Contents

[Introduction 3](#_Toc468803728)

[Pre-Requisites 3](#_Toc468803729)

[Database schema 3](#_Toc468803730)

[Tables 3](#_Toc468803731)

[Table Columns 3](#_Toc468803732)

[Client Side User Controls / Partial Views 5](#_Toc468803733)

[Entry point to Snapshots web page 5](#_Toc468803734)

[JavaScript Components 5](#_Toc468803735)

[Snapshot JavaScript file structure 5](#_Toc468803736)

[Object Models 5](#_Toc468803737)

[snapshot-filter.js: 5](#_Toc468803738)

[snapshot-model.js: 5](#_Toc468803739)

[UI Functions 5](#_Toc468803740)

[snapshots-control-ui.js: 5](#_Toc468803741)

[Core Functions 6](#_Toc468803742)

[snapshots-core.js: 6](#_Toc468803743)

[Process Flow: Capturing snapshot data 8](#_Toc468803744)

[Process Flow: Loading a Chart 9](#_Toc468803745)

[MVC Controller Events 10](#_Toc468803746)

[Data Access Objects 10](#_Toc468803747)

[SnapshotDAO 10](#_Toc468803748)

[GetSnapshotTypes 10](#_Toc468803749)

[GetPersonAndAccountStatesSnapshot 10](#_Toc468803750)

[PersonAccountStateSnapshotDAO 10](#_Toc468803751)

[GetPersonAndAccountStatesSnapshot 10](#_Toc468803752)

[Snapshot Specific TransactionType Definitions 10](#_Toc468803753)

[TransactionServiceProcess Implementations 11](#_Toc468803754)

[SnapshotProcessorBase 11](#_Toc468803755)

[CaptureAccountAndPersonStatusSnapshots 11](#_Toc468803756)

[Data Models 11](#_Toc468803757)

[SnapshotType 11](#_Toc468803758)

[DataSnapshot 11](#_Toc468803759)

[PersonAccountState 11](#_Toc468803760)

[Business Models 11](#_Toc468803761)

[ParameterTypes 11](#_Toc468803762)

[SnapshotParameter 11](#_Toc468803763)

[DateTimeSnapshotParameter 11](#_Toc468803764)

[MultiChoiceSnapshotParameter 11](#_Toc468803765)

[PersonAccountStateParameters 12](#_Toc468803766)

[Client View Models 12](#_Toc468803767)

[DataPoint 12](#_Toc468803768)

[ChartLine 12](#_Toc468803769)

[Chart 12](#_Toc468803770)

[SnapshotsViewModel 12](#_Toc468803771)

[SnapshotFilter 12](#_Toc468803772)

[SnapshotUserModel 12](#_Toc468803773)

[Creating a Data Snapshot 12](#_Toc468803774)

[Configuring the Solutions for Release 13](#_Toc468803775)

AccMan Data Snapshot Visualisations

Development Guide

## Introduction

The data snapshot visualisations component is part of the Acc Man Front End Web Application.

This guide is intended for the Software Developer working on the Acc Man Front End Web Application.

Acc Man Front End source code:

|  |  |
| --- | --- |
| Acc Man Project source: | Bit Bucket (stash):  <https://aaa45816@stash.salford.ac.uk:8447/scm/ac/accman.net.git> |
| Branch | feature/ARF-404-capture-account-person-status-data |

## Pre-Requisites

Visual Studio 2015

.NET Framework 4.5.2

MySQL AccMan database.

jQuery 1.4.2

jQuery UI 1.8.4

ChartJS

# Database schema

## Tables

|  |  |
| --- | --- |
| **Table** | **Description** |
| **snapshot\_types** | A list of unique types of snapshot e.g. Person account states |
| **data\_snapshot** | Unique snapshot instances taken on each scheduled capture. |
| **Account\_statuses** | A list of unique states for an account. |
| **Person\_statuses** | A list of unique states for a person. |
| **Account\_person\_state\_snapshot\_entry** | Snapshot data for account vs person statuses on each account / person in the AccMan database on that snapshot instance. |

