02.12.2019

# Test auf TEP: Success

Beim Test auf TEP konnte im Log ein Fehler gefunden werden und der Job lief weiter. Das "Update" der Klassen geschah ohne Deployment durch Austausch der .class Dateien ComplexResolveService.class und LcmRestSyncCustomerItemsBaseService.class im .jar-File ch.local.crm.server\_[Version.Build].jar sowie Austausch von **AbstractAddressRedirectServerResource.class** in einem Pfad von bsicrmserver.war.

**ComplexResolveService** in

\ch\local\crm\server\complex

**LcmRestSyncCustomerItemsBaseService** in

\ch\local\crm\server\interfaces\restsync\customer

Die Klasse **AbstractAddressRedirectServerResource.class** befindet sich im Pfad

delivery\deployment\webapps\WEB-INF\eclipse\plugins\ch.local.crm.server.rest\_[Version.Build]\ch\local\crm\server\rest\address

Auf TEP lief der Job durch ohne Abbruch trotz eines Fehlers mit CUSTOMER\_NR = 4089672869, mit der keine Complex\_NO gefunden werden kann.

Die beiden folgenden Abfragen liefern kein Ergebnis:

**SELECT** LE.X\_COMPLEX\_NO **FROM** BSI\_LEGAL\_ENTITY\_VIEW le **WHERE** LE.COMPANY\_NR = 4089672869

;

**SELECT** CMP.X\_COMPLEX\_NO **FROM** BSI\_COMPANY cmp **WHERE** CMP.COMPANY\_NR = 4089672869

;

Der Wert 4089672869 ist jedoch als Fremdschlüssel vorhanden in den folgenden Abfragen/Tabellen und wird vermultich aus diesem Zusammenhang eingeschleust.

**SELECT** \* **FROM** BSI\_X\_REST\_SYNC\_ITEM rsi **WHERE** RSI.ROOT\_ITEM\_KEY0\_NR = '4089672869'

;

**SELECT** \* **FROM** BSI\_RECEIPT\_CUST\_BALANCE\_MVIEW rcb **WHERE** RCB.COMPANY\_NR = '4089672869'

;

**SELECT** \* **FROM** BSI\_X\_COMPLEX\_PUBLISH xcp **WHERE** xcp.JOIN\_NR = '4089672869'

;

27.11.2019

# Vermeidung von Job-Abbruch bei unbekannter COMPANY\_NR

Nachdem die LOG-Ausgabe erweitert wurde, konnte der Fehler identifiziert werden: Der Job bricht ab bei einer COMPANY\_NR, zu welcher eine COMPLEX\_NO existiert. Der LOG-Eintrag in diesem Fall ist:

Nov 27, 2019 6:39:01 AM ch.local.crm.server.complex.ComplexResolveService.resolveComplexNo(ComplexResolveService.java:64)

SEVERE: Could not resolve COMPLEX\_NO for CompanyKey[8111262126], Query: SELECT le.complexNo FROM BsiXLegalEntityView AS le WHERE le.key = :b\_\_0

null

Die fehlerhafte COMPANY\_NR ist identifiziert – man könnte sie aus den Quelldaten herausnehmen.

Alternative: Das Exception THROW und damit der Job-Abbruch aus diesem Grund wird vermieden.

Modifikationen in Funktion

## Modifikationen in ComplexResolveService

Nachdem COMPLEX\_NO zu COMPANY\_NR ermittelt wurde, wird keine Exception mehr ausgelöst, sondern NULL zurückgegeben. Der Empfänger muss COMPLEX\_NO == NULL berücksichtigen.

