

# Developing Custom Forms in a Library

You can develop custom Pro-Code forms in a PAS Form Library to use these forms as Low-Code forms or sub-forms in the Designer. Either you can start from scratch, or you can export the form definitions of a Designer service (see [Modeling Forms](#)) as a Pro-Code PAS Form Library. The library already contains the necessary structure and meta-information to directly start coding.

The xUML Library Development Kit (`xlib`) helps you with this task. It can be used to build these libraries, and run them for testing purposes.

## Prerequisites



Developing custom form libraries for the **Scheer PAS** platform assumes a basic knowledge of **HTML**, **CSS**, **JavaScript/TypeScript** and the **Angular** framework. If you want to install the PAS xUML Library Development Kit, please contact our [Support team](#).

You need to have the following installed:

- [NodeJS](#)
- the [Angular CLI](#) (version 13.1.1)
- the PAS xUML Library Development Kit



### Expert Advice

For detailed information about multi-project workspaces, visit the [Angular documentation](#).

## Working with the xUML Library Development Kit (`xlib`)

After you have installed the `@pas/xuml-library-devkit`, you can switch to your project folder and execute the following commands to build a new multiple-projects workspace with a new library project in it.

### Creating Workspace and Library

The following table displays an overview on the main features of `xlib` to use with developing your own form libraries.

| Command                                    | Description  | Example                      |
|--|--|------------------------------|
| <code>xlib new &lt;library-name&gt;</code> | This command builds a new multiple-projects workspace with a new library project in it. The development kit automatically generates a test form and application. | <code>xlib new my-lib</code> |

### On this Page:

- [Prerequisites](#)
- [Working with the xUML Library Development Kit \(`xlib`\)](#)
  - [Creating Workspace and Library](#)
  - [Creating a Form](#)
  - [Building the Library](#)
- [Testing and Developing Your Form Component](#)
  - [Test Application Features](#)
  - [Starting the Test Application](#)
- [Using External Libraries](#)

### Related Pages:

- [Modeling Forms](#)

### Related Documentation:

- [test-lib.s.tar.gz](#)

Download the archived (empty) test project if you want to use it as a starting point for a new Pro-Code project.

- The project has been created by an `xlib new` command.
- Just perform an `npm install` to install the dependencies.

You will get the multiple-projects workspace <library-name> containing the following files:

#### Generated workspace files

```
/c/Projects/Devkit/my-lib (dev)
$ ls -la
drwxr-xr-x 1 u100106 1049089      0 Aug  9 14:46 ./
drwxr-xr-x 1 u100106 1049089      0 Aug  9 14:44 ../
/
-rw-r--r-- 1 u100106 1049089    274 Aug  9 14:44 .
editorconfig
drwxr-xr-x 1 u100106 1049089      0 Aug  9 14:44 .
git/
-rw-r--r-- 1 u100106 1049089    620 Aug  9 14:44 .
gitignore
drwxr-xr-x 1 u100106 1049089      0 Aug  9 14:44 .
vscode/
-rw-r--r-- 1 u100106 1049089   1139 Aug  9 14:45
angular.json
drwxr-xr-x 1 u100106 1049089      0 Aug  9 14:46
node_modules/
-rw-r--r-- 1 u100106 1049089   1285 Aug  9 14:46
package.json
-rw-r--r-- 1 u100106 1049089  843158 Aug  9 14:46
package-lock.json
drwxr-xr-x 1 u100106 1049089      0 Aug  9 14:45
projects/
-rw-r--r-- 1 u100106 1049089   1051 Aug  9 14:44
README.md
-rw-r--r-- 1 u100106 1049089    963 Aug  9 14:45
tsconfig.json
```

The library project is generated at path <library-name/projects/library-name>.

