialogic\cfg
Ntbcm.adt	Added	<*installation folder*>\dialogic\cfg
Ntm1.adt	Added	<*installation folder*>\dialogic\cfg
Rolm.adt	Added	<*installation folder*>\dialogic\cfg
Libd42mt.dll	Changed	<*installation folder*>\dialogic\lib winnt\system32\
Libd42mt.lib	Changed	<*installation folder*>\dialogic\lib
D42lib.h	Changed	<*installation folder*>\dialogic\inc
D82u.fwl	Changed	<*installation folder*>\dialogic\data
D82ucsp.fwl	Changed	<*installation folder*>\dialogic\data
D42ucsp.fwl	Changed	<*installation folder*>\dialogic\data
Lucent_2_wire.fwl	Changed	<*installation folder*>\dialogic\data
Lucent_4_wire.fwl	Changed	<*installation folder*>\dialogic\data
Mitel_DNIC_M430.fwl	Changed	<*installation folder*>\dialogic\data
Mitel_DNIC_M4150.fwl	Changed	<*installation folder*>\dialogic\data
Mitel_DNIC_M420.fwl	Changed	<*installation folder*>\dialogic\data
NEC_DTerm_III.fwl	Changed	<*installation folder*>\dialogic\data
Nortel_BCM.fwl	Changed	<*installation folder*>\dialogic\data

File name	Added/Changed	Location
Nortel_Meridian_1.fwl	Changed	<*installation folder*>\dialogic\data
Nortel_Norstar.fwl	Changed	<*installation folder*>\dialogic\data
Siemens_Hicom.fwl	Changed	<*installation folder*>\dialogic\data
Siemens_Rolm.fwl	Changed	<*installation folder*>\dialogic\data
Siemens_Rolm_9006.fwl	Changed	<*installation folder*>\dialogic\data

2 ADEPT OVERVIEW

The ANI/DNIS-Enabled Parsing Toolkit, ADEPT, is a set of software modules that functions as a display parser. ADEPT provides users a new way to control the parsing of CPID, ANI, and DNIS information. D/82 JCT-U and D/42 JCT-U boards support the ADEPT functionality.

The previous display parsers for the **d42_gtcallid( )** function in the PBX Integration Boards could not be modified without rebuilding the firmware. With the ADEPT implementation, CPID parsing is no longer controlled by the firmware. Instead, parsing control is managed in the d42 library by editing rules files. The rules files are implemented as simple text files, so a customer can change the display parsers just by editing the rules file. The rules file defines the rules used to extract CPID from the digital station set displays. ADEPT provides significant productivity advantages by eliminating the need to rebuild firmware. No recompilation is needed in any subsystem.

With ADEPT implementation, the Intel Dialogic System Release also provides a PC-based test tool. The test tool allows the developer to run the rules file to test the parsing rules against actual displays. Example displays can be gathered from traces, placed in a text file, and run through the test application. This provides a way for the rules to be verified, including a means to test the backward-compatibility of rules against known display examples. In this way, the developer can test the parser without having to run the actual product on-site.

With ADEPT functionality, the customer can add a display parsing rule in three simple steps:

1. Add the rule in the rules file.
2. Test the added rule with the ADEPT Test application to verify that the new rule works as designed.
3. Close all Intel Dialogic devices and open at least one device to ensure that the new rule has been read.

ADEPT supports the extraction of source and destination party information, call reasons and call origins (internal, external). The **d42_gtcallidex( )** function can be used to get call reasons and call type. The structure that **d42_gtcallidex( )** returns can be obtained from <installation folder>\dialogic\inc\d42lib.h header file.

The Intel Dialogic System Release provides default parsing rules files (*.adt) for each digital switch supported by the PBX Integration Board. The rules files are located in <installation folder>\dialogic\cfg folder. The names of the rules files are:

Rules file name	**Integration**
Hicom.adt	Siemens Hicom
Lucent.adt	Lucent 2-wire and Lucent-4 Wire
Mitel.adt	Mitel Superset 430, 420 and 4150
Nec.adt	NEC PBX and KTS
Ntbcm.adt	Nortel BCM
Ntm1.adt	Nortel M1
Rolm.adt	Rolm 9005 and 9006

2.1 The ADEPT module in libd42mt.dll

The ADEPT functionality is implemented in the libd42mt.dll and libd42mt.lib files with the appropriate constant declared in the d42lib.h file.

A rebuild of the application with the new libd42mt.lib and d42lib.h files is necessary to enable the ADEPT functionality.

- libd42mt.dll resides in:

  windows\system32 folder and <installation folder>\dialogic\lib folder.

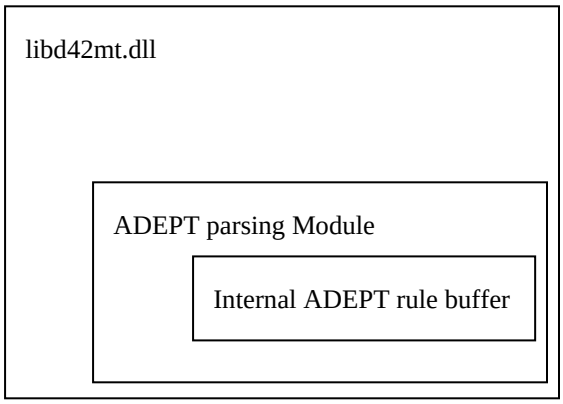
- libd42mt.lib resides in:

  <installation folder>\dialogic\lib.

