Web Client アプリケーションの開発

本マニュアルに記載の内容は、将来予告なしに変更することがあります。これらの情報について MSE（Magic Software Enterprises Ltd.）および MSJ（Magic Software Japan K.K.）は、いかなる責任も負いません。

本マニュアルの内容につきましては、万全を期して作成していますが、万一誤りや不正確な記述があったとしても、MSE および MSJ はいかなる責任、債務も負いません。

MSE および MSJ は、この製品の商業価値や特定の用途に対する適合性の保証を含め、この製品に関する明示的、あるいは黙示的な保証は一切していません。

本マニュアルに記載のソフトウェアは、製品の使用許諾契約書に記載の条件に同意をされたライセンス所有者に対してのみ供給されるものです。同ライセンスの許可する条件のもとでのみ、使用または複製することが許されます。

当該ライセンスが特に許可している場合を除いては、いかなる媒体へも複製することはできません。ライセンス所有者自身の個人使用目的で行う場合を除き、MSE または MSJ の書面による事前の許可なしでは、いかなる条件下でも、本マニュアルのいかなる部分も、電子的、機械的、撮影、録音、その他のいかなる手段によっても、コピー、検索システムへの記憶、電送を行うことはできません。

サードパーティ各社商標の引用は、MSE および MSJ の製品に対するコンパチビリティに関しての情報提供のみを目的としてなされるものです。

本マニュアルにおいて、説明のためにサンプルとして引用されている会社名、製品名、住所、人物は、特に断り書きのないかぎり、すべて架空のものであり、実在のものについて言及するものではありません。

Magic は Magic Software Japan K.K. の登録商標です。
Magic xpa は Magic Software Enterprises Ltd. のイスラエルその他の国での商標または登録商標です。
Magic xpa Enterprise Studio、Magic xpa Enterprise Client、Magic xpa Enterprise Server および Magic xpa RIA Server は Magic Software Japan K.K. の商標です。

一般に、会社名、製品名は各社の商標または登録商標です。
MSE および MSJ は、本製品の使用またはその使用によってもたらされる結果に関する保証や告知は一切していません。この製品のもたらす結果およびパフォーマンスに関する危険性は、すべてユーザが責任を負うものとします。

この製品を使用した結果、または使用不可能な結果生じた間接的、偶発的、副次的な損害（営利損失、業務中断、業務情報の損失などの損害も含む）に関し、事前に損害の可能性が勧告されていた場合であっても、MSE および MSJ、その管理者、役員、従業員、代理人は、いかなる場合にも一切責任を負いません。

Copyright 2021 Magic Software Enterprises Ltd.and Magic Software Japan K.K. All rights reserved.

2021 年 3 月 31 日

1 Web Client での行編集

2 Web Client でのルーティング

ルーティングとは **3**
Magic xpa でのルーティングについて **3**
1. ［ルート］イベント 3
2. ［ルート］イベントのロジックユニット 4
3. タスクのルート / ［名前］特性 4
4. パラメータ項目 / ［ルートに出力］特性 4
5. ［コール］処理コマンドの特性 4
6. ［ルート］イベントのロジックユニットのフロー 5
7. ［サブフォーム］コントロールのプロパティ 5
8. app.routes.ts ファイル 5
ルーティング処理 **6**

3 オーバーレイウィンドウのカスタマイズ

1. オーバーレイコンテナを用意する **9**
入力パラメータ 9
出力パラメータ 11
2. デフォルトの OverlayContainerMagicProvider の代わりに使用するプロバイダを指定します。 **11**
3. app.module.ts に次の変更を加えます。 **11**
デフォルトのオーバーレイウィンドウとカスタマイズされたオーバーレイウィンドウ 12
4. スタイルシート（my-overlay-container.component.css）を定義します。 **12**

4 ブラウザの履歴の操作

ブラウザの履歴を操作するための Angular のメソッド **14**
forward() 14
back() 14
Web ページを更新するための Java メソッド **14**
例 14