#### Projects Folder

```
/c/Projects/Devkit/my-lib/projects (dev)
$ ls -la
drwxr-xr-x 1 u100106 1049089 0 Aug  9 14:45 ./
drwxr-xr-x 1 u100106 1049089 0 Aug  9 14:46 ../
drwxr-xr-x 1 u100106 1049089 0 Aug  9 14:45 my-lib/
```

## Creating a Form

Your workspace project is created with an automatically generated PAS form from the @pas/xuml-library-devkit under the name **my-form**. To create your own form, switch to <library-name>/projects/<library-name>/src/lib> in your workspace project. You can now use the @pas/xuml-library-devkit to create a new PAS form using the following command:

| Command | Description | Example |
|---------|-------------|---------|
|---------|-------------|---------|

|   |  |                                      |
|---|--|--------------------------------------|
| <pre>xlib generate form &lt;form-name&gt;</pre> | <p>The Development Kit creates a new folder that contains all the files needed for a new PAS form.</p> <div data-bbox="380 205 883 758"> <p><b>Projects Folder</b></p> <pre>/C/Projects/Devkit/my-lib/projects/my-lib/src/lib/my-form (dev) \$ ls -la drwxr-xr-x 1 u100106 1049089  0 Aug 9 15:15 ./ drwxr-xr-x 1 u100106 1049089  0 Aug 9 15:15 ../ -rw-r--r-- 1 u100106 1049089  0 Aug 9 15:15 my-form.component.css -rw-r--r-- 1 u100106 1049089 270 Aug 9 15:15 my-form.component.html -rw-r--r-- 1 u100106 1049089 627 Aug 9 15:15 my-form.component.spec.ts -rw-r--r-- 1 u100106 1049089 767 Aug 9 15:15 my-form.component.ts -rw-r--r-- 1 u100106 1049089  57 Aug 9 15:15 my-form.interface.ts</pre> </div> | <pre>xlib generate form MyForm</pre> |
|---|--|--------------------------------------|

Use the `@PasForm` decorator in the **my-form.component.ts** file to set the name of the form in your Designer project and to define the events that can be used to trigger an execution in the BPMN process.

| @PASForm Decorator   |
|--|
| <pre>@PasForm( {   name: 'MyForm',   events: [     'submit'   ] })</pre> |

The Development Kit uses the **my-form.interface.ts** file to build the PAS Designer types for you, so use this file to define the types for this form.



Do not forget to export the form component class in the library interface file (**public-api.ts**) to make this new component accessible.



```
/*
 * Public API Surface of my-lib
 */

export * from './lib/my-lib.service';
export * from './lib/my-lib.component';
export * from './lib/my-lib.module';
export * from './lib/my-form/my-form.component';
```

## Building the Library

Make sure you generate the build folder before you build the library for the first time.

| Command | Description |
|---------|-------------|
|---------|-------------|

|            |  |
|------------|--|
| ng build   | <p>Generates the <b>build</b> folder.</p> <div>  You need to run this command once, before you can build the library for the first time. </div>   |
| xlib build | <p>Creates an <b>xlib</b> folder and builds the <b>&lt;library-name&gt;-library.xlib</b> package in the root folder.</p> <div>  Go to <a href="#">Administering Libraries</a> for detailed information on how to upload libraries. The usage of libraries is explained in detail on page <a href="#">Adding Libraries</a>. </div> |

## Testing and Developing Your Form Component

For testing purposes, your multiple-projects workspace comes with an automatically generated test application along with one Angular component.



### Expert Advice

For detailed information about multi-project workspaces, visit the [Angular documentation](#).

## Test Application Features

All mentioned files reside in folder **<name of the library>/projects/<name of the application>/src/app**.

| File                                    | Description  |
|---|--|
| <b>form-test-wrapper.component.html</b> | <p>Contains the <b>&lt;pas-child-form&gt;</b> element. This element is part of the <b>@pas/app-core</b> module and is also used by the Designer to show subform elements. With this element, you can test your Designer integration as close as possible.</p> <p>The <b>&lt;pas-child-form&gt;</b> element is preadjusted to use events in order to read the form's data. To test that, add text to the input field, open your browser console and click the submit button (see <a href="#">Starting the Test Application</a> below).</p> <div> <p><b>form-test-wrapper.component.html</b></p> <pre>&lt;pas-child-form [form]="form" (formEvent)="onFormEvent(\$event)" [formGroup]="reactiveForm" [data]="formData"&gt;&lt;/pas-child-form&gt;</pre> </div> |

