Willkommen zum Informix Newsletter

Liebe Leserinnen und Leser,

der Winter hatte uns im Januar fest im Griff, und es war nach Jahren endlich einmal wieder möglich auch am Bodensee die Schlittschuhe zu nutzen.
Die immer stärker werdenden Sonnenstrahlen lassen jedoch das Frühjahr schon erahnen.

Der Inhalt dieser Ausgabe des Newsletters wurde komplett durch Anfragen unserer Kunden bestimmt. Die eigentlich für diese Ausgabe vorgesehenen Themen haben wir auf das zweite Quartal 2026 verschoben.

Viel Spass mit dem aktuellen Newsletter !
Ihr TechTeam

Inhaltsverzeichnis

TechTipp: Ergänzung "InPlaceAlter" Task

Im INFORMIX Newsletter 2025_Q4 hatten wir einen Task erstellt, der "Pending Alter Tables" bereinigt.
Andreas Seifert der Cursor Expert Solutions GmbH hat uns darauf aufmerksam gemacht, dass es diese Funktion (allerdings ohne PDQ und ohne die Option "parallel") bereits als Task gibt. Die Funktionalität versteckt sich im Task
auto_crsd (Automatic Compress/Repack/Shrink and Defrag)
Vielen Dank für diese Ergänzung. Sie zeigt, dass INFORMIX viel mehr zu bieten hat, als man im Alltag nutzt und kennt. Selbst wir, die doch recht nahe an der Entwicklung stehen, lernen fast täglich dazu. Uns gehen daher auch die Themen nicht aus und damit stellen wir gleich den mitgelieferten Task und seine Nutzung vor.

Vielen Dank Andreas !

TechTipp: Task "auto_crsd"

Der Task "auto_crsd" kann dazu genutzt werden, Tabellen zu komprimieren, zu defragmentieren, die Daten mittels "repack" zusammenzuführen. Zudem können anschliessend ungenutzte Extents von Tabellen wieder dem DBSpace zurückzugeben werden.
Eine weitere Funktion ist, Tabellen bereinigen, auf denen ein "Pending Alter Table" vorhanden ist.
Die Aktivierung des Tasks erfolgt über zwei Stufen.
Zuerst wird der Task selbst in der Datenbank "sysadmin" aktiviert mittels:

```
update ph_task
set tk_enable = 't'
where tk_name = 'auto_crsd'
```

Dies allein bewirkt jedoch nur, dass der Task aufgerufen wird. Die eigentliche Arbeit wird über die Parameter in der Tabelle ph_threshold gesteuert:

AUTOCOMPRESS_ENABLED	- Für die Komprimierung von Tabellen
AUTOREPACK_ENABLED	- Für das Repack der Daten innerhalb der Extents
AUTOSHRINK_ENABLED	- Um ungenutzte Extents wieder frei zu geben
AUTODEFRAG_ENABLED	- Um Extents einer Tabelle zusammenzuführen
REMOVE_IPA_ENABLED	- Um ein Pending Alter bei Tabellen zu bereinigen

Je nachdem, welche dieser Parameter gesetzt wurden, werden die automatischen Arbeiten durch den Task ausgeführt.

TechTipp: ONCONFIG - NET_IO_TIMEOUT_ALARM

In der Konfigurationsdatei $ONCONFIG lässt sich ein Alarm aktivieren, der die Antwortzeiten von verteilten Datenabfragen überwacht.
Der Parameter NET_IO_TIMEOUT_ALARM existiert schon sehr lange (auch in Version 12.10), war jedoch bisher nicht in der Konfigurationsdatei onconfig.std zu finden.

Der Parameter kann folgende Werte annehmen:

- 0 = Disabled (Default)
- 1 = Enabled für Enterprise Replication Aktivitäten
- 2 = Enabled für Distributed Queries
- 4 = Enabled für HDR Aktivitäten
- 8 = Enabled für SMX Aktivitäten
- 16 = Enabled für Aktionen anderer Komponenten

Die Werte werden additiv gesetzt, so dass z.B. der Wert 7 die Aktivierung für Enterprise Replication (1) + Verteilte Abfragen (2) + HDR Replikation (4) darstellt.