5 警告と確認メッセージのカスタマイズ

この機能の目的 **17**
ConfirmationComponentMagicProvider **17**
警告と確認メッセージをカスタマイズする手順 **18**
1. Angukar コンポーネントの作成 18
2. 基本クラスの拡張 18
3. HTML ファイルの編集 18
4. CSS ファイルの編集 18
5. magic.gen.lib.module.ts ファイルの編集 19
カスタマイズされた警告および確認メッセージの結果 **21**
確認メッセージ 21

6 ログインとログアウトの概念

Magic の関数 **22**
IsLoggedIn() 22
Logout() 22
Angular 関数 **22**
mg.isLoggedIn() 22
getIsLoggedIn() 22

7 Web モジュールについて

ngModule の導入における考え方 23
Magic xpa の Web モジュール？ 23
ルートモジュールと Web モジュール 24
Angular と Magic のフォルダ構造と相関 24
Is-Web-Module プロパティを使用する場合と使用しない場合の Magic フォルダ 24
Web モジュールプロパティ 24
Web モジュール生成プロセス 25
Web モジュールの生成手順 25
Web モジュールの生成シナリオ 26

8 Web Client アプリケーションでのMagic コンポーネントの使用

はじめに 30
用語 30
Angular モジュールでコンポーネントの使用 30
ルーティングを使用したロードオンデマンドによるコンポーネントの使用 31

9 カスタムプロパティの値が変更されるとJavaScript コードをトリガする

