VANTIQ
Enabling the **Real-Time** Enterprise™

VANTIQ Blueprint – Field Service Management

Last Revision: October 9, 2018

Summary

VANTIQ Blueprints are working applications that are designed to be imported into the VANTIQ platform as a new project and then customized by users who are looking to create applications that will perform similar functions. The benefit of using a Blueprint is starting off with an application that has a completed code base to bootstrap a new project being undertaken. Although Blueprints are complete working applications they are not intended to be used "as-is", rather they are intended to speed up development of a new application by providing a completed and working set of base components and functionality to build upon and customize.

Field Service Management (FSM) describes a category of business that provides services for IT related hardware and software. This will commonly include technical support for hardware and software problems, installation and removal of hardware and software, management of licenses and consulting services.

The FSM Blueprint was built with common business practices for an FSM company in mind. These include:

- A Service Desk that receives service or support requests and provides customer support.
- Field technicians who are dispatched to handle problems that require an onsite presence.
- Varying skills between technicians, one technician may specialize in certain hardware, another in certain software.
- A structured process that supports customer support lifecycle.

Problem Statement

This project was built to provide an answer to the following challenges typical to today's FSM practices and solutions.

- Customer experience suffers from infrequent communication and lack of visible insight especially on long duration high priority issues.
- Dispatching of a field technician is a slow process with manual steps that are prone to human error.
- Inability to identify or predict response times to customer issues is preventing meeting SLA demands.
- Lack of accurate time and effort tracking for field technicians makes it difficult to staff appropriate to demand and meet the expected levels of customer service.

The FSM Blueprint is an application built to provide ideal solutions these challenges. If this describes the type of business that you are building application for or contains similarities this Blueprint will help jump start your project.

This Project Contains

- Web based ticketing system that supports creating tickets, Workorders, customers, technicians, service desk users, assets and reporting.
- Mobile application that use the VANTIQ mobile app to provide alerts, notifications and interactions with users and technicians.
- Automated technician dispatch capability that will identify the proper field technician for a Workorder based on their location and skillset.
- SLA monitoring that will escalate Workorders that are not being progressed in time.
- Reports that provide insights and analytics.
- A process/workflow for handling open tickets and Workorders.
- Security Profiles that provide ACL based security for users.

Application Description

A service desk user can log into the FSM web client once they are added to the system and assigned a security profile that enables read-write access. The service desk user can then begin using the system to insert data.

Navigation

The service desk user will have access to the entire FSM web application which is broken into menu's that will display a list of all the items in the database for that type. The pages use a similar layout for each type of record. A table list view that is selectable or clickable depending on the complexity of the record. Button will be available on this page to add a new record or view an existing record where the view page also supports editing the record. When the view record button does not exist, it is because clicking on a row in the table will automatically navigate to the Edit/View page. Here is a list of pages users can navigate to.

- Users
  - List Users
  - Edit/View Service Desk Users
- Tickets
  - List Tickets
  - Edit/View Tickets
- WorkOrders
  - List Workorders
  - Edit/View Workorder
- Assets
  - List Assets
  - Edit/View Asset
- Customers

    - List Customers
        - List/Add/Edit Locations
        - List/Add/Edit Contacts
    - Add/Edit Customer
- Technicians
    - List Technicians
        - List/Add/Edit Skills
    - Add/Edit Technicians
- Reports
    - Service Desk Details for Workorders and Tickets.

To Edit a record the service desk user will bring up the View page which will list all the data elements of that type. Clicking on an entry will open the Edit page for that item or select row which then requires clicking on the View button. Example, to make changes to an existing Workorder the user will navigate to Workorders and click on the entry they wish to edit. Use the "Search Workorders" field to filter the table list.

Popups are used to add referenced data to items to a record throughout the FSM application. For example each customer record can be associated with multiple locations which represent the various customer sites, buildings and addresses. When adding or editing a customer a the Add Location button will pop-up an input screen that allows new locations to be created.