Der Wert kann dynamisch mittels onmode -wf gesetzt werden und ist sofort wirksam.

Beispiel:

```
onmode -wf NET_IO_TIMEOUT_ALARM=31,300
Value for NET_IO_TIMEOUT_ALARM (31,300) was saved in config file.
Value of NET_IO_TIMEOUT_ALARM has been changed to 31,300.

onstat -g cfg full NET_IO_TIMEOUT_ALARM
Configuration Parameter Info

id   name                      type    maxlen   units   rsvd  tunable
310  NET_IO_TIMEOUT_ALARM      STRUCT  513                    *

     default : 0,1800
     onconfig: 31,300
     current : 31,300

     Description:
     Use the NET_IO_TIMEOUT_ALARM configuration parameter to control
     whether to be notified if network write operations have been blocked
     for 30 minutes or more.
```

Wird erkannt, dass eine verteilte Abfrage für länger als den definierten Zeitraum blockiert wurde, so wird mittels Alarmprogram der Alarm 82 ausgegeben.
Um diesen Alarm auszuwerten, muss im $ALARMPROGRAM eine Aktion für Alarm 82 hinzugefügt werden. Diese ist im Default Alarmprogramm nicht vorhanden.

Hinweis: Die Beschreibung spricht von 30 Minuten, was jedoch nur dem Default Wert entspricht und wie in unserem Beispiel geändert werden kann.

TechTipp: UTL_File Package (Datablade)

Datablades bzw. Packages sind Optionen, die Datentypen und Funktionen der Informix Instanz erweitern. Im Verzeichnis $INFORMIXDIR/extend finden sich einige dieser Datablades.
Eine recht unbekannte, aber sehr nützliche Erweiterung ist das UTL_FILE Package.
Dies umfasst die Funktionen:

- function utl_file_fopen # Öffnen einer Datei
- procedure utl_file_is_open # Abfrage, ob eine Datei geöffnet ist
- procedure utl_file_put # Einfügen einer Zeichenkette
- procedure utl_file_put_line # Einfügen einer Zeile in eine Datei
- procedure utl_file_new_line # Einfügen von "Newline" in die Datei
- procedure utl_file_fclose # Schliessen der Datei
- procedure utl_file_get_line # Lesen aus einer Datei

Damit das Package genutzt werden kann, muss die Funktionalität in der Datenbank einmalig
registriert werden:

```
EXECUTE FUNCTION sysbldprepare('excompat.*','create');
```

Zur Qualität der Dokumentation dieser Funktionalität äussert sich die Redaktion nicht.
Wir empfehlen als Anleitung diesen Newsletter, falls die Funktion verwendet werden soll.

Mit Test, der Hilfe der INFORMIX COMMUNITY, dem HCL Support und HCL Development ist es uns gelungen funktionsfähige Beispiele zu erstellen, die wir hier vorstellen wollen.

Die Anforderung kam von einem Kunden, der bisher Oracle im Einsatz hatte, und bei der Migration diese Protokollierung erhalten wollte.
Da Informix diese Funktionalität in einem Set zur Unterstützung der Migration zur Verfügung stellt, konnten wir die Funktionalität nahezu analog abbilden.

TechTipp: UTL_File - utl_file_fopen()

Um eine Datei zum Schreiben zu öffnen, wird die Funktion utl_file_fopen() genutzt.
Als Argumente werden übergeben:

- Das Verzeichnis, in dem die Datei angelegt werden soll
- Der Name der Datei
- Ein Flag mit den Optionen:
  - w - für "write" um eine neue Datei zu erstellen
  - a - für "append" um eine bestehende Datei zu öffnen
  - r - für "read" um eine bestehende Datei zum Lesen zu öffnen
- (optional) Die Zeilenlänge in Byte. Default ist 1024.

Der Rückgabewert ist vom Datentyp **utl_file_file_type** und verweist auf die geöffnete Datei.

