**CREATE THE CRUD PROCS**

@ Connect to dev\_ias/dev\_ias@DOCKER\_DB

@ Create YOUR\_ENTITY\_STORE package similar to DATA\_CHG\_SUB\_STORE.

Usually, we have 4 procs: SELECT list, SELECT a single entity, SAVE (one proc for INSERT & UPDATE) and DELETE, for example:

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

The above SAVE & DELETE procs demonstrate saving a single entity form dialog, which is the most popular.

If the data change dialog has multiple editable records, we use BULK SAVE & DELETE, sending a bunch of records.

The SAVE proc accepts them as a Json string:

A screenshot of a computer

Description automatically generated

The DELETE proc accepts the IDs of the entities as a comma-separated values string:

A white background with text

Description automatically generated

**CREATE THE .NET WEB SERVICE**

As an example, we will use DataChgSub objects (the "Data Change Subscriptions" screen in Admin).

**--== MT ==--**

@ Open the MT solution in VS and search for DataChgSubTbl in the Solution explorer - to see examples.

@ Open the MT solution in another instance of VS.

IF YOU ARE ADDING A NEW SCREEN:

**MT folders:**

@ Create the new entity's Solution Folder, like Admin > DataChangeSubscriptions:

Right-click the parent folder (like Admin) > Add > New Solution Folder.

Let's call that folder YourEntity.

@ Under YourEntity, create the next projects (how? read here: C:\\_MZ\DEV\HowTo\Steps - DEV - PT & MT.txt > CREATE A NEW NAMESPACE ("PROJECT") IN VISUAL STUDIO).

Suppose, the app root folder is Admin:

YourEntity.Admin.Domain - like Ias.DataChangeSubscriptions.Admin.Domain

YourEntity.Admin.Services.Contracts - like Ias.DataChangeSubscriptions.Admin.Services.Contracts

YourEntity.Admin.Services.MT - like Ias.DataChangeSubscriptions.Admin.Services.MT

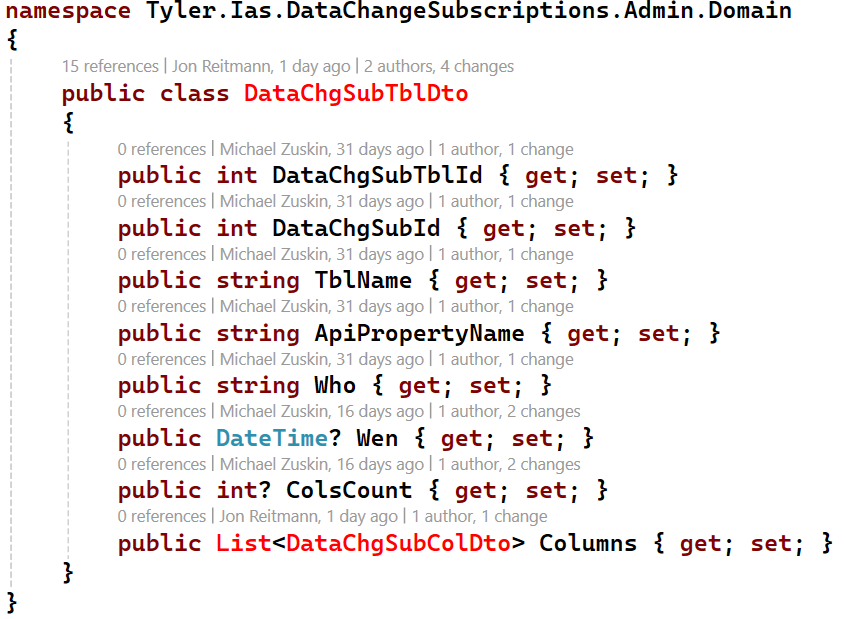
@ Search for Class1 in the Solution explorer > delete them all.

@ Search for DataChgSubTbl in the Solution explorer.