Tickets and Workorders

A ticket is used to describe a customer support issue or request and a Workorder is used whenever an onsite work is needed.

When a customer calls, emails or uses social media to report a problem the service desk user creates a new ticket using the Add Ticket menu. This form is used to document the customer report and contains field validation to prevent essential information from being left out. Once the ticket is added additional information can be added using the Comments section and state changes can be made in the Status dropdown menu.

If a ticket represents a situation that requires an onsite technician, then a Workorder will have to be created. Each ticket can have multiple Workorders and these can be prioritized or scheduled using the available fields in the Workorder record in order to specify a series of Workorders that need to be completed in a particular order in the event there are dependencies. A Workorder is always tied to a ticket and when escalations occur due to slow or missing response times the Service Desk Owner is the person who the Workorder escalates too.

Field Technician Dispatch

When a new Workorder is created the Technician field can be set to the value "-AutoAssign-" by clicking the Auto Assign button. This triggers an automated routine in the application that will attempt to automatically identify the appropriate technician using the customer location relative to the technician, the Tech Level and the skills for the Workorder as set in the Required Skills area.

A technician can be manually assigned using the Manual button. This will pop up an Events management calendar that will show a list of all the technicians who match the selected Skills table. A technicians name can then be drag and dropped onto the calendar which will set the workorder.scheduledArrivalDate and workorder.scheduledCompletionDate.

The technician's location is determined by the address provided when the technician is added. This provides a static location to use to determine if the technicians service area falls within the geographic radius of the customer location. This is preferred over the roaming GPS location of the technician's mobile phone since the technician may be traveling out of range temporarily. The geographic range is set per technician using the Support Radius field.

Once a suitable technician is identified they will receive a notification on the mobile application to request confirmation of assignment. The technicians can use the Accept or Reject buttons to respond. If Accept, then an additional notification is sent to the mobile app that contains all the information relating to the Workorder and requests scheduling input from the technician to set their expected arrival and completion times for the Workorder. This information will be communicated to the customer via Email.

An escalation will occur if no technician is found, if all suitable technicians Reject the assignment request or if no technician responds after a period of time. The escalation will notify the service desk user associated with the Workorders ticket of the situation, so they can manually intervene to resolve the assignment problem.

Technical Reference

VANTIQ is a fully integrated platform that contains elements for building the backend server logic, web and mobile, apps, data storage, data stream management and analytics. These categories are also represented in the VANTIQ IDE and detailed project features are described per-category.

Full Reference Documentation - https://dev.vantiq.com/docs/system/rules/index.html#rule-and-procedure-reference-guide

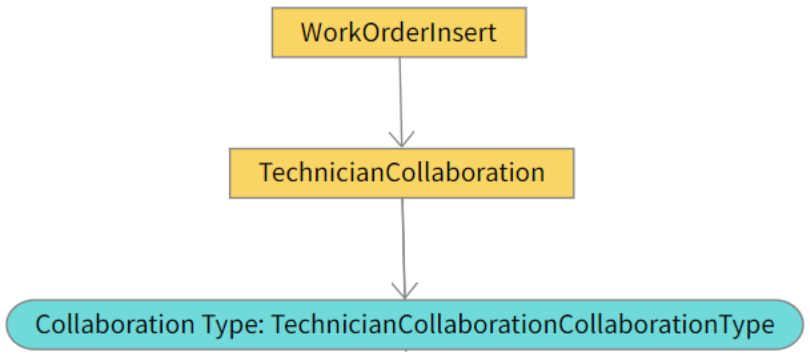
Customization

Blueprints are built specifically to provide an easy to customize starting point for new applications. Each category listed below will detail the functionality performed by that category and the names of the different assets contained. This will include everything from source code, web and mobile apps, external API connections and database definitions.

