Enate – Custom Card Introduction and Capabilities

..

Table of Contents

Enate Work Allocation

Introduction

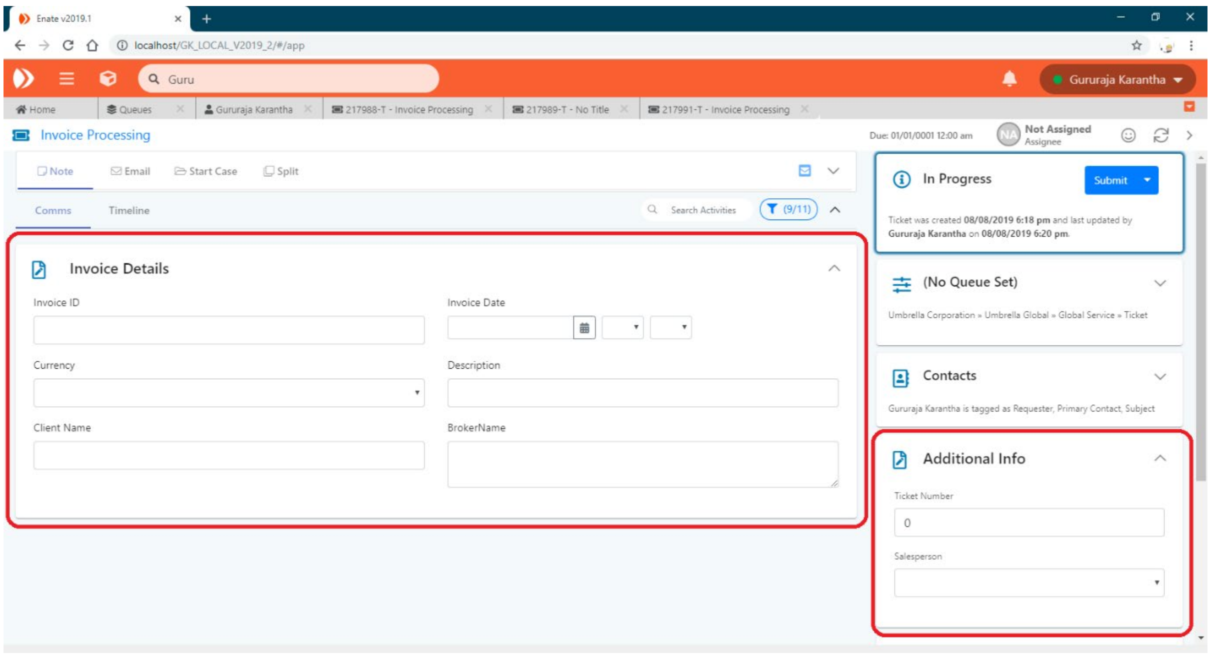
Enate Work Manager is the face of Enate, and is primary application used to process work-items in day to day activities. Though Work Manager is designed to be flexible and easy to for end users, it cannot cater for every need that business throws at it. One of the common requirements of many business processes is the ability to capture Custom Data in Work Manager.

Custom Cards can be added to Tickets, Cases & Actions to capture bespoke information (Custom Data) on these work items as they run through process. The information is displayable via Custom Cards. Cards can be set to display in the main section of the work item, and as a section of the side panel on the right side of the screens.

Custom cards are the configurable Work Manager sections which are displayed in Case, Ticket or Action screens. Custom cards can be created in Enate Builder and made to show in Work Manager.

These cards are designed with HTML, TypeScript and CSS.

An example Custom Card being displayed in Main and Side Section.



Ticket was created 08/08/2019 6:18 pm and last updated by Gururaja Karantha on 08/08/2019 6:20 pm.

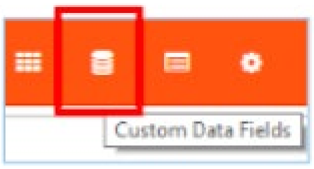
Custom Card

There are three steps involved in configuring Custom Card to display on Ticket, Case & Action screens:

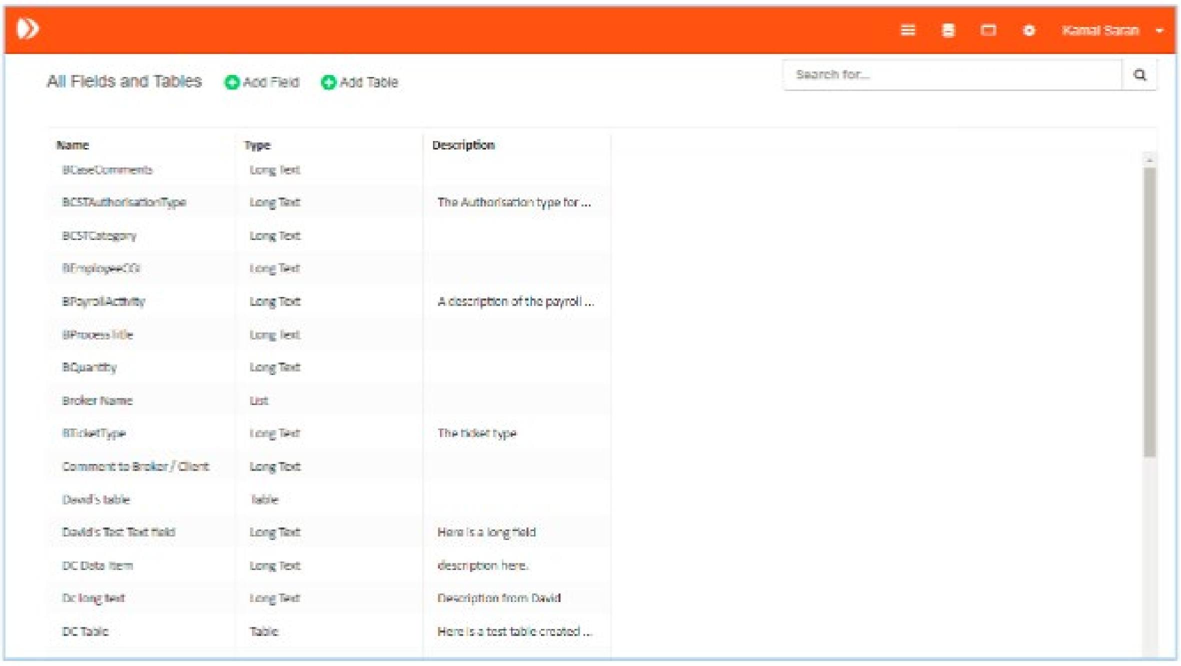
1. Creating Custom Data Fields
2. Creating Custom Cards to show these Data Fields / other content.
3. Linking Custom Cards to specific Case, Ticket & Action instances.
4. Viewing and Testing the Custom Card in Work Manager.

Creating Custom Data Fields

The first step to create and use Custom Card is to be ready with Custom Data fields that we need to capture. To create custom data fields, access this section from the Custom Data Fields link in the toolbar:



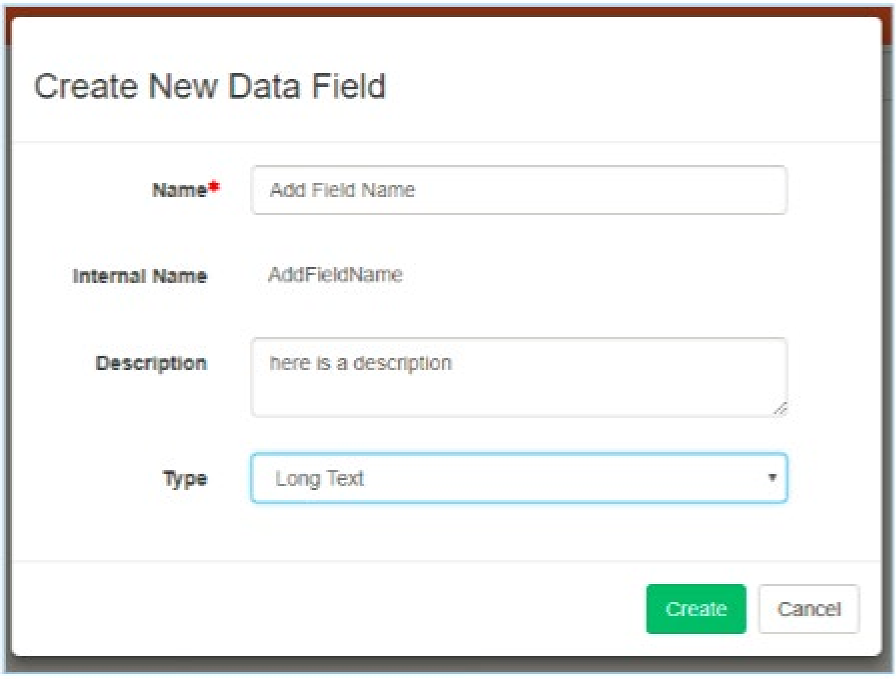
This will bring you to a list of all Custom Data fields (including tables of fields). This list can be filtered by field name.