**The DTO (“model”) class representing the entity:**

@ Create under YourEntity.Admin.Domain:

YourEntityDto.cs (like DataChgSubTblDto.cs)



**The interface describing the CRUD functions:**

@ Create under YourEntity.Admin.Services.Contracts:

IYourEntityService.cs (like IDataChgSubTblService.cs)

A screenshot of a computer screen

Description automatically generated

For bulk, the SAVE function accepts and returns an array of the entities, and the DELETE function accepts an array of IDs:

A screenshot of a computer

Description automatically generated

In this specific example, the DELETE proc returns an array due to a very specific need – yours must return void.

**WCF Web Service class:**

Directly calls the Oracle package procs.

@ Create under YourEntity.Admin.Services.MT:

YourEntityService.cs (like DataChgSubTblService.cs)

All the functions except DELETE return data: the SELECT list returns an array of entities, the SELECT single and SAVE – one record.

To avoid code duplication, incapsulate reading the recordset, returned by the proc, in ExecProc private function:

A screenshot of a computer program

Description automatically generated

Then call ExecProc from the relevant functions. Notice that the SELECT single and SAVE functions return the 1st element of the recordset (since there will be only one record returned by the proc):

A screenshot of a computer program

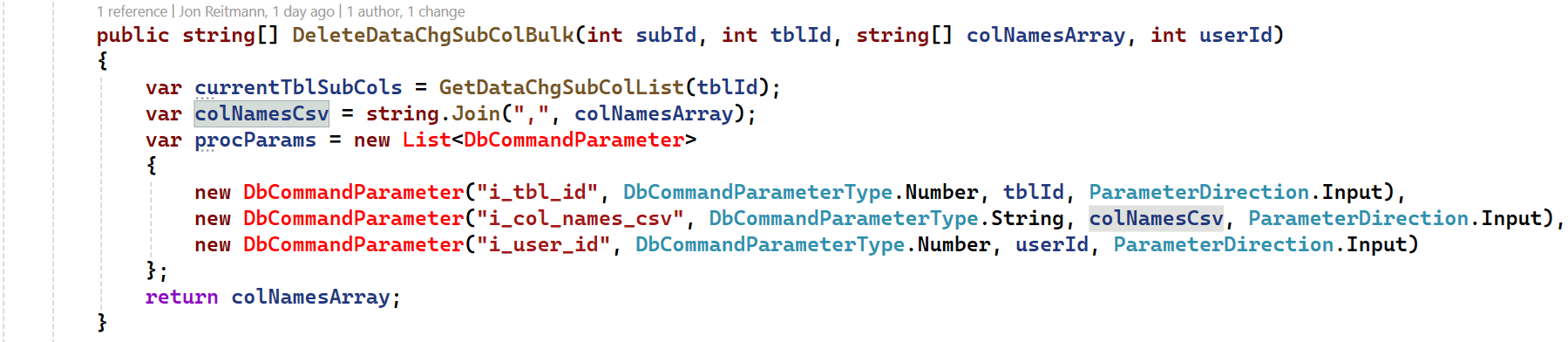
Description automatically generated

The next code demonstrates the SAVE & DELETE functions for bulk. The SAVE function converts the accepted array of entities into a Jason string:

A screenshot of a computer code

Description automatically generated

The DELETE function converts the accepted array of IDs into a comma separated list to send to the proc:



IF YOU ARE ADDING A NEW SCREEN:

@ Create the YourEntity folder under MiddleTier, like MiddleTier > DataChangeSubscriptions (C:\TylerDev\GitHub\ias-main\DotNet\MiddleTier\DataChangeSubscriptions):

Right-click MiddleTier > Add > New folder > name it YourEntity.

