trackmatic

Trackmatic

Integration Requirements Specification Document

trackmatic

Table of Contents

trackmatic

What we do

In short, Trackmatic provides unique tailor-made software solutions to fleet operators and fleet management of On-Road Execution™ regardless of the fleet size.

We offer a holistic business solution to our clients, meeting their unique and complex requirements. We work together with them to provide insight into the finer workings of their operations, thereby increasing efficiencies and enabling greater levels of satisfaction among their customers.

Resource optimisation and service excellence are key outcomes of the solution, resulting in higher profits and driving down costs. This is where the true value of the solution is gained.

How to use this guide

The graphical process shown below depicts the various sections and parts of the document which relate to the way in which you can integrate with trackmatic.

Following each node leads to the next process to be completed until having a smooth-running integration. The process of following a path in the diagram below shows what each node allows for the next steps to be followed.

Top left corner of the box shows which section to refer to while at the left of each box shows its relative part heading it falls under.

Sample scenario:

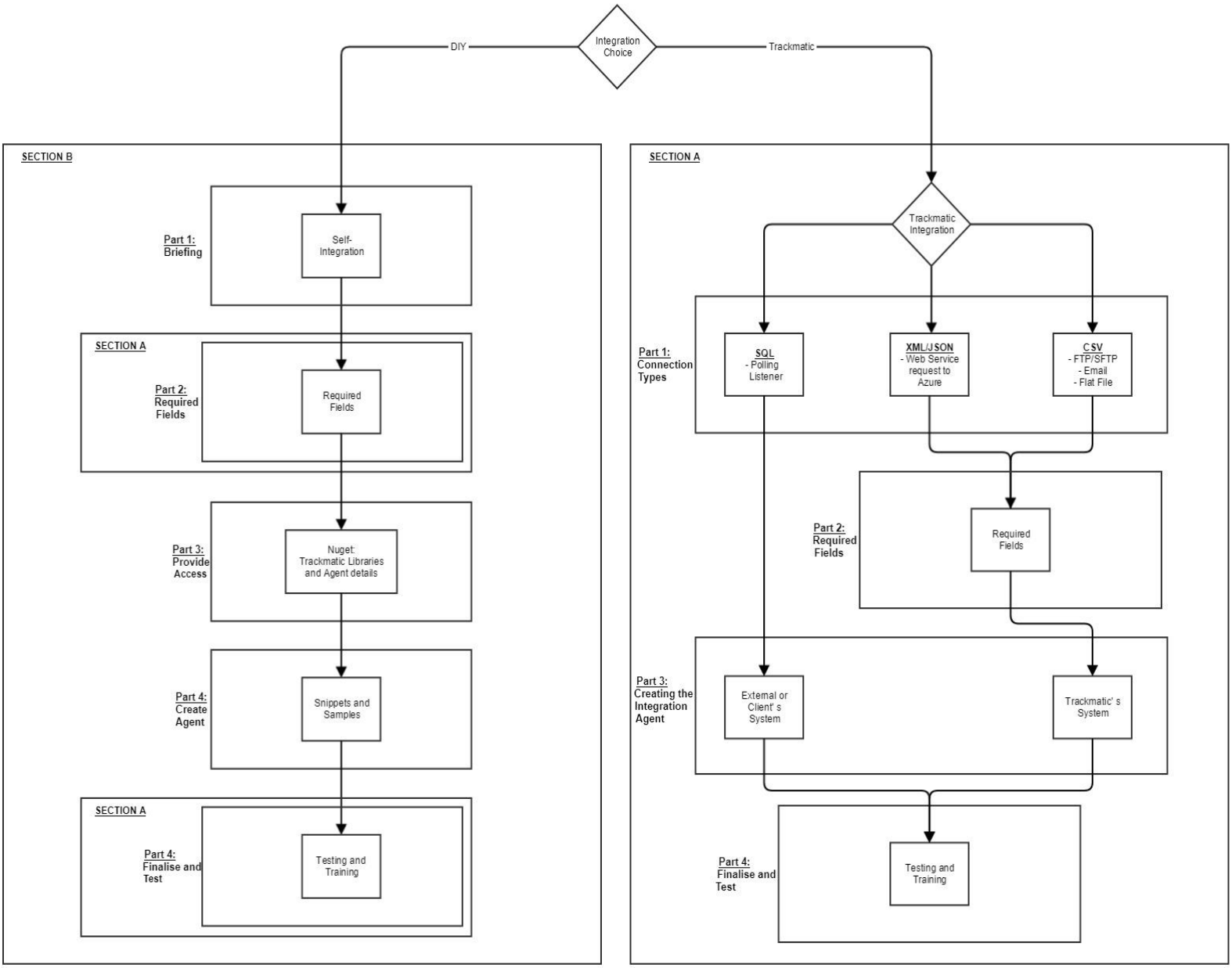
Integration with trackmatic where trackmatic creates the integration using a CSV file, connecting via Email to push data is in the following path:

- Trackmatic-> Trackmatic Integration -> CSV Email-> Required Fields -> Trackmatic' s System -> Testing and Training

trackmatic

Our graphical process used to integrate:

Follow a path and make reference to each section, part and label to complete the integration process.