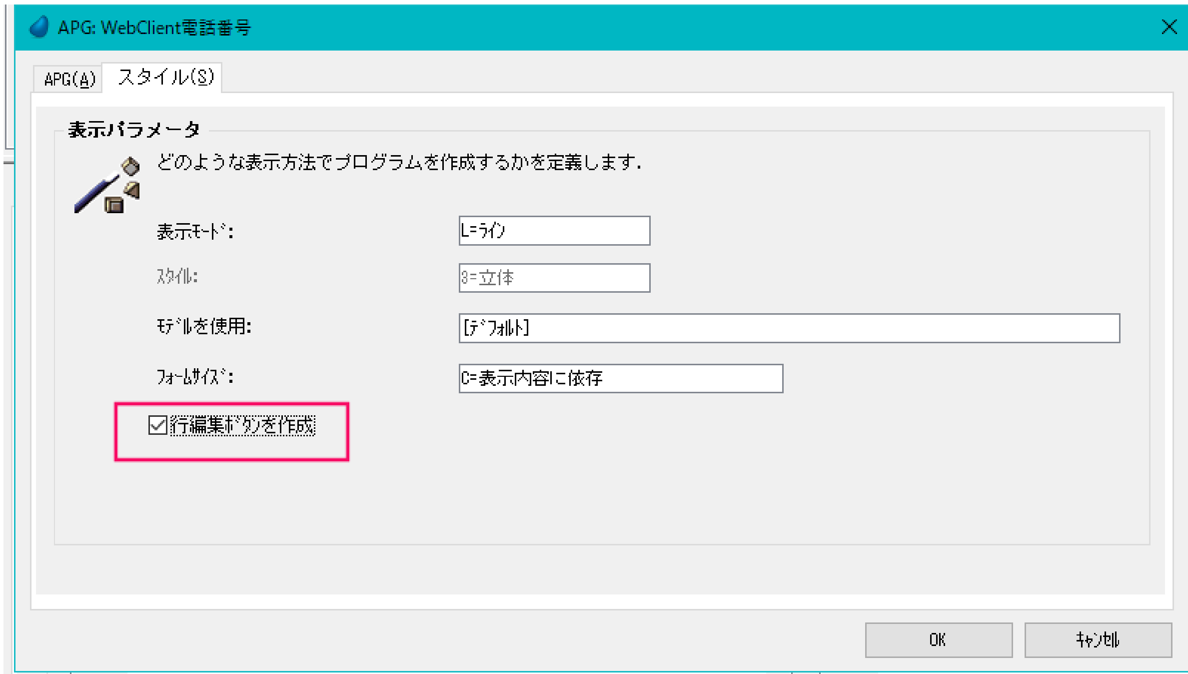
PropertyChanged 33

このホワイトペーパーは、Magic xpa の開発者が Web Client アプリケーションの開発中に重要な機能の実装を理解することを目的としています。

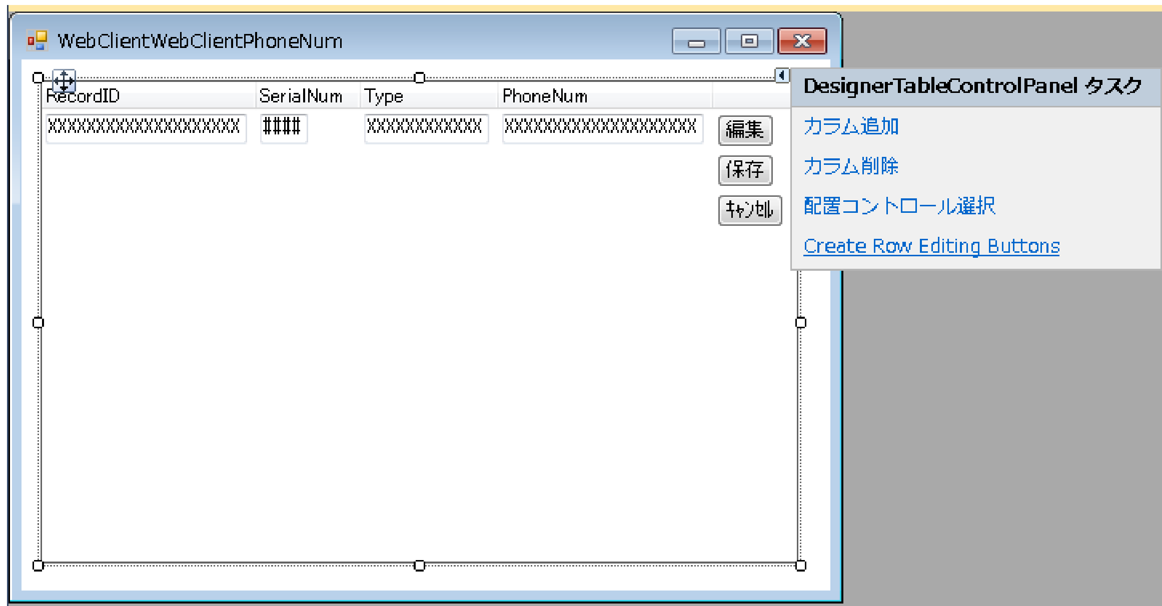
Web Client での行編集

このトピックは、Web Client での行編集の基本概念を理解するのに役立ちます。

APG で［行編集作成ボタンを作成］チェックボックスをオンにすると、フォーム上に［編集］、［保存］、［キャンセル］の 3 つのボタンが表示されます。



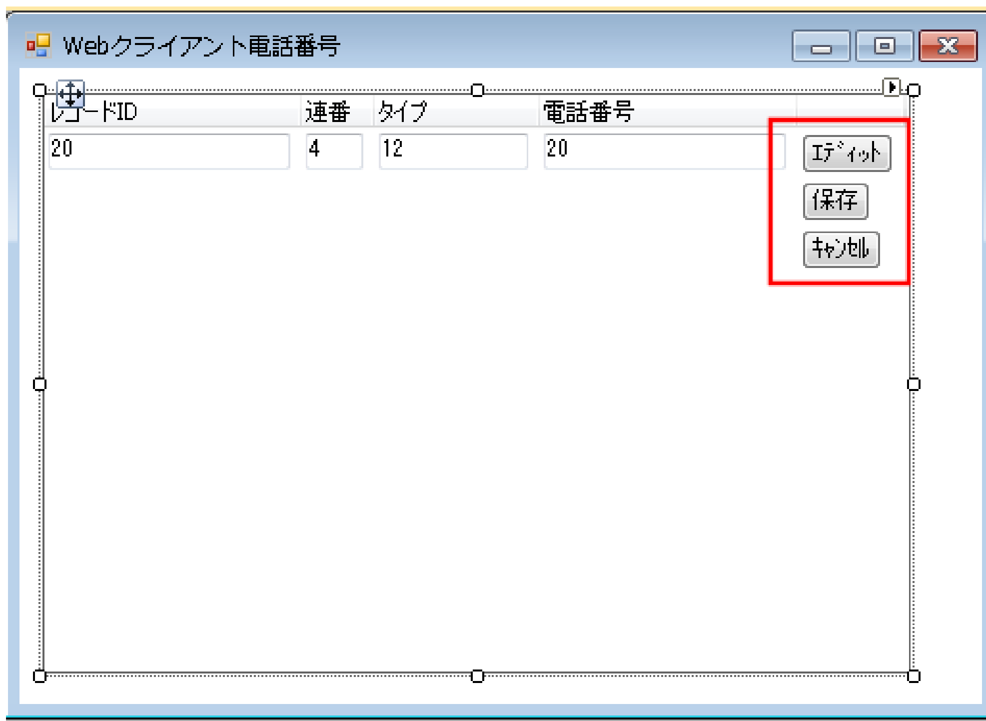
次のようにデザイナのアクションボタンをクリックして行編集ボタンを利用することもできます。



［行編集ボタンの作成］チェックボックスは、プログラム生成画面とフォーム生成画面の両方で使用できます。これはラインモード表示専用です。

これがフォームの編集ボタンの作成方法になります。

magic
OUTPERFORM THE FUTURE™



- 以下のイベントはそれぞれ 3 つのボタンに割り当てられています。
- 編集 …… ［行編集に入る］イベント
- 保存 …… ［レコード書込］イベント。［レコード再読込］パラメータを「False」に設定。.
- キャンセル …… ［キャンセル］イベント
  - ［行編集に入る］イベントは、行の状態を行編集の状態に変更します。レコードは、レコードを保存 / キャンセルするかレコードを終了するまで、行編集状態のままになります。