Beispiel:

```
select UTL_FILE_FOPEN("/tmp","IFX_Text_Output.txt","w",3000)
from systables
where tabid = 1;
```

In einer Prozedur könnte dies folgendermassen aussehen:

```
create or replace procedure health_statistik()
define logfile utl_file_file_type;
define is_open boolean;

select UTL_FILE_FOPEN("/tmp","IFX_Text_Output.txt","w")
     into logfile
from systables where tabid = 1;
...
```

TechTipp: UTL_File - utl_file_is_open()

Mit dieser Funktion lässt sich abfragen, ob eine Datei bereits geöffnet wurde. Der Rückgabewert ist ein Boolean ('t' für True und 'f' für False).
Dies kann z.B. im folgenden Kontext verwendet werden:

```
select UTL_FILE_IS_OPEN(logfile)
     into is_open
from systables where tabid = 1;
if is_open = 'f'
     then
          select UTL_FILE_FOPEN("/tmp","IFX_Text_Output.txt","w")
          into logfile
          from systables where tabid = 1;
end if;
...
```

TechTipp: UTL_File - utl_file_put() / utl_file_put_line()

In eine geöffnete Datei können Zeichenketten oder ganze Zeilen eingefügt werden.
Hierzu dient die Prozedur utl_file_put() bzw. utl_file_put_line().
Der Unterschied dieser Prozeduren besteht darin, dass put_line ein "newline" am Ende mit einfügt und somit die Zeile abschliesst.

Beispiele:

```
call UTL_FILE_PUT(logfile,"Analyse der Informix Instanz XX");
```

Inhalt der Datei:

```
Analyse der Informix Instanz XX
```

Der Text kann auch komplex mittels Query erstellt werden:

```
call UTL_FILE_PUT_LINE(UTL_FILE_FOPEN('$MAIL_DIR','$MAIL_FILE','a'),
      "INSTANZ: "||(
            select trim(cf_effective)
            from syscfgtab
            where cf_name = 'DBSERVERNAME'
            )||" - Uptime: "||(
            select (current - dbinfo('UTC_TO_DATETIME',
                 sh_boottime))||" - Last reset: "||
                 dbinfo('UTC_TO_DATETIME',sh_pfclrtime)
            from check_shmvals
            )
);
```

Inhalt der Datei:

```
DBSERVERNAME: test1 - Uptime: 1 00:25:45 - Last reset: 2026-02-09 01:22:04
```

TechTipp: UTL_File - utl_file_new_line()

Mit utl_file_new_line() lässt sich eine Zeile abschliessen, in die Werte mittels utl_file_put() geschrieben wurden.
Der Aufruf von utl_file_new_line() kann auch dazu verwendet werden, in die Datei zur besseren Übersicht Leerzeilen einzufügen.

Beispiel:

```
call UTL_FILE_NEW_LINE(UTL_FILE_FOPEN('$MAIL_DIR','$MAIL_FILE','a'));
```

TechTipp: UTL_File - utl_file_fclose()

Der Aufruf von utl_file_fclose() schliesst die Datei. Damit sind vor einem erneuten Öffnen keine Einträge mehr möglich mittels PUT bzw. PUT_LINE.

Anwendungsbeispiel:

Mit Hilfe dieser Funktionalität lässt sich ein Protokoll der Verarbeitung eines Statements erstellen. Die Ausgabe der aktuellen Zeit der Verarbeitungsschritte (current) und der Anzahl der verarbeiteten Datensätze je SQL Statement (mittels dbinfo() Funktion) ermöglichen die dynamische Erstellung einer Logdatei.

```
create procedure if not exists write_mail_file(
     mail_dir varchar(255),
     mail_file varchar(255),
     mail_flag char(1),
     txt lvarchar(255))
define xx int;
define x_file utl_file_file_type;

-- Open the file for write (mail_flag w=new file, a=existing file)
let x_file = UTL_FILE_FOPEN(mail_dir,mail_file,mail_flag,3000);

-- Write the line from txt into the file
call UTL_FILE_PUT_LINE(x_file,txt);

-- Close the file to delete the open file pointer
call UTL_FILE_FCLOSE(x_file);
end procedure;


execute procedure write_mail_file('/tmp','mail_test','w','Erster Test um '||cur-
rent year to second);
execute procedure write_mail_file('/tmp','mail_test','a','');
execute procedure write_mail_file('/tmp','mail_test','a','Zweiter Test dann
'||current year to second);
```

