Service Routines in Java

Customer How-to Guide 1.0 February 2020

Contents

1 About this how-to guide

The **Service Routines in Java Customer How-to Guide** describes how to write service routines in java to update fields of a locally developed application (L3).Prerequisites

This guide assumes that you:

- Have read the **Java Extensibility Framework Customer Overview**.
- Have a basic understanding of Java.
- Know how to configure and initiate a service in Transact (formerly known as T24).
- Have verified the JD product is installed in SPF.

1.1 Legal

1.2 History

Version	Date	Change	Author
1.0	February 2020	Initial release	Lizen Bista

2 Introduction

The **Service Routines in Java Customer How-to Guide** describes how to write service routines in Java to update fields of a locally developed application (L3).

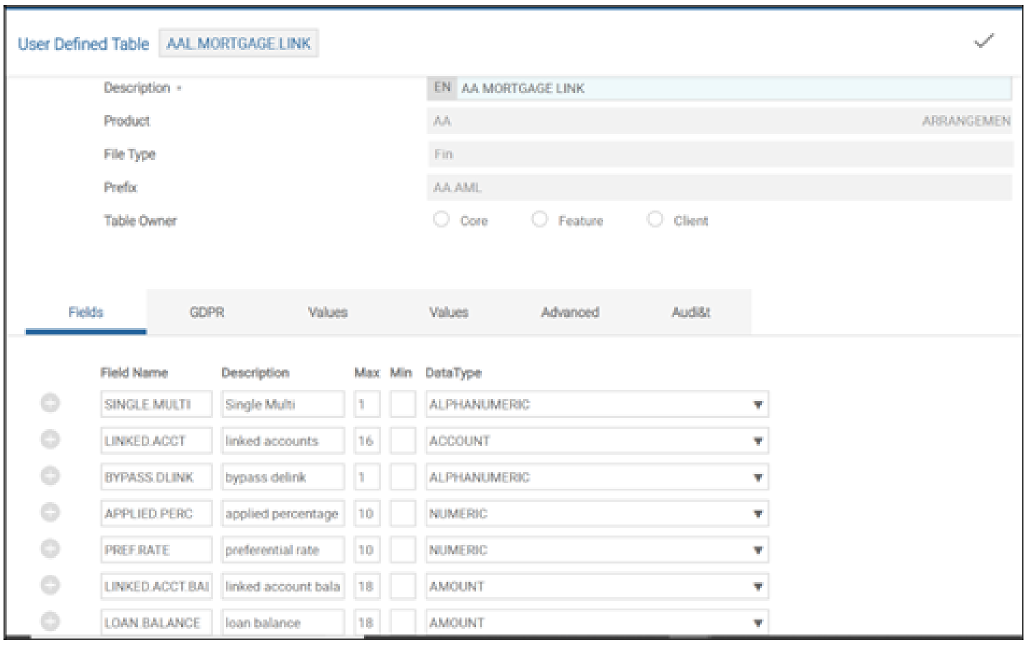
The use case in this document involves linking mortgage accounts with customer accounts (savings and current), so that they provide preferential rates up to a certain applied percentage of the loan balance.

2.1 L3 tables layout

We need to create two L3 tables for this use case.

2.1.1 AAL.MORTGAGE.LINK

This table records details of applied percentage and preferential interest rate for the linked account of a mortgage loan. The objective is to write service routines in java to update the fields LINKED.ACCT.BAL and LOAN.BALANCE of the L3 application with data from the corresponding core applications.

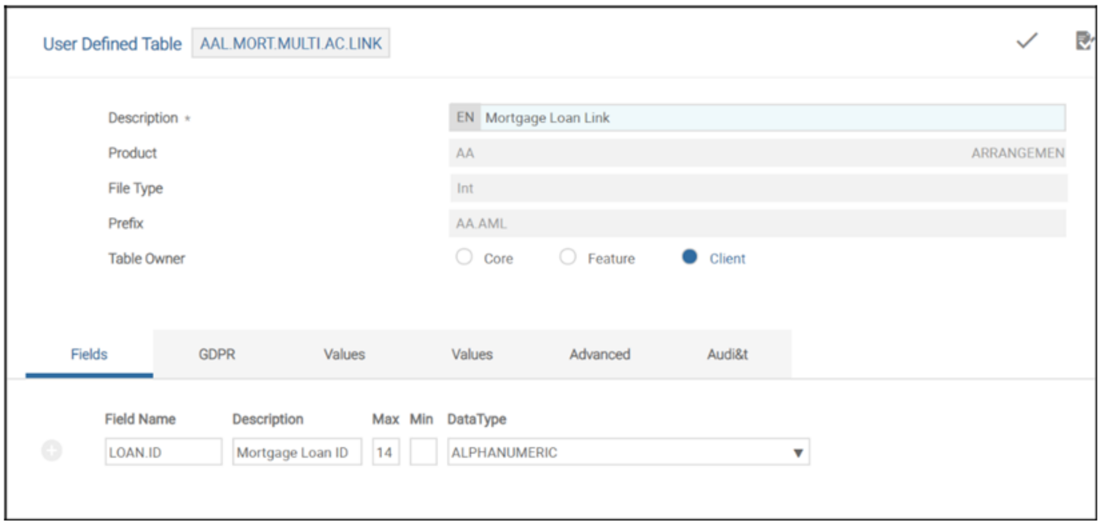

User Defined Table AAL.MORTGAGE.LINK

Description *	EN AA MORTGAGE LINK
Product	AA ARRANGEMEN
File Type	Fin
Prefix	AA.AML
Table Owner	Core Feature Client

Field Name	Description	Max	Min	DataType
SINGLE.MULTI	Single Multi	1		ALPHANUMERIC
LINKED.ACCT	linked accounts	16		ACCOUNT
BYPASS.DLINK	bypass delink	1		ALPHANUMERIC
APPLIED.PERC	applied percentage	10		NUMERIC
PREF.RATE	preferential rate	10		NUMERIC
LINKED.ACCT.BAI	linked account bala	18		AMOUNT
LOAN.BALANCE	loan balance	18		AMOUNT

2.1.2 AAL.MORT.MULTI.AC.LINK

This table records the linked account IDs of mortgage loans. The service routine uses this application to retrieve arrangement IDs.



