

PEGASYSTEMS PROVIDES THIS SOFTWARE 'AS IS', WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE, AND NONINFRINGEMENT. IN NO EVENT WILL PEGASYSTEMS, THE AUTHORS, OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES, OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT, OR OTHERWISE, ARISING FROM, OUT OF, OR IN CONNECTION WITH THIS SOFTWARE OR THE USE OR OTHER DEALINGS IN THIS SOFTWARE.

# Introduction

The Pega Angular Starter Kit shows how to construct an Angular application to utilize the Pega-APIs with layout and field information.

If a business user changes the layout of section, number of fields or type of fields; then, with the information in the Pega-API, the Angular application should be able to adapt without re-coding.

The Angular Starter Kit example builds a Case Worker portal which supports *any* simple Pega application.

## Pega-API

In Pega Infinity, Pega-API's for cases/casetypes/assignments were updated with new extension points to provide layout/field information. This information comes from Harness/Section model information and can be used as hints to allow your Angular displays to be model driven.

This allows your displays to be linked to the Pega application, such that if the business user changes the design via the Case Designer or App Studio, your display can instantly reflect the changes, without re-coding.

Layout and field information of a section/harness is passed through but organized differently to provide a simplified REST API JSON standard.

Support for Sections and Harness data as follows:

- Displays built via Case Designer
- Sections that utilize Dynamic Layouts
- Harnesses that utilize Screen Layouts
- "When" expressions that are run on server
- High level (limited) actions/event data
- Field information (control name, type, label, etc.)
- Repeating Dynamic Layouts

- Table (Grid)

Unsupported Layouts include:

- Layout group
- Columns
- Dynamic Layout Group
- Hierarchical List
- Dynamic Container
- AJAX Container
- Harness containers

Also unsupported at present:

- Expressions or client side when/expressions
- Non-auto generated sections
- Sections that include scripts
- Sections that are not guardrail complaint

New for 8.3

- DataPage for AutoComplete/Dropdown as embedded “options”
- ClipboardPage for AutoComplete/Dropdown as embedded “options”
- “newRow” for Repeating PageList/PageGroup
- Page Instructions for PageList/Page Group and Embedded Pages

Other

- `Work-.pyCaseDetails` does not work for UIKit:14.01 and beyond. So you can either keep you application at UIKit:13.01(or a lower version) or copy `Work-.pyCaseDetails` from UI-Kit-7:13-01-01 to your application ruleset.

Best practice is to avoid building a UI with deeply nested templates and sections (layouts inside of layouts).

Here are some examples of Angular generated UI based on the Pega-API.

## Main Portal

The screenshot shows the PegaApp Main Portal interface. The top navigation bar is purple and contains the PegaApp logo, version v 0.65, and a user menu for urtasks.manager. The left sidebar has a 'New +' button and a 'Recents' list. The main content area displays a 'Worklist' table with columns for Status, Category, and Urgency. Annotations in orange boxes highlight specific UI elements:

- /casetypes**: Points to the 'Bug' and 'TeamMember' buttons in the 'New +' section.
- /data/Declare\_pxRecents**: Points to the 'Recents' list in the sidebar.
- /authenticate-and-/data/D\_OperatorID**: Points to the user menu in the top navigation bar.

The 'Recents' list contains the following items:

- TeamMember | TM-397
- Bug | B-621
- Bug | B-629
- Bug | B-619
- Bug | B-618
- Bug | B-120
- Bug | B-87
- Bug | B-172
- Bug | B-93
- Run | R-14

The 'Worklist' table contains the following data:

Status	Category	Urgency
New	TeamMember	10
New	TeamMember	10
New	Task	10
New	TeamMember	10
New	TeamMember	10
Pending-Approval	Bug	10
New	TeamMember	10
New	Bug	10
New	Bug	10
New	Bug	10

The bottom of the page shows pagination information: 'Items per page: 10' and '1 - 10 of 185'.

## WorkList/WorkBasket

**Worklist**

Case	Status	Category	Urgency
TM-355	New	TeamMember	10
TM-356	New	TeamMember	10
T-54	New	Task	10
TM-354	New	TeamMember	10
TM-351	New	TeamMember	10
B-377	Pending-Approval	Bug	10
TM-352	New	TeamMember	10
B-413	New	Bug	10
B-108	New	Bug	10
B-423	New	Bug	10

Items per page: 10 1 - 10 of 186

## Work item

**Create a new bug**

**Bug**

Describe this bug

Steps to Reproduce

Urgency

**Creator**

Creator Email

Creator

My Link Label

Link Caption

Button

Cancel

Save

Submit

**Case details**

**Last updated by**

urtasks.manager in 4 hours

**Created by**

urtasks.manager in 4 hours

# Set Up and Installation

You will need to both create a simple application in Pega Infinity, then install Angular Starter kit on your local computer and configure the client to utilize your Pega application.