All Fields and Tables

Name	Type	Description
BCaseComments	Long Text	
BCSTAuthorisationType	Long Text	The Authorisation type for ...
BCSTCategory	Long Text	
BEmployeeCCs	Long Text	
BPayrollActivity	Long Text	A description of the payroll ...
BProcess title	Long Text	
BQuantity	Long Text	
Broker Name	List	
BTicketType	Long Text	The ticket type
Comment to Broker / Client	Long Text	
David's table	Table	
David's Test Text field	Long Text	Here is a long field
DC Data Item	Long Text	description here.
Dc long text	Long Text	Description from David
DC Table	Table	Here is a test table created ...

Creating a Field: Click the 'Add Field' link at the top of the screen to begin. This will bring up a popup to define the field data. (For existing fields, click on the field to bring up this popup for editing).



Create New Data Field

Name*	Add Field Name
Internal Name	AddFieldName
Description	here is a description
Type	Long Text

You can set the following field-level settings here:

Attribute	Description	Notes
Name	Name The field name	
Internal Name	The name by which the field should be referenced on Cards	The system will autogenerate this name as you type (a copy of the field name with spaces

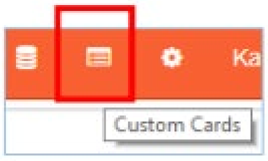
		removed). This Internal Name should be used in any subsequent Custom Card HTML / TypeScript references.
Description	A description for the field	
Type	The underlying data type for the field, e.g. long text / date time etc.	

The following Types of fields are supported:

Type	Description
Checkbox	Boolean marker
Date & Time	Stores both the date & time component.
Date Only	Stores only the date component
Decimal Number	e.g. 3.2
List	A dropdown list. If you select this type, you can manually enter a list of available dropdown items along with the field. This supports copy / pasting of tabular information from e.g. spreadsheets.
Multi-level List	A dropdown list with multiple levels (up to 3 levels supported).
Long Text	Text fields of over 255 characters.
Short Text	Text field limited to 255 characters.
Whole Number	e.g. 4

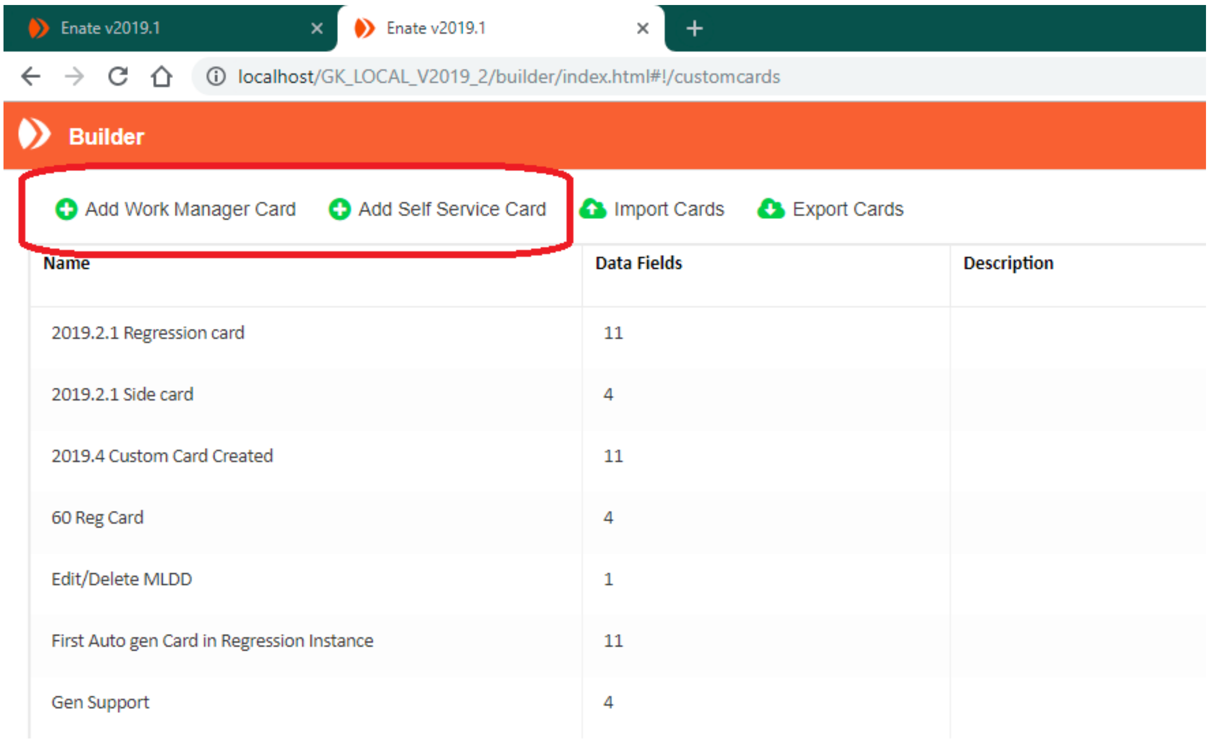
Creating Custom Card

After creating all the Data Fields needed by your Custom Card you are ready to proceed to the next step which is to create the Custom Card. To Create a custom card which can use the custom data that you previously created, access this section from the Custom Cards link in the toolbar:



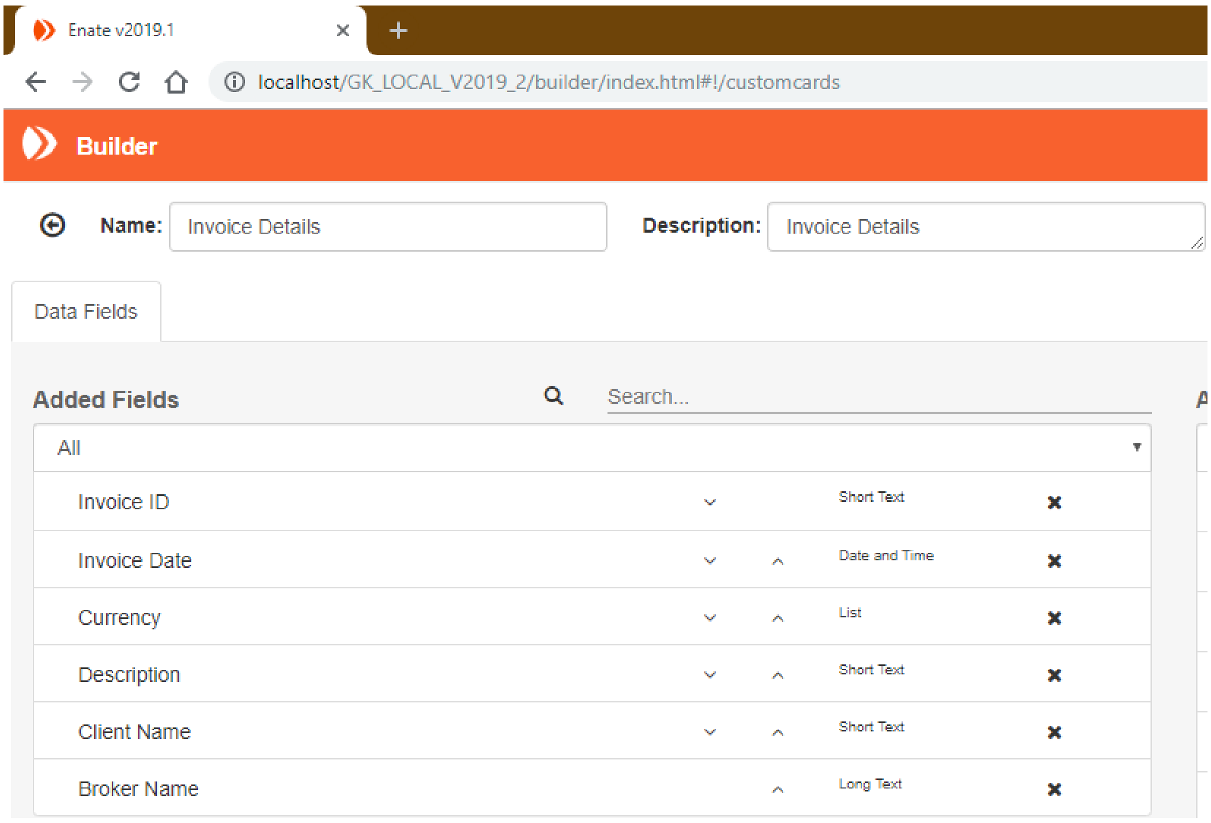
And Select the Type of Custom Card to create. There are two types of Custom Cards that can be created. One is **Work Manager Card** and other being **Self Service Card.** Work Manager cards are custom cards which will be displayed in Work Manager app and Self-Service Cards are cards which will be displayed in Self Service app.

In this document we mainly talk about Work Manager cards but the concepts and steps are identical to Self Service Card as well.



Name	Data Fields	Description
2019.2.1 Regression card	11	
2019.2.1 Side card	4	
2019.4 Custom Card Created	11	
60 Reg Card	4	
Edit/Delete MLDD	1	
First Auto gen Card in Regression Instance	11	
Gen Support	4	

Clicking the Add icon on the page brings up the ‘Create New Card’ (similarly the same ‘Edit’ card can be accessed by clicking the edit icon). These will bring up the card details screen:



Name: Invoice Details
Description: Invoice Details

Invoice ID	Short Text
Invoice Date	Date and Time
Currency	List
Description	Short Text
Client Name	Short Text
Broker Name	Long Text

The following information can be set for a Custom Card:

Type	Description
Name	The card name
Description	A description for the card
Customized	Specifies if you want to customize the HTML, TypeScript and CSS of the card being rendered.

`

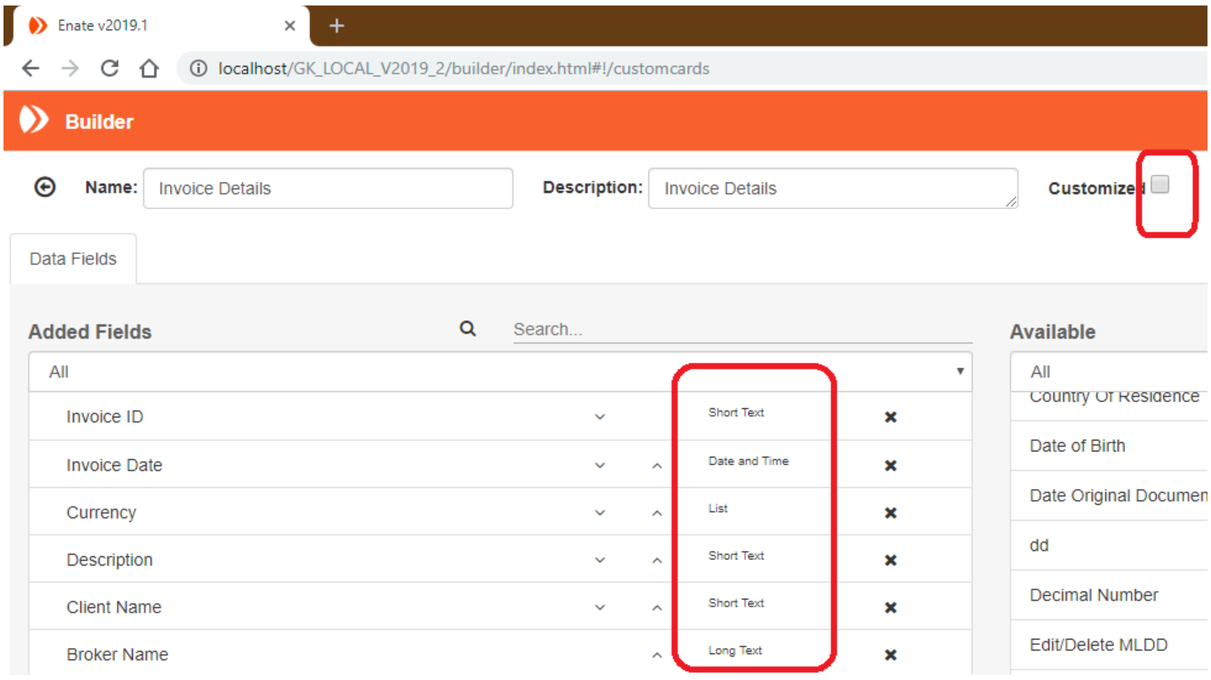
Standard and Customized cards

There are two types of custom cards in Enate. One is Standard Custom Card and other being Customized Custom Card.

Standard Cards

Standard custom cards are the cards in which we only configure the custom data that we want to display/capture in the Work Manager. The logic related to generating the HTML and supporting TypeScript is handled by the Work Manager.

In Standard Custom cards we just select the custom data that needs to be captured and select the order in which these custom data need to be displayed, that's is all we need to do. Once we provide these information Enate Work Manager will automatically decide the component to use in Work Manager based on the custom data type and displays it in the order that we have specified.



Name: Invoice Details
Description: Invoice Details
Customized ☐

Added Fields

All	
Invoice ID	Short Text
Invoice Date	Date and Time
Currency	List
Description	Short Text
Client Name	Short Text
Broker Name	Long Text

Available

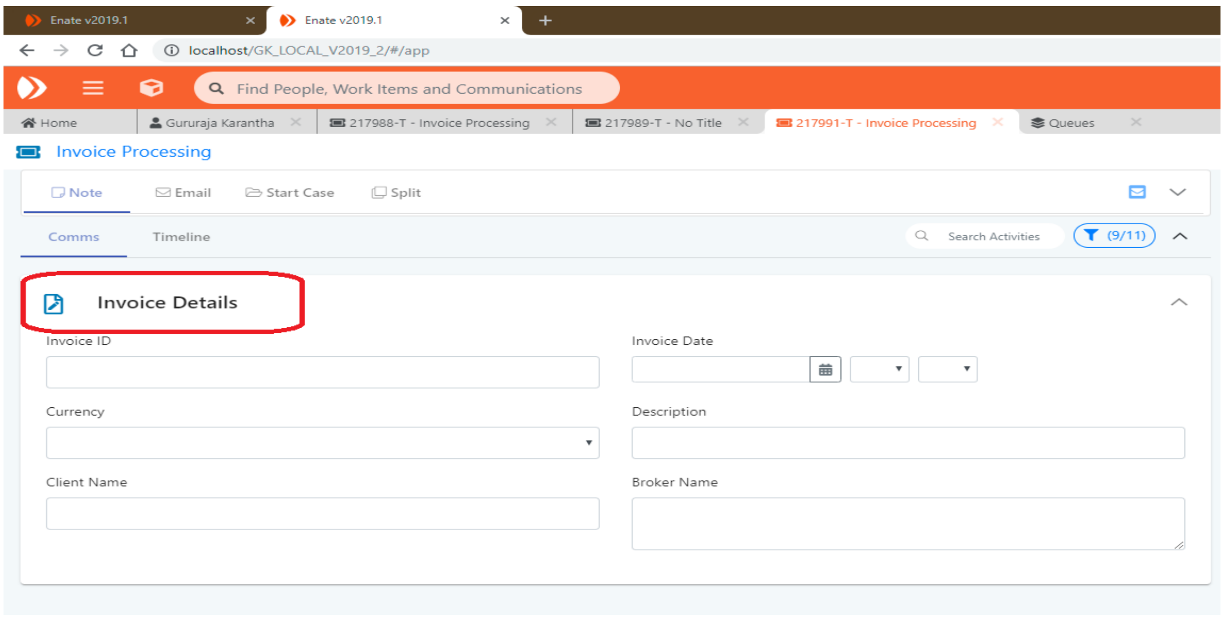
All
Country Of Residence
Date of Birth
Date Original Documen
dd
Decimal Number
Edit/Delete MLDD

In the above example we have created a **Standard Custom Card** named **Invoice Details.** To create a standard card, we just need to Create a card and "***Not Customize***" it. Clicking on the Customized checkbox converts a standard card to customized card.