User Defined Table AAL.MORT.MULTI.AC.LINK

Description *	EN Mortgage Loan Link
Product	AA ARRANGEMEN
File Type	Int
Prefix	AA.AML
Table Owner	Core Feature [✓ Client]

Field Name	Description	Max	Min	DataType
LOAN.ID	Mortgage Loan ID	14		ALPHANUMERIC

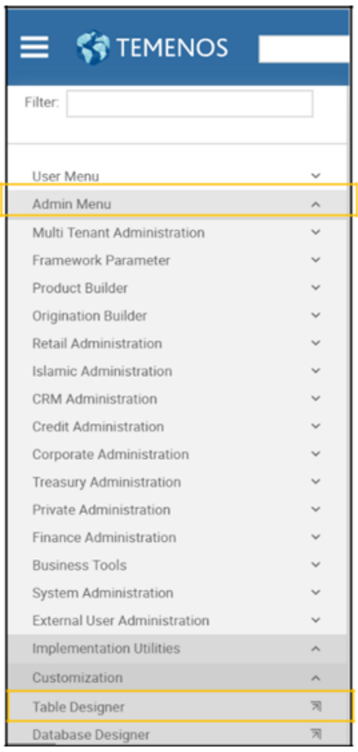
3 Creating the L3 tables

Use either the Table Designer or EB.TABLE.DEFINITION application to create the L3 tables AAL.MORTGAGE.LINK and AAL.MORT.MULTI.AC.LINK.

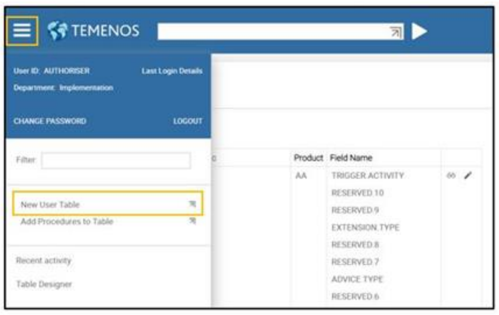
3.1 Table Designer

Procedure

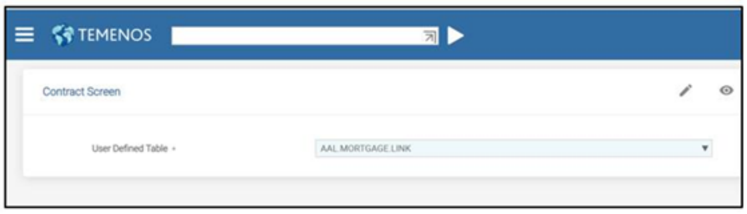
1. Go to **Admin Menu > Table Designer**



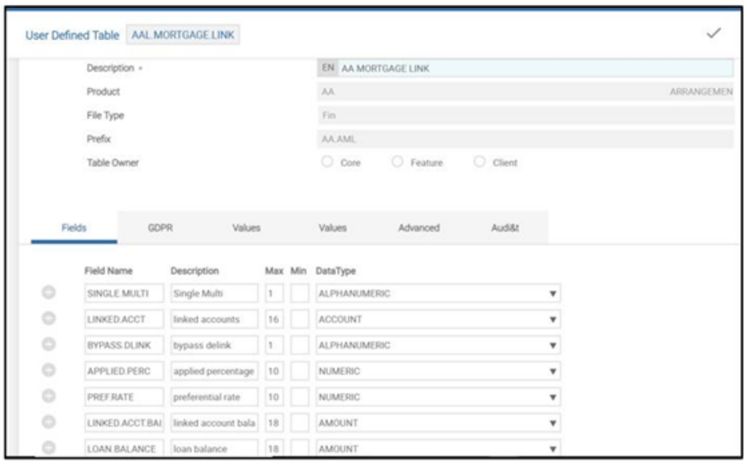
2. Click **New User Table.**



3. Create the table AAL.MORTGAGE.LINK and populate it with the required fields. Repeat the above steps to create the table AAL.MORT.MULTI.AC.LINK.


Contract Screen

User Defined Table *	AAL.MORTGAGE.LINK


User Defined Table AAL.MORTGAGE.LINK

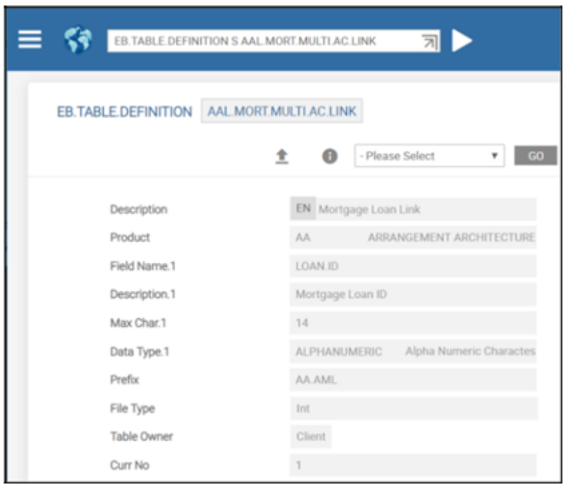
Description *	EN AA MORTGAGE LINK
Product	AA ARRANGEMEN
File Type	Fin
Prefix	AA.AML
Table Owner	○ Core ○ Feature ○ Client

Fields | GDPR | Values | Values | Advanced | Audit&t

Field Name	Description	Max	Min	DataType
SINGLE.MULTI	Single Multi	1		ALPHANUMERIC
LINKED.ACCT	linked accounts	16		ACCOUNT
BYPASS.DLINK	bypass delink	1		ALPHANUMERIC
APPLIED.PERC	applied percentage	10		NUMERIC
PREF.RATE	preferential rate	10		NUMERIC
LINKED.ACCT.BAI	linked account bala	18		AMOUNT
LOAN.BALANCE	loan balance	18		AMOUNT

3.2 EB.TABLE.DEFINITION application

Use the EB.TABLE.DEFINITION application to create the two L3 tables.