**The declaration (publication) of the web service:**

@ Create under the YourEntity folder under MiddleTier:

YourEntityService.svc (like DataChgSubTblService.svc)

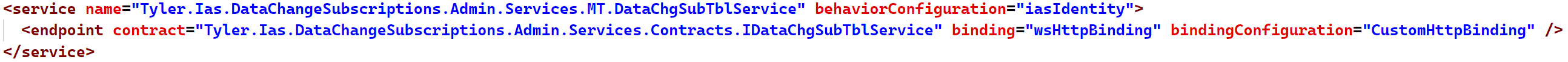


**Registering the web service in the MT’s Web.config:**

@ Under Maintain > Maintain.Services, open web.config (or just open C:\TylerDev\GitHub\ias-main\DotNet\MiddleTier\Web.config in VSC).

@ In the file, find DataChgSubTblService.

@ Copy the whole <service>...</service> section (3 lines), paste just after it, and customize by replacing DataChgSubTblService with YourEntityService (2 occurrences).



**Registering the web service in the MT’s IocConfiguration.cs:**

@ Under MiddleTier, open IocConfiguration.cs (or just open C:\TylerDev\GitHub\ias-main\DotNet\MiddleTier\IoCConfiguration.cs in VSC).

@ Find (Ctrl+F) "DataChgSubTblService".

@ Register YourEntityService in the same way.



@ Compile MT solution.

**--== PT ==--**

@ Open the PT solution in another instance of VS and search for DataChgSubTbl in the Solution explorer - to see examples.

@ Open the PT solution in another instance of VS.

IF YOU ARE ADDING A NEW SCREEN:

@ Create the new entity's Solution Folder: Right-click the app root folder (like Admin) > Add > New Solution Folder (name it YourEntity).

**Referencing PT projects (pointers to MT projects):**

@ Create the Domain project (a pointer to the Domain project in MT):

@ In MT, open YourEntityDto.cs > right-click it on the tab on the left side > Copy Full Path > return to PT.

@ On the right side, right-click the YourEntity folder > Add > Existing project.

@ Locate the folder you just copied in MT (without the file name) > double-click the .csproj file in it.

@ Create the Services.Contracts project (a pointer to the Services.Contracts project in MT):

@ In MT, open IDataChgSubService.cs > right-click it on the tab on the left side > Copy Full Path > return to PT.

@ On the right side, right-click the YourEntity folder > Add > Existing project.

@ Locate the folder you just copied in MT (without the file name) > double-click the .csproj file in it.

**PT folders:**

@ Under YourEntity folder, create the next new projects (supposing the app root folder is Admin):

Admin.Services

Admin.WcfClient

Admin.WebApi

@ Search for Class1 in the Solution explorer > delete them all.

@ Search for DataChgSubTbl in the Solution explorer.

**WCF Web Service client class:**

A Windows Communication Foundation proxy class that acts as a bridge between PT and MT. Calls the functions of the service object of MT described earlier.

@ Create under Admin.WcfClient:

YourEntityServiceClient (like DataChgSubServiceClient)



**A WCF Web Service client wrapper:**

An abstraction layer useful for decoupling and extensibility - for example, you can add additional logic to it.

While it adds a bit of boilerplate, it ensures your application is more adaptable to future changes.

The naming convention is simply “Service” since it represents the Service to the rest of the application, hiding the actual Service client class.

@ Create under Admin.Services:

YourEntityService (like DataChgSubService)

A screenshot of a computer program

Description automatically generated

**Web API Controller:**

Defines which .Net functions are called by different HTTP requests depending on the combination of the URL and the HTTP method.