- d42lib.h resides in:

  <installation folder>\dialogic\inc.

The library libd42mt.dll maintains an internal buffer to hold the rules read from the rule file for the appropriate integration. The diagram below shows the relationship of the ADEPT functionality to libd42mt.dll.



Please refer to the diagram below.

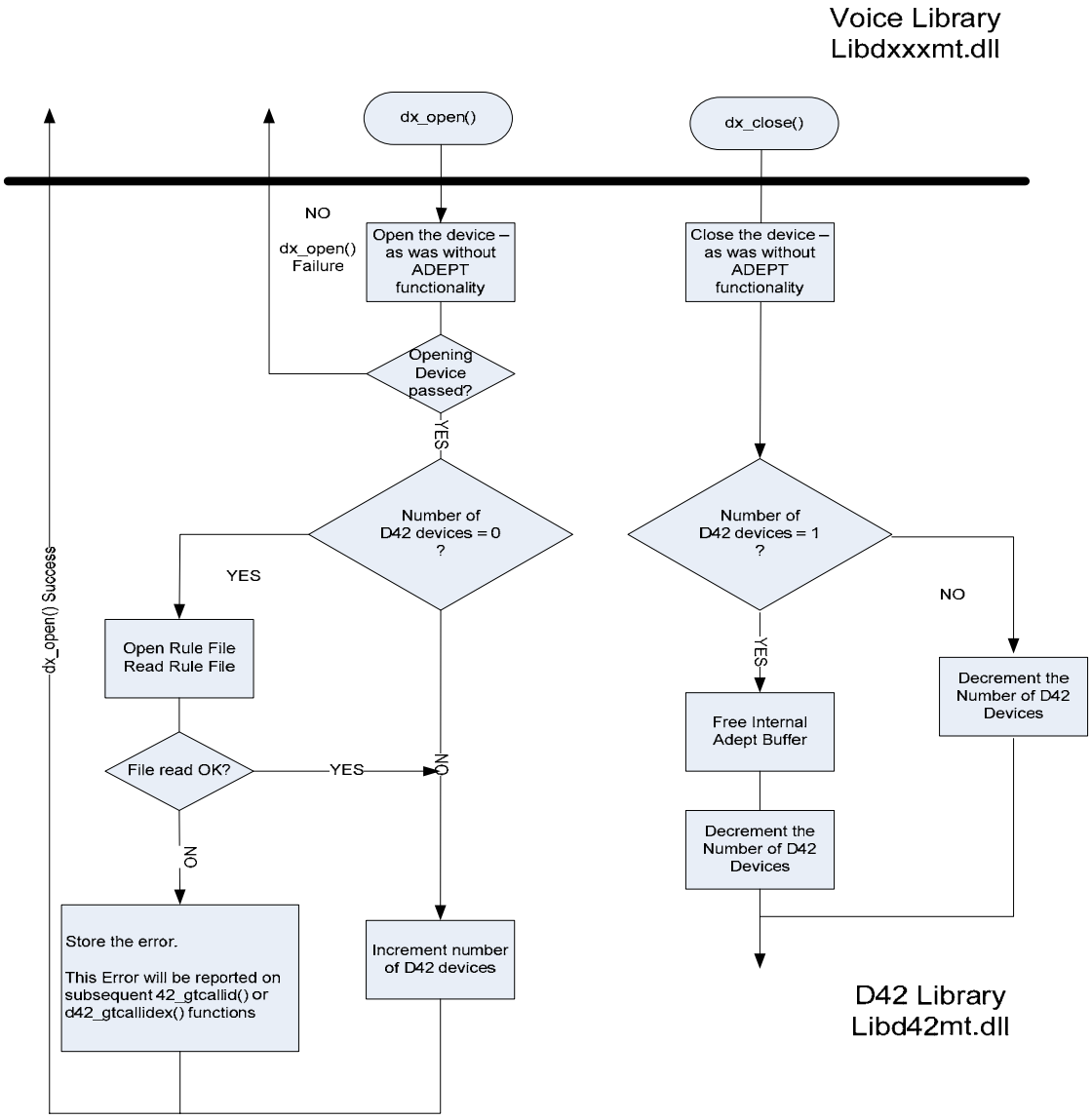
When the **dx_open()** function is first called, the d42 device is instantiated for the proper integration. At this time, the rules file is read and transferred to the internal buffer. The next **dx_open()** action increments the number of ADEPT devices opened but does not read the rules file anymore.

The **dx_close()** function decrements the number of d42 ADEPT devices and when that reaches 1, it decrements the number of devices to 0 and frees the ADEPT buffer for the rules file. At this time, a **dx_open()** reads the rules file again and fills the internal buffer with the rules from the rules file.