EB.TABLE.DEFINITION S AAL.MORT.MULTI.AC.LINK

EB.TABLE.DEFINITION AAL.MORT.MULTI.AC.LINK

Description	EN Mortgage Loan Link
Product	AA ARRANGEMENT ARCHITECTURE
Field Name.1	LOAN.ID
Description.1	Mortgage Loan ID
Max Char.1	14
Data Type.1	ALPHANUMERIC Alpha Numeric Charactes
Prefix	AA.AML
File Type	Int
Table Owner	Client
Curr No	1

4 Generating the Classes

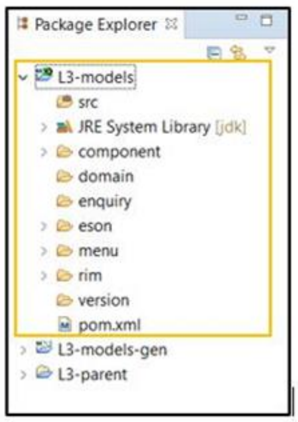
The following sections describe how to generate the classes for the two newly created L3 tables, AAL.MORTGAGE.LINK and AAL.MORT.MULTI.AC.LINK.

4.1 Create a Models Project and a Server Project

To import the application metadata of the newly created L3 tables, create a Models project in Design Studio.

Procedure

1. In the menu bar, click **File > New > Project**. Select **Design Studio > Design Studio Template Projects**.
2. Click choose a template drop down and select **Design Studio Model Project**.
3. In the **Project Name** field, type the name of the new project (for example, **L3**).
4. Click **Finish** to create the project. Package Explorer displays the new Design Studio project.



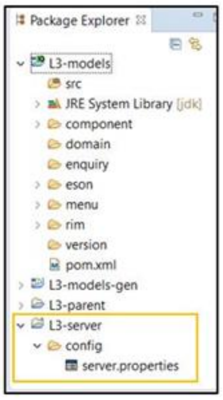
As well as the `*-models` project and `*-models-gen` project, you also need to add a `*-server` project to the workspace.

Procedure

1. In the menu bar, click **File > New > Project**.
2. Expand **Design Studio > Design Studio T24 server connectivity**.
3. Enter a name for the server project ending with `-server` (for example, `L3-server`).

4. Click **Next**. The **Choose the server connection type** dialog is displayed.

5. Select **T24 Server - Web service** as the connection type. Click **Finish**.



```
Package Explorer
v L3-models
    src
  > JRE System Library [jdk]
  > component
    domain
    enquiry
  > eson
  > menu
  > rim
    version
    pom.xml
> L3-models-gen
> L3-parent
v L3-server
  v config
      server.properties
```

6. Double click **server.properties** under ***-server** to open the properties file in the editor window:

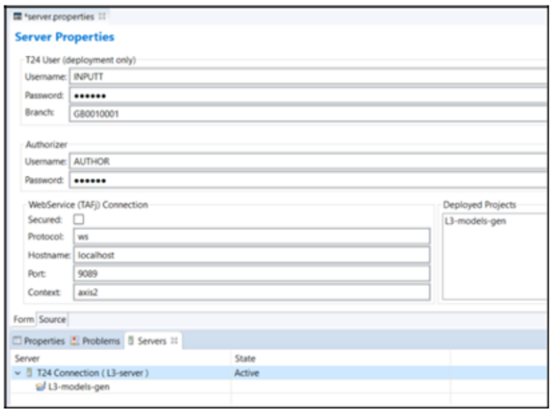
   a. Enter a valid T24 username, password and company code in the **T24 User** section.

   b. In WebService connection, set:

      - **Hostname = localhost** (or the IP address of the remote server).

      - **Port = 9089** (the JBoss port number).

   c. In the **Service view** pane, verify that the server connection is set to **Active.**



*server.properties

Server Properties

T24 User (deployment only)

Username:	INPUTT
Password:	••••••
Branch:	GB0010001

Authorizer

Username:	AUTHOR
Password:	••••••

WebService (TAFj) Connection

Secured:	☐
Protocol:	ws
Hostname:	localhost
Port:	9089
Context:	axis2

Deployed Projects

L3-models-gen

Form Source

Properties Problems Servers

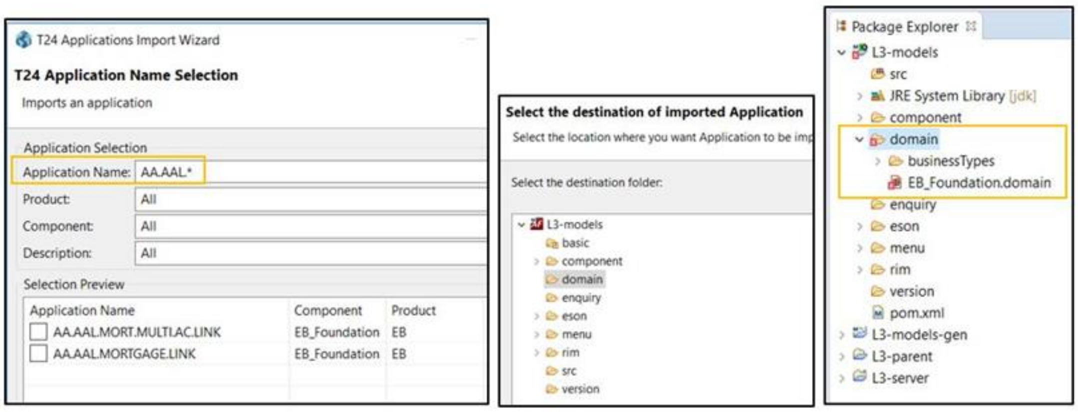
Server	State
T24 Connection (L3-server)	Active
L3-models-gen	

4.2 Import locally developed applications

You also need to import T24 application metadata into the design studio workspace.

Procedure

1. Select **File > Import > Design Studio - Import T24 Applications**.
2. Select the T24 server from the list and click **Next**. Design Studio connects to T24 and retrieves all existing applications.
3. Select the L3 application(s) to import.
4. Click **Next** and select the `*-model` project where you want the applications to be imported.
5. Click **Finish**.