@ Create under Admin.WebApi:

YourEntityController (like DataChgSubController)

In YourEntityController, endpoints should be created in the following manner:

|  |  |  |  |
| --- | --- | --- | --- |
| CRUD OPERATION: | HTTP: | MAIN (PARENT) ENTITY: | CHILD ENTITY: |
| Get all entities | GET | /entities | /entities/{entityId}/child\_entities |
| Get a single entity | GET | /entities/{entityId} | /entities/{entityId}/child\_entities/{childEntityId} |
| Add an entity | POST | /entities | /entities/{entityId}/child\_entities |
| Update an entity | PUT | /entities/{entityId} | /entities/{entityId}/child\_entities/{childEntityId} |
| Delete an entity | DELETE | /entities/{entityId} | /entities/{entityId}/child\_entities/{childEntityId} |

Notice that the same SAVE function is called with HttpPost (for INSERT) and HttpPut (for UPDATE):



A black and white image of a spiral

Description automatically generated

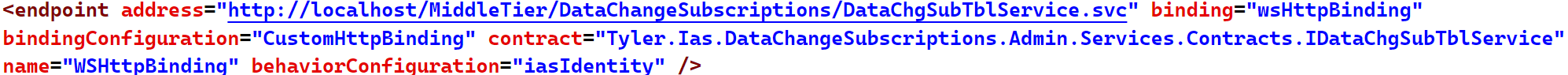
**Registering the endpoint in the PT’s Web.config:**

@ Solution explorer > find web.config - the very last one, under Web (or just open C:\TylerDev\GitHub\ias-main\DotNet\Web\Web.config in VSC).

@ In the file, find DataChgSubTblService.

@ Copy-paste the whole line and customize.

@ Copy the URL of the new service (like http://localhost/MiddleTier/DataChangeSubscriptions/DataChgSubTblService.svc) to the ticket's file - you will use it for test.



**Registering the web service in the PT’s IocConfiguration.cs:**

@ Under Web > Infrastructure, open IocConfiguration.cs (or just open C:\TylerDev\GitHub\ias-main\DotNet\Web\Infrastructure\IocConfiguration.cs in VSC).

@ Find (Ctrl+F) "DataChgSubTbl". You will find registration of DataChgSubTblService and DataChgSubTblController.

@ Register YourEntityService and YourEntityController in the same way.

A close-up of a text

Description automatically generated

@ Compile PT solution.

@ Run the URL of the new service in the browser to test it.

**CREATE THE ANGULAR OBJECTS**

IF YOU ARE ADDING A NEW SCREEN, CREATE A FOLDER FOR IT:

@ In C:\TylerDev\GitHub\ias-main\NgApps find the module folder.

Example: for Administration module it's C:\TylerDev\GitHub\ias-main\NgApps\admin (we will use "admin" in the path but it can be another app).

@ Under it, go to \src\app and find the folder of the section under which you need to add the screen.

Example: for Event Model section of Administration module, it's C:\TylerDev\GitHub\ias-main\NgApps\admin\src\app\eventmodel

We will call it \your\_section\_folder (C:\TylerDev\GitHub\ias-main\NgApps\admin\src\app\your\_section\_folder).

**CREATE FILES WHICH ARE OUTSIDE OF COMPONENTS:**

REMARK #1: If it's unclear what should be in the files, just comment out what you have copy-pasted from data-chg-sub and customize later.

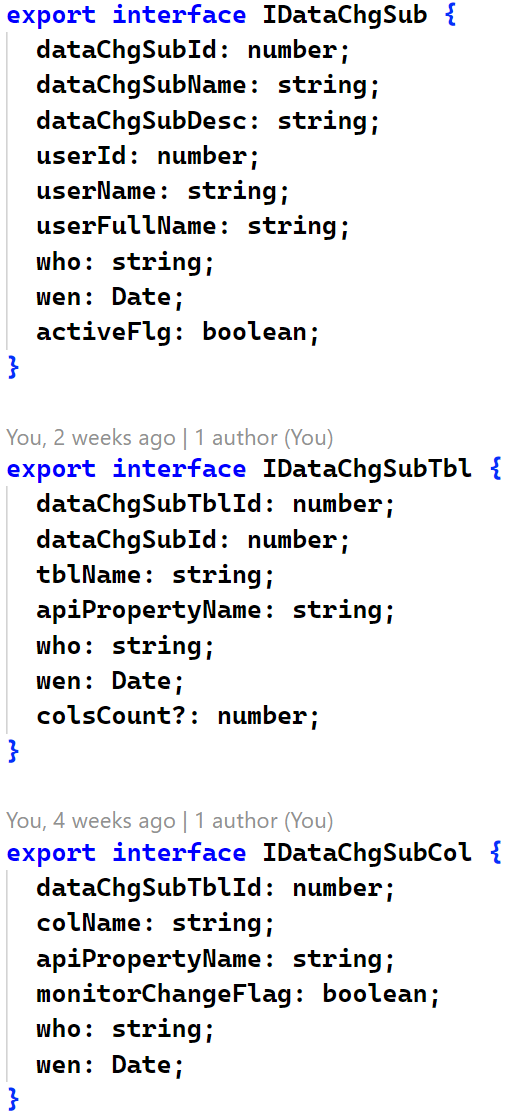
REMARK #2: Create the next 6 files only once, when creating the parent component. When you create child components later, add them to these files.