## Set Up

### Pega Infinity Set Up

You should create a simple application in Pega Infinity utilizing the Case Designer. Having an application that has a few steps and stages which contains fields to gather data, will help you understand how the Angular application works.

A sample Application, “CableConnect” has been provided to show case some of the functionality.

NOTE: With the latest release of this Angular Starter Kit v1.30, it is highly recommended that you use Pega Infinity 8.1.3 and above, as new functionality was added to the Pega-API DX. To utilize 8.3 functionality, you will need Pega Infinity 8.3 and above.

Your application should only utilize supported features (described above.) In addition, your simple application should:

- Be guardrail compliant
- See “Layouts” section below for currently supported “controls” and “layouts”
- If your case does not “skip new”, then you need to create a “New” harness that is simple:
  - Your “New” harness **MUST** be a screen layout
  - Suggest you include “pyNewCaseMain” section if you just want “relevant fields” (out of the box “New” includes this section) – For Pega-API, “pyNewCaseMain” is special and shouldn’t be copied.
- You will need a very simple “Confirm” harness
  - **MUST** be a screen layout
- If your application uses PageList/Group, you should have a default row to start with, even if the values are blank. (This can be avoided in 8.3 if you utilize “newRow” described in 8.3 Additional Functionality.)

Once you have created your application, it must have the following:

- The access group of the user you will login as in the Angular app, **MUST** include the role XXXX:PegaAPI (XXX is your application, you can reference PegaRULES:PegaAPI as a starter)
- If you want POST security, you can add privileges to your XXXX:PegaAPI role (see below)

## Security Privileges (add to your access role)

Access Class	Read instances	Write instances	Delete instances	Read rules	Write rules	Delete rules	Execute reports	Execute activities	Privileges	Settings	
Pega-API	5	5	5	5	5	5	5	5	pxCreateCaseDX (5) pxPerformAssignmentDX (5) pxUpdateCaseDX (5)	None	

+

These privileges will block POST data elements there are not part of the screen being posted.

There are 3 individual privileges:

- pxCreateCaseDX – will block when creating a case
- pxPerformAssignmentDX – will block when performing an assignment
- pxUpdateCaseDX – will block when updating a case

## Authentication

For demonstration of the starter kit, we will be using Basic Authentication. This very simplistic security and not recommended for any production application, but sufficient for early application development.

On your Pega Infinity system, open the Service Package record called “api” and edit the following:

- Authentication type: *Basic*
- Uncheck *Require TLS/SSL for REST services*
- Save the record

Next, open the Endpoint-CORS policy mapping record

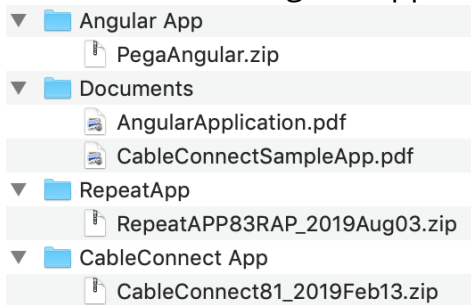
(Configure>Integration>Services>Endpoint-CORS policy mapping):

- the endpoint `api/` should have a policy of *APIHeadersAllowed*
- if there is a "cases" endpoint, delete it

## Angular Set Up

In order to edit/modify/run this Angular application, or create your own Angular application, you need to do the following:

- Install `yarn` on your computer if you don't already have it. Can be found here: <https://yarnpkg.com/en/>
- Get the Pega Angular Starter Pack zip file from Pega Exchange/Marketplace
- Should contain: Angular App, CableConnect & Documents directories.



- Retrieve `PegaAngular.zip` from Angular App directory.
- Unzip the starter kit into a separate directory
  - This will create subdirector called "PegaAngular"
- Execute the following commands, from your directory in a terminal window:

```
$ cd PegaAngular
$ yarn install
$ npm install
```

- In an IDE like Visual Studio Code or Atom, point to the directory `pegaapp-ng` find the subdirectory `/src/app/_services` and open the TS file "endpoints.ts"
  - Change the BASEURL to point to your server, noting that in the URL, after `prweb`, you should have `/api/v1` as in the example
  - Save
- Back to your terminal window, type :

```
$ yarn start
```

NOTE: as of this writing, if you are seeing errors like:

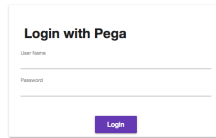
**ERROR in node\_modules/rxjs/internal/types.d.ts(81,44): error TS1005: ';' expected.**

Then you will need to revert rxjs...

```
$ npm install rxjs@6.0.0 --save
$ yarn start
```

- In your browser, go to <http://localhost:4200>
- Pega Angular application should display as the following:



A login form titled "Login with Pega". It contains two input fields: "User Name" and "Password". Below the "Password" field is a purple "Login" button.