  - ［レコード書込］イベントに［レコード再読込］という名前のオプションパラメータが導入されました。イベントの実行時にデータベースからレコードを再読込するかどうかを指定することができます。
  - Magic xpa に IsRowEditing 関数が追加されました。現在パーク中の行が行編集状態の場合に True を返します。

mg.isRowInRowEditing という Angular 関数が追加されました。これは、行が行編集状態にある場合に True を返します。これは生成された html ファイルで使用され、編集、保存およびキャンセルボタンの表示を制御します。

次のコードスニペットは、Angular 関数の mg.isRowInRowEditing の使用方法を示しています。

```
<button
     mat-raised-button
    *ngIf="!mg.isRowInRowEditing(row)"
     color="primary"
     magic="EnterRowEditing"
    [rowId]="row.rowId"
    class="EnterRowEditingProps"
>
```

注意：

- 行を編集した後、他のレコードをクリックすると、編集したレコードへの変更が自動的に保存されます。
- 行編集モードは修正モードでのみ有効です。

Web Client でのルーティング

一般的にルーティングは、Web アプリケーションを作成する際に非常に重要になります。? ルーティングは、Web SPA（Single Page Application）のさまざまなページにアクセスできるようにするために不可欠です。Magic xpa には、Web アプリケーションにルーティングを実装するための機能があります。