## Table Columns

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Table: snapshot\_types** | | | | | | | |
| Name | Type | Size | Primary Key | Auto Increment | Not Null | Default/Expression | Unique |
| Id | Mediumint | 9 | Yes | Yes | Yes |  |  |
| Type | Varchar | 30 |  |  | Yes | ‘’ | Yes |
| Name | Varchar | 45 |  |  |  | ‘’ | Yes |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Table: data\_snapshot | | | | | | | |
| Name | Type | Size | Primary Key | Auto Increment | Not Null | Default/Expression | Unique |
| Id | Mediumint | 9 | Yes | Yes | Yes |  |  |
| Snapshot\_type\_id | Mediumint | 9 |  |  | Yes |  |  |
| Snapshot\_datetime | datetime |  |  |  |  | Current\_timestamp |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Table: account\_statuses | | | | | | | |
| Name | Type | Size | Primary Key | Auto Increment | Not Null | Default/Expression | Unique |
| Id | Mediumint | 9 | Yes | Yes | Yes |  |  |
| Account\_status | Varchar | 9 |  |  | Yes | ‘’ | Yes |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Table: person\_statuses | | | | | | | |
| Name | Type | Size | Primary Key | Auto Increment | Not Null | Default/Expression | Unique |
| Id | Mediumint | 9 | Yes | Yes | Yes |  |  |
| Person\_status | Varchar | 10 |  |  | Yes | ‘’ | Yes |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Table: Account\_person\_state\_snapshot\_entry | | | | | | | |
| Name | Type | Size | Primary Key | Auto Increment | Not Null | Default/Expression | Unique |
| Id | Mediumint | 9 | Yes | Yes | Yes |  |  |
| Snapshot\_id | Mediumint | 9 |  |  | Yes |  |  |
| Account\_status\_id | Mediumint | 9 |  |  | Yes |  |  |
| Person\_status\_id | Mediumint | 9 |  |  | Yes |  |  |
| Count | Mediumint | 9 |  |  | Yes |  |  |

# Client Side User Controls / Partial Views

## Entry point to Snapshots web page

The user is able to access the snapshot visualisations from the AccMan main menu > Health Checks > System Status.

On the System Status page, the Snapshots tab allows the user to see the visualisations.

# JavaScript Components

## Snapshot JavaScript file structure

In the project AccMan root, the Scripts folder contains the following files used for snapshot data visualisation:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Scripts*\ | | | | | |  | |  |  |
|  | *common*\ | | |  | | | | |  |
|  |  | **date-helper.js** | | | | | | |  |
|  | *snapshots*\ | | |  | | | | |  |
|  |  | *model*\ | | |  | | | | |
|  |  |  | **snapshot-filter.js** | | | | | | |
|  |  |  | **snapshot-model.js** | | | | | | |
|  |  | **snapshots-control-ui.js** | | | | |  | | |
|  |  | **snapshots-core.js** | | | | |  | | |

## Object Models

The client side Snapshots functionality relies on some JavaScript objects which are used to defined parameters for a visualisation and the snapshot visualisation itself.

These are transmitted as JSON to the controller actions to generate the required snapshot data model.

### snapshot-filter.js:

A snapshotFilter represents a user selection in the snapshot parameters.

The functional initialiser of a snapshotFilter object sets default values to the properties of the snapshotFilter.

The FilterGroupName serves purpose for both the client side control id/name and for the field name when posted to the controller action.

### snapshot-model.js:

A snapshotModel represents the snapshot type and it’s filters.

The controller action SystemController.GetSnapshotData expects this model structure as its parameter. The simple object snapshotModel is the structure which specifies the snapshot type id and the filters selected by the user.

## UI Functions

### snapshots-control-ui.js:

Functions in this file should only manipulate ui controls for the snapshots components.

**Page load/ready**

*Performs event binding to page elements.*

Page load initialisation is handled in this script using the jQuery document ready shorthand. Within this function, the following happens:

1. The snapshot drop down list is bound to change event to retrieve the parameter controls ui.
2. The view-snapshot-button button is bound to on click event to call the LoadSnapshot function.

**SetParameterControls(html)**

Renders UI controls to allow the user to select snapshot parameters.

This function constructs and appends html to the snapshot user control page. This will serve as the container for the snapshot parameters user controls.

## Core Functions

### snapshots-core.js:

**GetSnapshotParametersControls(callback, snapshotId)**

Retrieves UI controls to allow the user to select snapshot parameters.

This calls the snapshot controller action GetSnapshotParametersControls in order to obtain the html for the snapshot visualisation parameters. The user is then able to select parameters for this specific snapshot type.