Sehen wir uns den Inhalt der Logdatei an:

```
cat /tmp/mail_test
Erster Test um 2026-02-18 06:26:31

Zweiter Test dann 2026-02-18 06:26:31
```

Achtung:
Wird die Datei nicht geschlossen, sondern immer wieder geöffnet, sammeln sich offene File pointer an, die Ressourcen benötigen.
Im folgenden Beispiel werden immer wieder Dateien geöffnet, aber nie geschlossen.

Beispiel:

```
call UTL_FILE_PUT(UTL_FILE_FOPEN("/tmp","IFX_Log.txt","w",2000),"First line of
the test in a file and START at: "||(select current from systables where tabid =
1));
call UTL_FILE_PUT(UTL_FILE_FOPEN("/tmp","IFX_Log.txt","a"),"... Second entry af-
ter PUT");
call UTL_FILE_NEW_LINE(UTL_FILE_FOPEN("/tmp","IFX_Log.txt","a"));
call UTL_FILE_PUT_LINE(UTL_FILE_FOPEN("/tmp","IFX_Log.txt","a"),"... one more
line after NEW_LINE using PUT_LINE");
call UTL_FILE_PUT_LINE(UTL_FILE_FOPEN("/tmp","IFX_Log.txt","a"),"... and the
next one with PUT_LINE at: "||(select current from systables where tabid = 1));
call UTL_FILE_NEW_LINE(UTL_FILE_FOPEN("/tmp","IFX_Log.txt","a"));
call UTL_FILE_NEW_LINE(UTL_FILE_FOPEN("/tmp","IFX_Log.txt","a"));
call UTL_FILE_NEW_LINE(UTL_FILE_FOPEN("/tmp","IFX_Log.txt","a"));
call UTL_FILE_PUT_LINE(UTL_FILE_FOPEN("/tmp","IFX_Log.txt","a"),"... and the
last one after 3x NEW_LINE at: "||(select current from systables where tabid =
1));
call UTL_FILE_FCLOSE(UTL_FILE_FOPEN("/tmp","IFX_Log.txt","a"));
```

onstat -g iof zeigt diese:

```
13  /tmp/IFX_Log.txt

14  /tmp/IFX_Log.txt

15  /tmp/IFX_Log.txt

16  /tmp/IFX_Log.txt

17  /tmp/IFX_Log.txt

18  /tmp/IFX_Log.txt

19  /tmp/IFX_Log.txt

20  /tmp/IFX_Log.txt

21  /tmp/IFX_Log.txt
```

Der abschliessende Aufruf von UTL_FILE_FCLOSE auf den zehnten File Pointer schliesst nur diese, aber nicht alle Anderen.

TechTipp: UTL_File - utl_file_GET_LINE()

Mittels der Prozedur "utl_file_get_line()" kann eine Datei zeilenweise gelesen werden. Die Funktionalität get_lines() wurde nicht implementiert, lässt sich jedoch mittels While-Schleife einfach realisieren.

Das folgende Beispiel zeigt, wie wir die im vorigen Beispiel geschriebene Datei wieder lesen können:

```
create or replace procedure read_file()
returning lvarchar(32672);

define logfile utl_file_file_type;
--define txt lvarchar(32672); -- Siehe Hinweis unten --
define txt char(32672);
define x boolean;

on exception in (-937)
end exception;

let txt='';
select UTL_FILE_FOPEN("/tmp","IFX_Log.txt","r",2000)
     into logfile
from systables where tabid = 1;

select UTL_FILE_IS_OPEN(logfile)
     into x
from systables where tabid = 1;

while txt is not null and x='t'
     call UTL_FILE_GET_LINE(logfile,txt);
     return trim(txt) with resume;
end while

call UTL_FILE_FCLOSE(logfile);

end procedure;
```