Application

Apps are used in VANTIQ to handle data streams and live events. Apps are built graphically in the App Builder and it allows users to layout the general flow of data through their system. Each step in the App represents a task and these tasks are used to manipulate the data streams in order to identify business situations that can drive collaborations.

FSM Blueprint uses a basic App that will trigger on an Insert into the workorder type.


WorkOrderInsert
TechnicianCollaboration
Collaboration Type: TechnicianCollaborationCollaborationType

Any new workorderwill trigger the collaboration steps.

Customization Example: Add a filter after the WorkOrder Insert step to only pass workorders where the technician name is set to "-AutoAssign-". A list of technicians can then be added to that menu for manual selection.

Collaboration Type

VANTIQ collaborations describe aspects of the applications designed to integrate systems to people, people to people and systems to systems. Collaborations are often utilized when real time business decisions need to be made either using automation or manual input from a person.

In FSM the collaborations focus on the field technician dispatch and handling of communication, time tracking, and events throughout the workorder process. Here is a description of each step.

- <u>ProcessAssignment</u> – Performs a Recommendation task to identify a technician based on skills and geolocation of the work order and the support radius for the technician. The technician must match the skills and be inside the support radius. This step is only performed if the Technician Assigned To field is set to -AutoAssign-.

- EscalationAssignmentTimer – Technicians have a configurable duration to accept/acknowledge assignment or the task is escalated to the work order's service desk owner.
  - NotifyServiceDeskNoAssignment – Notify the service desk owner that no one has responded to the workorder assignment request. This will be sent to the Workorders "Service Desk Owner" field who will then be required to manually set the technician assignment.
    - CompleteManualAssignment – Service Desk owner acknowledges and manually assigns the work order to a technician.
  - Notification.retractPayload - Remove the accept-reject request from all users once escalation time limit has expired or a response is received by a technician.
- AcceptRejectWorkorder – Push notification containing the Accept – Reject assignment request is sent to all recommended technicians.
  - Notification.retractPayload - Remove the accept-reject request from all users once escalation time limit has expired or a response is received by a technician.
  - AssignTechRole – Take the first technician response and assign it to the technician. The responses come back as an array in the event of any race condition where two techs click the Accept button at the same time only the first responded technician in the response array is assigned.
    - UpdateWorkorderAssignment – Updates the work order status to "Assigned" along with setting the status date.
    - CollaborationUtils.cancelEscalation - Cancel the EscalationAssignment escalation timer after an assignment is made. The cancelEscalation is a built-in service. An escalation is a countdown timer that triggers the steps below it when the countdown completes. Once an assignment is made, we cancel the escalation. If an assignment is not made before the escalation time limit is reached, then the Assignment request payloads are removed from the mobile apps and a new form is sent to the Service Desk Owner user to manually make an assignment.
    - ScheduleArrivalDateTime – This date is used to notify the customer of the appointment as well as to determine when to start location tracking of the technicians current GPS coordinates. The default will be 1 hour prior to the estimated arrival.
      - CustomerSchedulingAlert - Send customer notification based on the notification preferences in the workorder. SMS, Email or Both.
      - SetTrackingTime – Sets the point in time in the future when to start location tracking. The location tracking sends a request to the Google Maps API to determine travel time. Location tracking will begin 15 minutes prior to the ETA returned by Google to consider changes in traffic.
        - UpdateWorkorderEnroute – Updates the work order status to "En Route" along with setting the status date.

        - BeginLocationTracking – Tracks the technician's location default to every 10 minutes.
            - TrackProgress – This procedure will run with each new GPS update and determine the technician's current location which will update the map. This procedure will also use Google Maps API to get the ETA using current location and destination location. If the ETA will result in a time that is past the scheduled arrival time, we will notify customer and technician of the late arrival.
            - (On arrival) CompleteWorkorderNotification – Sends a form to the mobile app that the tech can use to complete the workorder when the job is done.
                - UpdateWorkorderCompleted - Updates the work order status to "En Route" along with setting the status date.
                - SaveSignature – Saves the confirmed completion signature.
                - CollaborationUtils.CloseCollaboration - Close out the collaboration.
            - (On arrival) UpdateWorkorderStatus – Updates the work order status to "On-Site" along with setting the status date.
                - CustomerNotifyTechArrival – Notify the customer that the technician is on-site.