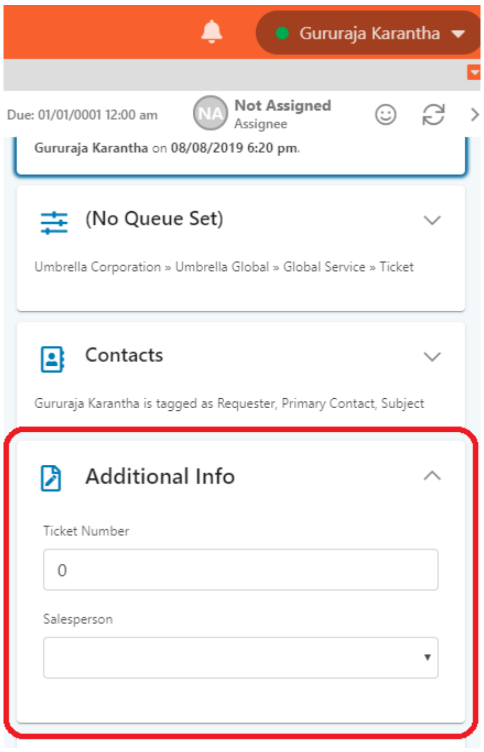
After giving the card name one needs to select the custom data that he needs to capture in Work Manager. After adding the fields into the Added Fields section, rearrange the fields in the order that it needs to be displayed in Work Manager. Use the Up and Down buttons to order the data fields. After finishing the ordering save the card using the save button.

Once the card is created user must link it with Ticket, Case and Action that he needs to display it. After linking the card one can see the card being rendered in Work Manager like this.

An Example card showing in Main-Section



An example card showing in Side-Section


Gururaja Karantha
Due: 01/01/0001 12:00 am
NA
Not Assigned
Assignee
Gururaja Karantha on 08/08/2019 6:20 pm.
(No Queue Set)
Umbrella Corporation » Umbrella Global » Global Service » Ticket
Contacts
Gururaja Karantha is tagged as Requester, Primary Contact, Subject
Additional Info
Ticket Number
0
Salesperson

If you look at the above example the Custom Card has automatically picked the right component to display based on the data type of Custom Data that we have added. For example, in case of List View data field 'Currency' the custom card has automatically picked right component 'HTML Dropdown' to show the available options and render them as well. Enate has pre-defined components for all the data field types we can create in Data Field section.

A standard card displays the right component for all the fields selected with value bound to the same field in Packet object. In all a standard card automatically takes care of showing and saving custom data for end users.

Customized Cards

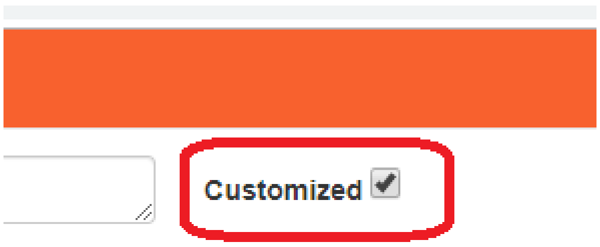
Enate always recommends using standard cards over customized cards as standard cards are lot more safe when compare to customized cards, but there will be cases where in we need to add extra business logic to the custom card or change the appearance of the cards, in those kinds of cases you can customize your existing cards and add extra logic using TypeScript code or change the appearance of your card by writing your own HTML. Any standard card when customized gives us the ability to write our own HTML, TypeScript and CSS.

Users should customize a card only when you are sure that the standard card does not meet the requirement and the customer is ready to take the risk of writing and maintaining custom HTML and TypeScript code.

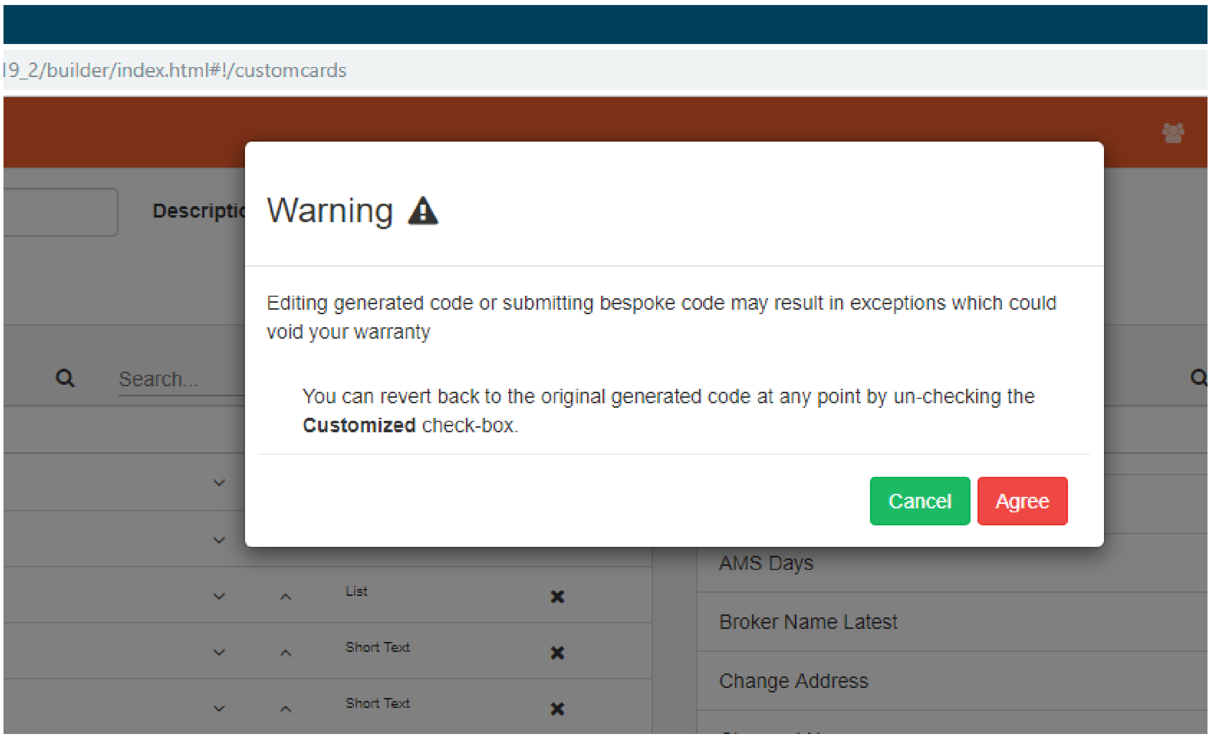
How to Customize a standard card

To customize a card first create a standard card and save it. After saving the standard card click on the customize checkbox to customize the standard card and see the HTML, TypeScript and CSS code.

In the example below we will be converting our previous standard card "Invoice Details" to be a customized card.