Hence, if the rules file is changed for adding a new rule, it is highly recommended to:

- stop the application,
- stop and restart the Intel Dialogic system service,
- start the application back up again.

These steps will ensure that the new rules file has been read. The procedure should be to close all the D42 devices and then to open at least one device to read the new rules file.



When the **d42_gtcallid( )** or **d42_gtcallidex( )** function is called, the following steps are executed in the libd42mt.dll library:

1) Check if an error was stored from the **dx_open( )** function. If yes then, report that error. In this case the function fails.
2) If no error was stored from the **dx_open( )** action:
    a. Get the latest display from firmware
    b. Pass this display to the ADEPT parsing module
    c. ADEPT parses the display based on the rules file that was read in it's internal buffer
    d. The ADEPT parsing module returns the parsed CPID information back to libd42mt.dll and libd42mt.dll returns this value to the application.

2.2 ADEPT Supported reason codes and origin from D42 Library

The structure returned by **d42_gtcallidex( )** is as follows:

```
#define CALLIDLEN 16
typedef struct tagCALLIDEX
{
        char called_id[CALLIDLEN];
        char calling_id[CALLIDLEN];
        int  call_type;
        int  reason_code;
} CALLIDEX;
```

ADEPT supports the following reason codes:

Constant	**Value**	**Description**
RSN_DIRECT	0x01	Direct call
RSN_FBUSY	0x02	call Forwarded because busy
RSN_FNOANS	0x03	call forwarded because no answer
RSN_SUPR_XFER	0x04	Supervised call transfer
RSN_UNSUPR_XFER	0x05	call transfer because
RSN_FALL	0x06	call Forwarded always
RSN_TIME	0x07	display contains time info only (no CPID)

ADEPT supports the following call types:

Constant	**Value**	**Description**
CALL_TYP_INTERNAL	0x01	Internal call
CALL_TYP_EXTERNAL	0x02	external call

NOTE: In order to get the above mentioned return values, the rules file must have the case-sensitive translation-rules.

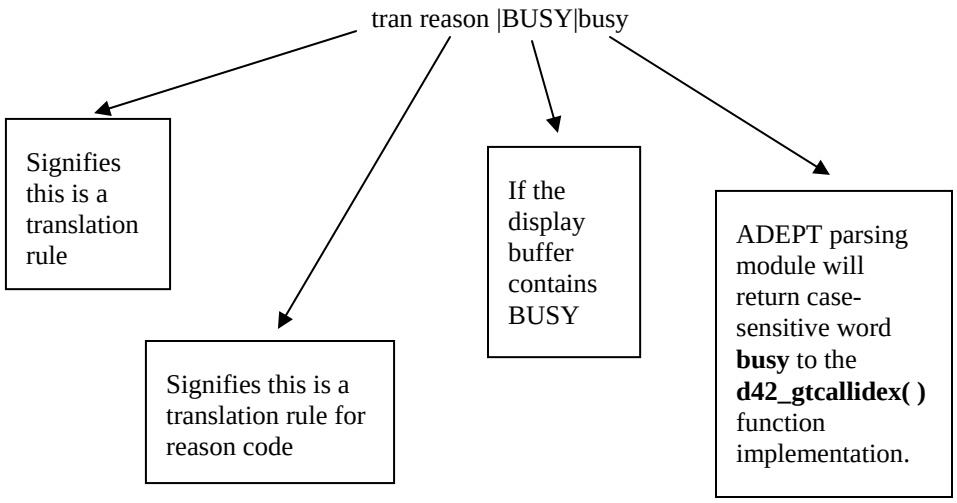
A typical translation section in an ADEPT configuration file (.adt) looks like this:

```
# Reason translations
tran reason |BUSY|busy
tran reason |NO ANS|no-answer
tran reason |DO NOT DISTURB|no-answer
tran reason |ALWAYS|fwd-all
tran reason |FWD|fwd-all
tran reason |FORWARD|fwd-all
tran reason |FORWARDED|fwd-all
tran reason |default|no-answer
tran origin |default|internal
```

For example, the line

tran reason |BUSY|busy

defines a translation rule. If we look closely, the meaning of each literal is as follows:



This means that, when an ADEPT module finds the word **BUSY** in the display buffer, it will pass the case sensitive word **busy** as the reason code to the **d42_gtcallidex( )** function in libd42mt.dll. This function has been designed only to convert **busy** to a call reason of RSN_BUSY. If **Busy (**B in uppercase**)** is written in the last part of the rule (or the return value from the rule), the **d42_gtcallidex( )** function will not return RSN_BUSY. Instead, it will return the default reason code, RSN_DIRECT. This is because the reason code is case sensitive and must match exactly.

The last part of the translation in the rules file must be one of the following for call reason:

Reason Code	Description
busy	Forwarded busy
no-answer	Forwarded no-answer
fwd-all	Forwarded on both busy and no-ans
xfr-supr	Supervised Transfer
xfr-unsupr	Unsupervised Transfer
time	This is for regular time update
	NOTE: If the reason code does not match any of the codes mentioned above the default reason code, RSN_DIRECT will be returned by **d42_gtcallidex( )**.