Data Type

Data types refer to the internal storage system used by VANTIQ. Types contain a schema definition which are called Properties. Records that are stored in Types can be accessed programmatically or using the Show --> Find Records dialog.

The FSM Blueprint uses Type fields to store records that are utilized by the Web Client such as Ticket and Customer. The following is a list of all the Type fields defined in this Blueprint, most Types will map directly to the related page on the FSM Web Application.

- Contacts
- Customers
- Users
- Locations
- Assets
- Skills
- Technicians
- Tickets
- Workorders

- NotificationTemplates
- Events

Procedure

Procedures contain user or system generated source code that can be executed on demand. Procedures built by users are often created to supplement the graphical programming tools. They provide users a way to create very specific routines and services that can be executed as part of an App or a Collaboration. A common example is connecting to the Google Maps API to convert an address into GeoCoordinates. System generated Procedures are created from the graphical programming tools and by default marked as Hidden.

Procedures are used frequently in the FSM Blueprint to provide customized functionality that is not already provided by the graphical programming tools. The following is a description of each user generated procedure.

- GoogleDirections – Uses the Google Maps API to get directions between two points
- GoogleDirectionsDuration – Use the Google Maps API to calculate travel time
- GoogleGeoCode – Uses the Google Maps API to convert an address into geojson coordinates
- GoogleTimeZone – Uses the Google Maps API to determine timezone for localization purposes.
- clientLoginValidation - Determines if a user has rights to access the FSM web client
- collab.changeWorkOrderStatus -Used to update the workorder status during the collaboration
- collab.scheduleLocationTracking - Determines the date-time to start technician location tracking
- collab.customerNotify - Used to send a custom email/sms notification to contacts
- collab.setScheduledDate - Validates the scheduled arrival date to prevent invalid entries such as setting a date in the past
- collab.completeManualAssignment - Used when an Assignment is manually made via escalation.
- collab.weightedNearbyRecommendations - Custom event used to identify technicians based on skills and location. Also identified the difference between automated and manual assignments.
- generateUUID - Creates a UUID

*System generated procedures are not listed.

Some additional procedures that are not currently used were left in the project as examples of how to perform different tasks. For example, the GooglePlaceId procedure could be used to implement a type ahead list of locations to select from when adding an address.

Rules

A Rule is source code that is triggered by an event such as when a message arrives, a publish occurs or an insert occurs. The EventStream task in the Application Builder performs the same function and it's common for many VANTIQ applications to not contain any user generated rules and instead using the App Builder approach.

The FSM Blueprint does not contain any user generated rules however the collaboration and app builder generate rules.

Clients – Web, Mobile and Both

Clients are created in VANTIQ using the Client Builder and can be specifically Web only, Mobile only or a mix of both. Web clients are accessible via the VANTIQ RTC (run-time-client) URL. The mobile clients are used to send push notifications to a mobile app in the form of custom pages. The Client Builder uses standard HTML/JavaScript which allows for high degree of customization in both.

The FSM Blueprint contains a web application for the service desk and several mobile clients used for primarily for push notifications.

Web Clients

- Service Desk – FSM Web client

Mobile Clients

- AcceptRejectAssignment - Accept or Reject assignment request
- ScheduleArrivalDateTime – Used to schedule estimated arrival time to the customer site.
- CompleteWorkorder – Provides form to complete and close the workorder
- Workorders – Displays a technician's assigned Workorder details

Projects

The VANTIQ graphical editing environment is called Modelo. Modelo is organized by Projects which contain a set of assets such as Procedures and Types that are helpful to the developer. Projects are for organizational purposes only and useful to make it easier to work on specific items in the VANTIQ platform by having specific layout of windows and dialogs open for each project.