Das Ergebnis:

```
execute procedure read_file();

(expression)  First line of the test in a file and START at: 2026-01-09
01:21:03.455... Second entry after PUT
(expression)  ... one more line after NEW_LINE using PUT_LINE
(expression)  ... and the next one with PUT_LINE at: 2026-01-09 01:21:03.482
(expression)
(expression)
(expression)
(expression)  ... and the last one after 3x NEW_LINE at: 2026-01-09 01:21:03.513

7 row(s) retrieved.
```

Hinweis:
In der Dokumentation steht, dass jede Zeile als VARCHAR(32672) zurückgegeben wird. Da der Datentyp VARCHAR bei INFORMIX eine maximale Länge von 255 nicht übersteigen kann, ging unser Blick zur Definition der Prozedur.

```
dbschema -d health_check -f utl_file_get_line

create procedure "informix".utl_file_get_line (utl_file_file_type,out lvarchar)
external name "$INFORMIXDIR/extend/excompat.1.2/excompat.bld(utl_file_get_line)"
language c;
```

Die Verwendung von LVARCHAR führte jedoch ebenfalls zu einem Fehler.

An dieser Stelle bedanken wir uns herzlich beim HCL Development, das uns den Tipp gegeben hat, dass wir CHAR(32672) verwenden müssen, damit die Funktion korrekt läuft.

Versionshinweis: 14.10.FC13 verfügbar

Seit dem 26.12.2025 ist die Version 14.10.FC13 verfügbar und kann über FixCentral mittels Download heruntergeladen werden.

https://www.ibm.com/support/fixcentral

Versionshinweis: 15.0.1.0 verfügbar

Seit dem 26.12.2025 ist die Version 15.0.1.0 verfügbar und kann über FixCentral mittels Download heruntergeladen werden.
Diese Version beinhaltet einige neue Features, die wir im Newsletter 2026_Q2 ausführlich behandeln wollen.
Ein ganz wichtiges Feature ist die Option STRING_TRUNCATE_ERROR, die überwacht, ob die eingefügte Zeichenkette in die entsprechende Spalte passt, ohne dass Zeichen abgeschnitten werden müssen.

https://www.ibm.com/support/fixcentral

Nutzung des INFORMIX Newsletters

Die hier veröffentlichten Tipps&Tricks erheben keinen Anspruch auf Vollständigkeit.
Die IUG hat sich dankenswerterweise dazu bereit erklärt, den INFORMIX Newsletter auf ihren Webseiten zu veröffentlichen.
Da uns weder Tippfehler noch Irrtümer fremd sind, bitten wir hier um Nachsicht, falls sich bei der Recherche einmal etwas eingeschlichen hat, was nicht wie beschrieben funktioniert.
Die gefundenen Tippfehler dürfen zudem behalten und nach Belieben weiterverwendet werden.
Eine Weiterverbreitung in eigenem Namen (mit Nennung der Quelle) oder eine Bereitstellung auf der eigenen HomePage ist ausdrücklich erlaubt. Alle hier veröffentlichen Scripts stehen uneingeschränkt zur weiteren Verwendung zur Verfügung.
Wir würden uns über eine Information freuen, wann und wo unsere Inhalte weiterverbreitet werden.

Die Autoren dieser Ausgabe

Andreas Legner **INFORMIX Development**
HCL Software

Martin Fuerderer **Database Development**
HCL Software

Gerd Kaluzinski **Delivery Consultant**
IBM Expert Labs
gerd.kaluzinski@de.ibm.com +49-175-228-1983

Herzlichen Dank an die vielen Helfer im Hintergrund.

Nicht zu vergessen der Dank an die Informix User Group, ohne die es den INFORMIX Newsletters heute nicht mehr geben würde, und die dankenswerterweise die Verteilung übernimmt.

Foto Nachweis:
Goldplatz Lindau im Winterschlaf (Gerd Kaluzinski)