**public** String resolveComplexNo(ILegalEntityKey key) **throws** ComplexNoResolvingException {

BsiXLegalEntityView\_aliased le = **new** BsiXLegalEntityView\_aliased("le");

LcmBinds binds = **new** LcmBinds();

String result = "";

**final** CharSequence qrySeq = QL.*select*(le.complexNo).from(le.TABLE).where(*eq*(le.key, binds.setDomainKey(key)));

**final** HQuery<String> query = JPA.*createSuperUserSQL92Query*(String.**class**, qrySeq);

*LOG*.error("(START - ComplexResolveService.resolveComplexNo): Query: "

+ query.getQueryString()

+ " with COMPANY\_NR = " + key.toString());

**try** {

**final** String complexNo = binds.applyQueryParameters(query).uniqueResult();

result = complexNo;

} **catch** (Exception ex) {

*LOG*.error("Could not resolve COMPLEX\_NO for ComplexResolveService.resolveComplexNo " + key.toString() + ", Query: " + query.getQueryString() + "\n" + ex.getMessage());

**throw** **new** ComplexNoResolvingException("Could not resolve COMPLEX\_NO for " + key.toString() + ", Query: " + query.getQueryString() + "\n" + ex.getMessage());

}

**return** result;

}

## Modifikationen in Klassen, welche ComplexResolveService verwenden

Die Methode ComplexResolveService.resolveComplexNo wird von 2 Klassen verwendet:

* LcmRestSyncCustomerItemsBaseService
* AbstractAddressRedirectServerResource

In beiden Klassen ist NULL als Rückgabe aus der Funktion nicht vorgesehen. Die Klasse LcmRestSyncCustomerItemsBaseService wird im Zusammenhang mit dem Job SF-Synch-Items aufgerufen.

### Modifikationen in LcmRestSyncCustomerItemsBaseService

In dieser Klasse wird eine List<CustomerItem> erstellt. Die erste Funktion, loadCustomerItems, ruft dazu eine Hilfsfunktion, loadCustomerItem auf, welche ein CustomerItem zurückgibt.

loadCustomerItem ruft ComplexResolveService.resolveComplexNo auf und muss für eine Rückgabe von NULL modifiziert werden. Hier wird eine falsche CUSTOMER\_NR im Log ausgegeben mit dem Hinweis, dass der Job fortgesetzt wird.

**private** CustomerItem loadCustomerItem(LcmCustomerSyncItem syncItem) **throws** ProcessingException {

CustomerItem customerItem = **new** CustomerItem();

**if** (isCompany(syncItem)) {

**try** {

SERVICES.*getService*(ISchedulerBaseService.**class**).logJobInfoMessage(**null**, "Try to load LcmCustomerSyncItem " + syncItem.toString());

**final** String complex\_no = SERVICES.*getService*(IComplexResolveService.**class**).resolveComplexNo(syncItem.getCustomerKey());

**if** (complex\_no != **null**) {

customerItem.setComplexNo(complex\_no);

customerItem.setName(getCompanyName(syncItem.getCustomerKey()));

customerItem.setMergedComplexNoList(getMergedIds(customerItem.getComplexNo()));

setParentCompanyId(syncItem, customerItem);

setExternalCustomers(syncItem, customerItem);

} **else** {

*LOG*.error("ERROR (LcmRestSyncCustomerItemsBaseService.loadCustomerItem) "

+ "No COMPLEX\_NO found for CUSTOMER\_NR = " + syncItem.toString()

+ " \*\*\* CONTINUE WITH NEXT COMPANY\_NR \*\*\*");

customerItem = **null**;

}

} **catch** (ProcessingException ex) {

**\*\*\***

}

} **else** {

**throw** **new** ProcessingException("Sync service supports only companies at the moment and not persons " + syncItem.getCustomerKey());

}

**return** customerItem;

}

### Modifikationen in AbstractAddressRedirectServerResource

complex\_no wird vor der Verwendung im return Statement ermittelt, so dass auf eine Rückgabe von NULL reagiert werden kann.