The call origin must be as follows:

Origin	Description
External	External call
Default or internal	Internal call

NOTE: All of the letters are lowercase. For a detailed understanding of a sample rule file, e.g., how a rule is defined, display is parsed, how to define internal or external call origin etc., please see the next section with example.

2.3 Basic structure of a rule and a rule file

Consider the following translation section of a rules file;
tran origin |default|internal
tran reason |default|direct
tran reason |BFWD|busy
tran reason |FWD|fwd-all
tran reason |NAFWD|no-answer

Consider a rule as follows:
Rule 1:
rule FROM\s\(\d(3,3)\)\d(3,3)\-\d(4,4)\n\b~*\b\d*\s.*
src_number 1 2 3
dst_number 4
reason 1
origin external

This rule states that ADEPT will parse successfully if we get a string

- that starts with the word "FROM"
- followed by white space characters [\s]
- followed by 3 digits within parenthesis followed by 3 more digits [\d(3,3)]
- followed by the character – [\-]
- followed by 4 digits [\d(4,4)] then a newline [\n]
- followed by a word boundary with a reason code [\b~*\b]
- followed by a number (any numbers of digit)
- followed by string

For example, if we get the following display:

FROM (111)222-3333
FWD 2007 TEST 3

ADEPT will parse the display (always from left to right) successfully because:

- It starts with the word "FROM"
- Then white space characters,
- Then 3 digits within parenthesis [\(\d(3,3)\)] – the 1st parsed number 111 from the above display
- Then 3 more digits [\d(3,3)] – the 2nd parsed number 222
- Then the character – [\-]
- Then 4 digits [\d(4,4)] – the 3rd parsed number 3333
- Then a newline [\n] – this ends the first line of the display
- Then a word boundary with a reason code [\b~*\b] – the reason code FWD
- Then by a number (any numbers of digit) – the 4th number parsed, 2007
- Then a string

- src_number 1 2 3: This line means the 1st, 2nd and 3rd parsed number will be source number and from this example it will be 111 222 3333. If the format of the src number is 1 2 3 with spaces, the number will be 111 222 3333. If the src_number is 1-2-3 with dashes, the number will be 111-222-3333. The number will correspond to the way the src_number is defined.
- dst_number 4 : This means that the 4th parsed number will be destination number

- Reason Code will be fwd-all because that matches the word FWD in the display, as defined in the rule:
  reason 1 [the first parsed reason code]
- The origin (or call type) will be external because this rule states that if a display with a string is parsed, this will be treated as an external call by the line:
  origin external

Consider another rule:

Rule 2:
rule \d*\s*.*
src_number 1
reason direct

This rule states that if a display that has a number (any number of digits) followed by any number of white spaces, followed by any number of characters (of any type), then the source number will be the first parsed digit, the reason will be direct and , the origin will be internal (because there is no mention of the call origin).

Please note that in a rules file, rules must be written in order from very specific to very generic, from top to bottom. The first rule will be used to parse a display first, if the display does not match that rule, the second rule will be used to parse, and so on. So, if the second rule in the example above is placed as the first rule in the rules file, **Rule 1** in the examples above will never be executed because the display would have already been parsed by the very generic **Rule 2**.

The rules file should have a basic structure as follows:

Translation section	tran origin |default|internal tran reason |default|direct tran reason |BFWD|busy tran reason |FWD|fwd-all tran reason |NAFWD|no-answer
Very specific Rule	rule FROM\s\(\d(3,3)\)\d(3,3)\-\d(4,4)\n\b~*\b\d*\s.* src_number 1 2 3 dst_number 4 reason 1 origin external
Less specific rule	: : :
Very general rule	rule \d*\s*.* src_number 1 reason direct

Points to remember

- The top rule in the rules file will be used first. If a display could be parsed with this rule, the parsing will stop and the next rule in the rules file will not be executed. Parsing is based on the first match in the rules file. In the rules file, the specific rules should be defined first, followed by more general rules.
- The display is parsed from left to right by a rule.

2.4 ADEPT Limitations

The following known limitations must be considered when writing a rule:

- The maximum size of a rule is 32 characters. If the rule has more than 32 characters, the character corresponding to the 33rd element is not processed by ADEPT. No error is returned, but ADEPT truncates the rule to 32 characters and uses the partial rule to parse the display.
- The maximum size of the rule file is 8096 bytes (8K). If the file size is more than the 8K limit, ADEPT does not return an error code, but ADEPT truncates the rule file to 8K and uses whatever is contained in the 8K buffer to parse the display.

These two limitations currently impose the restriction of using comments in the rule file as little as possible.