If you have created a simple Pega application as described above, you should be able to login with an operator configured to use the application.

## Angular

The following Angular 6 example, utilizes the Angular CLI to build components, services, etc. If you want to edit the starter kit, you should install the Angler CLI, to facilitate adding components, services, etc.

```
$ yarn global add @angular/cli
```

The application utilizes “yarn” to build and package. You can set up Angular to use “yarn”:

```
$ ng config -g cli.packageManager yarn
```

A good idea to use saas (style sheet builder) instead of css, and we’ve used it in this application.

```
$ ng config schematics.@schematics/angular:component.styleext scss
```

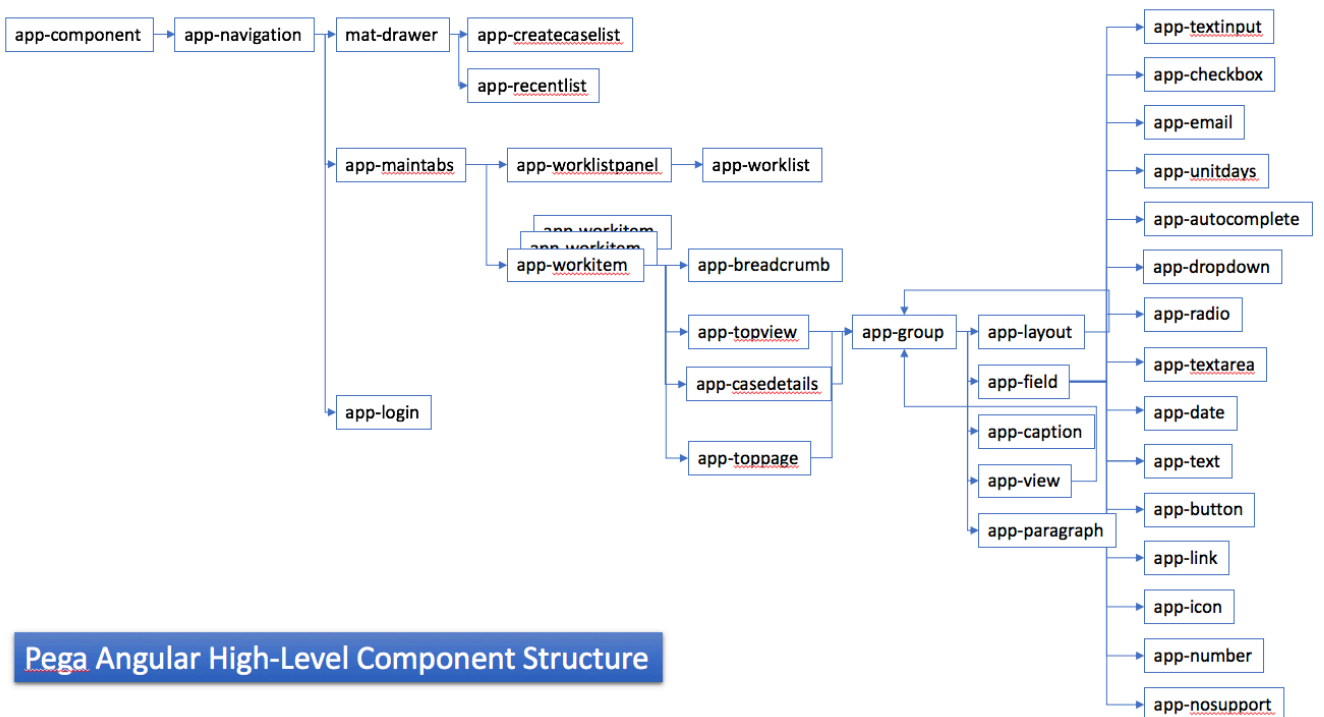
The application angular UI components are rendered using the Google’s Material framework as the basis for the display framework.

- <https://material.angular.io>

The Pega Angular Starter Kit, supports a number of different commonly used layouts, formats and controls and demonstrate how to utilize the data/hints from the Pega-API to create a dynamic Angular application. You can utilize these examples in your own Angular application.

# Structure

Angular application is built upon components. These components in turn can include other components, html, etc. Below is a very high-level structure diagram of the angular components created in this application and their inclusion in other components.



The following shows where the components would be located within the Angular application. This tree diagram doesn't show all the components, but gives a representation of the general layout and location.