The FSM Blueprint contains several projects.

- FSM_APPBUILD – Displays the App which is triggered by the workorder "Insert" rule
- FSM_CLIENT – Contains the FSM Web client
- FSM_COLLAB – Contains the technician dispatch collaboration
- FSM_MOBILE_CLIENTS – Lists all the mobile clients
- SOURCES – Google, Twilio and Email sources – set your API keys here
- LOGS_ERRORS – Debug, log and error windows

- TYPES – List of all the Type definitions

Source

A Source represents the saved connection settings to an external data source which can be used for taking data in or sending data out. VANTIQ supports several data sources which can include your own mobile applications, rest and web sockets, MQTT, chatbot and even email. A VantiqPushNotification is always created by default and provides push notifications to the VANTIQ mobile app.

The FSM Blueprint uses the Google Maps API and will require an SMTP server to deliver email notifications.

- GoogleDirections - Google Maps API, will not work until you provide your own API key
- GoogleGeoCode - Google Maps API, will not work until you provide your own API key
- GoogleTImeZone - Google Maps API, will not work until you provide your own API key
- VantiqPushNotification - Push notification service for VANTIQ mobile apps
- email – SMTP server, will require you provide your own settings

Analytic Source

Analytics sources are the same as sources except customized for specific external services like Microsoft MLStudio.

The FSM Blueprint does not contain any Analytic Sources.

RCS Page

RCS Pages are simplified widget-based page layouts used for mobile push notifications.

The FSM Blueprint does not contain any RCS pages.

Scheduled Event

VANTIQ supports setting up scheduled events which can be one time or reoccurring events that run at some time in the future. The event will send a message to a user specified topic which can then trigger an App or a Rule to execute.

The FSM Blueprint does not contain any Scheduled Events.

User and Security Management

This Blueprint uses two profiles to assign security access to users.

- Technician – Will have access to the web and mobile clients.
- User – Will have access to the web and mobile clients.
- Customer – Will not have access to either web or mobile clients.

Appendix

Coding Examples, Tips and Tricks

Global Functions

Some commonly used JavaScript functions can be added as .js files in Custom Assets or a similar effect is to a add code to the Properties --> Custom Code.

One technique used is have a function for all the common rest api calls like SELECT, INSERT, UPSERT.

```
function select(resource, params, client, callback){
   var http = new Http();
   http.setVantiqUrlForResource(resource);
   http.setVantiqHeaders();

   // if no params then limit to 50 sort by create date, newest first
   if (!params){
      params = {
         sort:{"ars_createdAt":-1},
         limit:50
      };
   }

   http.select(params,function(response)
   {
      console.log("SUCCESS: " + JSON.stringify(response));
      if (callback)
         callback(response);
   },
   function(errors)
   {
      // client.showHttpErrors(errors,"Doing a select on: " + resource);
```

```
        console.log(errors,"Doing a select on: " + resource);
    });
}
```

Function can then be called from the Page's On Start using:

```
select("Type-Field-Name", params, client, function(response){
    console.log(response);
});
```

Parameters are an object that determine the search criteria for example to find a customerid that equals "134" you would use:

```
params = {where:{customerid:134}};
```

List View Pages

When a user navigates to a page from the nav buttons they will first see a DataTable widget populated with the data for that page, such as Assets, Customers. Some pages have two or more tables to show additional content related to that record such as all the Work Orders for the selected Ticket.

When the page loads the a select query is run against the record type and used to populate an "On Client Event" data stream.

```
select("Customers", null, client, function(response){
    client.sendClientEvent("ce_Customers",response);
});
```

This is preferred over timed queries which reset any special properties or sorting on the table you may have applied each time the query updates.

In "Data Streams" there will be a entry for ce_Customers that is a On Client Event mapped to the Customers type.