2.5 ADEPT Related ERROR from D42 Library

The **d42_gtcallid( )** and **d42_gtcallidex( )** functions return various ADEPT related errors. If these functions fail, the return code will be -1. The error code and error message can be extracted using

ATDV_LASTERR and ATDV_ERRMSGP macros as follows:

```
char szErrMsg[256];
:
:
if (d42_gtcallidex(d4x_desc[0][chan].desc, &callidex)==-1)
{
        sprintf(szErrMsg, "%s (0x%X)",  ATDV_ERRMSGP(rcode), ATDV_LASTERR(rcode));
}
```

In case of an ADEPT error, the d42 library will return the following errors:

Error	**Value**	**Description**
ED42_ADEPTFAILURE	0x0514	General ADEPT failure
ED42_ADEPTNOFILE	0x0515	ADEPT failure – no rule file
ED42_ADEPTNORULE	0x0516	ADEPT failure – no rule in config handles this display
ED42_ADEPTTOOBIG	0x0517	ADEPT failure - display is too big for parsing

2.6 The ADEPT Configuration File

An ADEPT configuration file (*.adt) is a text file that contains expressions that represent different types of displays on the target digital telephony switch interface. The configuration file contains translations between switch-specific tokens and normalized application tokens, rules, and expressions that specify the location of the source, destination, and reason information in the display text.

Multiple configuration files may be written, each supporting a different digital switch interface or specific site requirements.

Comments can start with either a pound sign (#) or a semi-colon (;).

Example Configuration File

```
### Adept.cfg file - Siemens Hicom
#
#
# Reason translations
tran reason |default|direct
tran origin |default|internal
# ignore name fields of 'Set', since switch uses that when there
is no name field
tran src_name |Set|
tran dst_name |Set|
# time in first row, ignore these
rule .*\d+:\d+.*\n.*
reason time

# 501 calling
# George calling
rule \b\w*\b\d*\bcalling\b\n.*
src_number 1
src_name 1

# 501 > 203
rule \b\w*\b\d*\b>\b\w*\b\d*\b\n.*
src_number 1
src_name 1
dst_number 2
dst_name 2

# 501
rule \b\w*\b\d*\b\n.*
src_number 1
src_name 1
```

3 CONFIGURATION FILE SYNTAX

3.1 Syntax Overview

The ADEPT Configuration File Syntax is used to describe a display parser. The syntax of the rules is based on the regular expression syntax of the programming language, Perl.

The syntax supports the extraction of the following information:

1. The Source Party ID (can include ANI information)
2. The Destination Party ID (can include DNIS information)
3. The reason code (no-answer, busy, etc.)
4. The call origin (internal, external).

A configuration file consists of translation descriptors and call-class parsing rules.

Translation descriptors define the translations of switch-specific reason code strings into normalized strings that the **d42_gtcallidex( )** function wishes to see. For instance, the **d42_gtcallidex( )** function in libd42mt.dll wants no-answer displays to be tagged with the "no-answer" string as described in 2.2, while the telephony switch may use the string "NoAns" in its displays. The translation section can define the translation between the switch-specific "NoAns" string and the **d42_gtcallidex( )** required "no-answer" string.

Call-class parsing rules are Perl-like expressions that define displays. A configuration file may contain multiple rules, each representing a different type of display. In each different display, the CPID information may be located in different fields.

The ADEPT algorithm attempts to match an input display string to a display rule defined in the configuration file. If a match is made, the ADEPT algorithm uses the call-party, reason code, and call origin specifies of the matching rule. In this manner, ADEPT can extract the CPID from the correct locations in the display.

3.2 Translation Descriptors

Translation descriptors define translations between telephony switch-specific display tokens and strings that the application uses. Translation descriptors are global translations that govern all display parsing rules.

Example:

```
trans reason |FWD|fwd-all
```

This defines a translation of a switch-specific reason code of "FWD" to the application string "fwd-all". If the token "FWD" is found in the reason-code section of a display, the reason code presented to the application will be "fwd-all". This provides a means of normalizing different switch-specific reason codes into strings that can be recognized by the application. Note, token comparison is case sensitive.

Example:

```
trans origin |default|internal
```

If the switch-specific code is "default", then the application string of the translation is set as the default translation result if no switch-specific string is found. In this case, the default call origin for all displays is "internal".

Example:

```
trans dst_number |408|
```

If there is no translation output text, then the input text is essentially translated to "". This can be used to ignore certain tokens or extensions. For instance, if 408 is a hunt group, this translation entry would ignore it in the display and not present it to the application as a call party extension.

3.3 Call Class Rules

On a telephony switch, there can be several different display formats for call information. Each format must be parsed differently, and it is not always easy to discern which format is being used. Displays which use the same format are said to be in the same "Call Class". That is, two displays can be parsed using the same rules if they are in the same call class. If a display cannot be parsed using the same rules, then a new call class must be declared.

3.3.1 Regular Expressions

Call classes will be recognized using user-defined regular expressions. ADEPT uses standard regular expression "meta-characters" (special characters that are used to describe sequences of regular characters) and ADEPT specific meta-characters.

The display string will be parsed from left to right – the meta characters all have the same precedence in a rule, they are used in a rule from left to right.