## Top level Case Work Portal

The screenshot shows the top-level Case Work Portal interface. The header bar is purple and contains the PegaApp logo, a version number (v 0.65), and a user profile (urtasks.manager). The left sidebar is white and contains a 'New +' button, a 'Bug' button, a 'TeamMember' button, a 'Get Next Work' button, and a 'Recents' section. The 'Recents' section lists several items, including 'TeamMember | TM-397' and various 'Bug | B-...' items. The main content area is white and contains a 'Worklist' table. The table has columns for 'Status', 'Category', and 'Urgency'. The table lists several items, including 'New', 'Pending-Approval', and 'Bug'. Annotations with orange boxes and arrows point to various elements: 'mat-drawer' points to the sidebar, 'app-createscaselist' points to the 'New +' button, 'app-recentlist' points to the 'Recents' section, and 'app-login' points to the user profile in the header.

Annotations:

- mat-drawer
- app-createscaselist
- app-recentlist
- app-login

Status	Category	Urgency
New	TeamMember	10
New	TeamMember	10
New	Task	10
New	TeamMember	10
New	TeamMember	10
Pending-Approval	Bug	10
New	TeamMember	10
New	Bug	10
New	Bug	10
New	Bug	10

## Top level navigation

The screenshot shows the top-level navigation interface. The header bar is purple and contains the PegaApp logo, a version number (v 0.65), and a user profile (urtasks.manager). The left sidebar is white and contains a 'Dashboard' button, a 'B-636' button, and a 'B-637' button. The main content area is white and contains a 'Worklist' table. The table has columns for 'Case', 'Status', 'Category', and 'Urgency'. The table lists several items, including 'TM-355', 'TM-356', 'T-54', 'TM-354', 'TM-351', 'B-377', 'TM-352', 'B-413', 'B-108', and 'B-423'. Annotations with orange boxes and arrows point to various elements: 'app-maintabs' points to the 'Dashboard' button, 'app-worklistpanel' points to the 'Worklist' table, and 'app-worklist' points to the 'Urgency' column header.

Annotations:

- app-maintabs
- app-worklistpanel
- app-worklist

Case	Status	Category	Urgency
TM-355	New	TeamMember	10
TM-356	New	TeamMember	10
T-54	New	Task	10
TM-354	New	TeamMember	10
TM-351	New	TeamMember	10
B-377	Pending-Approval	Bug	10
TM-352	New	TeamMember	10
B-413	New	Bug	10
B-108	New	Bug	10
B-423	New	Bug	10

## Work item

The screenshot shows the 'Create a new bug' form in the PegaApp. The form is divided into two main sections: 'Bug' and 'Creator'. The 'Bug' section contains fields for 'Describe this bug', 'Steps to Reproduce', 'Urgency', and a 'Link Caption' field with a 'Button' button. The 'Creator' section contains fields for 'Creator Email', 'Creator', and 'My Link Label'. The form also includes a 'Cancel' button and a 'Submit' button. Annotations point to various parts of the form: 'app-breadcrumb' points to the breadcrumb trail 'Creation > Triage > Team Assign > Repair > Resolve'; 'app-workitem' points to the 'Bug' section; 'app-topview' points to the 'Cancel' button; and 'app-casedetails' points to the 'Case details' section on the right, which includes 'Last updated by' and 'Created by' information.

app-breadcrumb

app-workitem

app-topview

app-casedetails

## Layouts/Groups/Fields/Controls

The screenshot shows the 'Create a new bug' form in the PegaApp, focusing on the layout and controls. The form is divided into two main sections: 'Bug' and 'Creator'. The 'Bug' section contains fields for 'Describe this bug', 'Steps to Reproduce', 'Urgency', and a 'Link Caption' field with a 'Button' button. The 'Creator' section contains fields for 'Creator Email', 'Creator', and 'My Link Label'. The form also includes a 'Cancel' button and a 'Submit' button. Annotations point to various parts of the form: 'app-layout' points to the overall form structure; 'app-group' points to the 'Creator' section; and 'app-field/app-dropdown' points to the 'Urgency' field.

app-layout

app-group

app-field/app-dropdown

# Navigation

After the standard “app-component”, the top-level component is “app-navigation”. app-navigation delivers standard Pega case work portal functions for:

- Side drawer (slides out) – containing New, Get Next Work and Recents
- Log in/out drop down
- Main Tabs

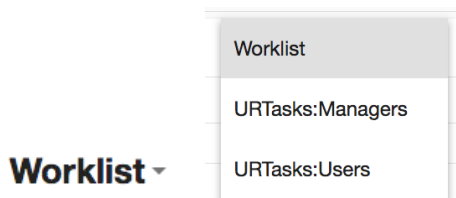
“app-login” component uses the Pega-API /authenticate along with basic authentication to login and uses local storage to save the authenticated user.

Following this, the login component will retrieve the Operator record via Pega-API /data for getting data pages using the data page “D\_OperatorID”. We will store the access group, work group and workbaskets (work queue) for later use in local storage.