ルーティングとは

ルーティングとは、SPA Web アプリケーション内での Web ページ間の移動操作です。

Magic xpa でルーティング機能を使用すると、Web Client アプリケーション内で次のことを実行することができます。

- ブラウザのアドレスバーに URL を入力して、目的のページに移動します。
- ページ上のハイパーリンクをクリックして、目的のページに移動します。
- クライアント（Angular、JS など）によるルートイベントを発行します。
- ブラウザの［戻る／進む］ボタンをクリックして、訪問したページの履歴を前後に移動します。

Magic xpa でのルーティングについて

Magic xpa でルーティングを実装するためには、ルーティング機能の次の重要な要素を理解する必要があります。

1. ［ルート］イベント
2. ［ルート］イベントのロジックユニット
3. タスクのルート /［名前］特性
4. パラメータ項目 / ［ルートに出力］特性
5. ［コール］処理コマンドの特性
6. ［ルート］イベントのロジックユニットのフロー
7. ［サブフォーム］コントロールのプロパティ
8. app.routes.ts ファイル

1. ［ルート］イベント

これは Web Client で導入された新しい内部イベントです。［ルート］イベントは、Magic で他のイベントを発生させる場合と同じように（つまりプッシュボタンに割り当てたり、Web ブラウザ（クライアント）に URL を設定したり、Angular でルートコマンドをアクティブにすることで）発生させることができます。全てのイベントとタスクイベントのリストで［ルート］イベントを検索することができます。

magic
OUTPERFORM THE FUTURE™

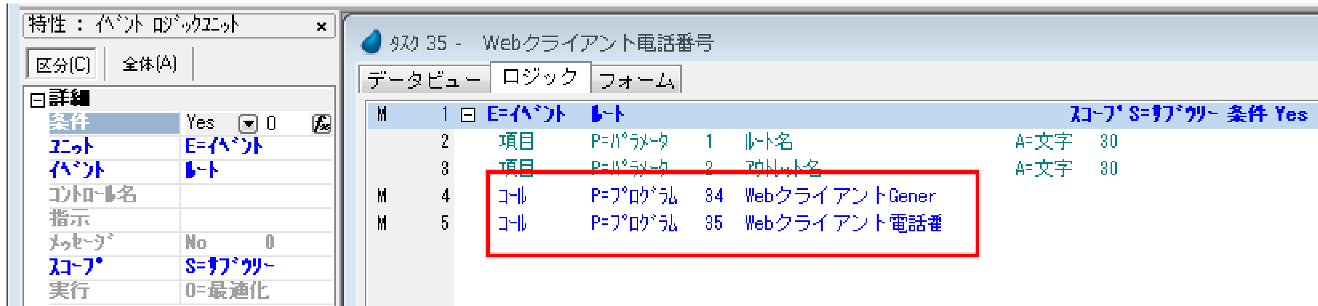
2. ［ルート］イベントのロジックユニット

「Web アプリケーションの設定］ダイアログの［スタートアッププログラム］に設定されたプログラムには、［ルート］イベントのロジックユニットを定義する必要があります。

［ルート］イベントのロジックユニットの作成時には、「ルート名」と「アウトレット名」という名前のサイズ 30 の 2 つの読み取り専用文字列パラメータが作成されます。これらのパラメータは、［特性］パレットおよびタスクロジックでも読み取り専用になります。

このロジックユニットには、［コール（プログラム / サブタスク）］、［ブロック（if,Else,End）］、［項目更新］の各処理コマンドのみを含めることができます。（サポートバージョン：4.8.1）

このロジックユニットは、ルーティングに関連したすべての［コール］処理コマンド含めることができます。



3. タスクのルート / ［名前］特性

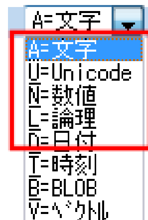
パスを指定するために、［タスク特性］の［インタフェース］タブの［ルート］グループに［名前］特性が追加されました。この特性は、タスクコンポーネントへのマッピングには直接使用されることはありません。必要に応じて、呼び出し先のプログラム／タスクに設定された特性値が［コール］処理コマンドの［ルート名］特性にコピーされます。