After this click on he Agree button to continue and create the customized card

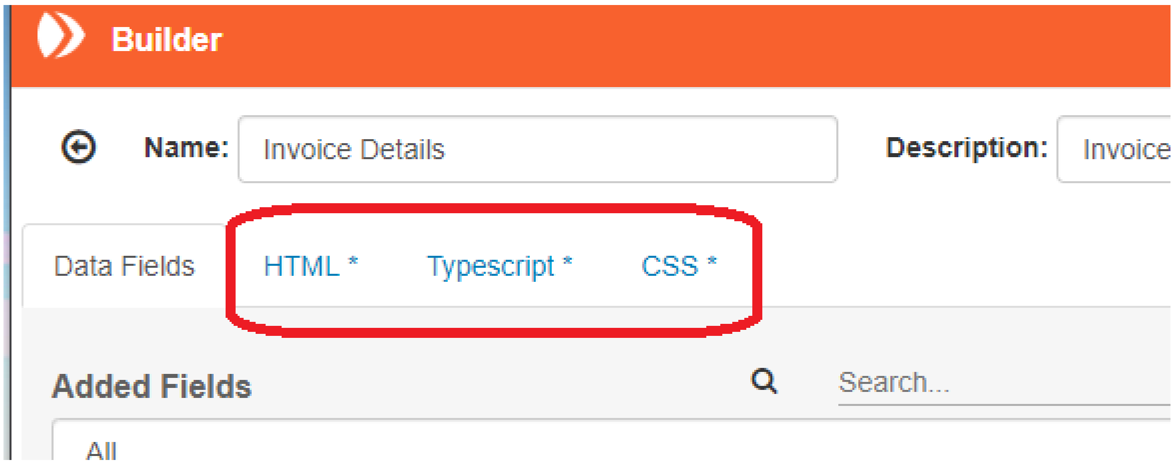


Warning

Editing generated code or submitting bespoke code may result in exceptions which could void your warranty

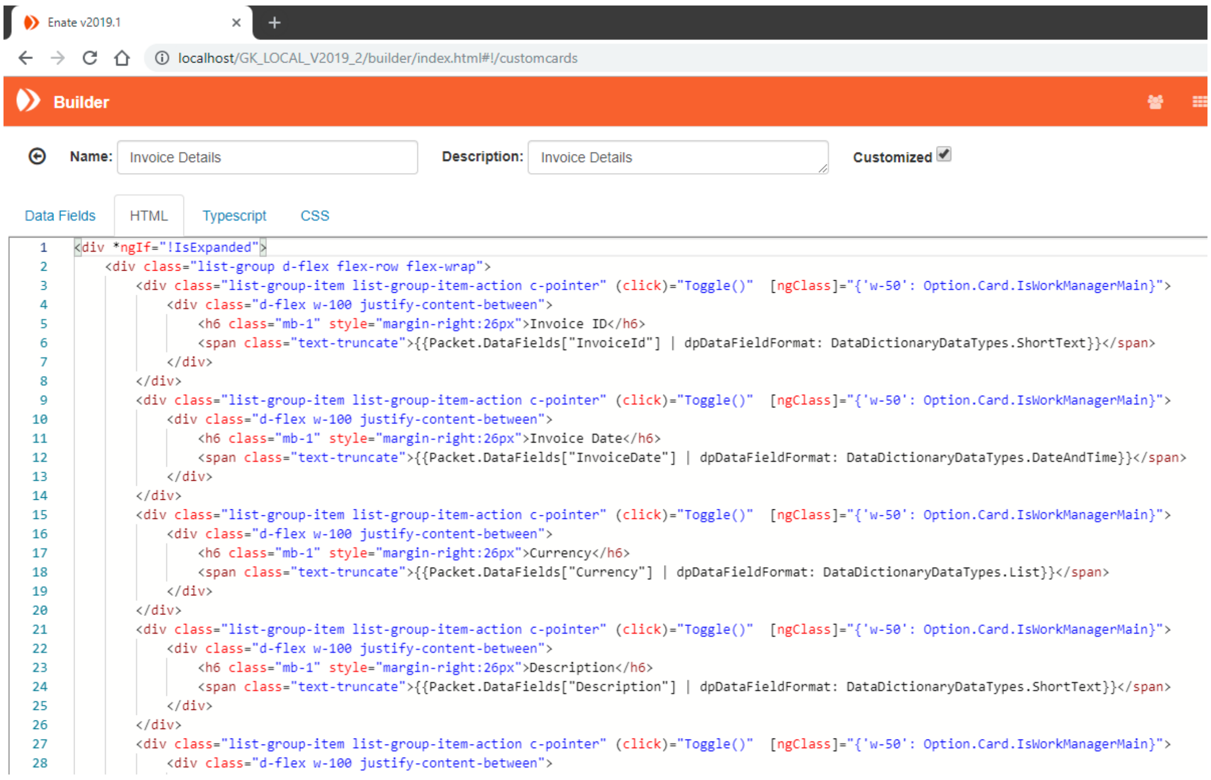
You can revert back to the original generated code at any point by un-checking the **Customized** check-box.

After this you will get to see the HTML, TypeScript and CSS of the card



After you customize the card you can see the actual HTML, TypeScript and CSS that Work Manager was using to render the card and change it according to your need.

HTML Code section that the card was using, allowed for modification by users.