|                                       |   |
|---------------------------------------|---|
| <b>form-test-wrapper.component.ts</b> | <p>Contains an import of your form component from your library's <b>public-api.ts</b> file and a form variable which allows you to use your component <b>MyFormComponent</b> in the <b>&lt;pas-child-form&gt;</b> element.</p> <pre> <b>form-test-wrapper.component.ts</b>  import { Component, Input, OnInit, Type } from '@angular/core'; import { AbstractFormComponent, FormEvent } from "@pas/app-core"; import { FormGroup } from '@angular/forms'; import { MyFormComponent } from "../../my-lib/src/public-api";  @Component({   selector: 'app-form-test-wrapper',   templateUrl: './form-test-wrapper.component.html',   styleUrls: ['./form-test-wrapper.component.scss'] }) export class FormTestWrapperComponent implements OnInit {   public form: Type&lt;AbstractFormComponent&gt; = MyFormComponent;   constructor() { }    ngOnInit(): void {    }    @Input() public reactiveForm: FormGroup = new FormGroup({});   public formData: any = {};    async onFormEvent(event: FormEvent) {     console.log(this.reactiveForm.value);   } } </pre> |
| <b>app.module.ts</b>                  | <p>Contains an import of <b>@pas/app-core module</b> which makes the <b>&lt;pas-child-form&gt;</b> element accessible.</p> <pre> <b>app.module.ts</b>  import { NgModule } from '@angular/core'; import { BrowserModule } from '@angular/platform-browser';  import { AppRoutingModule } from './app-routing.module'; import { AppComponent } from './app.component'; import { FormTestWrapperComponent } from './form-test-wrapper/form-test-wrapper.component'; import { AppCoreModule } from "@pas/app-core";  @NgModule({   declarations: [     AppComponent,     FormTestWrapperComponent   ],   imports: [     BrowserModule,     AppRoutingModule,     AppCoreModule,   ],   providers: [],   bootstrap: [AppComponent] }) export class AppModule { } </pre>   |

## app.component.html

Contains the selector of your **form-test.wrapper.component** to make your component appear.

### app.component.html

```
<app-form-test-wrapper></app-form-test-wrapper>
```



You can find how your HTML element is called in the .ts file of your generated **form-test-wrapper** component. It is the **selector** entry in the **@Component** decorator.

```
@Component({
  selector: 'app-form-test-wrapper' ,
  templateUrl: './form-test-wrapper.component.html' ,
  styleUrls: ['./form-test-wrapper.component.scss' ]
})
```

## Starting the Test Application

To start your test application, use:

```
ng serve my-form-test --configuration development
```

This outputs

```
** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/
**
```

## MyForm

Some text:

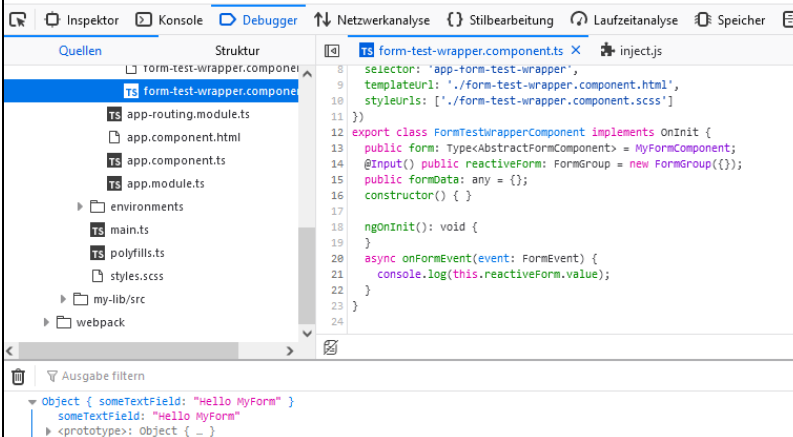
Submit

Open the mentioned URL **http://localhost:4200/** in your browser and you will see the following screen:

### MyForm

Some text: Hello MyForm

Submit



Enter some text in the input field in your form.

Open the developer console of your browser and click **Submit**. The entered text will be displayed in the console.

Now you can continue extending your form with individual elements and test them in the Designer.

# Using External Libraries

If you want to use external libraries in your Pro-Code forms, you need to add the external library to your own library project.

| Path                   | Step | Description   | Example Code  |
|------------------------|------|---|---|
| my-lib/projects/my-lib | 1    | Go to <b>my-lib/projects/my-lib</b> and execute the install command for the external library.                         | <pre>npm install ng2-pdfjs-viewer --save</pre>  |
|                        | 2    | Adjust the <b>ng-package.json</b> in the library folder to make the external library available for your applications. | <pre>"allowedNonPeerDependencies": [   "@pas/xuml-library-devkit",   "ng2-pdf-viewer" ]</pre> |