trackmatic

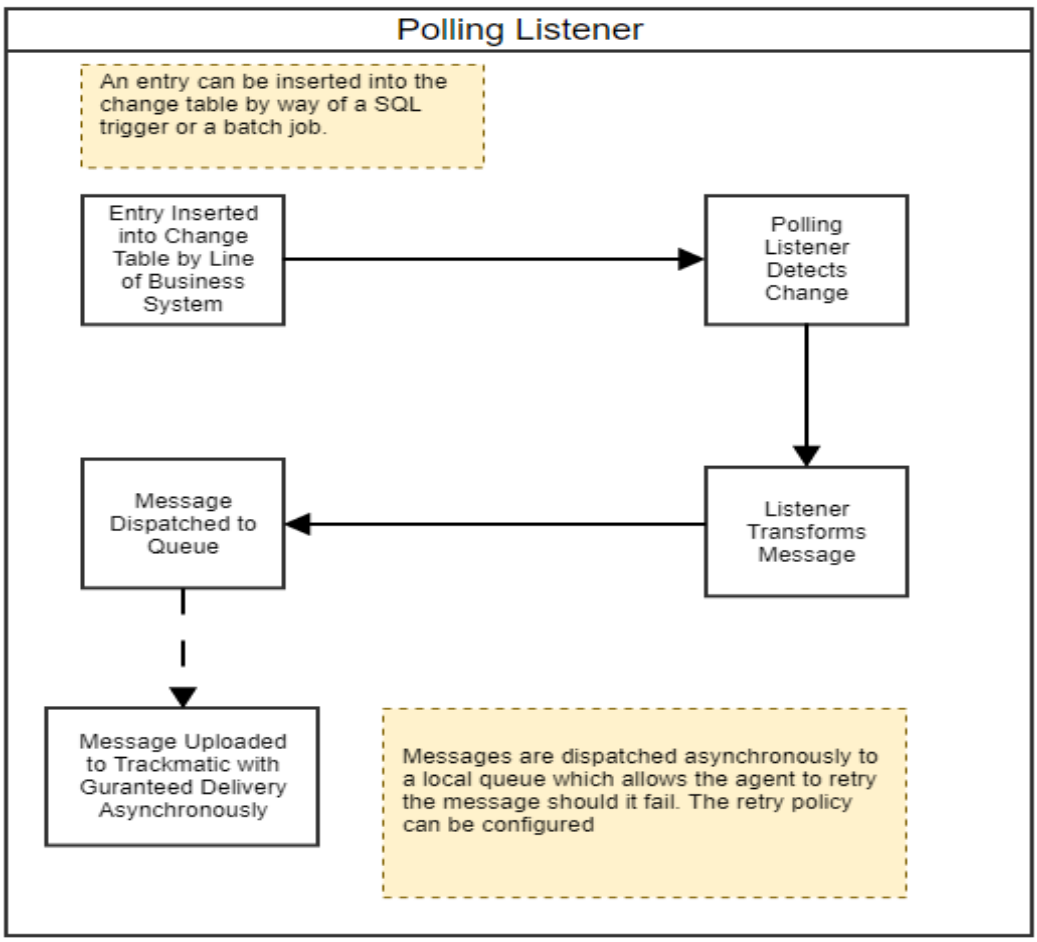
Section A

Part 1: Connection Types

SQL

Polling Listener

This is installed as a service on the client's local server or machine. It listens to a 'Trackmatic-Changes' table placed within the client's database. This table records all the changes been made to the relevant client's tables needed to use Trackmatic' s services. These changes in relation to the tables are then traced, picked up and updated in Trackmatic.

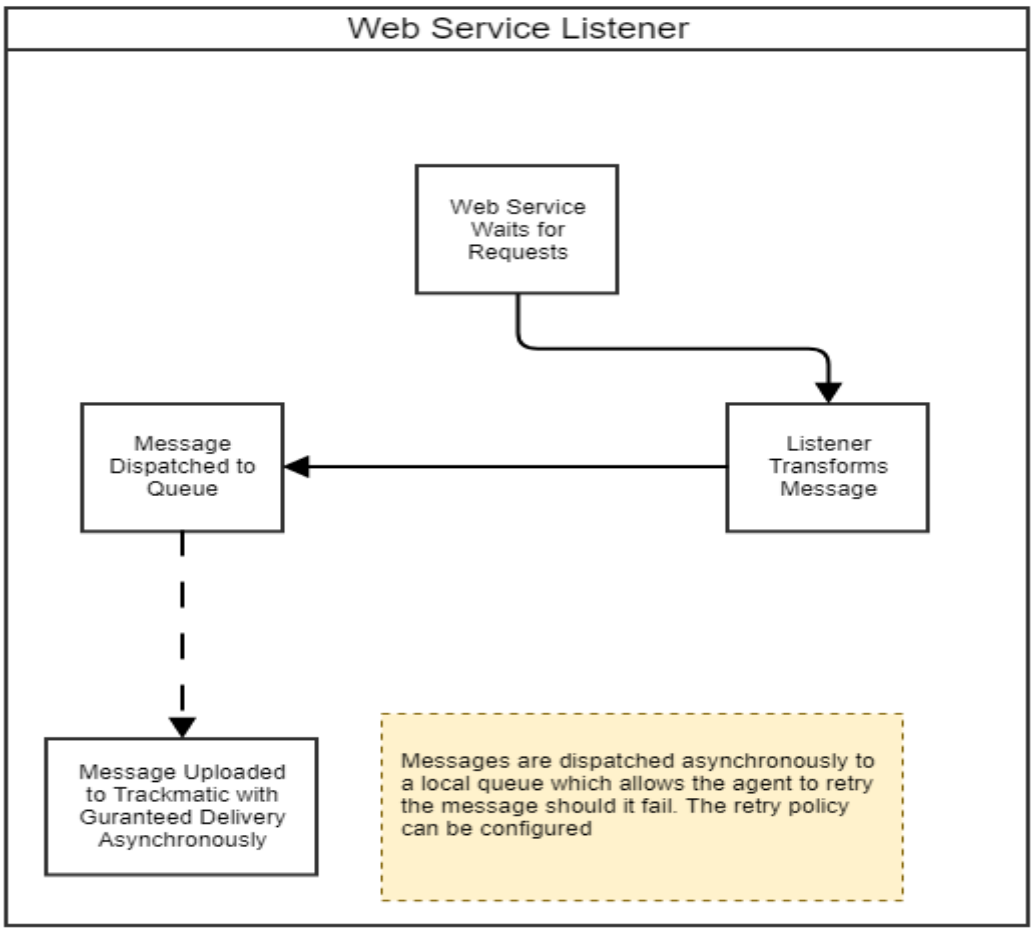


trackmatic

XML or JSON

Web Service Request To Azure

The client presents the required data to our web service in XML or JSON. Upon receiving it, it is then mapped and uploaded into Trackmatic.