4. パラメータ項目 / ［ルートに出力］特性

［ルートに出力］という名前の特性がタスクのパラメータ項目に追加されました。値は、Yes/No（式なし）で設定します。

この特性が「Yes」に設定されている場合、ルーティングを行う場合の URL にはこのパラメータが必要です。

ルートでのパラメータの使用



出力できるのは、単純型のパラメータ、つまり、文字型、Unicode 型、数値型、または論理型です。最小のパラメータをルートに公開する必要があります。ルートでのパラメータの標準的な用途は、呼び出されたプログラム / タスク内のレコードの範囲／位置付です。

5. ［コール］処理コマンドの特性

［コール］処理コマンドの標準的な特性に加え、2 つの新しい特性が追加されました。

- ルート名 …… この特性は、呼び出されたプログラム / タスクの［ルート / 名前］特性から初期値を取得します。この値は変更することができます。
- ルートパラメータ …… これは、呼び出されるタスクへのルーティングに必要なパラメータを表す読み取り専用の特性です。パラメータは、呼び出されたプログラム / タスク内の各パラメータの［ルートに出力］特性の値が「Yes」の場合、自動的に定義されます。

［コール］処理コマンドの「ルート名」特性に指定されたルート名で識別されるプログラムを呼び出すことができます。［ルート名］特性は、URL によるルーティングで呼び出す場合、URL 内のルート名としてこの名前を指定する必要があります。

［出力先］特性は、ルーターアウトレットとなる［サブフォーム］コントロールを指定してプログラムを呼び出す必要がある場合に使用することができます。

6. ［ルート］イベントのロジックユニットのフロー

［ルート］イベントのロジックユニットで［コール］処理コマンドを実行すると、独特の動作が発生します。

- ［条件］特性 …… この特性で入力した式の上に、Magic xpa は内部条件を追加します。この条件に基づいて、Magic xpa は［コール］処理コマンドの［ルート名］特性の値とパラメータ項目「ルート名」を一致させます。
- ［出力先］特性 …… この特性が空の場合、出力先は［ルート］ロジックユニットが配置されているタスクのデフォルトルートのサブフォーム（［デフォルトアウトレット］プロパティが「True」）になります。ルートをデフォルト以外のアウトレットで実行させる場合は、ルートアウトレット（［ルータアウトレット］プロパティが「True」）として定義されているサブフォームのコントロール名を指定し、［デフォルトアウトレット］プロパティを「Falsei に設定する必要があります（複数のサブフォームがある場合でも）。
- コールプログラム / タスクのパラメータ …… ［コールプログラム / タスク］処理コマンドで任意のパラメータを送ることができます。これらのパラメータは、他の Magic パラメータのように実行時に評価されます。ただし、パラメータを［ルートに出力］として定義した場合、それらのパラメータは URL に指定されているものと見なされ、関連するパラメータの実行時の値が上書きされます。つまり、Magic xpa が呼び出されたプログラム / タスクに送られる値は、ルートトリガーによって変わります。Magic xpa 内から［ルート］イベントを発生させた場合、値は Magic に基づきます。

［コール］処理コマンドの［ルート名］特性がパラメータ項目「ルート名」と一致し、［コール］処理コマンドの［出力先］特性が空またはパラメータ項目「アウトレット名」と一致し、追加の条件式（存在する場合）が True と評価された場合、［コール］処理コマンドが実行されます。

7. ［サブフォーム］コントロールのプロパティ

サブフォームは、ルーティングタスクすなわちルータアウトレットのコンテナとして使用されます。2 つのプロパティ（［ルータアウトレット］と［デフォルトアウトレット］）が、Web Client タスクの［サブフォーム］コントロールに追加されています。