T24 Applications Import Wizard

T24 Application Name Selection

Imports an application

Application Selection

Application Name:	AA.AAL*
Product:	All
Component:	All
Description:	All

Selection Preview

Application Name	Component	Product
☐ AA.AAL.MORT.MULTI.AC.LINK	EB_Foundation	EB
☐ AA.AAL.MORTGAGE.LINK	EB_Foundation	EB

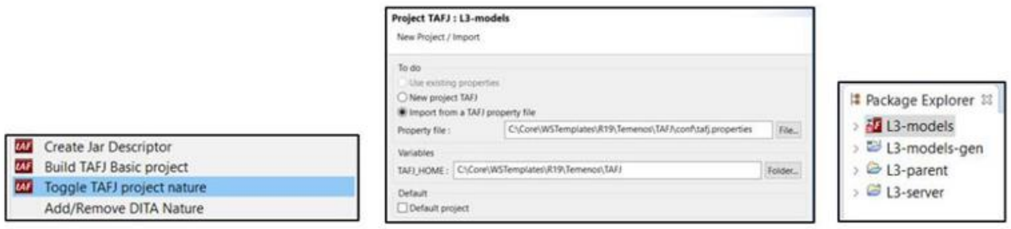
Select the destination of imported Application

Select the location where you want Application to be imp

Select the destination folder:

Ignore the error on the `*-models` project. The imported domain has dependencies on other domains which are missing in the workspace.

6. Toggle the `*-models` project to TAFJ nature.



Project TAFJ : L3-models

New Project / Import

To do

☐ Use existing properties

○ New project TAFJ

● Import from a TAFJ property file

Property file : C:\Core\WSTemplates\R19\Temenos\TAFJ\conf\tafj.properties File...

Variables

TAFJ_HOME : C:\Core\WSTemplates\R19\Temenos\TAFJ Folder...

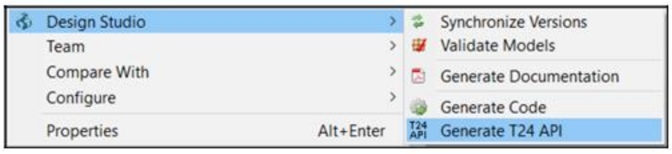
Default

☐ Default project

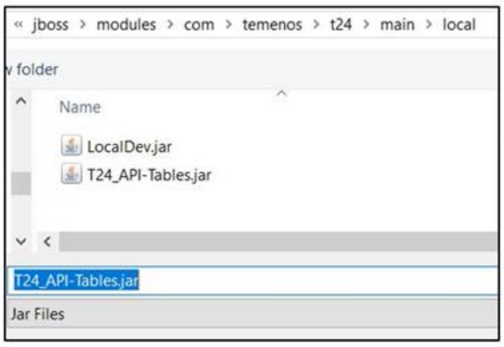
4.3 Generate API for imported applications

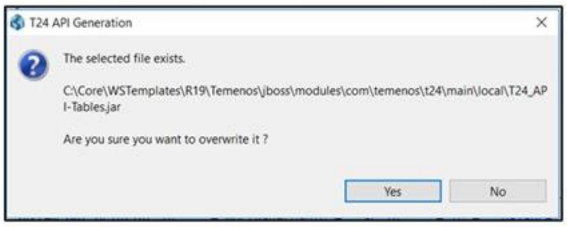
Procedure

1. Right-click the ***-models project > Design Studio > Generate T24 API**.



2. Provide a location for the jar. The API is generated.




T24 API Generation

The selected file exists.

C:\Core\WSTemplates\R19\Temenos\jboss\modules\com\temenos\t24\main\local\T24_API-Tables.jar

Are you sure you want to overwrite it ?

5 Writing the Java implementation

To write the service routine in Java, start Design Studio and switch to the Java perspective..

5.1 Create a new Java project

Procedure

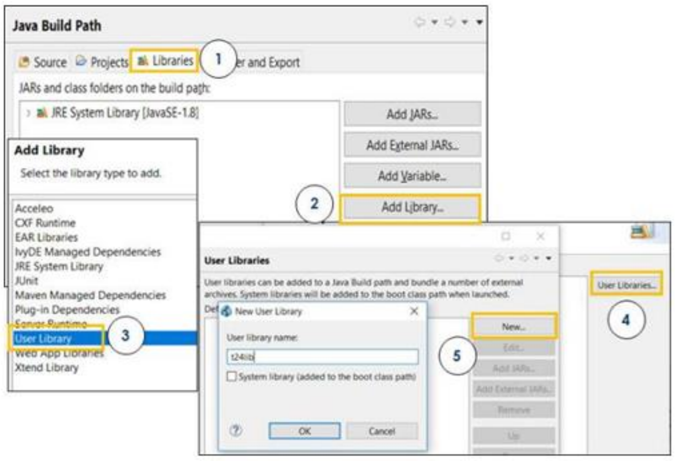
1. Create a new Java project (**File > New > Java project**). In the wizard supply a project name.

2. Configure the build path settings for the Java project to add dependent T24 and TAFJ libraries.

    a. Right click the project, for example **L3JAVA > Build path > Configure build path**.

    b. Click **Libraries** tab **> Add Library > User Library > User Libraries**.

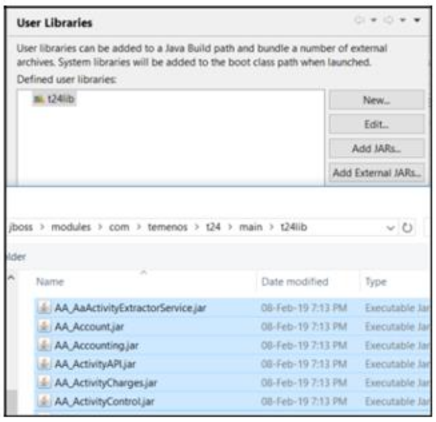
    c. In the **User Libraries** window, click **New** and give the library a name, for example, **t24lib**.



    d. Click **Add external jars**. Navigate to the T24 libraries folder under `%JBOSS_HOME%/modules`. Select all the jars and click **Open**.