**Model:**

The file declares the data types which describe the entities used in the module (screen). Usually, they are the DTOs of .Net translated from C# to TypeScript.

@ Go to C:\TylerDev\GitHub\ias-main\NgApps\admin\src\app\your\_section\_folder\models and add your-entity.model.ts file

(like C:\TylerDev\GitHub\ias-main\NgApps\admin\src\app\eventmodel\models\data-chg-sub.model.ts).



**State**

***The data type which will be used for the module’s Store – a singleton containing the whole data of the module as it is at the current moment (i.e. the current State).***

@ Go to C:\TylerDev\GitHub\ias-main\NgApps\admin\src\app\Store\states\ and add your-entity.state.ts file

(like C:\TylerDev\GitHub\ias-main\NgApps\admin\src\app\Store\states\data-chg-sub.state.ts).

A screenshot of a computer

Description automatically generated

**Action**

***The file declares Actions – constants, representing different directives which are dispatched in one part of the application and captured in others to execute operations.***

***Each CRUD operation of each entity has its dedicated Action.***

@ Go to C:\TylerDev\GitHub\ias-main\NgApps\admin\src\app\Store\actions and add your-entity.action.ts file

(like C:\TylerDev\GitHub\ias-main\NgApps\admin\src\app\Store\actions\data-chg-sub.action.ts).

*Firstly, the file declares a private enum each constant of which describes an Action.*

*Each description must be unique per the whole application, so its first part (“[Data Change Subscription Component]”) is the module description.*

*Notice that each main Action has its own …Success action. The Success action is dispatch if its main Action has been executed successfully.*



Then, the file declares the Actions themselves.

The props parameter to the createAction function usually describes the input and output data of the operation executed and used to enforce type safety.

The props parameter of a main Action describes the input data (example: getDataChgSubTblListAction accepts the Subscription whose Tables must be retrieved).

The props parameter of a Success Action describes the output data (example: getDataChgSubTblListSuccessAction returns the retrieved Tables list).

A close up of text

Description automatically generated

**Service**

***The file declares the class with functions which call Web Services directly.***

@ Go to C:\TylerDev\GitHub\ias-main\NgApps\admin\src\app\your\_section\_folder\services and create your-entity-services folder

(like C:\TylerDev\GitHub\ias-main\NgApps\admin\src\app\eventmodel\services\data-chg-sub-services).

@ In the services folder, create your-entity.service.ts file - the endpoints must fit YourEntityController.cs in PT

(like C:\TylerDev\GitHub\ias-main\NgApps\admin\src\app\eventmodel\services\data-chg-sub-services\data-chg-sub.service.ts).

*Notice that the http function (get, post, put & delete) must fit [HttpGet], [HttpPost] etc. in the PT’s Web API controller, as well as the URL must fit the [RoutePrefix] and [Route] directives there.*

A screenshot of a computer program

Description automatically generated

**Effect**

***The file declares the class which captures a dispatched main Action and calls the corresponding function of the Service class.***

***On successful execution, the class dispatches the respective Success Action.***

@ Go to C:\TylerDev\GitHub\ias-main\NgApps\admin\src\app\Store\effects and add your-entity.effect.ts file

(like C:\TylerDev\GitHub\ias-main\NgApps\admin\src\app\Store\effects\data-chg-sub.effect.ts).