Table 3-1 - ADEPT Regular Expression Meta-Characters

Character	Purpose
.	Matches any single character.
?	0 or 1 of preceding character.
*	0 or more of preceding character.
+	1 or more of preceding character.
~	A reason code or call origin character.
\	Used when a meta-character must be matched. Example: **\.** matches a period, **\?** matches a question mark, \\ matches a back slash.
\n	Matches a new line character.
\nnn	Specifies the numeric value (in octal) of the match character. Example: \101 matches the letter A. However, it is typically used to match non-displayable or extended ASCII characters. Example: \011 matches the vertical tab character. Allowed values are 000 to 377, any value outside this range yields indeterminate results.
\d	Digit character.
\D	Non-digit character.
\w	Word character. Must start with alpha, then can be alpha, digit, -, comma, space, period, or _.
\s	Whitespace character. Space, newline, or tab.
\b	Word boundary. Whitespace, punctuation, beginning or end of text. Used to specify that strings are bounded at start or end of string. Not explicitly searched for. Checked on reasons, digits, and literal strings.
(min,max)	Optionally follows other chars (meta or non). For a match, character length must be within min/max. Min may be zero. Example: **\d(7,10)** matches a number from seven to ten digits long.
[min,max]	Optionally follows other chars (meta or non). For a match, character length must be within min/max. Min may be zero. Example: **\d[7,10]** matches a number from seven to ten digits long.

3.3.2 ADEPT Specific String Literals

The ADEPT rules file is composed of translation definitions and rules. This rules file (.adt file) contains a one-time translations section for the reason codes and origins and a rules section containing several rules to parse a display. In order to define the translation and a rule, the ADEPT rules file needs special case-sensitive string literals. They are explained in the following table.

ADEPT String Literal (case-sensitive)	**Explanation**
default	The default reason or origin
src_number	Source number or the calling party
dest_number	Destination number or the called party in a forwarded call
src_name	Source name – not supported in PBX Integration Board
dest_name	Destination name in a forwarded call – not supported in PBX Integration Board
reason	Reason code, such as, forwarded busy, no-answer etc.
origin	Origin of the call, such as, internal call, external call etc.
tran	Translation section for reason and origin in a ADEPT rule file
rule	Define a parsing rule

3.3.3 Rule Syntax

A call-class rule starts with the tag **rule.** All characters following the tag define the rule. Following the rule are specifiers that define the location in the display of the CPID information.

For example, to describe the display:

[a= JOE 123 to BILL 456 b]

the lines might read:

```
rule .= \w*\s\d* to \w*\s\d*\s*~*
src_number        1
src_name          1
dst_number        2
dst_name          2
reason            1
origin            internal
```

This means that when the regular expression is satisfied:

- The first digits string is the source number - `.= \w*\s`**`\d*`**` to \w*\s\d*\s*~*`
- The second digits string is the destination number - `.= \w*\s\d* to \w*\s`**`\d*`**`\s*~*`
- The first word string is the source name - `.= `**`\w*`**`\s\d* to \w*\s\d*\s*~*`
- The second word string is the destination name - `.= \w*\s\d* to `**`\w*`**`\s\d*\s*~*`
- The reason code is at the end - `.= \w*\s\d* to \w*\s\d*\s*`**`~*`**
- The origin of the call is “internal”.

4 HOW IT WORKS

4.1 Rule Order

ADEPT attempts to match an input display string with a rule in the configuration file. The rules are compared to the display from the top rule to the bottom rule. Because of this, rules that contain the most specific information should be listed in the configuration file first. For instance, rules that have the most static text in them should be listed first, followed by rules that contain more wildcards. In this way, exact matches can occur before a more generic 'catch-all' rule is reached.

If a rule does not match, ADEPT attempts to match the next rule in the list. If no rules match, then the display parsing fails. For this reason, the last rule in the rule list should be a very generic rule that provides the best possibility of extracting the desired information from the display.

For instance, a rule such as:

```
rule \s*CALL FROM:\s*\d*
```

contains a static string 'CALL FROM:'. This rule should be located in the configuration file before a more generic rule such as:

```
rule \s*.*\s*\d*
```

because a match to the first rule would provide a much more reliable result.

4.2 Expression Evaluation

ADEPT attempts to match a display string with a rule by examining the rule's expressions in an explicitly prioritized order. The expressions are examined in the following order:

1) Literal Strings
2) Literal Non-Space Characters
3) Call Party Numbers '\d'
4) Call Party Reason '~'
5) Call Party Name '\w'
6) Literal Space Characters ' '
7) Non Digits '\D'
8) White-Space '\s'
9) Any '.'

The expressions are examined in this order so that the most specific (non-wildcard) expressions may be matched first. For example, if a rule is specified as:

```
rule .= \d* to \d*
```

and a display contains:

```
a= 334 to 234
```

the literal string 'to' is matched first, since this will provide a landmark for the remaining un-solved expressions.

As expressions are solved, ADEPT attempts to solve the remaining un-solved expressions by searching between the already-solved expressions. In the previous example, ADEPT would first locate the 'to' string, then the '=' literal character. It now has landmarks from which to search for the call party numbers. ADEPT would search for the first '\d' number expression between the location of the '=' literal character and the 'to' literal string.