In order to maintain separation between ui and core logic, the success function of this methods’ internal ajax call fires the callback function. It is expected by design this callback does something with the html, in this case it just renders it inside the placeholder div ‘snapshotParameters’.

**GetUserFilters()**

Collects the users’ snapshot parameter selections as a snapshotFilter array.

The snapshot parameters controls are rendered with custom data-\* attributes.

These attributes are

* data-snapshotelement=param
* data-paramtype=datetime

Various other ones may be added/used as required, noting not to conflict with any other JavaScript framework/component data-attributes.

The filters are collected by first checking for checkboxes with the data attribute ‘data-snapshotelement’ with value: ‘param’.

Then the text boxes are collected and converted into the same snapshotFilter model as the checkboxes.

**LoadSnapshot()**

Collects user selected parameters/filters and retrieves the chart data from the SystemController.

GetUserFilters is called and then passed to the SystemController as JSON parameter. The resulting chart data is passed into the LoadChart(data) function.

**LoadChart(chartData)**

Parses the chart data to construct a charts labels, data points and axis values.

With the chart data received, this function generates a chart model ‘lineChart’ with the labels to display as a parameter.

Then for each chart label, the data points, label caption and cosmetic values are passed to the addLineToChart function.

**lineChart(labels)**

A ChartJS object defining the third party components required config model for a chart.

This function returns a chart instance as a config object with property names required by ChartJS.

**addLineToChart(chartConfig, lineDataPoints, lineColor, lineLabel)**

Adds a line to the line chart object config by setting XYDataPoint data and labels.

This function essentially populates the ChartJS configs’ data.datasets property by adding each line’s data, label and colour.

# Process Flow: Capturing snapshot data

This process() is called within each ConsoleSchedulerService schedule trigger event.

**Process()**

Get Snapshot Data

Create data\_snapshot record

Insert snapshot data records

**SnapshotProcessorBase**

PrepareSnapshotEntry()

**SnapshotDAO**

Get Snapshot type specific data

# Process Flow: Loading a Chart

LoadSnapshot detail

**Render chart**

**Add chart data**

**Configure chart**

**Delegate(filters) based on snapshot type**

**GetSnapShotData**

SystemController

**GetUserFilters**

**Set snapshot model filters**

**LoadChart data**

**Select Snapshot type [user]**

Retrieve parameters ui controls.

**Page load [mvc]**

Bind events to snapshot dropdown

Bind event to Go button.

**Set parameters [JS]**

**Go (Show chart) [User]**

LoadSnapshot

# MVC Controller Events

The SystemController is responsible for serving the initial page to load the Snapshot data visualisations.

The sequence of actions is as follows:

1. .NET MVC first receives the request to serve the System HealthChecks view. Within the HealthChecks view, the Snapshots link anchor control allows the user to load the snapshots controls.
2. The snapshots controls are delivered from the Snapshots PartialView action.
3. After the user selects the Snapshot type the System controller then returns the GetSnapshotParametersControls PartialViewResult object “~/views/system/snapshot/parameters.ascx”. This partial view renders the user selectable filters for the selected snapshot type.
4. The System controller then handles the request to load a visualisation by calling the GetSnapshotData JsonResult method/action. This calls the relevant delegate snapshot data implementation and returns the visualisation data as JSON.

# Data Access Objects

## SnapshotDAO

The purpose of the SnapshotDAO is to interact with the Snapshot data tables.

### GetSnapshotTypes

To get the Snapshot types available as stored in the database, call GetSnapshotTypes. This will return list of type SnapshotType.

### GetPersonAndAccountStatesSnapshot

When a snapshot is required this method will call a query on the Acc Man database person and account tables summarising the person to account state combinations currently held. The resulting object is list of type PersonAccountState.

## PersonAccountStateSnapshotDAO

### GetPersonAndAccountStatesSnapshot

This method is responsible for grouping Person and Account state combinations by types and returning a list of type PersonAccountState. The list represents the person state, account state and count of occurrences in the Acc Man database at the time of execution. This data is intended for the insertion into the account\_person\_state\_snapshot\_entry database table.

# Snapshot Specific TransactionType Definitions

In uk.ac.salford.accman.Models.Enumerations.TransactionType enumeration, add a new definition to represent the new snapshot type being implemented. These definitions are required to allow ConsoleSchedulerService to process the corresponding snapshot Quartz job.