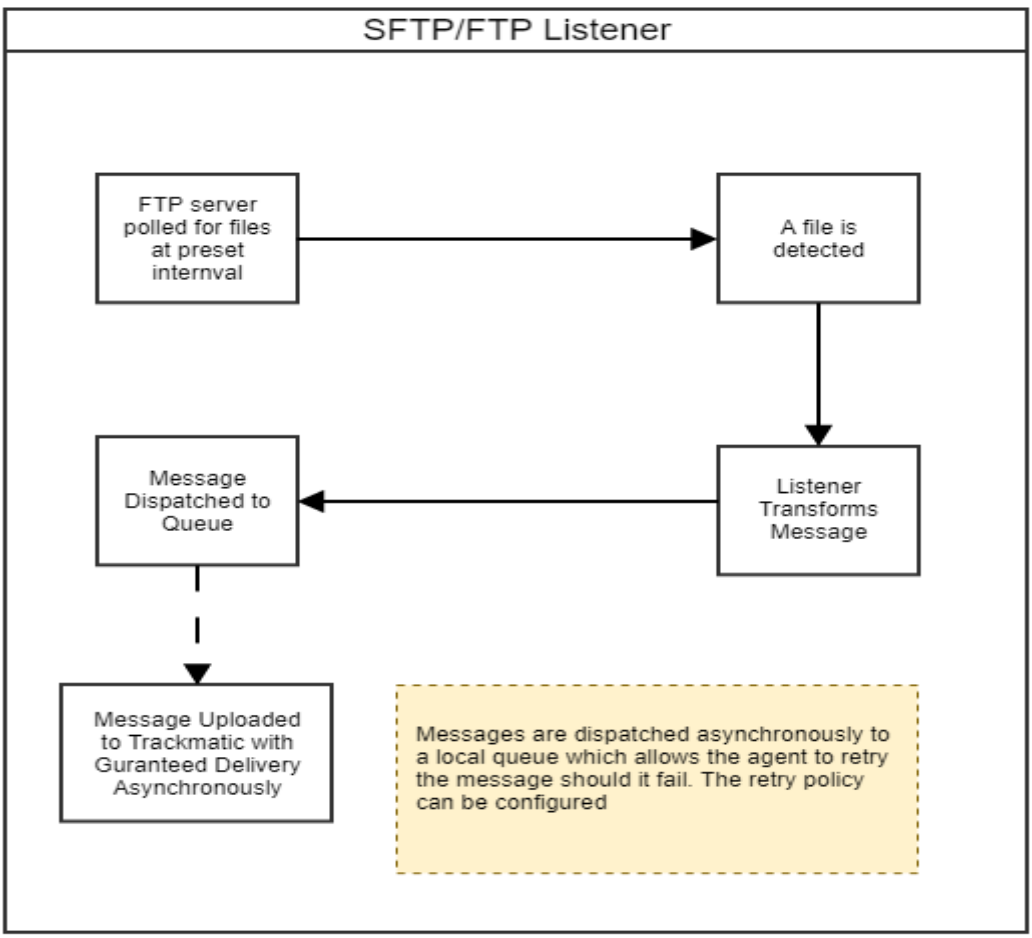


trackmatic

CSV

SFTP/FTP

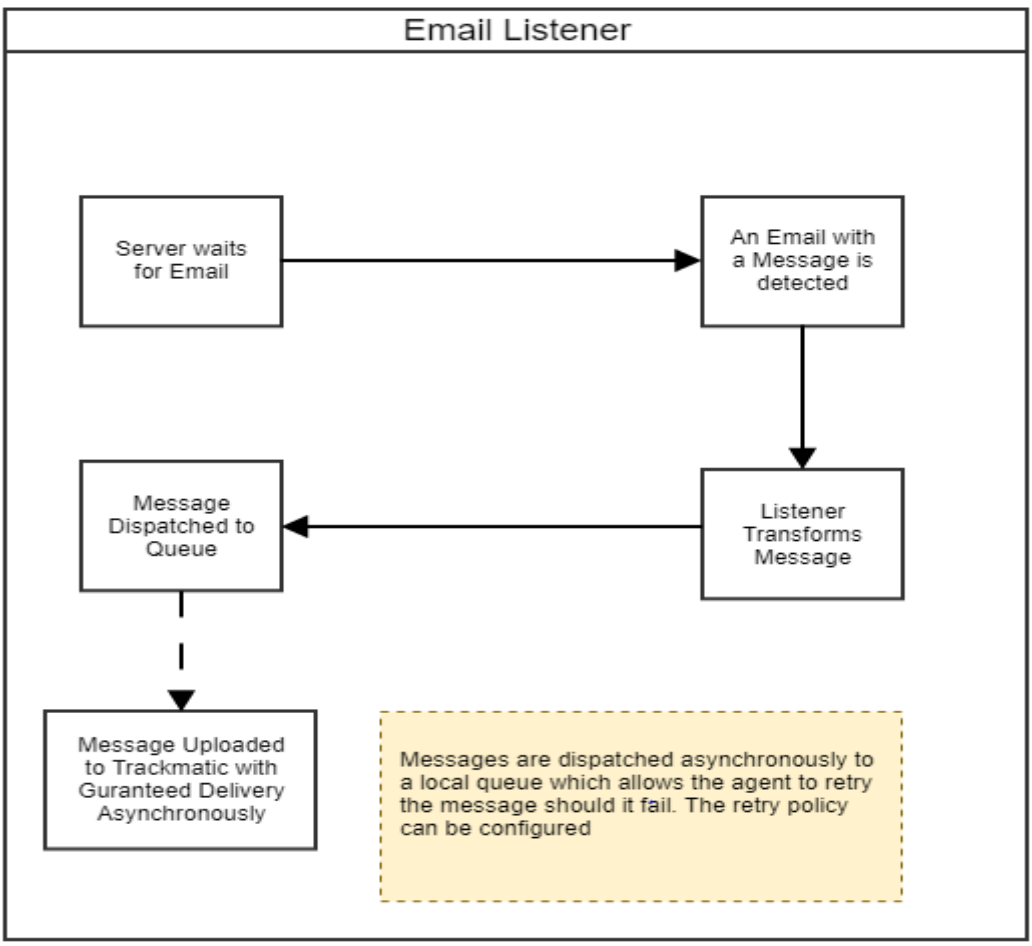
The client drops an excel extract (CSV) to the SFTP/FTP server containing all the relevant data to use Trackmatic services. The integration agent polls the directory of the SFTP/FTP for the file. If an extract is picked up, it is then read in and uploaded into Trackmatic. The client can choose to use their own SFTP/FTP server (preferable) or Trackmatic' s SFTP/FTP server.