- ルータアウトレット …… サブフォームをルーターアウトレットとして機能させる場合は、このプロパティは「True」に設定されています。つまり、このプロパティを「True」に設定すると、サブフォームはルーティングされたコンテンツのプレースホルダのように機能します。
- デフォルトアウトレット …… 1 つのフォームに複数のルータアウトレットを含めることができますが、［ルート］イベントに対する［イベント］ロジックユニットでデフォルトアウトレットとして定義できるのは 1 つだけです。［コール］処理コマンドの［出力先］特性を設定せず、パラメータ項目「アウトレット名」をイベントに送らなかった場合は、デフォルトのアウトレットが使用されます。2 つを指定した場合は、同等のコントロール名が定義されたサブフォームが使用されます。

8. app.routes.ts ファイル

app.route.ts ファイルはルートパスを格納します。

このファイルは、<Web アプリケーションフォルダ >¥< アプリケーション名 >¥src¥app フォルダに作成されます。

Web アプリケーション作成のウイザードで［ルートマップを作成］をチェックすることで作成／更新されます。ルートマップが正しく作成されていないと、以降のルーティング処理は実行されません。

ファイルには、次の 3 つのパラメータが含まれています。

- Routes …… Routes は path（'/persons' や '/cart' など）を Angular コンポーネントにマッピングします。任意の URL A/B/C において、文字列「A」、「B」、および「C」は、各 Angular コンポーネントを識別するルート名となります。.
- Component …… Magic xpa はルーティングのために、ここでは汎用名（RouterContainerMagicComponent）を使用します。コンポーネント /Magic xpa タスクは、実行時に Magic xpa 自身によって決定されます。
- Outlet …… これは、当該コンポーネントが表示される HTML 内の領域です。［サブフォーム］コントロールのコントロール名が設定されます。

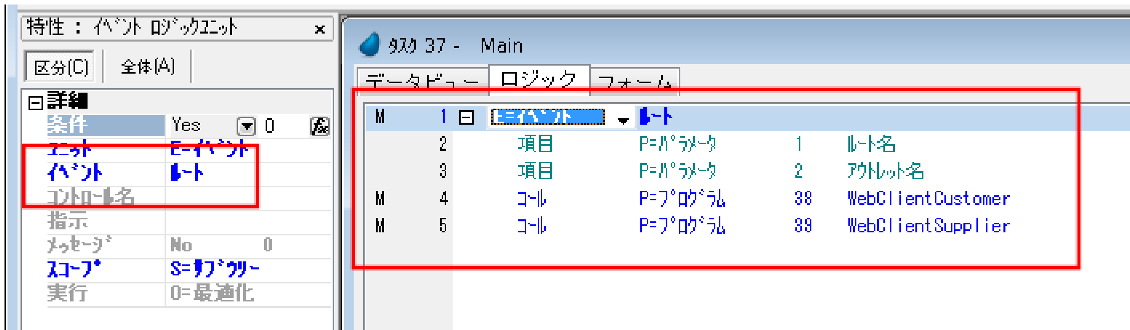
app.route.ts ファイルの内容は、次のようになります。

```
import {
    Routes,
    RouterModule
} from '@angular/router';
```

```
import {
    RouterContainerMagicComponent
} from "@magic-xpa/angular";

import {
    CommonModule
} from "@angular/common";

import {
    NgModule
} from '@angular/core';

export const routes: Routes = [
    {
        path: 'Customer/:Parameter1/:Parameter2',
        component: RouterContainerMagicComponent,
    },
    {
        path: 'Supplier/:Parameter1/:Parameter2',
        component: RouterContainerMagicComponent,
    },

];

......
```