**protected** ICustomerAddressItemReference execCreateReference() **throws** JsonException {

ILcmAddressBean<?> addressBean;

**try** {

addressBean = SERVICES.*getService*(ILcmAddressBaseService.**class**).loadActiveAddressBeanWithDuplicates(getAddressNo());

} **catch** (ProcessingException e) {

**throw** **new** JsonException("Error loading address", e);

}

**if** (addressBean == **null**) {

**throw** **new** AddressNoNotFoundExeption(getAddressNo());

}

**final** ILegalEntityKey legalEntityKey = (ILegalEntityKey) addressBean.getSelf().getItemKey();

**try** {

**final** String complex\_no = SERVICES.*getService*(IComplexResolveService.**class**).resolveComplexNo(legalEntityKey);

**if** (complex\_no != **null**) {

**if** (addressBean **instanceof** LcmPhysicalAddressBean) {

**return** CustomerPhysicalAddressJsonItemReference.*create*(

AbstractCustomerJsonItemReference.*create*(complex\_no, legalEntityKey),

addressBean.getAddressNo(), addressBean.getSelf().getAddressKey());

}

**else** {

**return** CustomerElectronicAddressJsonItemReference.*create*(

AbstractCustomerJsonItemReference.*create*(complex\_no, legalEntityKey),

addressBean.getAddressNo(), addressBean.getSelf().getAddressKey());

}

} **else** {

**throw** **new** AddressNoNotFoundExeption(getAddressNo());

}

} **catch** (ComplexNoResolvingException e) {

**throw** **new** AddressNoNotFoundExeption(getAddressNo());

} **catch** (ProcessingException ex) {

**throw** **new** AddressNoNotFoundExeption(getAddressNo());

}

}

07.2019

|  |  |
| --- | --- |
| Klasse | Methode |
| LCMRestSyncItemsJob | prepareSyncItems , insertOrUpdateItems |
| LCMRestSyncItemsBaseService |  |
| SchedulerBaseService.class |  |

## Abfragen

-- LcmRestSyncItemsBaseService

**SELECT** \* **FROM** BSI\_X\_EXT\_PRODUCT;

-- prepareSyncItems

**SELECT** \* **FROM** BSI\_X\_REST\_SYNC\_ITEM

**WHERE** ROOT\_ITEM\_TYPE\_ID **IN** (318594, 318596)

**ORDER** **BY** ITEM\_KEY0\_NR, ITEM\_TYPE\_ID, ROOT\_ITEM\_KEY0\_NR, ROOT\_ITEM\_TYPE\_ID;

**SELECT** \* **FROM** bsicrm.BSI\_X\_REST\_SYNC\_ITEM

**WHERE** ITEM\_TYPE\_ID = 318594 **AND** ROOT\_ITEM\_TYPE\_ID = 318594

**AND** **NOT** **EXISTS** (**SELECT** 0 **FROM** bsicrm.BSI\_COMPANY xcm **WHERE** XCM.COMPANY\_NR = ROOT\_ITEM\_KEY0\_NR);

01.07.2019 Log-Datei Auswertung

Die ersten Zeilen enthalten die letzten Ausrufe vor Abbruch des Import-Jobs:

01.07.2019 04:30 Customer Sync - Start

01.07.2019 05:08 null

at ch.local.crm.server.complex.ComplexResolveService.resolveComplexNo(ComplexResolveService.java:53)

at ch.local.crm.server.interfaces.restsync.customer.LcmRestSyncCustomerItemsBaseService.loadCustomerItem(LcmRestSyncCustomerItemsBaseService.java:54)

at ch.local.crm.server.interfaces.restsync.customer.LcmRestSyncCustomerItemsBaseService.loadCustomerItems(LcmRestSyncCustomerItemsBaseService.java:46)

at ch.local.crm.server.interfaces.restsync.LcmRestSyncItemsBaseService$LcmCustomerSyncItemTransformer.transformToJsonObject(LcmRestSyncItemsBaseService.java:615)

Alle Methoden enthalten bis auf «throws Exception» keine Fehlerbehandlung. Die Exceptions werden dazu «von Hand» in einem if – else Block geworfen, wenn das erwartete Ergebnis nicht erhalten wurde.