trackmatic

Email Listener

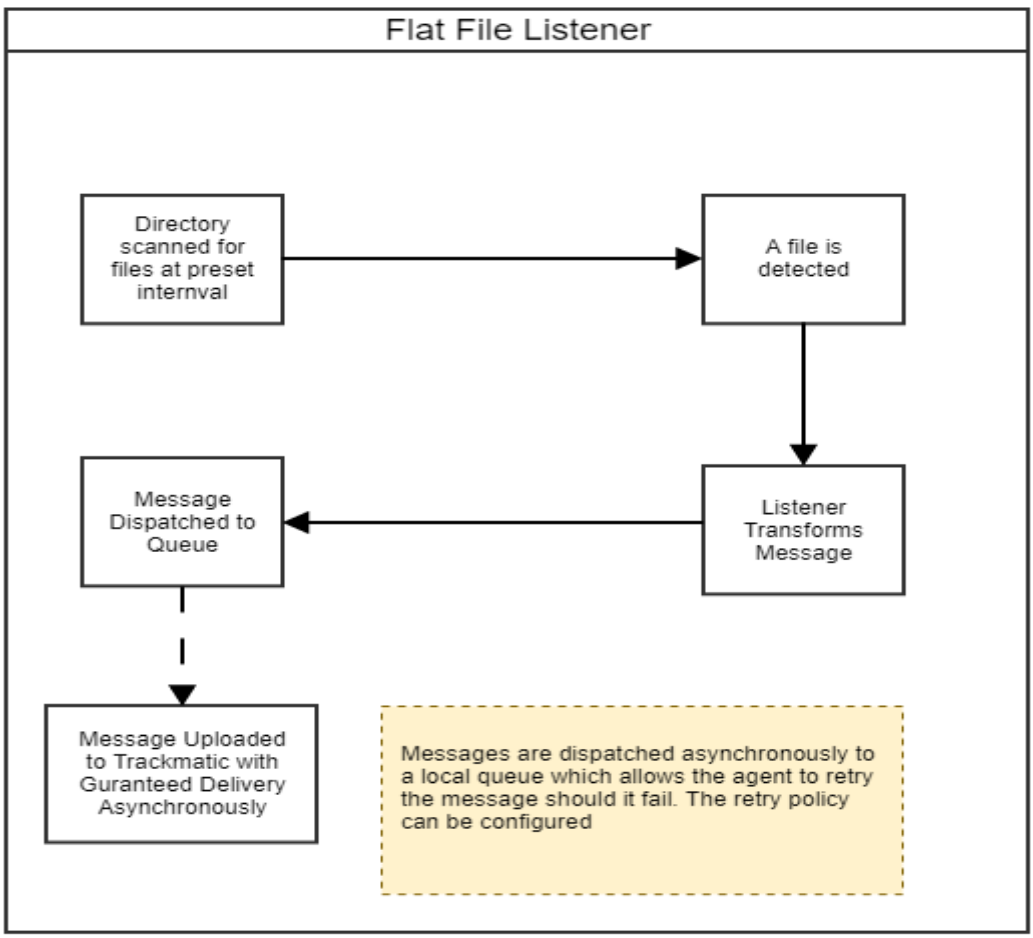
The client emails an excel extract (CSV) containing all the relevant data to use Trackmatic' s services to an email address provided by Trackmatic. The integration agent which is running on Trackmatic' s server will then pick up this email. If an extract is picked up within the email, it will then be read in and uploaded into Trackmatic.



trackmatic

Flat file Listener

Very like SFTP/FTP, the client drops an excel extract (CSV) containing all the relevant data to use Trackmatic' s services onto a local directory in Trackmatic' s server. The integration agent which is running on Trackmatic' s server will poll the directory for the file. If an extract is picked up, it is then read in and uploaded into Trackmatic.