    Click **OK** and finish.

Alternatively, just add the required T24 hook jars like EB_TemplateHook.jar, T24_API-Tables.jar etc using Add External JARs.


User Libraries

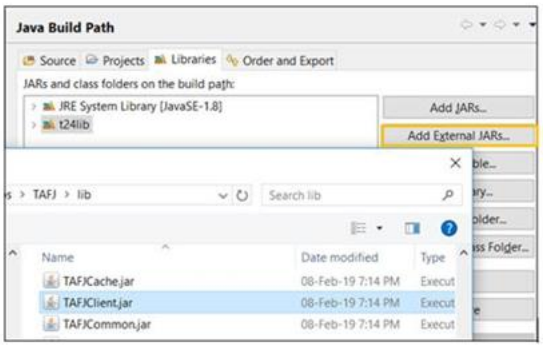
User libraries can be added to a Java Build path and bundle a number of external archives. System libraries will be added to the boot class path when launched.

Defined user libraries:

t24lib

Name	Date modified	Type
AA_AaActivityExtractorService.jar	08-Feb-19 7:13 PM	Executable Jar
AA_Account.jar	08-Feb-19 7:13 PM	Executable Jar
AA_Accounting.jar	08-Feb-19 7:13 PM	Executable Jar
AA_ActivityAPI.jar	08-Feb-19 7:13 PM	Executable Jar
AA_ActivityCharges.jar	08-Feb-19 7:13 PM	Executable Jar
AA_ActivityControl.jar	08-Feb-19 7:13 PM	Executable Jar

e. In the **Libraries** tab, click **Add External JARs** and add `TAFJClient.jar` from `%TAFJ_HOME%/lib`

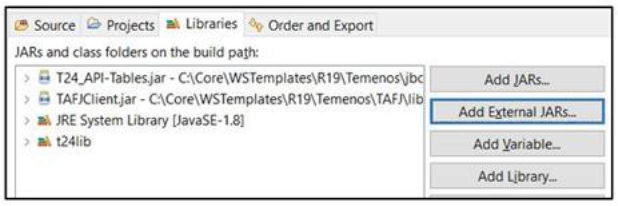

Java Build Path

JARs and class folders on the build path:

- JRE System Library [JavaSE-1.8]
- t24lib

Name	Date modified	Type
TAFJCache.jar	08-Feb-19 7:14 PM	Execut
TAFJClient.jar	08-Feb-19 7:14 PM	Execut
TAFJCommon.jar	08-Feb-19 7:14 PM	Execut

f. In the **Libraries** tab, click **Add External JARs**, navigate to the `T24_API-Tables.jar` folder and click open.



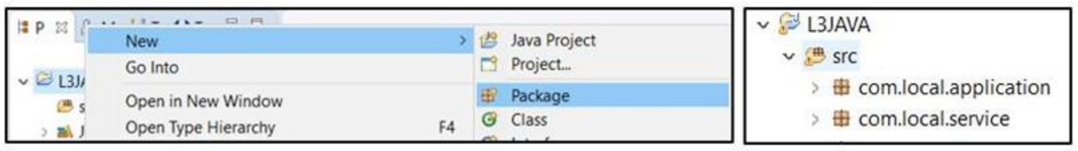
JARs and class folders on the build path:

- T24_API-Tables.jar - C:\Core\WSTemplates\R19\Temenos\jbc
- TAFJClient.jar - C:\Core\WSTemplates\R19\Temenos\TAFJ\lib
- JRE System Library [JavaSE-1.8]
- t24lib

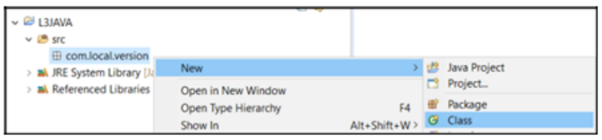
5.2 Extend the Superclass ServiceLifecycle

Procedure

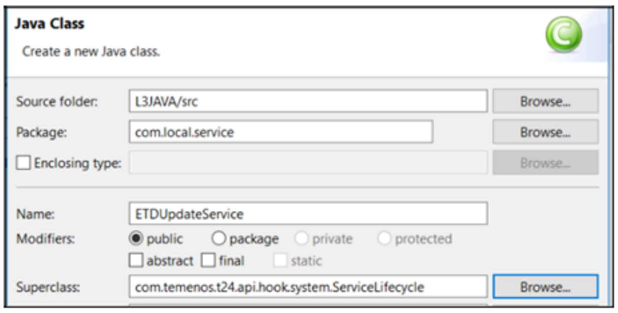
1. Create a new Java package. Right click the **project > New > Package** and supply a name.



2. Right click the package and add a new Class to the package, for example, `ETDUpdateService`.



3. Extend the superclass `ServiceLifecycle` for your class.

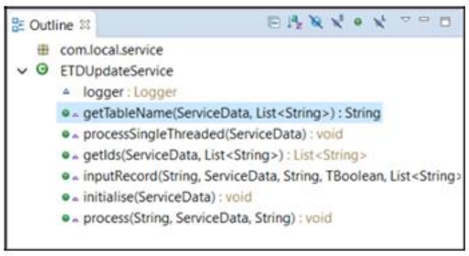


Java Class

Create a new Java class.