The data table will have an onSelect code added to it which will either load the selected row into a Edit/View page or will load the related data objects such as loading Work Orders when a Ticket is selected. In this later case a button is used to open the Edit/View page and the contents of the row click are passed into the goToPage("Ticket", selectedRow);

Add, Edit/View Page

This page will contain all the input fields for the database type. The same page is used for adding the page and for updating fields on the page. A couple techniques are applied here to set the display options for buttons and fields based on how the page is loaded.

When a page transition occurs using goToPage parameters can be passed in from the selected row using the extra argument.

client.goToPage(“ticketspage”, extra);

This will pass the selected row into the ticketspage. The selected row will contain all the data for the ticket record even if that data is not currently displayed on the table.

When the ticketspage is loaded an if/else condition will look for incoming values in the form of “parameters”.

If parameters exist it identifies an Edit/View situation and the parameters are loaded into a data object for that page. Since the “page” is not yet available we use “this”.

this.data.Tickets.copyMatchingData(parameters);

Once all the parameters are loaded into the Tickets data object we can then add the corresponding fields to the page and by setting the data binding in each widgets properties which allows the content of that field will appear.

Data binding for the Summary field: page.data.Tickets.summary

If no “parameters” exist we reset the page to its default values using:

this.data.Tickets.initializeToDefaultValues();

This will reset all the forms on the page to their defaults, which usually means blank or in the case of droplist menu’s “-select-”. This will prevent issues with cached fields that can commonly happen in web browser form fields.

Buttons and text on the page can be set in the if/else conditions so that the button text will show Submit for new records and Update for editing existing records. This is done by first collecting the properties of the button or text widgets:

var atb = client.getWidget("addticketBtn");

We can then set the button or text properties that are available such as isVisible or buttonLabel.

atb.buttonLabel = "Update Ticket";

To see what options you can set this way see the following documentation page, this one is pointing directly to the button widget. https://dev.vantiq.com/docs/system/cbref/index.html#button

Submit Button

A couple different variations of how data gets submitted is used based on the specific requirements of each action being performed. In some instances, the API command UPSERT is used to update/insert records, in other instances individual INSERT or UPDATE commands are used, often as part of an if/else condition to determine if the action should be INSERT or UPDATE. The UPSERT is more convenient and requires less code and works so long as the natural key is present (usually the “id” field) where the entire data object is passed into the command. The UPDATE supports changing specific fields instead of using the entire data object but also required the internal database record id which is always the “_id” field. Often we set the value of this field on the page start code using a database object called “dbid”.

UPSERT example:

```
    upsert("Locations", page.data.Locations, client, function(response){
      client.closePopup();
    });
```

INSERT/UPDATE example:

```
  if (page.data.dbid){
    update("Assets", page.data.Assets, page.data.dbid, client, function(){
      client.goToPage("assetslist");
    });
  } else {
    insert("Assets", page.data.Assets, client, function(response){
      client.goToPage("assetslist");
    });

  }
```

Droplists

Droplists can be configured with a hard coded set of Values and Labels from the properties pane of the widget itself. Dynamically populating the menu’s requires running a query against the database type that contains the values that should be loaded into the menu. For example when you add new Users to the system they will appear in the Add Ticket page under the Assign To droplist.

To populate this menu we first initialize the widget properties as a variable, similar to how we dynamically set the button properties, then we load the Value and Label of the droplist entries using a for loop through the result set of a SELECT query.

```
//SELECT query that returns all the users from the Users type
select("Users", null, client, function(response){
   var json = response;
   var users = [{"value":"-select-", "label":"- select -"}];
   $(json).each(function(index, item) {
      var name = item.fname + " " + item.lname;
      var obj = {value:item.username, label:name};
      //Build an array of value, label combinations
      users.push(obj);
   });
   //Get the properties of the droplist using the getWidget operation
   var assignmentList = client.getWidget("assignmentList");
   //Set the droplist list using the enumeratedList property.
   assignmentList.enumeratedList = users;
});
```

When the droplist is populated this way the label will show the Users name as “Firstname Lastname”. But the value of that tag is set using the Users username (which is a unique id) which will be used when the record type is added or updated.