Enate v2019.1

localhost/GK_LOCAL_V2019_2/builder/index.html#!/customcards

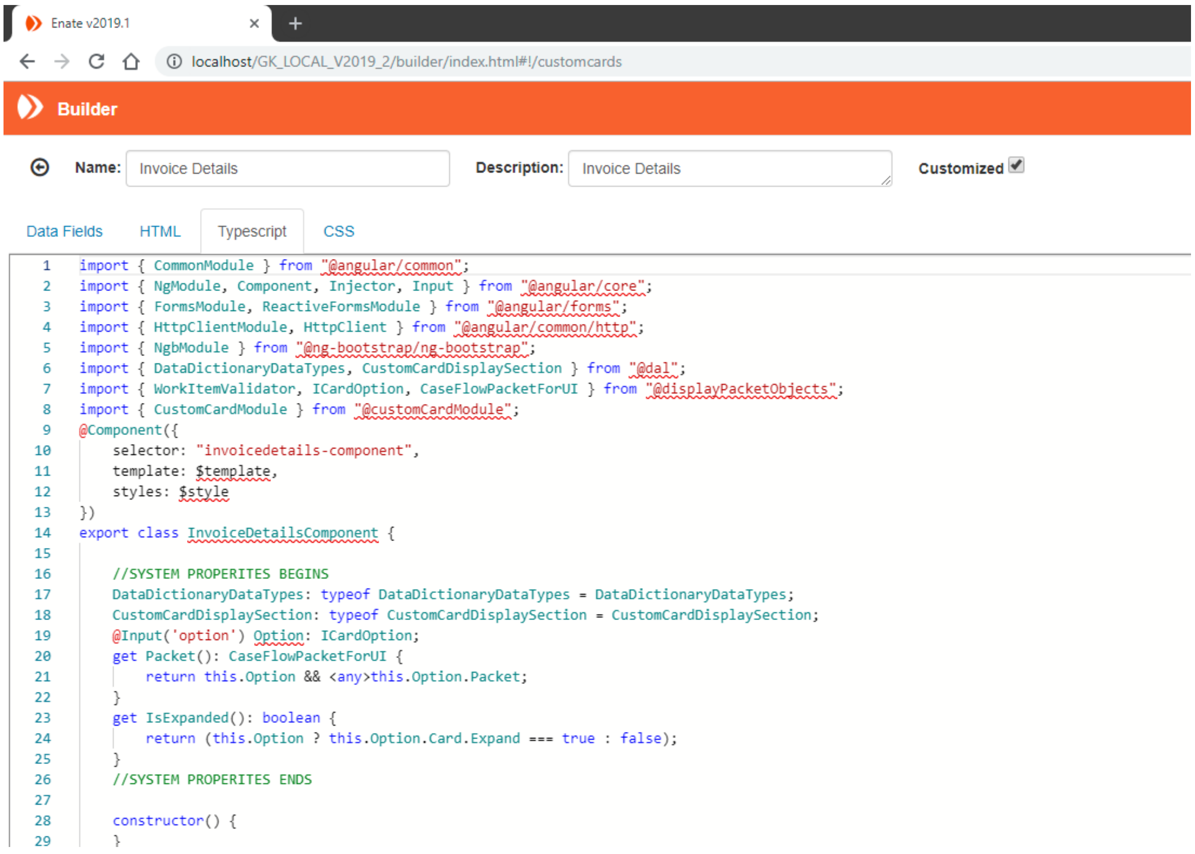
Builder

Name: Invoice Details Description: Invoice Details Customized ✓

Data Fields | HTML | Typescript | CSS

```html
<div *ngIf="!IsExpanded">
    <div class="list-group d-flex flex-row flex-wrap">
        <div class="list-group-item list-group-item-action c-pointer" (click)="Toggle()"  [ngClass]="{'w-50': Option.Card.IsWorkManagerMain}">
            <div class="d-flex w-100 justify-content-between">
                <h6 class="mb-1" style="margin-right:26px">Invoice ID</h6>
                <span class="text-truncate">{{Packet.DataFields["InvoiceId"] | dpDataFieldFormat: DataDictionaryDataTypes.ShortText}}</span>
            </div>
        </div>
        <div class="list-group-item list-group-item-action c-pointer" (click)="Toggle()"  [ngClass]="{'w-50': Option.Card.IsWorkManagerMain}">
            <div class="d-flex w-100 justify-content-between">
                <h6 class="mb-1" style="margin-right:26px">Invoice Date</h6>
                <span class="text-truncate">{{Packet.DataFields["InvoiceDate"] | dpDataFieldFormat: DataDictionaryDataTypes.DateAndTime}}</span>
            </div>
        </div>
        <div class="list-group-item list-group-item-action c-pointer" (click)="Toggle()"  [ngClass]="{'w-50': Option.Card.IsWorkManagerMain}">
            <div class="d-flex w-100 justify-content-between">
                <h6 class="mb-1" style="margin-right:26px">Currency</h6>
                <span class="text-truncate">{{Packet.DataFields["Currency"] | dpDataFieldFormat: DataDictionaryDataTypes.List}}</span>
            </div>
        </div>
        <div class="list-group-item list-group-item-action c-pointer" (click)="Toggle()"  [ngClass]="{'w-50': Option.Card.IsWorkManagerMain}">
            <div class="d-flex w-100 justify-content-between">
                <h6 class="mb-1" style="margin-right:26px">Description</h6>
                <span class="text-truncate">{{Packet.DataFields["Description"] | dpDataFieldFormat: DataDictionaryDataTypes.ShortText}}</span>
            </div>
        </div>
        <div class="list-group-item list-group-item-action c-pointer" (click)="Toggle()"  [ngClass]="{'w-50': Option.Card.IsWorkManagerMain}">
            <div class="d-flex w-100 justify-content-between">
```

TypeScript section that the card was using, allowed for modification by users.


Enate v2019.1

localhost/GK_LOCAL_V2019_2/builder/index.html#!/customcards

Builder

Name: Invoice Details Description: Invoice Details Customized ✓

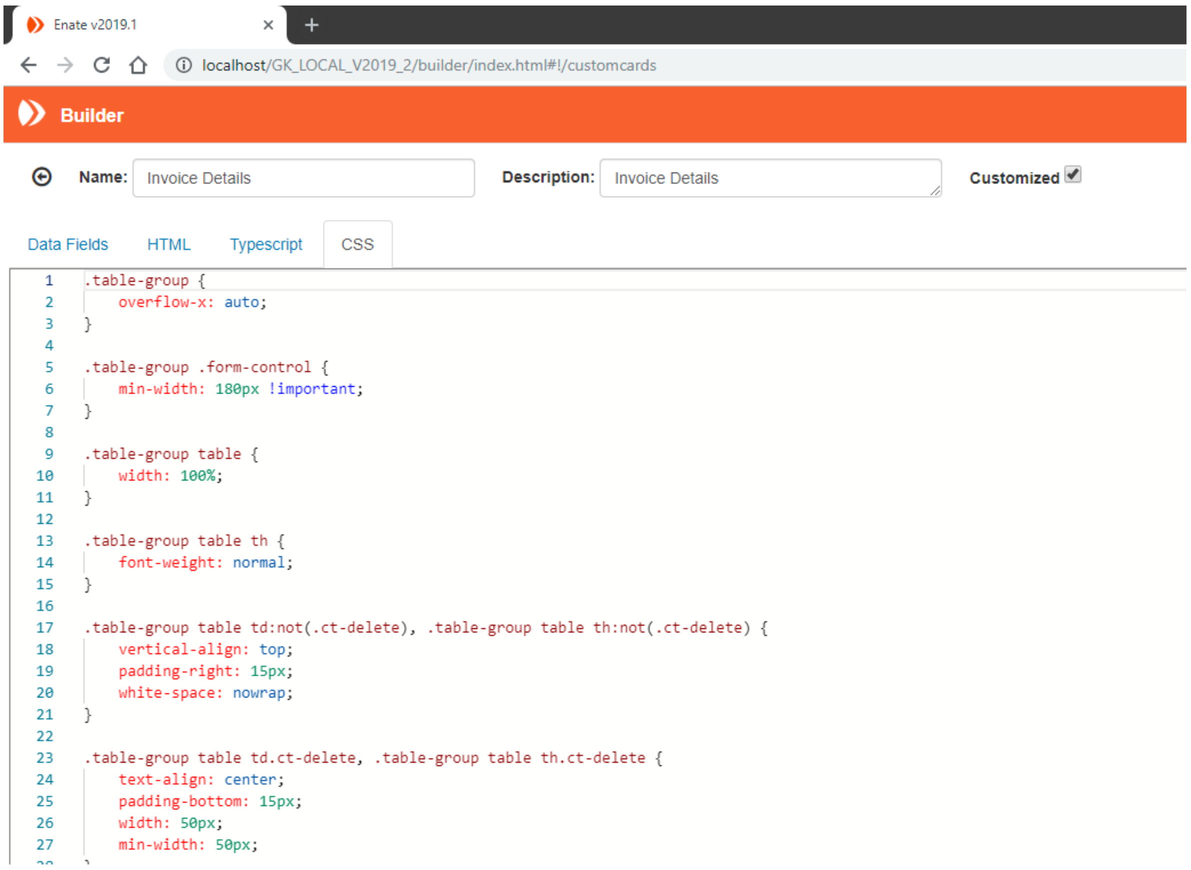
Data Fields | HTML | Typescript | CSS

```typescript
import { CommonModule } from "@angular/common";
import { NgModule, Component, Injector, Input } from "@angular/core";
import { FormsModule, ReactiveFormsModule } from "@angular/forms";
import { HttpClientModule, HttpClient } from "@angular/common/http";
import { NgbModule } from "@ng-bootstrap/ng-bootstrap";
import { DataDictionaryDataTypes, CustomCardDisplaySection } from "@dal";
import { WorkItemValidator, ICardOption, CaseFlowPacketForUI } from "@displayPacketObjects";
import { CustomCardModule } from "@customCardModule";
@Component({
    selector: "invoicedetails-component",
    template: $template,
    styles: $style
})
export class InvoiceDetailsComponent {

    //SYSTEM PROPERITES BEGINS
    DataDictionaryDataTypes: typeof DataDictionaryDataTypes = DataDictionaryDataTypes;
    CustomCardDisplaySection: typeof CustomCardDisplaySection = CustomCardDisplaySection;
    @Input('option') Option: ICardOption;
    get Packet(): CaseFlowPacketForUI {
        return this.Option && <any>this.Option.Packet;
    }
    get IsExpanded(): boolean {
        return (this.Option ? this.Option.Card.Expand === true : false);
    }
    //SYSTEM PROPERITES ENDS

    constructor() {
    }
```

CSS section that the card was using, allowed for modification by users.