*Notice that the parameter(s) passed to the Success Actions are forced by the props of the Actions.*

*The parameter(s) passed to the main Actions are forced by their props when those Actions are dispatched from the Component as a response to the user interactions.*

******

A screenshot of a computer code

Description automatically generated

**Reducer**

***The file declares the reducer function which captures a dispatched Action and creates a new State which reflects the result of the Action.***

@ Go to C:\TylerDev\GitHub\ias-main\NgApps\admin\src\app\Store\reducers and add your-entity.reducer.ts file

(like C:\TylerDev\GitHub\ias-main\NgApps\admin\src\app\Store\reducers\data-chg-sub.reducer.ts).

Don't forget to change the type of the states to IYourEntityState you created in your-entity.state.ts file.

*Main CRUD Actions usually just set the …Loaded flag to false to display the progress bar.*

*Success CRUD Actions usually set the …Loaded flag to true and update the State with the data returned form proc:*

* *SELECT a list populates the respective array in the State.*
* *SELECT a single entity populates the respective object in the State.*
* *INSERT refreshes the respective object populating the ID, WHO and WEN generated by the DB, and adds the entity to the list array.*
* *UPDATE refreshes the respective object populating the WHO and WEN, and finds and updates the entity in the list array.*
* *DELETE removes the DELETED entity the respective object and from the list array.*

A screenshot of a computer

Description automatically generated

**CREATE THE SCREEN'S PARENT COMPONENT (ONLY IF YOU ARE ADDING A NEW SCREEN, OTHERWISE IT ALREADY EXISTS):**

@ In C:\TylerDev\GitHub\ias-main\NgApps\admin\src\app\your\_section\_folder\components (like C:\TylerDev\GitHub\ias-main\NgApps\admin\src\app\eventmodel\components),

create the your-entity folder (like C:\TylerDev\GitHub\ias-main\NgApps\admin\src\app\eventmodel\components\data-chg-sub).

@ In it, create four the files for the parent component (the screen itself) like the data-chg-sub... files;

**CREATE A SCREEN'S CHILD COMPONENTS:**

@ Go to the screen's parent component folder - C:\TylerDev\GitHub\ias-main\NgApps\admin\src\app\your\_section\_folder\components\your-entity

(like C:\TylerDev\GitHub\ias-main\NgApps\admin\src\app\eventmodel\components\data-chg-sub).

@ In it, create a sub-folder for the new child component (the screen area)

like the data-chg-sub... sub-folders in C:\TylerDev\GitHub\ias-main\NgApps\admin\src\app\eventmodel\components\data-chg-sub.

**CHANGE EXISTING FILES:**

@ Open C:\TylerDev\GitHub\ias-main\NgApps\admin\src\app\app.module.ts

@ Find "DataChgSubComponent" (2 occurrences!).

@ Register YourEntityComponent in the same way.

ONLY IF YOU ARE ADDING A NEW SCREEN:

@ Open C:\TylerDev\GitHub\ias-main\NgApps\admin\src\app\Store\storeConfig.ts

@ Find "DataChgSubEffect" (2 occurrences!).

@ Register YourEntityEffect in the same way.

@ Find "dataChgSubReducer" (2 occurrences!).

@ Register YourEntityReducer in the same way.

@ Go to \config folder of the module (like C:\TylerDev\GitHub\ias-main\NgApps\admin\src\app\config)

@ Open pages.ts file in it (like C:\TylerDev\GitHub\ias-main\NgApps\admin\src\app\config\pages.ts).

@ Find "DataChgSubComponent" (2 occurrences!).

@ Add code which displays the link which takes to the new screen in a similar way.