The “app-navigation” component contains the “app-maintabs” component. This component contains an initial tab “Dashboard” (which cannot be dismissed), followed by other tabs which are created and dismissed as work items.

## WorkList

The “Dashboard” consists of a work list and a work list drop down. If the operator has multiple work queues (retrieved from operator id), this drop down will be visible and allow the user to switch between the worklist and different work baskets (work queues). If not, then only the “Worklist” label will be present.



Worklists are retrieved via the Pega-API /data for getting data pages. Here the datapage is “D\_Worklist”.

Workbaskets (work queues) are retrieved via the Pega-API /data for getting data pages, Here the datapage is "D\_WorkBasket" with a parameter of "WorkBasket" to determine what workbasket to show.

## Components

The whole Pega Angular application is built as components that contain components. When creating new/additional components, we use "ng cli":

```
$ng g component xxxx
```

where "xxxx" is the name of the component with an optional path.

The main components live in "app" directory. Sub components have been broken down into the following sub directories:

- \_subcomponents – components used by the main components
- \_fieldcomponents – components used by the field component (these are controls)

## Layouts

The "app-layout" component supports section layouts and their formats.

The following section layouts are supported:

- Dynamic Layout
- Repeating Dynamic Layout
- Table (Grid)
- Embedded Section

### Layout Formats

There are a number of different layouts that can be support, along with custom layouts. In this application, we are supporting the following layouts simplistically, so might not perfectly match what is shown via Pega Infinity:

- Stacked
- Action area

- Mimic a sentence
- Simple list
- Inline middle
- Inline grid double
- Inline grid 70 30
- Inline grid 30 70
- Inline grid triple

## Screen Layouts

For harnesses, Pega API support is limited to Screen Layouts (no Harness containers). The sample application supports the following screen layouts:

- header
- header\_footer
- content\_only

## Repeating Layouts

For repeating layouts, the sample application is supporting the following:

- REPEATINGROW (Table Layout – Grid)
- REPEATINGLAYOUT (Dynamic Repeating Layout)

NOTE: For repeats, you should have an initial row, even if the values are blank. If you've used the Case Designer to create this application, there should be a *pySetFieldDefaults* Data Transform; just set the first value in the first row to blank, should be sufficient. (eg. `.myRepeat(1).myProp equal to ""`). Again, 8.3 eliminates this need if you utilize "newRow" functionality.

## Controls

The following is a list of supported controls. These controls may not support ALL Pega functionality. These supported controls give you an example of how that is accomplished.

- pxTextInput
- pxCheckbox
- pxEmail
- pxUnitDays
- pxAutoComplete
- pxDropdown
- pxRadioButtons
- pxTextArea
- pxDate
- pxText
- pxButton
- pxLink
- pxIcon
- pxNumber
- pxPhone
- pxInteger
- pxCurrency
- pxUrl

## Control Error Handling

Control components utilize FormControl object to handle errors. For “required” field and a special case for “email” field, there is local validation using “validators” built in Angular. These local validations happen when “submit” button is pressed. We iterate through all the controls and call markAsTouched, then check if any are invalid. FormControl will display the error.

First Name \*

---

You must enter a value

Anything invalid, prevents a submit.



For server-side validation, the submitted form response will report validation errors in list format. The angler sample iterates through those errors, finds the reference control and add the error to the corresponding FormControl.

Phone

a

Invalid value specified for Phone. Value doesn't adhere to the Validate: ValidPhoneNumber

This type of validation utilizes both property validation and flow action validation expressed in your Pega application, and is recommended.

For generic page errors (with no specific control), the Snackbar control is used to display errors.

Validation failed: Errors Detected. .Contacts(1): Validation failed:  
Errors Detected. .Contacts(1).Phone: Invalid value specified for  
Phone. Value doesn't adhere to the Validate: ValidPhoneNumber

Ok

## Action Handling

Actions such as “click”, “change”, etc. will go to the actionsHandler class. This class will check for a corresponding event/action set and send out a corresponding action message. “app-workitem” component will handle the action messages.

## Classes

For code that is not a component, we create a TypeScript class file for it. We use ng cli to generate a class:

```
$ ng g class xxxx
```

where “xxxx” is the name of the class with an optional path.

The following classes are supported:

- handleactions – handles event/action set and sends corresponding message

- reference-helpers – similar to React, list of functions to convert full reference key/value pairs (flat, page list and page groups) to json context data

## Services

When retrieving data from the server, via REST API calls (Pega-API), we use angular services. Angular services are created via the angular ng cli:

```
$ ng g service xxxx
```

where “xxxx” is the name of the service including an optional path if desired.

Services definitions are stored in sub directory “\_services”.

The “ng cli” command, creates the correct files and injects the service into the root. In the service, the HttpClient component utilizes “GET, POST and PUT” HTTP commands to communicate with the Pega Infinity Pega-API rest service.