Enate v2019.1

localhost/GK_LOCAL_V2019_2/builder/index.html#!/customcards

Builder

Name: Invoice Details

Description: Invoice Details

Customized ✓

Data Fields | HTML | Typescript | CSS

```css
.table-group {
    overflow-x: auto;
}

.table-group .form-control {
    min-width: 180px !important;
}

.table-group table {
    width: 100%;
}

.table-group table th {
    font-weight: normal;
}

.table-group table td:not(.ct-delete), .table-group table th:not(.ct-delete) {
    vertical-align: top;
    padding-right: 15px;
    white-space: nowrap;
}

.table-group table td.ct-delete, .table-group table th.ct-delete {
    text-align: center;
    padding-bottom: 15px;
    width: 50px;
    min-width: 50px;
```

About Customised Card HTML

When we customize a card Enate provided the HTML that it was using to render that card. This HTML tab will show the Labels and Components that it was using to render the Data Fields that the user had selected. Users can change the HTML to their needs in the HTML tab.

Users can change the HTML to change the appearance of the Card like changing the colour, label name or the display layout etc. Or write completely new sections of HTML and use native HTML elements as well.

But it is always advised to use Enate's components to capture and display Custom Data. Enate's components starts with **en8** prefix. Below given are some of the Enate's components.

`<en8-short-text>`

`<en8-date-time>`

`<en8-multi-level-select>`

About Customised Card TypeScript

When we customize a card Enate provided the TypeScript that it was using to render the standard card. Typescript is the backbone of a custom card, A TypeScript component is what gules the HTML, Packet Data and CSS all together to render the card.

Users should only **add** functions and properties to the TypeScript code they should never change the default code given by Enate.

Some of the common things that can be done in TypeScript is

- Perform validation on the data fields in the custom card.
- Provide information toast about the card data
- Red packet data and provide default values to the custom card contents.
- Respond to the common events provided by Work Manager.

About Customised Card CSS

When we customize a card Enate provided the places holder for you to put your custom CSS. Users are free to add the CSS that they will be using in card HTML.

Standard Events

When writing custom card code, one usually wants the card to react to some of the common events that happen in the screen. Enate provides these below standard events that custom card developers can utilize.

- StatusChanged -Fired when the Status of the packet gets changed.
- CategoryChanged - Fired when the Category is changed.
- OperationChanged - Fired when Packet operations tab gets changed.
- ContactModified - Fired when Contacts are either added or removed.
- FilesModified - Fired when Files are either added or removed.

Standard or Customized card?

One question that every user has is, should I use a Standard Card or Customized card or when should I customize a card. There is no easy answer to the above but below we list the advantages and disadvantages of both Standard and Customized cards.

Below we list the advantages of Standard card over customized card.

1. **HTML, TypeScript is internal to Enate** - In Standard cards the HTML, TypeScript and CSS is maintained by Enate and it is regularly update by for bug fixes, browsers support and other performance optimizations.
2. **Localization** - Localization is supported by Standard Card whereas Customized cards are generated only for English language as of now.
3. **Enate's breaking changes** - Breaking Changes by Enate is automatically handled by Enate as we maintain the HTML rendering logic internally.
4. **Angular's breaking changes** – Enate uses Angular as the Web technology to render the Work Manager website. And Angular which is developed by Google can introduce breaking changes which might make the TypeScript code written in Custom cards to break. These breaking changes introduced by Angular has be taken care by custom card developers in case of Customized card. Whereas with standard cards Enate will modify the card to cater for breaking change by Angular and it would be a seamless upgrade.

When it comes to customized cards the biggest advantage is the flexibility to write custom code and HTML, with this the possibilities are endless. But this also brings the disadvantage of maintaining the code.

Keeping in mid these above advantages/disadvantages it is always advised to use Standard Cards, if the feature is missing in standard card please raise a request for the same.

Custom Card Customisation

This section assumes that the reader is well versed with the usage of Custom Cards in Enate and knows HTML, TypeScript and CSS to further customise Enate's auto generated cards.

Custom Card Capabilities

Custom cards are mainly built to capture custom data in Enate. Custom Cards can be added to Tickets, Cases & Actions to capture bespoke information (Custom Data) on these work items as they run through process.

Coming to the types of custom cards, we can broadly classify them in to two categories

1. Data Update Cards
2. Integration Cards

Data Update Cards

A data update card is a custom card built solely to update data in Enate. These cards mainly display defined custom data and allow the users to update them using the basic controls like Text box, drop down etc.

In Data Update cards there are two flavours

1. Autogenerated cards
2. Customized cards

Autogenerated cards are those, which display the custom data by automatically generating the HTML layout in the backend.

Customized cards are those where in users change/modify the autogenerated HTML/TypeScript to add their own functionally for example configuring validations to custom data.

Configure Validation

This section explains and provides sample code for configuring validations for custom data.

Typescript is the backbone of a custom card, A TypeScript component is what gules the HTML, Packet Data and CSS all together to render the card.

When a card is customized, Enate by default provides the TypeScript that it was using to render the standard card and users should only add custom functions within the "`//YOUR CUSTOM CODE BEGINS` and `//YOUR CUSTOM CODE ENDS`" section and do not change the default rendered code.

Below is a sample example where you can see there are two custom data fields “Software Change” which is of data type “Short Text and “Fund Code” which is of data type “List” and we will add validations messages and make these custom data fields mandatory.

To make both of the fields mandatory, I have written the “CustomValidations” function within the export class Demo component and called the function within “ngOnInIT()” function.

Please Note:

1. For ease of code maintenance we have first defined the function and called the function within “ngOnInIT()” but alternatively you can write the entire validations within the “ngOnInIT()” itself.
2. Data field name to use in typescript will always be the safe name of the data field.

```
export class DemoComponent {
    //SYSTEM PROPERITES BEGINS
    DataDictionaryDataTypes: typeof DataDictionaryDataTypes = DataDictionaryDataTypes;
    CustomCardDisplaySection: typeof CustomCardDisplaySection = CustomCardDisplaySection;
    @Input('option') Option: ICardOption;
    get Packet(): CaseFlowPacketForUI {
        return this.Option && <any>this.Option.Packet;
    }
    get IsExpanded(): boolean {
        return (this.Option ? this.Option.Card.Expand === true : false);
    }
    //SYSTEM PROPERITES ENDS

    constructor() {
    }

    ngOnInit() {
        //YOUR CUSTOM CODE BEGINS
        this.CustomValidations();
        //YOUR CUSTOM CODE ENDS
    }
    CustomValidations() {
        this.Option.Card.Validators.push((packet: CasePacketForUI, cardOptions: ICardOption) => {
            const errors: string[] = [];
            if (!this.Packet.DataFields.SoftwareChange) {
                errors.push("Please provide Software Change Value");
            }
            if (!this.Packet.DataFields.FundCode) {
                errors.push("Please select Fund Code Value");
            }
            return WorkItemValidator.ERRORS(<CardForUI>cardOptions.Card, errors);
        }
        );
    }
}
```

Above validation example code is standard across all custom data types with the exception to multi-level drop down list.