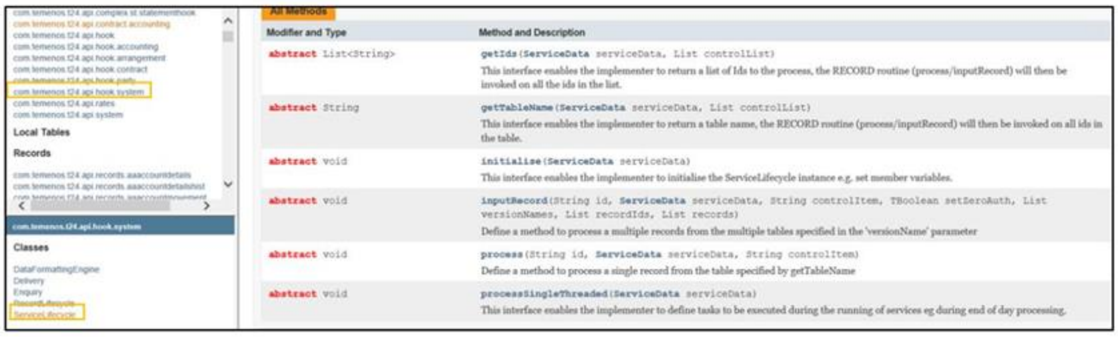
Source folder:	L3JAVA/src
Package:	com.local.service
Enclosing type:	
Name:	ETDUpdateService
Modifiers:	[✓ public] package private protected; abstract final static
Superclass:	com.temenos.t24.api.hook.system.ServiceLifecycle

4. Click Finish. The method outlines of the superclass `ServiceLifecycle` is generated. The class has several methods that can be overridden.



```
Outline
com.local.service
ETDUpdateService
  logger : Logger
  getTableName(ServiceData, List<String>) : String
  processSingleThreaded(ServiceData) : void
  getIds(ServiceData, List<String>) : List<String>
  inputRecord(String, ServiceData, String, TBoolean, List<String>
  initialise(ServiceData) : void
  process(String, ServiceData, String) : void
```

To find out more about the class `ServiceLifecycle` and what methods can be overridden, click the `com.temenos.t24.api.hook.system` package in L3 API documentation in the browser.


com.temenos.t24.api.complex.st.statementhook
com.temenos.t24.api.contract.accounting
com.temenos.t24.api.hook
com.temenos.t24.api.hook.accounting
com.temenos.t24.api.hook.arrangement
com.temenos.t24.api.hook.contract
com.temenos.t24.api.hook.party
com.temenos.t24.api.hook.system
com.temenos.t24.api.rates
com.temenos.t24.api.system
Local Tables
Records
com.temenos.t24.api.records.aaaccountdetails
com.temenos.t24.api.records.aaaccountdetailshist
com.temenos.t24.api.hook.system
Classes
DataFormattingEngine
Delivery
Enquiry
RecordLifecycle
ServiceLifecycle
All Methods

Modifier and Type	Method and Description
abstract List<String>	getIds(ServiceData serviceData, List controlList) This interface enables the implementer to return a list of Ids to the process, the RECORD routine (process/inputRecord) will then be invoked on all the ids in the list.
abstract String	getTableName(ServiceData serviceData, List controlList) This interface enables the implementer to return a table name, the RECORD routine (process/inputRecord) will then be invoked on all ids in the table.
abstract void	initialise(ServiceData serviceData) This interface enables the implementer to initialise the ServiceLifecycle instance e.g. set member variables.
abstract void	inputRecord(String id, ServiceData serviceData, String controlItem, TBoolean setZeroAuth, List versionNames, List recordIds, List records) Define a method to process a multiple records from the multiple tables specified in the 'versionName' parameter
abstract void	process(String id, ServiceData serviceData, String controlItem) Define a method to process a single record from the table specified by getTableName
abstract void	processSingleThreaded(ServiceData serviceData) This interface enables the implementer to define tasks to be executed during the running of services eg during end of day processing.

5.3 Writing the Java implementation for the batch job

Procedure

1. Override the inherited methods:

   - `getTableName()` to return all IDs from AAL.MORT.MULTI.AC.LINK (SELECT routine).
   - `process()` to update LINKED.ACCT.BAL and LOAN.BALANCE in AAL.MORTGAGE.LINK (RECORD routine).

2. Click Ctrl + S to save the code. See code sample below.

```
package com.local.service;

import java.util.List;
import java.util.logging.Logger;

import com.temenos.api.TBoolean;
import com.temenos.api.TStructure;
import com.temenos.api.exceptions.T24CoreException;
import com.temenos.api.exceptions.T24IOException;
import
com.temenos.t24.api.complex.aa.contractapi.BalanceMovement;
import
com.temenos.t24.api.complex.eb.servicehook.ServiceData;
import com.temenos.t24.api.contract.accounting.Contract;
import
com.temenos.t24.api.hook.system.ServiceLifecycle
; import
com.temenos.t24.api.records.account.AccountRecor
d; import com.temenos.t24.api.system.DataAccess;
import com.temenos.t24.api.tables.aalmortgagelink.AalMortgageLinkRecord;
import com.temenos.t24.api.tables.aalmortgagelink.AalMortgageLinkTable;
import com.temenos.t24.api.tables.aalmortmultiaclink.AalMortMultiAcLinkRecord;
public class ETDUpdateService extends ServiceLifecycle {

    Logger logger = Logger.getLogger("T24");

    @Override
    public String getTableName(ServiceData serviceData, List<String>
        controlList) { return "F.AA.AAL.MORT.MULTI.AC.LINK";
         }
    @Override
    public void process(String id, ServiceData serviceData, String controlItem) {
        DataAccess da = new DataAccess(this);
        try {
            AalMortMultiAcLinkRecord accountLinkRecord = new AalMortMultiAcLinkRecord(
                    da.getRecord("AA.AAL.MORT.MULTI.AC.LINK", id));
            String mortgageLoanId = accountLinkRecord.getLoanId().getValue();
            AccountRecord accountRecord = new AccountRecord(da.getRecord("ACCOUNT",
            id)); String actualBalance =
            accountRecord.getOpenActualBal().getValue().toString();
            AalMortgageLinkTable mortgageTable = new AalMortgageLinkTable(this);
            AalMortgageLinkRecord mortgageRecord = new AalMortgageLinkRecord(

                  da.getRecord("AA.AAL.MORTGAGE.LINK", mortgageLoanId));
```

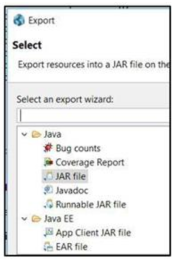
```
                mortgageRecord.setLinkedAcctBal(actualBalance
            ); Contract contract = new Contract(this);
            contract.setContractId(mortgageLoanId);
            List<BalanceMovement> loanBalance =
            contract.getBalanceMovements("CURACCOUNT", ""); int dateBalance =
            loanBalance.get(0).getBalance().intValue();
            String balanceAsString =
            Integer.toString(dateBalance);
            mortgageRecord.setLoanBalance(balanceAsString);
           mortgageTable.write(mortgageLoanId, mortgageRecord);
        } catch (T24IOException e) {
            System.out.println("Write failed " + e);
        } catch (T24CoreException tce) {
            System.out.println("File does not exist " + tce);
        }

    }
}
```