We have divided up the services into the following files:

- assignment
- case
- datapage
- user

These files contain all the Pega-API calls we have used in this application.

## Messages

Messages are another form of “services” described above. However, instead of communicating with a server, we are communicating with different components. Messages create a “pub/sub” functionality. This allows components to communicate without sharing variables, etc. Components publish messages, and those component(s) that register, will receive that message and can react/respond accordingly.

These are created with the ng cli to create a message (service):

```
$ ng g service xxxx
```

where “xxxx” is the name of the message including an optional path.

Messages are found in the sub directory “\_messages”.

There are many message service components, because there are many components that need to communicate. Messages implement the following patterns:

- messages that begin with “open” are usually received by the “app-maintabs” component (to create a new tab). After the tab is created, a corresponding “get” message is sent out, so the “app-workitem” component then follow up. (ie. openassignment is followed up with a “getassignment”)
- most messages are either received by “app-workitem” or “app-maintabs”

## Pipe

An Angular pipe, “safehtml” is used to handle that content that requires “innerHTML” processing. The safeHTML utilizes “bypassSecurityTrustHtml” for innerHTML processing. This currently is used for the “paragraph” component. Pega paragraph control is html with inline styles. Direct insertion of html into an html component (like a div/span) within Angular prevents inline styles from being applied. This pipe overrides this issue. (NOTE: if you are familiar with React, this is analogous to *dangerouslySetInnerHTML* )

This was created with the ng cli to create a pipe:

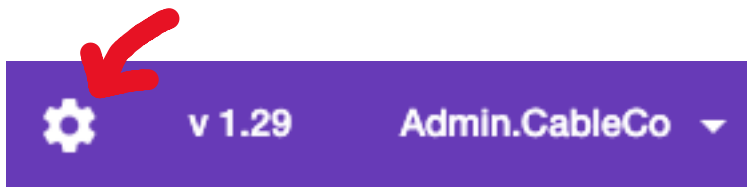
```
$ ng g pipe xxxx
```

where “xxxx” is the name of the pipe including an optional path.

## 8.3 Additional Functionality

New functionality has been added the Pega-API. To take advantage of these, but still keep compatibility with prior releases, we've added a settings option. Checking the new functionality in the settings dialog, will turn on usage of the new functionality. You switch back and forth to compare old vs new. However, new functionality will only work on an 8.3 and higher Pega Infinity systems.

NOTE: These settings are here for example only. Normally, in your code you would utilize new functionality if you choose without checkboxes to determine its usage.

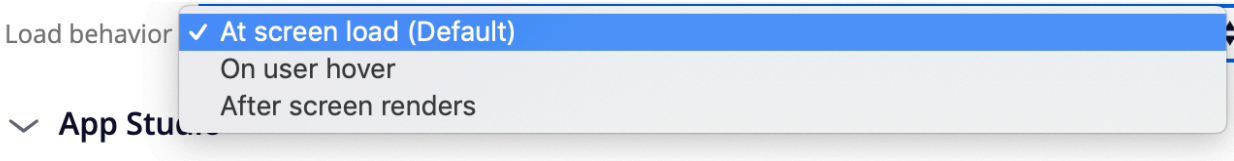


Pressing the settings icon will bring up the settings dialog. You can activate any of the new features by checking the checkbox. Ok will save the settings in local storage for the given user. They should remain after you log off. They are local to your browser and operator not your operator on the Pega Infinity platform.

**Application Settings**  
  
**Optimizations for 8.3+**  
  
☐ PageList/Group New Row  
☐ PageList/Group Use Page Instructions  
  
☐ Embedded Page Use Page Instructions  
  
☐ Autocomplete/Dropdown use local options for Data Page  
☐ Autocomplete/Dropdown use local options for Clipboard Page  
  
Ok      Cancel

## Embedded Options for Autocomplete/Dropdown

In 8.3, for Drop Downs that use Data Pages or Clipboard Page and have “Load behavior” set to “At screen load” for cell properties, the data page/clipboard page information will be now embedded in “modes/options”, similar to Prompt List/Local List. For Autocompletes that use Data Page or Clipboard Pages, page information will also be now embedded in “modes/options”.



What this means is now the data to populate the dropdown/autocomplete can be found embedded in the api request for the main view without having to make a second call to retrieve it for a data page or retrieve data from “content” for a clipboard page.

In addition to populating “key” and “value” properties in modes/options, there is also a “tooltip” property as well.

You can always retrieve data manually (like pre 8.3) from the server via a second call, if you are chaining drop downs, etc. where the selection of the first drop down influences (like a data page parameter) the results for the second drop down, and so on.