Here is the sample code for multi-level drop down list.

```
        if (!this.Packet.DataFields.FieldName || this.Packet.DataFields.FieldName.indexOf('null') !== -
        1 || this.Packet.DataFields.FieldName.indexOf('Select') !== -1) {
            errors.push("Your error message here ");
        }
```

Here is the Enate Default Rendered TypeScript Example:

```
        import { CommonModule } from "@angular/common";
        import { NgModule, Component, Injector, Input } from "@angular/core";
        import { FormsModule, ReactiveFormsModule } from "@angular/forms";
        import { HttpClientModule, HttpClient } from "@angular/common/http";
        import { NgbModule } from "@ng-bootstrap/ng-bootstrap";
        import { DataDictionaryDataTypes, CustomCardDisplaySection } from "@dal";
        import { WorkItemValidator, ICardOption, CaseFlowPacketForUI } from "@displayPacketObjects";
        import { CustomCardModule } from "@customCardModule";
        @Component({
            selector: "demo-component",
            template: $template,
            styles: $style
        })
        export class DemoComponent {

            //SYSTEM PROPERITES BEGINS
            DataDictionaryDataTypes: typeof DataDictionaryDataTypes = DataDictionaryDataTypes;
            CustomCardDisplaySection: typeof CustomCardDisplaySection = CustomCardDisplaySection;
            @Input('option') Option: ICardOption;
            get Packet(): CaseFlowPacketForUI {
                return this.Option && <any>this.Option.Packet;
            }
            get IsExpanded(): boolean {
                return (this.Option ? this.Option.Card.Expand === true : false);
            }
            //SYSTEM PROPERITES ENDS

            constructor() {
            }
            ngOnInit() {
                //YOUR CUSTOM CODE BEGINS

                //YOUR CUSTOM CODE ENDS
            }
        }
        @NgModule({
            declarations: [DemoComponent],
            entryComponents: [DemoComponent],
            imports:     [CommonModule,     FormsModule,     ReactiveFormsModule,     HttpClientModule,
        CustomCardModule],
            providers: [HttpClient,
                {
                    provide: "widgets",
                    multi: true,
                    useValue: [{ name: "component-demo", component: DemoComponent }]
                }]
        })
export default class DemoModule { }
```

Integration Cards

Integration cards are those wherein we write completely bespoke code in Enate's custom cards to integrate with third party web apps or with third party WebAPIs.

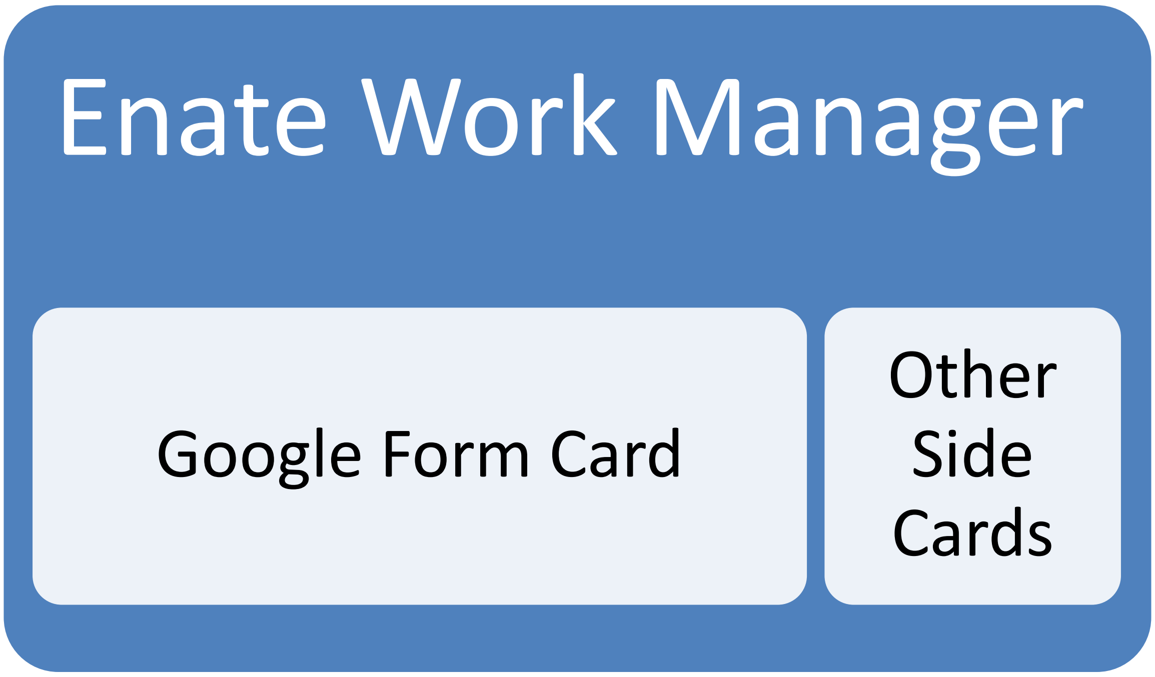
In integration cards there can to be two flavours

1. Loading third-party pages
2. Updating third-party data

Loading third party pages

In this method the Enate's custom card will be act like a host which just loads a third-party apps web page in an IFRAME inside the Enate. To be able to run show other systems apps in Enate the integrating application must allow their webpage to be loaded in an IFRAME.

One of the very common examples of this is loading of Google Forms inside an Enate Custom Card. Since Google forms properly allow loading and saving of webpages in other systems this can be very easily achieved in Enate.

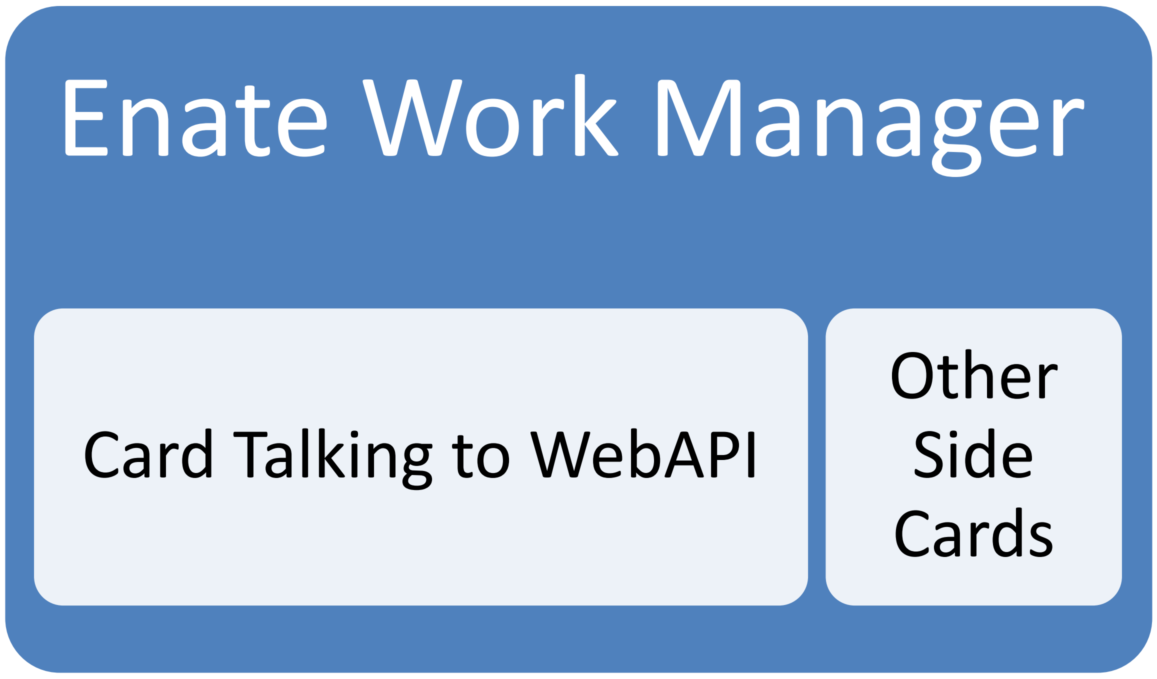


Updating third-party data

In this method the Enate's custom card will load its own UI but read and update data of a third-party system using their publicly documented WebAPIs. For this method to work the third-party system must expose well documented APIs which have a secure authentication and authorization mechanism.

The WebAPIs that Enate cards can integrate should follow industry standards and be **RESTfull** WebAPIs. Also, these WebAPIs should return and update data using industry standard data exchange format of **JSON** for Enate to be able to talk to it.

Through cards we should never try to integrate with a non-documented APIs of third-party systems



Conclusion

Custom cards are one of the most pivotal features of Enate, we will only further add more feature and make it more versatile and Enate's goal is provide the most user friendly experience out of the box at the same time allowing end users to extend it as well. Custom cards are the key bits which meet that. Thank you.