4.3 Literal Strings

Literal strings are any strings of more than one literal character. A literal string could be, for example, ‘Call From’, or ‘IS CALLING’. Literal strings provide specific landmarks for locating the calling party information in the display.

4.4 Literal Non-Space Characters

Literal non-space characters are any literal characters that are not a space character. They could be, for example, ‘=’, ‘-‘, or ‘a’. Literal non-space characters provide specific landmarks for locating the call party information in the display.

4.5 Call Party Numbers

A call party number is a string of digit chars, and ‘*’, ‘#’, or ‘-‘, in the display string of the phone emulation.

4.6 Call Party Reason

A call party reason string must match one of the switch-specific strings in the ‘reason’ translation list. Therefore, call party reason strings are similar to literal strings, except that they may be one of a number of switch-specific strings. For instance, if a configuration file contained the following translation descriptors:

```
tran reason |FWD|fwd-all
tran reason |BFWD|busy
tran reason |NA|no-answer
```

then the reason strings ‘FWD’, ‘BFWD’, and ‘NA’ would be searched for in the display if the rule specifies a ‘~’ in the expression.

4.7 Call Party Name

A call party name is a string that begins with an alpha character, but can then contain alphanumeric characters, or space, comma, ‘-‘, or ‘_’.

NOTE: The ADEPT **d42_gtcallidex( )** function for PBX integration board does not support this Call Party Name.

4.8 Bounded or Un-Bounded

An expression can be bound or un-bound by specifying the \b meta-character. This can be used to specify if a literal string, a call party number, a call party name, or a call reason must be bound by delimiting characters (non alphanumeric, non-‘.’)

For instance, a rule such as:

```
rule .= \s*\w*\d*
src_number 1
```

would match a display:

a = Joe123 143

The result would return a call party number of 123. This may be incorrect if the ‘123’ is actually part of the call party name and the ‘143’ is the actual source number.

Changing the rule to:

```
rule .= \s*\b\w*\b\d*\b
src_number 1
```

would specify that the call party name and number must be bound by delimiting characters. Therefore, ADEPT would skip over the ‘123’ string since it is not bound at its start. The ‘143’ would match since it is bound by a space at both its beginning and end.

5 THE ADEPT FILE TESTER

When creating a new configuration file or modifying an existing file, the file tester application can be used to verify the new rule definitions as well as to test the backward-compatibility of the configuration file against known displays.

The input display file (*.in) contains captured displays in the following format:

[display 1, line 1]
[display 1, line 2]

[display 2, line 1]
[display 2, line 2]

The application uses the [ ] delimiters for each display row. If the display is followed by a tag ‘must=’, then the application compares the resulting ADEPT display parse with the data on the ‘must=’ line. This is useful for testing backward compatibility. For instance:

```
[a=   Eng. Lab 1  to  Eng. Lab 508    d]
must=[][Eng. Lab 1]->[508][Eng. Lab][no-answer][internal]
```

If the application result and the ‘must=’ text do not match, then the application exits with an error. The –d command line parameter can be used to determine why the parsing failed.

Format of must statement: **must=[a][b]->[c][d][e][f]**

where:

a = source number

b = source name

c = destination number

d = destination name

e = reason code

f = origin code

notes:

1. Any of a, b, c, d, e, or f can be null.

```
rem forward
[HUBERT>FRANZ O  ]
[                ]
must=[][HUBERT]->[][FRANZ O][no-answer][internal]

rem direct
[24 calling      ]
[                ]
must=[24][]->[][][direct][internal]

rem fowrard
[HUBERT>233      ]
[                ]
must=[][HUBERT]->[233][][no-answer][internal]
```

The file tester application is located in the <installation folder>\Dialogic\Samples\ directory.

The file tester application is invoked in the following manner:

```
adepttest <.adt file location and prefix> [-n] [-d]
```

The suffix '.adt' is appended to the config file prefix parameter by the application. The input display file is assumed to have the same prefix as the config file but has an '.in' suffix.

Valid .adt file prefixes are given in the following table:

.ADT Prefix	**Supported PBX**
Mitel	Mitel
Ntm1	Nortel M1
Ntns	Nortel Norstar
Lucent	Lucent
Magix	Lucent Magics
Optiset	Optiset, Rolm

If the –n parameter is specified, a config file is not used. In its place, the application uses the default rules for the specified PBX .

NOTE: PBX integration boards do not support this argument because there is no embedded default rule supported by PBX Integration boards.

If the –d parameter is specified, detailed trace information is printed to the console during application execution.

NOTE: For debugging newly written rules, the -d parameter should always be used.

NOTE: All files with the extension .in must be located in the same directory as the AdeptTest executable.