5.4 Placing the implementation in a library and loading in JBoss Classpath

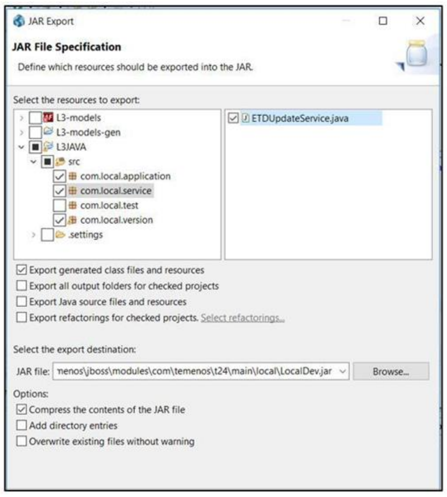
Procedure

1. After the Java code is written, right click the **project > Export > JAR** file. Select the export destination.



Export resources into a JAR file on the

Select an export wizard:



Define which resources should be exported into the JAR.

Select the resources to export:

☑ Export generated class files and resources
☐ Export all output folders for checked projects
☐ Export Java source files and resources
☐ Export refactorings for checked projects. Select refactorings...

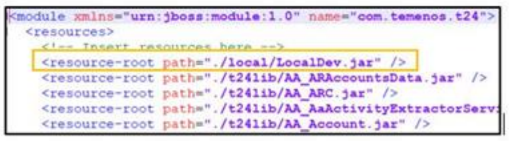
Select the export destination:

JAR file: nenos\jboss\modules\com\temenos\t24\main\local\LocalDev.jar

Options:

☑ Compress the contents of the JAR file
☐ Add directory entries
☐ Overwrite existing files without warning

2. Update `module.xml` in `%JBOSS_HOME%/modules` with the new jar path and name. Restart JBoss.

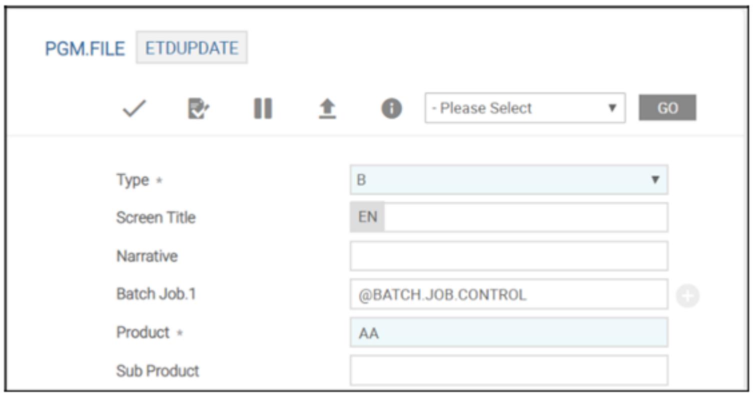


```
<module xmlns="urn:jboss:module:1.0" name="com.temenos.t24">
  <resources>
    <!-- Insert resources here -->
    <resource-root path="./local/LocalDev.jar" />
    <resource-root path="./t24lib/AA_ARAccountsData.jar" />
    <resource-root path="./t24lib/AA_ARC.jar" />
    <resource-root path="./t24lib/AA_AaActivityExtractorServ
    <resource-root path="./t24lib/AA_Account.jar" />
```

5.5 Link the Java Methods to Service Workflow

Procedure

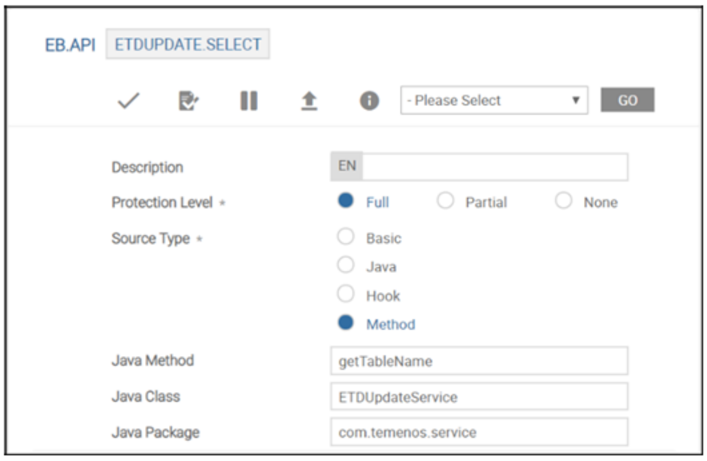
1. Create a PGM.FILE record for the job.



PGM.FILE ETDUPDATE

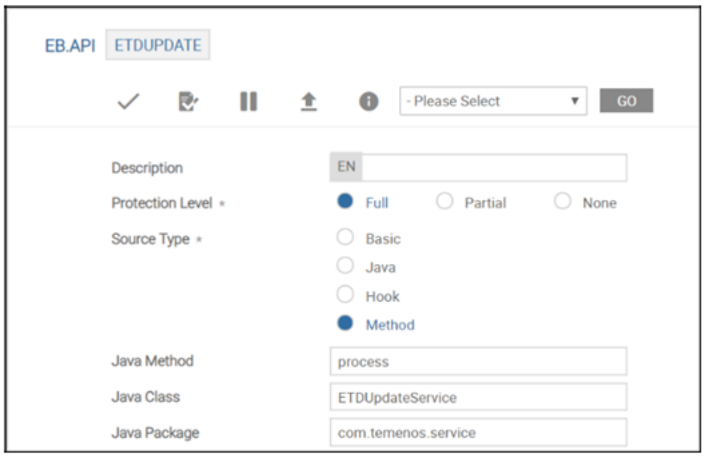
Type *	B
Screen Title	EN
Narrative	
Batch Job.1	@BATCH.JOB.CONTROL
Product *	AA
Sub Product	

2. Create EB.API records for each of the overridden methods (select and record routine). Append .SELECT to EB.API ID for the select routine. (For a load routine, append .LOAD to the ID).