Checking the boxes “Autocomplete/Dropdown use local options for Data Page” and “Autocomplete/Dropdown use local options for Clipboard Page”, will utilize the embedded “modes/options” data and won’t make a possible second call to the server. Unchecking these boxes, reverts to original pre 8.3. Looking at the angular code, you can see how this is performed. You can check your server traffic, and you should will notice when these boxes are checked, there is no additional requests to the server for data pages if you are using dropdowns/autocompletes that are populated via data pages.

## newRow object for PageList/PageGroup

Prior to 8.3, you could not have a repeating PageList that didn't have a least one row. This is because you needed a row to sort of "clone" to send to the server, via `"/refresh"`, to add a row. And you needed to call the server for each new row.

With 8.3, we've added a `"newRow"` object to the `"layout"` object for repeating grids and dynamic repeating layouts. The `"newRow"` object is meant to be cloned and added or inserted into the existing `"rows"` object without having to go to the server via `"/refresh"`. The `"newRow"` object also has `"simple"` default values if they exist. By inserting this row in the `"rows"` object, your code can take the `"rows"` object and render the display.

When cloning the `"newRow"` and appending or inserting into `"rows"` you need to replace the `"token"` with a row number for a PageList and a row name for a PageGroup. The token is contained in the property `"listIndex"` for a PageList and `"groupIndex"` for a PageGroup. You should be able to `"stringify"` the JSON object, do a string replace and the parse it back into a JSON object.

Also note, if you chose to do an insert (as opposed to an append), your code will have to be more complex. As with an insert, all subsequent rows will have to be updated with the correct row number (all references, `refreshFor`, etc.). We have a working example in the Angular application.

Checking the `"PageList/Group New Row"` option in the settings will utilize the `"newRow"` object. If you check your network traffic, after this option is turned on, you will see that there is not any network traffic when rows are appended or inserted.

We've included RepeatApp simple application that demonstrates this repeating page functionality. RepeatApp should only be used on Pega Infinity 8.3 and above.

NOTE: Login/passwords are `user.repeat/pega` and `admin.repeat/pega`

## Page Instructions for PageList/PageGroup and Embedded Pages

Page Instructions was added in 8.3 as an additional return `"object"` to `"content"`. This object consists of a list of `"instructions"` to be carried out on the server. This was designed to overcome the need to POST the entire content when only a small

amount of data was changed. It over comes a Pega API limitation on large Embedded pages, when a single change of one element, requires the entire page to be returned, or data will be lost.

Page Instructions can be applied to an Embedded Page or a PageList/PageGroup. For PageList/PageGroup, we can “insert, append, delete, move” rows independently without posting the entire PageList/Group.

Checking the “PageList/Group Use Page Instructions” checkbox (you MUST check “PageList/Group New Row) will utilize Page Instructions when editing PageLists or PageGroup. If you turn this on, edit a Repeating Grid and check your network traffic, you will see the “pageInstructions” object as opposed to data changes in the “content” object.

Checking the “Embedded Page Use Page Instructions” will utilize Page Instructions when editing Embedded Pages. If you turn this on, an edit an Embedded Page, and check your network traffic, you will see the “pageInstructions” object as opposed to data changes in the “content” object.

You can examine the coding in the Angular app to get an example of how to utilize the Page Instructions. There some minor optimizations such that if you continuously operate on the same row (add row, change, data, etc.), then all will be combined to one pageInstruction, otherwise, there will be a separate Page Instruction for each operation. You can optimize further as your needs warrant.

NOTE: for “/refresh”, the Page Instructions are sent, but kept. Because when the final “submit” happens, the Page Instructions need to be resent, they were not saved via the “/refresh”.

# API Example

Here is a quick example of a display that is rendered like:

Address	
Street	
City	
State	
Postal Code	
Phone number	
(###) ###-####	

## JSON

Calling `/api/v1/assignments/{ID}/actions/{ACTIONNAME}` would return something similar to below JSON with layout/field information.

```
{
  "view": {
    "reference": "",
    "validationMessages": "",
    "viewID": "Address",
    "visible": true,
    "name": "Address",
    "appliesTo": "CableC-CableCon-Work-Service",
    "groups": [{
      "layout": {
        "visible": true,
        "titleFormat": "h2",
        "containerFormat": "NOHEADER",
        "groups": [{
          "field": {
            "validationMessages": "",
            "visible": true,
            "labelReserveSpace": true,
            "readOnly": false,
            "control": {
              "modes": [{
                "modeType": "editable",
                "controlFormat": "Standard",
                "textAlign": "Left",
                "tooltip": "",
                "maxChars": "",
                "formatType": "none",
                "specifySize": "auto",
                "minChars": ""
              }, {
                "modeType": "readOnly",
                "tooltip": "",
                "showReadOnlyValidation": "false",
                "formatType": "none"
              }
            ],
            "actionSets": [],