### Refactoring

In einem ersten Wurf wurde eine try – catch Fehlerbehandlung hinzugefügt, welche eine aussagekräftige Fehlermeldung enthält.

Weiterhin wird der komplette Import abgebrochen, wenn sich z.B. eine Complex\_No nicht ermitteln liess. Der Import sollte an dieser Stelle nicht abbrechen, sondern solche Objekte qualifiziert in einer Tabelle speichern.

## 02.07.2019: Analyse der Fehler in Logfiles

Das vollständige Log befindet sich auf cl01s004.ltvintra.ltv.ch (Remote-Desktop) im Verzeichnis D:\Transfer.

**Logfile:** 201907011700\_PROD\_BSICRM\_lcmserver-0.log

Anzahl Zeilen: 485.294

* Das Wort «ERROR» befindet sich 10.500 mal im Log
* Die Fehlermeldung «host minus domain may not contain any dots», Vorkommen: 2452
* ClassNotFoundException: LcmCustomerCategoryCodeType, Vorkommen: 6
* CommunicationServiceJsonException[ProcessingStatus[ERROR code=0 'No e-mail address for given complex\_no found., Vorkommen: 245
* SEVERE: Couldn't read decision configuration for stepUid Uid[2757013308, class ch.local.crm.shared.process.LcmProcessCodeType], Vorkommen: 3

### Meldung: value is out of integer range

Vorkommen: > 22.000

Die Klasse

com.bsiag.crm.shared.core.configuration.code.Uid

enthält eine Methode

**public** **static** **int** intValue(Uid uid)

Diese läuft ständig auf einen Fehler beim Casting von Long auf Integer:

SEVERE: accessing int value of Uid 4394947802 (class com.bsiag.crm.shared.core.emailimport.EmailRuleCodeType) but value is out of integer range

### ClassNotFoundException: LcmCustomerCategoryCodeType

ProcessingException[ProcessingStatus[ERROR code=0 Deserialization of byte array failed. java.lang.ClassNotFoundException: ch.local.crm.shared.legalentity.codetype.LcmCustomerCategoryCodeType]]

Die Klasse LcmCustomerCategoryCodeType war nicht vorhanden und wurde neu erstellt.

**public** **class** LcmCustomerCategoryCodeType **extends** CustomerCategoryCodeType {

**private** **static** **final** **long** *serialVersionUID* = 3219974716217426254L;

**public** LcmCustomerCategoryCodeType() {

**super**();

}

/\*\*

\* **@param** callInitializer

\*/

**public** LcmCustomerCategoryCodeType(**boolean** callInitializer) {

**super**(callInitializer);

}

@Override

**protected** Class<? **extends** ICodeFolder> getConfiguredCodeFolder() {

**return** **super**.getConfiguredCodeFolder();

}

}

## 09.07.2019 LOG-Ausgabe erweitert

Die LOG-Ausgabe in den zuletzt aufgerufenen Funktionen wurde erweitert, so dass der Fehler besser nachvollzogen werden kann.

Der hinzugefügte oder geänderte Code ist grau hinterlegt.

ComplexResolveService

@Override

**public** String resolveComplexNo(ILegalEntityKey key) **throws** ComplexNoResolvingException {

*LOG*.info("START: Resolving COMPLEX\_NO for " + key.toString());

BsiXLegalEntityView\_aliased le = **new** BsiXLegalEntityView\_aliased("le");

LcmBinds binds = **new** LcmBinds();

String result = "";

**try** {

HQuery<String> query = JPA.*createSuperUserSQL92Query*(String.**class**, QL.*select*(le.complexNo).from(le.TABLE)

.where(*eq*(le.key, binds.setDomainKey(key))));

**final** String complexNo = binds.applyQueryParameters(query).uniqueResult();

**if** (complexNo == **null**) {

*LOG*.error("Could not resolve COMPLEX\_NO for " + key.toString());