EB.API ETDUPDATE.SELECT

Description	EN
Protection Level *	[✓ Full] Partial None
Source Type *	Basic Java Hook [✓ Method]
Java Method	getTableName
Java Class	ETDUpdateService
Java Package	com.temenos.service

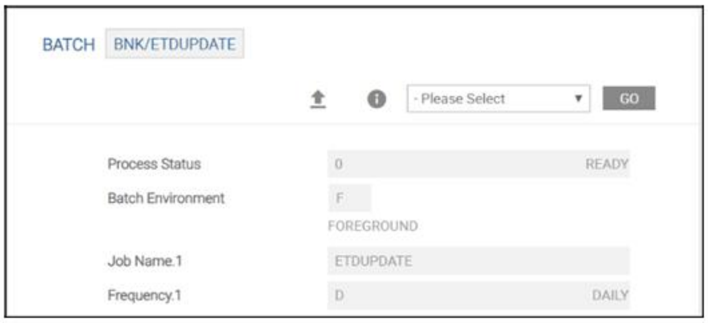


EB.API ETDUPDATE

- Please Select GO

Description	EN
Protection Level *	[● Full] ○ Partial ○ None
Source Type *	○ Basic ○ Java ○ Hook [● Method]
Java Method	process
Java Class	ETDUpdateService
Java Package	com.temenos.service

3. Add EB.API record to the service workflow.

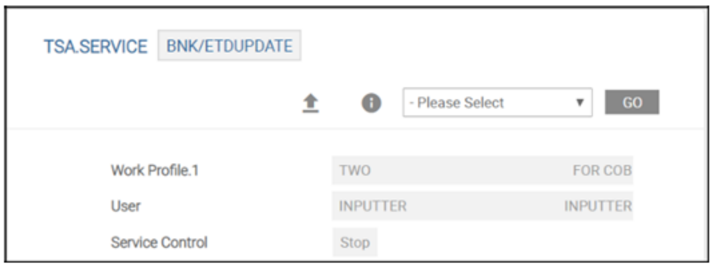


BATCH BNK/ETDUPDATE

- Please Select GO

Process Status	0	READY
Batch Environment	F	FOREGROUND
Job Name.1	ETDUPDATE	
Frequency.1	D	DAILY

4. Create a TSA.SERVICE record with the same ID as BATCH.



TSA.SERVICE BNK/ETDUPDATE

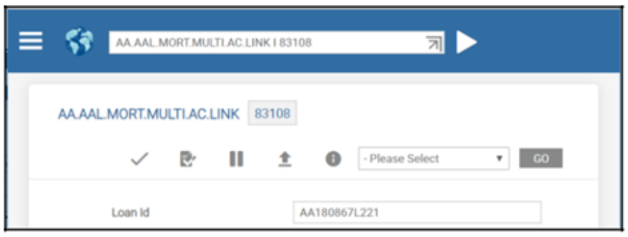
- Please Select GO

Work Profile.1	TWO	FOR COB
User	INPUTTER	INPUTTER
Service Control	Stop	

6 Testing the service

Procedure

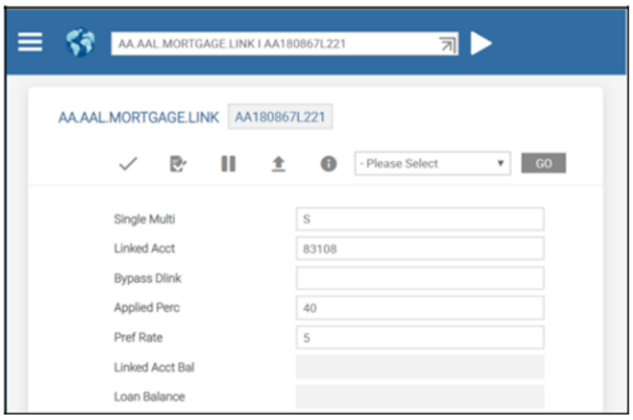
1. Manually add records in AAL.MORT.MULTI.AC.LINK and AAL.MORTGAGE.LINK to test the service.



AA.AAL.MORT.MULTI.AC.LINK I 83108

AA.AAL.MORT.MULTI.AC.LINK 83108

Loan Id	AA180867L221

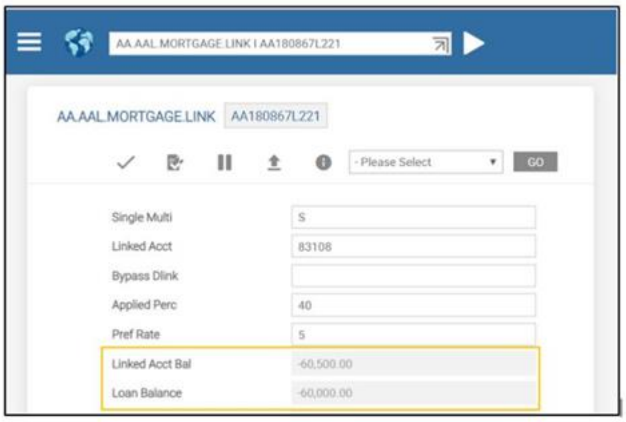


AA.AAL.MORTGAGE.LINK I AA180867L221

AA.AAL.MORTGAGE.LINK AA180867L221

Single Multi	S
Linked Acct	83108
Bypass Dlink	
Applied Perc	40
Pref Rate	5
Linked Acct Bal	
Loan Balance	

2. After running the service, the local table fields **Linked Acct Bal** and **Loan Balance** are updated for the Arrangement.



AA.AAL.MORTGAGE.LINK I AA180867L221

AA.AAL.MORTGAGE.LINK AA180867L221

Single Multi	S
Linked Acct	83108
Bypass Dlink	
Applied Perc	40
Pref Rate	5
Linked Acct Bal	-60,500.00
Loan Balance	-60,000.00