Current snapshot specific TransactionType definitions:

135 - CAPTURE\_ACCOUNT\_AND\_PERSON\_STATUS\_SNAPSHOTS

# TransactionServiceProcess Implementations

These implementations are so that console scheduler service can correctly load the class configured in the Quartz job to query the snapshot data and to write it to the data\_snapshot and specific table(s).

## SnapshotProcessorBase

Use this class to derive a new snapshot specific TransactionServiceProcess type. An example is CaptureAccountAndPersonStatusSnapshots.cs.

This base class offers generic methods to create a data\_snapshot record and to clean up too in case of exceptions.

## CaptureAccountAndPersonStatusSnapshots

The account and person state summary snapshot data is processed in this class. Specifically the process override handles the creation of the data\_snapshot record and the specific entries in account\_person\_state\_snapshot\_entry.

# Data Models

## SnapshotType

A SnapshotType

## DataSnapshot

A DataSnapshot defines the snapshot id, the type and when it was captured. This reflects the data table columns of data\_snapshot.

## PersonAccountState

This is related to the snapshot data capture of the person and account state. It defines the snapshot date time and the account and person state as well as its total count. This model reflects the data table columns of account\_person\_state\_snapshot\_entry.

# Business Models

## ParameterTypes

This model class defines the code interface and implementations of the type of visual controls that a user can see for a specific snapshot visualisation. These are generally serialised to JSON for client side code usage.

### SnapshotParameter

Any parameter on the UI which can be rendered and used like or as a text box input/ free text.

### DateTimeSnapshotParameter

Child class defining itself as a date time specific field for user entry of a date.

### MultiChoiceSnapshotParameter

Exposes a collection property ‘Options’ to allow user to select from a choice of values for the snapshot parameter.

## PersonAccountStateParameters

Simple class defining a list of type ISnapshotParameter as ‘Parameters’. An instance of this class can contain any type of ISnapshotParameter type object in ‘Parameters’ thus allowing many different parameters UI control types on the client side.

# Client View Models

After the business logic processes service requests, the client view models are constructed and populated to meet the structure expected by client side logic. The following view models are required and used by the front end controls.

## DataPoint

A DataPoint class object specifies the intersection of two axis x and y for the purpose of visual charting.

## ChartLine

A ChartLine object is a collection of DataPoint objects along with any visual configuration e.g. colour, label.

## Chart

A Chart object is a collection of the data items e.g. ChartLine and XLabels; a collection of labels to use to depict the values of the X-axis items.

## SnapshotsViewModel

A simple collection of SnapshotTypes defined as the SnapshotTypeId and its string text representation e.g. [1,”PersonAccountStates”].

## SnapshotFilter

This is the definition of a filter for which the snapshot should apply as a data filter on the snapshot data.

## SnapshotUserModel

A collection of filters for a snapshot data set and the associated Snapshot Id.

# Creating a Data Snapshot

1. Create a new TransactionType definition.
2. Implement a TransactionServiceProcess class to collect the data similar to CaptureAccountAndPersonStatusSnapshots.
3. In the TransactionServiceProcess implementation override the Init method to set the TransactionType to the new enum definition.
4. In the Quartz config file, add a new job configuration to enable ConsoleSchedulerService to create the snapshots.
5. Override the Process method.
6. Implement a snapshot specific method in SnapshotDAO similar to GetPersonAndAccountStateSnapshot.
7. Back in the TransactionServiceProcess implementation, in the Process override, call this new SnapshotDAO method specific to this snapshot type. This will return the data to capture.
8. Handle data\_snapshot parent record creation and clean up in case failure.
9. Write to the data\_snapshot table.
10. Write the snapshot data returned from the specific SnapshotDAO method snapshot specific database table.

# Configuring the Solutions for Release

Both .NET Code Library (AccMan Back End) and AccMan.Net Front End will require building and releasing for any new visualisations.

1. Ensure the Quartz config file contains the new configuration for the new snapshot capture TransactionServiceProcess implementation.
2. Build the .NET Code Library solution
3. Build the AccMan.NET solution
4. Set the Web.Config of AccMan.NET to log the messages accordingly as required.
5. Set the App.Config of ConsoleSchedulerService app to log the messages accordingly as required and also for the new TransactionType definition. (message log path set in quartz config for the specific TransactionType).