Popup Pages

Popup pages are frequently used to provide users a convent way to add or update some of the simpler data objects such as Assets, Locations, Skills.

When editing a record the id of the database object is passed in as a paramter for the record and when the popup is loaded the content of the existing record is set from a SELECT query.

```
client.popupPage("contactslistpopup", "Select a Customer Contact for the Ticket", params, function(response)
{
          //Steps to perform after popup window closes
}
```

Often a datatable is reloaded to show any changes to that record. For example if a new contact is added to the Customers page or if a contact is added to the list of contacts on the Tickets page. A client event schema will have to be defined the same way they are for the list view pages (when possible the same definition can be used) and when the popup closes the table is updated.

```
client.sendClientEvent("ce_tpcontactslist", arr);
```

Field Validation

Validation for required fields and some content type is provided in the project. The code is written in the Custom Code section of the Properties for the client. This is initially invoked via the following function:

```
function checkForRequiredFields(pagechildren){
   Errors = "";
   searchforerrors(pagechildren);
   return Errors;
}
```

Which can be called from the Submit buttons.

```
var fieldErrors = checkForRequiredFields(page.children);
if (fieldErrors !== ""){
  client.infoDialog("You need to supply values for: " + fieldErrors);
} else {
  //Perform INSERT/UPDATE steps
}
```

This code will identify which required fields were left blank and provide the user an error with a | pipe separated list of field names that need to be set before Submit can complete.

All of the input form fields are placed inside containers. Containers can return a list of all the widgets inside it using a ".children" operation. This makes it easy to iterate over the fields and check for content.

Form field properties have an Is Optional setting. When that box is checked it can be left empty.

Additional validation is performed to provide user friendly messages when non-integers are used on integer only fields.

Field validation is not currently implemented to check for content structure, for example phone numbers must be xxx-xxx-xxxx or dates must be yyyy-mm-dd or emails in the form "string@string.com". These can always be added by invoking similar methods of the fields with the proper regex string matching on the input to determine if it matches the required format or not.

Collaboration Custom Events

This project tried to use as many of the features as out-of-the-box as possible. In some cases this was not possible and customization was required. Several collaboration steps will execute procedures to perform custom tasks such as the TrackProgress step.

The ProcessAssignment uses a custom event that requires some explanation. This is a Recommendation type task that is used to build a list of users to contact to accept a Workorder assignment. The build in service does not allow us to account for Manual assignments vs Automated assignments, or to evaluate an Array field for the matching.

To solve this problem we created a custom event called collab.weightedNearbyRecommendations. This is a procedure that will determine if the assignment type is Automated or Manual based on the techId field (-AutoAssign- will always appear for automated).

A query using the more advanced API syntax is performed that will return a list of technicians that are inside the Max Support radius (which is 150 miles) and contain all the skills selected in the workorder.

```
for each s in workorder.skills do
   var obj = {"$elemMatch":{"skill":s.skill}}
   push(skills, obj)
end

var query = {
   "skills": {
      "$all" : skills
   }
}
```

FOR (c IN SELECT * FROM @candidateType where geolocation geoNear destination and query)

This For loop then goes through each result and ensures that the support radius set in the technician record is not greater then the distance to the customers location, and if true they are added to an array object. The result of this array is then passed into the next step of the collaboration which is a notification step that will request the list of technicians to accept the workorder assignment. The first response to that notification is then assigned to the collaboration.

Google Procedures

Google Maps API is commonly used in this project. This API service is used to provide geocoding of addresses, route and distance calculations.

Before these procedures will work users need to provide their own API key for Google Maps.