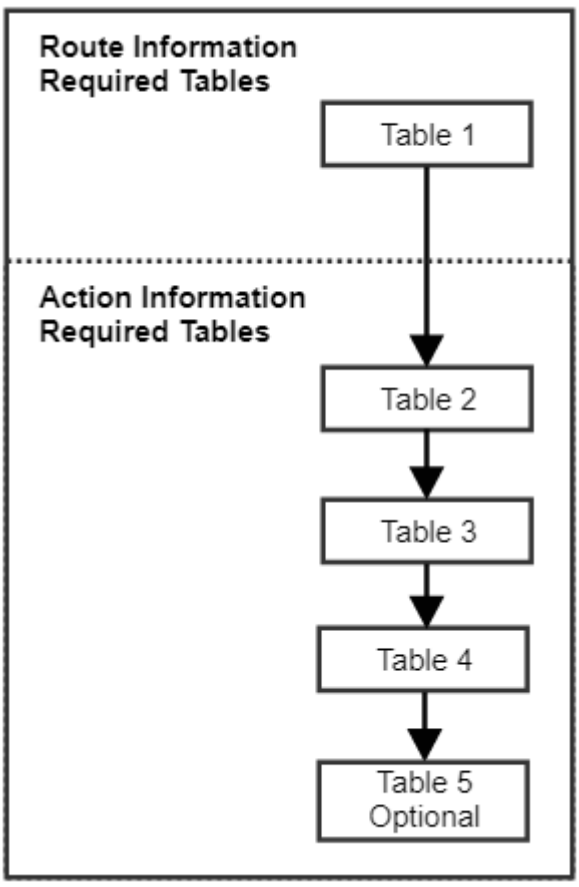
trackmatic

Part 2: Required Fields

Below shows a diagram of the fields that will be required depending on the data you have.

A simple scenario of the two cases:

1. **Route information** may include the data received from a Planning tool which provides you with a vehicles route and stops to be made for a specific period. This also including action information defined in point 2 below.
2. **Action information** includes data about the stop to be made. This may be invoice data, delivery notes, interbranch transfers, collections etc. Additional information with this is required such as the address of the stop to be made as well as any extra functionalities that trackmatic can provide per stop. These extra functionalities can be seen in the required fields.



trackmatic

ROUTE INFORMATION

Route information is comprised of four fundamental sets of data and one optional:

Table 1: Route – This is the header information for the sequence of stops for a vehicle within an allocated period.

Table 2: Action – Type of stop to be made. (Delivery, Collection, IBT, etc)

Table 3: Entity – This is essential the sell to.

Table 4: Deco – This is essential the ship to.

Table 5: Handling Units – Individual items been delivered (optional)

Table 1:

ROUTE			
A basic example of a route in trackmatic is the travel path in which a delivery is to be made.			
	Field Name	**Mandatory**	**Description**
1	Reference	Yes	A unique reference number for the route
2	Planned Start	Yes	Planned start date and time
3	Registration	No	Registration number of the vehicle being assigned to the route
4	Name	No	A name or number of the route
Crew			
5	Reference	Yes	Personnel Unique Identifier issued by Client
6	Name	Yes	Personnel Name
7	Type	Yes	Personnel Type (Driver or Crew)
8	Identity No.	No	Identity number of crew member
9	Cell No.	No	Cell phone number of crew member

trackmatic

ACTION INFORMATION

Action information is comprised of three fundamental sets of data and one optional:

Table 1: Action – Type of stop to be made. (Delivery, Collection, IBT, etc)

Table 2: Entity – This is essential the sell to.

Table 3: Deco – This is essential the ship to.

Table 4: Handling Units – Individual items been delivered (optional)

Table 2:

ACTION			
A basic example of an action in trackmatic is an invoice along with its details for who the customer is for.			
	Field Name	**Mandatory**	**Description**
1	IsCod	Yes	Cash On Delivery Indicator
2	Reference	Yes	Unique reference number associated with the action
3	Volumetric Mass	No	The volume of the parcel
4	Weight	No	Weight of the parcel
5	Pieces	No	Number of pieces in the action
6	Unit	No	Unit of measure
7	Instructions	No	Special instructions
8	Customer Reference	No	Client supplied reference number
9	Customer Code	No	Unique client code
10	Expected Delivery	No	Date the action is expected to be executed
11	Pallets	No	Number of pallets associated with the action
12	Amount Incl	No	Monetary value of the action including vat
13	Amount Excl	No	Monetary value of the action excluding vat
14	Reference_Internal	No	Internal reference number for workflow i.e. picking slip, sales doc etc

trackmatic

Table 3:

ENTITY			
A basic example of an entity in trackmatic is the individual stores/customers details within an area(Edgars in Mall Of Africa).			
	Field Name	**Mandatory**	**Description**
1	Name	Yes	The name of the entity
2	Reference	Yes	A unique reference number for the entity (Sell-to identifier)
Contact			
3	First Name	No	First name of the contact
4	Last Name	No	last name of the contact
5	Tel No	No	Telephone number of the contact
6	Cell No	No	Cell phone number of the contact
7	Email	No	Email address of the contact
Requirements			
8	Action Debrief	No	Success or Failure of the action that was to be executed.Per entity, it can be stated if you would like an entity debrief.
9	Cod Debrief	No	Review of Cod deliveries 1) EFT POP 2) Cash 3) No Cash 4) Accouting Pin. Per entity, it can be stated if you would like a Cod debrief.
10	Signature	No	Sign On Glass required after an action is debriefed. Per entity, it can be stated if you would like the signature feature.

trackmatic

Table 4:

DECO			
A basic example of a Deco in trackmatic is the area/location of the entities (Mall Of Africa has entities like Edgars, Woolworths etc.).			
	Field Name	**Mandatory**	**Description**
1	Reference	Yes	A unique reference number (Ship-to Identifier)
2	Name	Yes	Common name of the DECO
Address			
3	AddressId	Dependant	Mandatory if 3PL or you may have multiple delivery addresses
4	IsAdhoc	Yes	Once of stop
5	Unit No.	No	Unit No.
6	Building Name	No	Building Name
7	Street No.	No	Street Number
8	Sub Division No.	No	Sub Division Number
9	Street	No	Street Number
10	Suburb	No	Street
11	Province	No	Province
12	Postal Code	No	Postal Code
13	Map Code	No	Map Code
14	Latitude	No	Latitude
15	Longitude	No	Longitude
16	MST	No	Maximum stop time

trackmatic

Table 5: optional

Handling Units			
A basic example of a handling unit in trackmatic is the individual items/materials/goods that are been delivered.			
	Field Name	**Mandatory**	**Description**
1	Barcode	Yes	Barcode of the handling unit
2	Unit of measure	Yes	Describes the type of handling unit (carton, parcel, box)
3	Customer Reference	No	Client supplied ref number
4	Quantity	No	Amount of parcels in the handling unit
5	Weight	No	Weight of the handling unit
6	Status	No	Status of the handling unit (pending, endorsed, rejected, missing)
7	Description	No	Description of handling unit
8	Volume	No	Volume of the parcel
9	Volumetric Mass	No	Volumetric mass of the parcel
Dimensions			
10	Height	No	Height of handling unit
11	Width	No	Width of handling unit
12	Length	No	Length of handling unit

trackmatic

Part 3: Creating the Integration Agent

Trackmatic' s System

The pre-requisites of this stage are to:

1. Have a chosen Connection Type (Part 1)
2. Populate the Required Fields (Part 2) in a sample file that will be in the format relative to your choice in the above point then provide it to the trackmatic professional developers leading the integration.
3. The sample file should be accompanied by its relevant documentation (invoices, delivery notes, trip sheets etc.) for data integrity measures.

Note:

- This is the longest process of integration and a specific timeline can be acquired from the project manager showing the tasks that is needed to be completed.

- Due to ensuring that this integration is done efficiently and effectively whilst bearing in mind that every client is unique in the way they carry out their processes, trackmatic works iteratively to meet the needs and functionalities that we can provide to our client. Therefore, key personnel should be assigned the task of working and communicating with a trackmatic professional to get the best out of our services.

After having met the pre-requisites, Trackmatic professional developers will create a baseline agent in which users of trackmatic can provide the data to pushed and uploaded into trackmatic' s API. This baseline agent will be iteratively worked till all functionalities are catered for.

Depending on the Connection Type chosen, Trackmatic will provide you with credentials and details for the system you will use (URL, Email, FTP/SFTP) to push data to.

External or Client' s System

1. If trackmatic is to integrate into external systems, then Trackmatic will need to be provided with the details for the systems. For example, IP addresses, port numbers, logon credentials and firewall rules etc.
2. Guidance on where we can obtain the Required Fields (Part 2) will be needed.
3. Relevant documentation (invoices, delivery notes, trip sheets etc.) in relation to the Required Fields (Part 2) will be need to be provided for data integrity measures.

trackmatic

Part 4: Finalise and Test

Testing and Training

Upon completing the task of Creating the Integration Agent (Part 3), automation of providing the file to the system we have provided (URL, Email, FTP/SFTP) will need to be set up from the client's side.

Once data is being uploaded regularly, training and testing of data integrity as well as functionalities can commence:

- Training on how to use trackmatic and all its features can be discussed with the project manager leading the integration. Training can only commence once data is being uploaded into trackmatic' s API on its appropriate time intervals.
- During the testing and training phases, any discrepancies, bugs or additional features to be made will be applied and re-tested with the new implementation of these changes. Once again, this task may be iterative to accommodate and fix and unforeseen circumstance that may arise.

Finally, if both the client and trackmatic is pleased with the outcome of the integration, where-by the focus points and business requirements are met, we can then move over into live production. This will be scheduled to occur over a controlled period, preferably when the client is least busy. It is advised to have both the staged and live environment run concurrently until such the live production is stable.

trackmatic

Section B

Part 1: Creating Your Own Integration Agent

Briefing

When creating your own integration, make sure you fully understand the requirements of what data needs to be pushed into trackmatic.

Creating the integration agent process works as follows:

- Prepare the fields needed as can be seen in Section A – Part 2
- Select the way in which you want to push data (read from CSV, database etc.)
- Create baseline code that will read in the fields in the format you provide
- Add trackmatic' s libraries and authentication details as in part 3 below
- Create the model and transform it into trackmatic' s format as in part 4 below
- Place agent in secure environment which is stable
- Test and confirm data been sent to trackmatic

If there are any issues when trying to create your own integration to trackmatic then feel free to contact any of the trackmatic professional who are part of the handling the project.

Part 2:

Same as Section A – Part 2

trackmatic

Part 3: Authentication

Provide Access

Upon completing the task of determining which Required Fields (Section A – Part 2) is needed, you will need to start creating your own integration.

In order to make use of our framework as well as push data to Trackmatic' s API, you will require access to its libraries and authentication to its API:

1. Trackmatic' s libraries are accessible from NuGet use the following link: http://nuget.trackmatic.co.za/nuget
2. Packages you will need to make use of are:
   - Trackmatic.Rest.Client
   - Trackmatic.Api.Agent.Host
   - Trackmatic.Api.Agent
   - Trackmatic.Api.Agent.Polling use if you are pulling data directly from a database. Trackmatic to provide Scripts if needed to use our triggers and table to listen to upon a change or insert.
3. Trackmatic will provide you with the following when authenticating to push data to our API:
   - Client ID
   - Integration Username
   - Integration Password
   - Api Address: https://rest.trackmatic.co.za/api/v1

trackmatic

Part 4: Create Agent

Snippets and Samples

Authentication:

Authentication should be placed in the configuration file as follows:

Authentication

```
<appSettings>
    <add key="api/address" value="https://rest.trackmatic.co.za/api/v1" />
    <add key="api/username" value="0000000000999" />
    <add key="api/password" value="PassW999" />
</appSettings>
```

The authentication will occur in the background of the API which is built into the interface IAgent which will be used by the listener. All other instances can also be injected to the listener. Below depicts how the listener class should implement this:

Implement IAgent

```
public class Listener : BaseListener
{
  Public Listerner(IAgent agent) : base(agent)
}
```

trackmatic

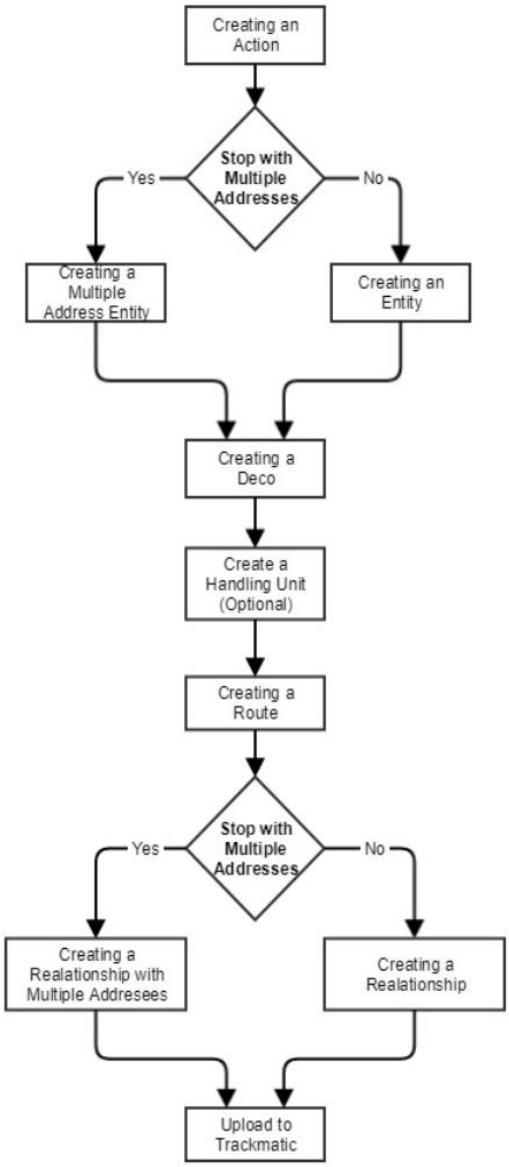
Sample Model:

Note:

This shows a sample of how these models are created.

The model been created is dependent on Section A - Required Fields (Part 2).

The below diagram shows the flow to be taken of which models to use for each type of stop (one to one, multiple address, adhoc). Each node in the below diagram represents the label of the snippets of the models below.



trackmatic

- **Creating an Action:**

Create An Action

```
private Action CreateAction ()
{
  return new Action
  {
    Id = //string Required: Action reference prefixed with Client Id,
    ClientId = //string Required: Client Id,
    Name = //string: Name given to this action,
    Reference = //string Required: Line-of-business generated reference number for this action,
    Nature = //anything Required: Valid values: "Unknown" "Product" "Service",
    IsCod = //boolean Required: This is true if cash on delivery else false,
    VolumetricMass = //double : VolumetricMass of the item,
    Volume = //double: Volume of the item,
    Weight = //double: Weight of the item,
    Pieces = //int: Number of pieces,
    Unit = //string: Unit of measure,
    Instructions = //string: Specify any special (delivery) instuctions, pertaining to this action,
    InternalReference = //string: Internal reference number for workflow i.e. picking slip, order number, sales doc etc,
    CustomerReference = //string: Reference number the customer uses to internally refer to this specific action,
    CustomerCode = //string: A reference to the customer,
    ExpectedDelivery = //string: ISO 8601 DateTime string, representing expected delivery of action,
    ActionTypeId = //string: The Trackmatic Id assosiated with the Action Type Name. This is the branch id, suffixed with the ActionTypeId,
    ActionTypeName = //string: Describes what action type this action is i.e. collection, delivery, ctc etc.,
    Pallets = //int: Number of pallets,
    AmountIncl = //double: Number of the amount including vat,
    AmountEx = //double: Number of the amount excluding vat,
    CreatedOn = //string: ISO 8601 DateTime string, representing when action was captured in line-of-business ERP system,
    Direction = //anything: Valid values: "Outbound" "Inbound",
    HandlingUnitIds = //string List: If using handling units this is a list of each of the Id's,

    Owner = new ActionOwner // Owner of this particular action
    {
      Name = //string: Name of the owner,
      Reference = //string: Reference of the owner,
      Contact = //string: Contact detail of the owner,
    }
  }
}
```

trackmatic

- **Creating an Entity:**

Create An Entity

```
private Entity CreateEntity ()
{
  return new Entity
  {
    Id = //string Required: Entity reference prefixed with Client Id,
    Name = //string Required: Name of the entity,
    Reference = //string Required: Line-of-business reference for a particular entity,

    Contacts = new ObservableCollection<EntityContact>
    {
      new EntityContact
       {
        Id = //string: ID of the contact,
        FirstName = //string: FirstName of the contact,
        LastName = //string: LastName of the contact,
        TelNo = //string: Telephone Number of the contact,
        CellNo = //string: Cell Number of the contact,
        Email = //string: Email of the contact,
        WorkNo = //string: Work Number of the contact,
        PositionHeld = //string: Position of the contact,
       }
    }

    Requirements = new ObservableCollection<EntityRequirement>() //Used to determine which entity requirements apply to this entity
    {
          new RequireActionDebrief(),// Verifying each transaction
          new RequireSignature(),// Sign on glass debrief
          new RequireActionImageDebrief(),//
          new RequireCodDebrief(),// Cash on Delivery debrief
          new RequireHandlingUnitDebrief(),// If using handling units - handling unit debrief
          new RequireImageCapture(),// Image capture debrief
          new RequireRatingSimple(),// Good/Bad rated debrief
          new RequireRatingSmiley(),// Simley rated debrief
          new RequireRatingStar()// Star rated debrief
    }
  }
}
```