**throw** **new** ComplexNoResolvingException("Could not resolve COMPLEX\_NO for " + key.toString());

}

result = complexNo;

} **catch** (ComplexNoResolvingException ex) {

*LOG*.error("Could not resolve COMPLEX\_NO for " + key.toString() + "\n" + ex.getMessage());

**throw** **new** ComplexNoResolvingException("Could not resolve COMPLEX\_NO for " + key.toString() + "\n" + ex.getMessage());

} **catch** (Exception ex) {

*LOG*.error("Could not resolve COMPLEX\_NO for " + key.toString() + "\n" + ex.getMessage());

**throw** **new** ComplexNoResolvingException("Could not resolve COMPLEX\_NO for " + key.toString() + "\n" + ex.getMessage());

}

**return** result;

}

LcmRestSyncCustomerItemsBaseService

**private** **static** **final** IScoutLogger *LOG* = ScoutLogManager.*getLogger*(ComplexResolveService.**class**);

@Override

**public** List<CustomerItem> loadCustomerItems(Collection<LcmCustomerSyncItem> syncItems) **throws** ProcessingException {

**if** (syncItems == **null** || syncItems.isEmpty()) {

**return** Collections.*emptyList*();

}

List<CustomerItem> customers = **new** ArrayList<CustomerItem>();

**for** (LcmCustomerSyncItem syncItem : syncItems) {

**try** {

customers.add(loadCustomerItem(syncItem));

} **catch** (ProcessingException ex) {

*LOG*.error("Error in " + syncItem.toString() + "\n" + ex.getMessage());

**throw** **new** ProcessingException("Error in " + syncItem.toString() + "\n" + ex.getMessage());

} **catch** (Exception ex) {

*LOG*.error("Error in " + syncItem.toString() + "\n" + ex.getMessage());

**throw** **new** ProcessingException("Error in " + syncItem.toString() + "\n" + ex.getMessage());

}

}

**return** customers;

}

**private** CustomerItem loadCustomerItem(LcmCustomerSyncItem syncItem) **throws** ProcessingException {

CustomerItem customerItem = **new** CustomerItem();

**if** (isCompany(syncItem)) {

**try** {

SERVICES.*getService*(ISchedulerBaseService.**class**).logJobInfoMessage(**null**, "Try to load LcmCustomerSyncItem " + syncItem.toString());

customerItem.setComplexNo(SERVICES.*getService*(IComplexResolveService.**class**).resolveComplexNo(syncItem.getCustomerKey()));

customerItem.setName(getCompanyName(syncItem.getCustomerKey()));

customerItem.setMergedComplexNoList(getMergedIds(customerItem.getComplexNo()));

setParentCompanyId(syncItem, customerItem);

setExternalCustomers(syncItem, customerItem);

} **catch** (ProcessingException ex) {

*LOG*.error("Error in " + syncItem.toString() + "\n" + ex.getMessage());

**throw** **new** ProcessingException("Error in " + syncItem.toString() + "\n" + ex.getMessage());

} **catch** (Exception ex) {

*LOG*.error("Error in " + syncItem.toString() + "\n" + ex.getMessage());

**throw** **new** ProcessingException("Error in " + syncItem.toString() + "\n" + ex.getMessage());

}

} **else** {

**throw** **new** ProcessingException("Sync service supports only companies at the moment and not persons " + syncItem.getCustomerKey());

}

**return** customerItem;

}

## 18.07.2019 Fehler in TEP: Bei gleicher Kundennummer wie vorher

Das Log-Skript vom 18.07.2019, 14:15 enthält folgende Zeile:

SEVERE: Could not resolve COMPLEX\_NO for CompanyKey[19128510]

Die gleiche Fehlermeldung gab es bereits am 09.07.

Mit der folgenden Abfrage werden auf TEP keine Daten erhalten, wohl aber auf PROD:

**SELECT** \* **FROM** BSI\_LEGAL\_ENTITY\_VIEW **WHERE** COMPANY\_NR = 19128510