A big development caveat is that the GeoJSON format used by VANTIQ uses a different order of coordinates then Google. VANTIQ uses [long, lat] while google uses [lat, long] which means a request or response will need to be reformatted to switch around the ordering of the cordinates.

Here is an example request we send to google to get the timezone of geo coordinate.

```
// origin coordinates
var olon = location.coordinates[0]
var olat = location.coordinates[1]

var queryparams = {}
queryparams.location = format("{0},{1}", olat, olon)

// time in seconds
queryparams.timestamp = (date(time) / 1000).toString()

var timezone = SELECT FROM SOURCE GoogleTimeZone WITH query = queryparams
```

CSS, Theme and Style

This project uses the page Properties Theme settings and a pair of .css files, one for the servicedesk web page and one for mobile pages to apply style.

The .css files can be downloaded from Model under Show --> Advanced --> Documents. Download the filenames:

servicedesk.css
Servicedeskmobile.css

The .css uses the default VANTIQ classnames to change the properties of common widget types. For example to set the style for all buttons that are added from the client builder but will not impact any non-VANTIQ buttons that appear on the page such as one's that commonly show up on the Map widget.

```
/* Vantiq Button class */
.vantiqButton {
	text-transform: none;
	border-radius: 5px;
	background-color: #EBAF57 !important;
	border-width: 0px;
	color: #FFFFFF !important;
	padding: 8px;
}
```

Some properties may conflict with existing HTML DOM properties. In order to set the priority to your CSS entries use the "!important" designation.

If any changes are made to the .css files they must be reuploaded to VANTIQ. There is an upload icon that will allow you to select the css file and will automatically set the destination name and path for you. If using a manual approach the name and destination of the file should match what is currently listed...

public/css/servicedesk.css

public/css/servicedeskmobile.css

Notification Templates

The email and sms notifications for customers are used to provide alerts and status updates. The email and sms approach is used because the default behavior of this project is that customers do not have a sign in. If customers had access to a mobile application for example they would not need sms and email notifications as the mobile alerts provided through push_notifications would suffice and in some cases this may be desirable since it provides the customer with real-time insight into the state of the workorder.

A lightweight template system was created in the collab.notifyCustomer procedure. There is a NotificationTemplates type that contains the subject, body, type for messages you want to send with some examples already added.

The string parser format() is used to do a simple substitution of field tags that can be added to the Body of the messages "{0}". The substitution field is determined based on its order in the format() command.

To help simplify this for users there is a list of index positions for data fields available in the collab.notifyCustomer procedure.

* Index of values for use in template text
0 = Technician Name
1 = Scheduled Arrival Date
2 = Summary
3 = Assets
4 = Customer Address
5 = Priority
6 = unused
7 = unused
8 = unused
9 = unused
10 = Technician Phone **
11 = Technician Email **

Most these fields relate to a field inside the workorder so an array is used to set the index location for the format() operation. Those that are part of other types or need to be recalculated or set specifically in the format().

var str = notifications[x].body
	//The order in which a value appears in the format() list determines its index {0} number. The first item workorder.techname will replace any occurrences of {0} in the message body.

```
        notifications[x].body = format(str, workorder[subs[0]], arrivaltime, workorder[subs[2]], assets,
workorder[subs[4]], workorder[subs[5]], workorder[subs[6]], workorder[subs[7]], workorder[subs[8]],
workorder[subs[9]], phone, email)
    }
```

For the NotificationTemplate now just specify the message you want to send as the body field by using the substitution index. For example:

Hello you technician {0} is set to arrive at address {4} at the following date and time {1}

If the message is an email HTML tags must be used, by default all emails are sent in HTML format only.

<html><head></head><body><p>Hello you technician {0} is set to arrive at address {4} at the following date and time {1}</p></body></html>

These are simple examples but HTML can be used to provided rich text content to customers.

If using Twilio check the character limit for sms messages to make sure you do not exceed this limit with messages that are too long.