trackmatic

- **Creating a Multiple Address Entity**

Multiple Addresses

- Note if there are multiple addresses, replace the above "CreateEntity" method with "CreateShipTo" and "CreateSellTo" methods. Also update "CreateRelationship" method to the one below.

```
private Entity CreateShipTo()
{
  return new Entity
    {
    Id = //string Required: It must take the following form when there is multiple addreses [client id]/[Reference]/[address id].,
    Name = //string Required: Name of the entity,
    Reference = //string Required: Line-of-business reference for a particular entity,

    Contacts = new ObservableCollection<EntityContact>
    {
      new EntityContact
       {
        Id = //string: Name of the contact,
        FirstName = //string: FirstName of the contact,
        LastName = //string: LastName of the contact,
        TelNo = //string: Telephone Number of the contact,
        CellNo = //string: Cell Number of the contact,
        Email = //string: Email of the contact,
        WorkNo = //string: Work Number of the contact,
        PositionHeld = //string: Position of the contact,
       }
    }

    Requirements = new ObservableCollection<EntityRequirement>() //Used to determine which entity requirements apply to this entity
    {
          new RequireActionDebrief(),// Verifying each transaction
          new RequireSignature(),// Sign on glass debrief
          new RequireActionImageDebrief(),//
          new RequireCodDebrief(),// Cash on Delivery debrief
          new RequireHandlingUnitDebrief(),// If using handling units - handling unit debrief
          new RequireImageCapture(),// Image capture debrief
          new RequireRatingSimple(),// Good/Bad rated debrief
          new RequireRatingSmiley(),// Simley rated debrief
          new RequireRatingStar()// Star rated debrief
    }
}
```

```
private Entity CreateSellTo()
{
   return new Entity
    {
      Id = //string Required: Entity reference prefixed with Client Id,
      Reference = //string Required: Reference of the Sell to Entity,
      Name = //string Required: Name of the Sell to Entity,
    };
}
```

trackmatic

- **Creating a Deco:**

Create A Deco

```
private OLocation CreateDeco ()
{
  return new OLocation
  {
    Id = //string Required: It must take the following form, [client id]/[address id]. When the location is adhoc it should take the following form
[client id]/$tmp/[address id].,
    Name = //string Required: Name of the deco,
    ClientId = //string Required: Client Id,
    IsActive = //boolean: true to show on routing

    Entrance = new OCoord()
    {
      Latitude = //double: Latitude,
      Longitude = //double: Longitude
    }

    Coords = new SpecializedObservableCollection<OCoord>()
    {
      new OCoord()
       {
         Latitude = //double: Latitude,
         Longitude = //double: Longitude,
         Radius = //double: Radius
       }
    },

    StructuredAddress = new StructuredAddress()
     {
       UnitNo//string: Unit Number,
       BuildingName = //string: Building Name,
       StreetNo = //string: Street Number,
       SubDivisionNumber = //string: Subdivision Number,
       Street = //string: Street,
       Suburb = //string: Suburb,
       City = //string: City,
       Province = //string: Province,
       PostalCode = //string: Postal Code,
       MapCode = //string: Map Code,
     }
  }
}
```

- **Creating Handling units (Optional):**

Create A handling Unit

```
private HandlingUnit CreateHandlingUnit()
{
  return new HandlingUnit
  {
    Id = //string Required: Route reference prefixed with ClientId,
    Barcode = //string Required: Barcode of item,
    CustomerReference = //string: Reference to a customer,
    InternalReference = //string: internal Reference,
    Quantity = //int: Quantity,
    Weight = //double: Weight,
    Status = //anything: pending,

    Dimensions = new HandlingUnitDimensions()//Dimensions of the item
    {
      Height =//double: Height,
      Length =//double:Length,
      Width = //double: Width,
    }
  };
}
```

- **Creating a Route**

Create A Route

```
private RouteModel CreateRoute()
{
  return new RouteModel
  {
    Id = //string Required: Route reference prefixed with ClientId,
    Reference = //string Required: Route reference,
    Name = //string Required: Name of Route,
    ClientId = //string Required: Client Id,
    Registration = //string: Vehicle registration,
    StartDate = //string: ISO DateTime string, representing route's scheduled start date and time,
    TemplateId = //string Required: ISO DateTime string, representing route's scheduled start time,
    Schedule = //boolean: true
  };
}
```

- **Creating a Relationship**

Create the Relationship

```
private UploadModel CreateRelationship()
{
  var model = new UploadModel();
  model.Route = CreateRoute();
  var location = CreateDeco();
  var entity = CreateEntity();
  var action = CreateAction();
  var handlingUnit = CreateHandlingUnit() //If handling units will be used

  model.Add(location);
  model.Add(entity);
  model.Add(action);
  model.Add(handlingUnit);//If handling units will be used

  var relationship = Relationship
                  .Link(action)
                  .To(entity)
                  .At(location);
  model.Add(relationship);

  model = CreateHandlingUnit(); // Add if handling units are been used

  return model;
}
```

- **Creating a Relationship with Multiple Addresses**

Create Multiple Address Relationship

```
private UploadModel CreateRelationship()
{
  var model = new UploadModel();
  var route = CreateGroupRoute();
  var action = CreateGroupAction();
  var deco = CreateGroupDeco();
  var shipTo = CreateGroupShipTo();
  var sellTo = CreateGroupSellTo();
  var handlingUnit = CreateHandlingunit();//If handling units will be used

  model.Add(action);
  model.Add(deco);
  model.Add(shipTo);
  model.Add(sellTo);
  model.Add(handlingUnit);//If handling units will be used
  model.Route = route;

  var relationship = Relationship
                  .Link(action)
                  .To(shipTo)
                  .At(deco)
                  .SellTo(sellTo);
  model.Add(relationship);

  model = CreateHandlingUnit(); // Add if handling units are been used

  return model;
}
```

- **Upload to Trackmatic API:**

In order to upload the transformed data to trackmatic, it has to be placed in the pipeline within the listener. See below how data is transformed then dispatched to be uploaded into trackmatic.

Uploading the transfomed data

```
private void ProcessPipeline()
{
  var pipeline = new CreatePipeline();
  var uploadModel = new UploadModel();
  var transformedModel = uploadModel.Transform();
  var element = new UploadElement (ClientId, transformedModel);
  pipeline.Add(element);
  Dispatch(pipeline);
}
```