SAMPLE RESULTS: C:\Program Files\Dialogic\SAMPLES>AdeptTest ..\cfg\rolm -d

SAMPLE RESULTS

Translations:

tr[4]=default to internal

tr[3]=default to direct

tr[3]=BFWD to busy

tr[3]=FWD to fwd-all

tr[3]=NAFWD to no-answer

Regexp: FROM\s\(\d(3,3)\)\d(3,3)\-\d(4,4)\n\b~*\b\d*\s.*

FROM (4,4)

\s (1,1)

((1,1)

\d (3,3)

) (1,1)

\d (3,3)

- (1,1)

\d (4,4)

\n (1,1)

\b (1,1)

~ (0,200,Opt)

\b (1,1)

\d (0,200,Opt)

\s (1,1)

. (0,200,Opt)

null (1,1)

src num : 1 2 3

dst num: 4

reason: 1

origin: external

==========

FROM (214)632-6052

FWD 2007 UNITY TEST 3

==========

ADEPT_parse_display()

[FROM (214)632-6052

FWD 2007 UNITY TEST 3]

Rule->[FROM\w*\n\b~*\b\d*]

Exp->[FROM,0<->66] failed

Rule->[FROM\s\(\d(3,3)\)\d(3,3)\-\d(4,4)\n\b~*\b\d*\s.*]

[FROM (214)632-6052

FWD 2007 UNITY TEST 3]

[FROM\s\(\d(3,3)\)\d(3,3)\-\d(4,4)\n\b~*\b\d*\s.*]

Result: [2146326052][]->[2007][][fwd-all][external]

FROM (0,4) = [FROM]

\s (4,1) = [ ]

((5,1) = [(]

\d (6,3) = [214]

) **(9,1) = [)]**

\d **(10,3) = [632]**

- **(13,1) = [-]**

\d **(14,4) = [6052]**

\n **(25,1) = [**

]

\b **(0,-1) = []**

~ **(26,3) = [FWD]**

\b **(0,-1) = []**

\d **(30,4) = [2007]**

\s **(34,1) = [ ]**

. **(35,17) = [UNITY TEST 3]**

Result: [2146326052][]->[2007][][fwd-all][external]

TEST COMPLETE

6 EXAMPLE CONFIGURATION FILES

6.1 Lucent

```
### Lucent
################################################
#
#
tran reason |d|no-answer
tran reason |c|fwd-all
tran reason |s|fwd-all
tran reason |f|fwd-all
tran reason |b|busy
tran reason |B|busy
tran reason |h|no-answer
tran reason |default|direct
tran origin |default|internal
# ignore name fields of 'EXT', since switch uses that when there
is no name field.
tran src_name |EXT|
tran dst_name |EXT|
#
# a = Eng. Lab to Eng. Lab 505  d
rule .=\s*\w*\b\d*(3,12)\b.*\bto\b\w*\b\d*(3,12)\b.*\b~*\b
src_number 1
src_name 1
dst_number 2
dst_name 2
reason 1

# a= CALL FROM 716-689-6700
rule .=\s*\bCALL FROM\b\s*\d*
src_number 1
tran origin external

# a = Eng. Lab 504
rule .=\s*\w*\s*\b\d*(3,12)\b.*
src_number 1
src_name 1
```

6.2 Mitel

```
### Adept.adt file - MITEL
##########################################
#
#
# Reason translations
tran reason |BUSY|busy
tran reason |NO ANS|no-answer
tran reason |DO NOT DISTURB|no-answer
tran reason |ALWAYS|fwd-all
tran reason |FWD|fwd-all
tran reason |FORWARD|fwd-all
tran reason |FORWARDED|fwd-all
tran reason |default|direct
tran origin |default|internal
# time in first row, ignore these
rule .*\d+:\d+.*
reason time

# TRUNK 102 IS CALLING  FWD FROM JOEY ALWAYS
rule \bTRUNK\b\d*\b\D*\bFROM\b\d*\b\w*\b~*\b
src_name Outside_Call
dst_number 2
dst_name 2
reason 1
origin external

# JOE IS CALLING  FWD FROM JOEY ALWAYS
rule \b\d*\b\w*\bIS\b\D*\bFROM\b\d*\b\w*\b~*\b
src_number 1
src_name 1
dst_number 2
dst_name 2
reason 1

# 455 IS CALLING
# JOE IS CALLING blah blah blah
# JOE IS RINGING YOU BACK
rule \b\d*\b\w*\bIS\b\D*
src_number 1
src_name 1

# JOE     FORWARDED FROM 732
rule \b\d*\b\w*\D*\bFORWARDED FROM\b\d*\b\w*\b~*\b
src_number 1
src_name 1
dst_number 2
dst_name 2
reason 1

# JOE     FWD FROM 732
rule \b\d*\b\w*\D*\bFWD FROM\b\d*\b\w*\b~*\b
src_number 1
src_name 1
dst_number 2
dst_name 2
reason 1
```

6.3 Nortel M1

```
#### NT M-1
#
# Reason translations
tran reason |AFWD|fwd-all
tran reason |B|busy
tran reason |N|no-answer
tran reason |A|fwd-all
tran reason |F|fwd-all
tran reason |default|direct
tran origin |default|internal
# JOE 501 BOB 203 "F "
rule \s*\b\w*\b\d*\b\s*\b\w*\b\d*\b\s*\b~*\b
src_number 1
src_name 1
dst_number 2
dst_name 2
reason 1
```