```



```

        "type": "pxTextInput"
    },
    "label": "Street",
    "type": "Text",
    "required": false,
    "reference": "Street",
    "labelFormat": "Standard",
    "disabled": false,
    "value": "",
    "maxLength": 256,
    "expectedLength": "",
    "fieldID": "Street"
}
}, {
    "field": {
        "validationMessages": "",
        "visible": true,
        "labelReserveSpace": true,
        "readOnly": false,
        "control": {
            "modes": [{
                "modeType": "editable",
                "controlFormat": "Standard",
                "textAlign": "Left",
                "tooltip": "",
                "maxChars": "",
                "formatType": "none",
                "specifySize": "auto",
                "minChars": ""
            }, {
                "modeType": "readOnly",
                "tooltip": "",
                "showReadOnlyValidation": "false",
                "formatType": "none"
            }
        ],
        "actionSets": [],
        "type": "pxTextInput"
    },
    "label": "City",
    "type": "Text",
    "required": false,
    "reference": "City",
    "labelFormat": "Standard",
    "disabled": false,
    "value": "",
    "maxLength": 256,
    "expectedLength": "",
    "fieldID": "City"
}
}, {
    "field": {
        "validationMessages": "",
        "visible": true,
        "labelReserveSpace": true,
        "readOnly": false,
        "control": {
            "modes": [{
                "dataPagePrompt": "pyFieldValue",
                "groupOrder": "asc",
                "listSource": "datapage",
                "textAlign": "Left",
                "tooltip": "",
                "enableGrouping": false,
                "groupBy": "",
                "minChars": "",
                "modeType": "editable",
                "dataPageValue": "pyFieldValue",
                "controlFormat": "",
                "dataPageTooltip": "",
                "dataPageID": "D_GetStates",
                "maxChars": ""
            }
        ]
    }
}

```

```

        "formatType": "none",
        "specifySize": "auto"
    }, {
        "modeType": "readOnly",
        "tooltip": "",
        "showReadOnlyValidation": "false",
        "formatType": "none"
    }],
    "actionSets": [],
    "type": "pxDropdown"
},
{
    "label": "State",
    "type": "Text",
    "required": false,
    "reference": "State",
    "labelFormat": "Standard",
    "disabled": false,
    "value": "",
    "maxLength": 256,
    "expectedLength": "",
    "fieldID": "State"
}
}, {
    "field": {
        "validationMessages": "",
        "visible": true,
        "labelReserveSpace": true,
        "readOnly": false,
        "control": {
            "modes": [{
                "modeType": "editable",
                "controlFormat": "Standard",
                "textAlign": "Left",
                "tooltip": "",
                "maxChars": "",
                "formatType": "none",
                "specifySize": "auto",
                "minChars": ""
            }, {
                "modeType": "readOnly",
                "tooltip": "",
                "showReadOnlyValidation": "false",
                "formatType": "none"
            }],
            "actionSets": [],
            "type": "pxTextInput"
        },
        "label": "Postal Code",
        "type": "Text",
        "required": false,
        "reference": "PostalCode",
        "labelFormat": "Standard",
        "disabled": false,
        "value": "",
        "maxLength": 256,
        "expectedLength": "",
        "fieldID": "PostalCode"
    }
}, {
    "field": {
        "validationMessages": "",
        "visible": true,
        "labelReserveSpace": true,
        "readOnly": false,
        "control": {
            "modes": [{
                "modeType": "editable",
                "controlFormat": "",
                "textAlign": "Left",
                "tooltip": "(###) ###-####",
                "maxChars": "",

```

```

        "formatType": "tel",
        "specifySize": "auto",
        "minChars": ""
    }, {
        "modeType": "readOnly",
        "controlFormat": "",
        "tooltip": "",
        "showReadOnlyValidation": "false",
        "formatType": "tel",
        "obfuscated": false
    }],
    "actionSets": [],
    "type": "pxPhone"
},
{
    "label": "Phone number",
    "type": "Text",
    "required": false,
    "reference": "PhoneNumber",
    "labelFormat": "Standard",
    "disabled": false,
    "value": "",
    "maxLength": 0,
    "expectedLength": "",
    "fieldID": "PhoneNumber"
}
}],
"groupFormat": "Stacked",
"layoutFormat": "SIMPLELAYOUT",
"title": ""
}
}],
},
"caseID": "CABLEC-CABLECON-WORK S-64",
"name": "Address",
"actionID": "Address"
}

```

## Corresponding Components

Stepping through the above JSON we would have the following in “app-workitem”:

- app-view
  - app-groups
    - app-layout
      - app-groups
        - app-field
          - app-textinput
        - app-field
          - app-textinput
        - app-field
          - app-dropdown
        - app-field
          - app-textinput
        - app-field
          - app-textinput