ルーティング処理

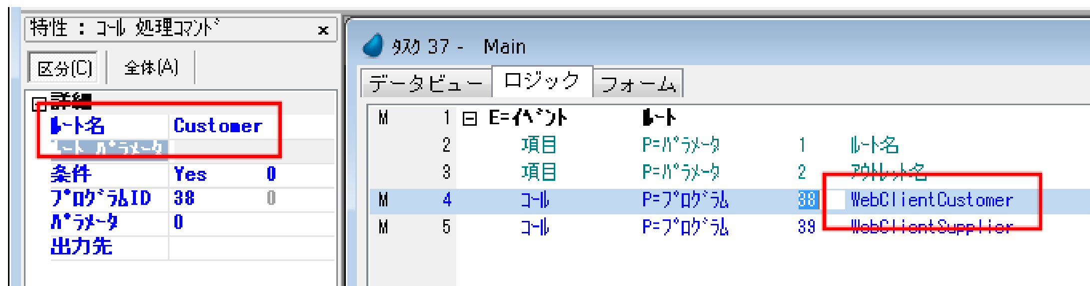
Magic のルーティング処理を理解するために、以下のようにルートタスクと 2 つのルーティングされるタスクがある例を見てみましょう。



ルートタスクには、そのフォームに関する以下のコントロールが含まれています。

- 2 つの［ボタン］コントロール："Customer" と "Suplier"。どちらも［ルート］イベントを割り当てます。イベントに対するパラメータとして「ルート名」に［コール］処理コマンドの［ルート名］特性に指定されている名前を渡すように指定します。
- ［サブフォーム］コントロール

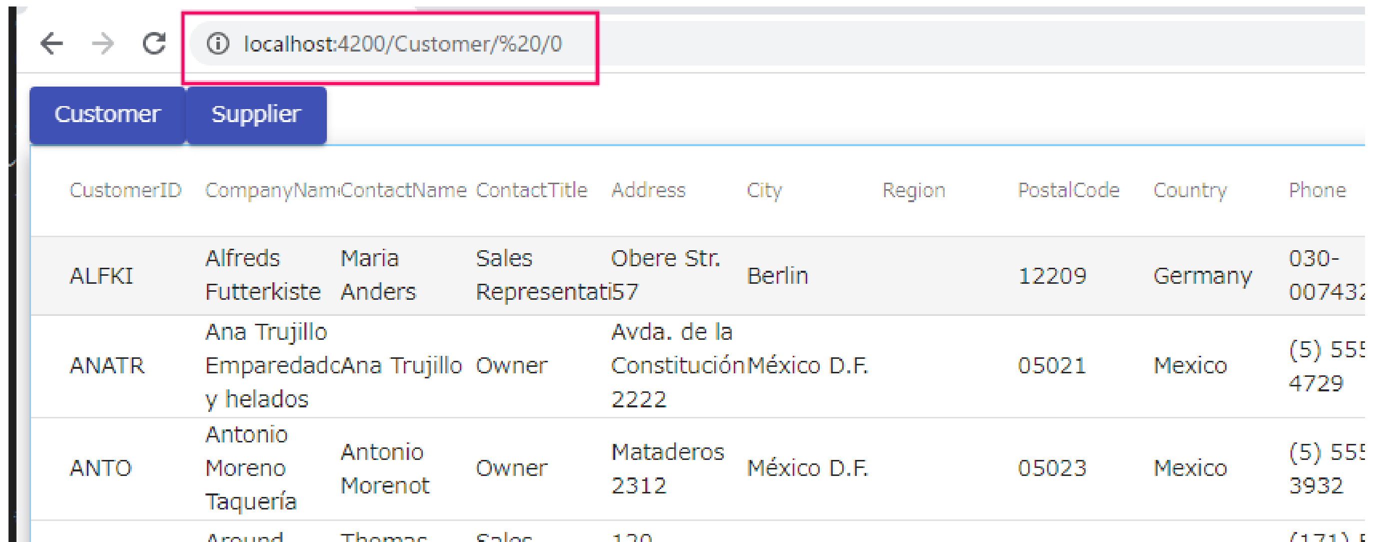
また、WebClientCustomers と WebClientSuppliers の 2 つのルーティングされるプログラム（またはタスク）もあります。どちらのプログラムも同じアウトレット（サブフォーム）で呼び出されます。

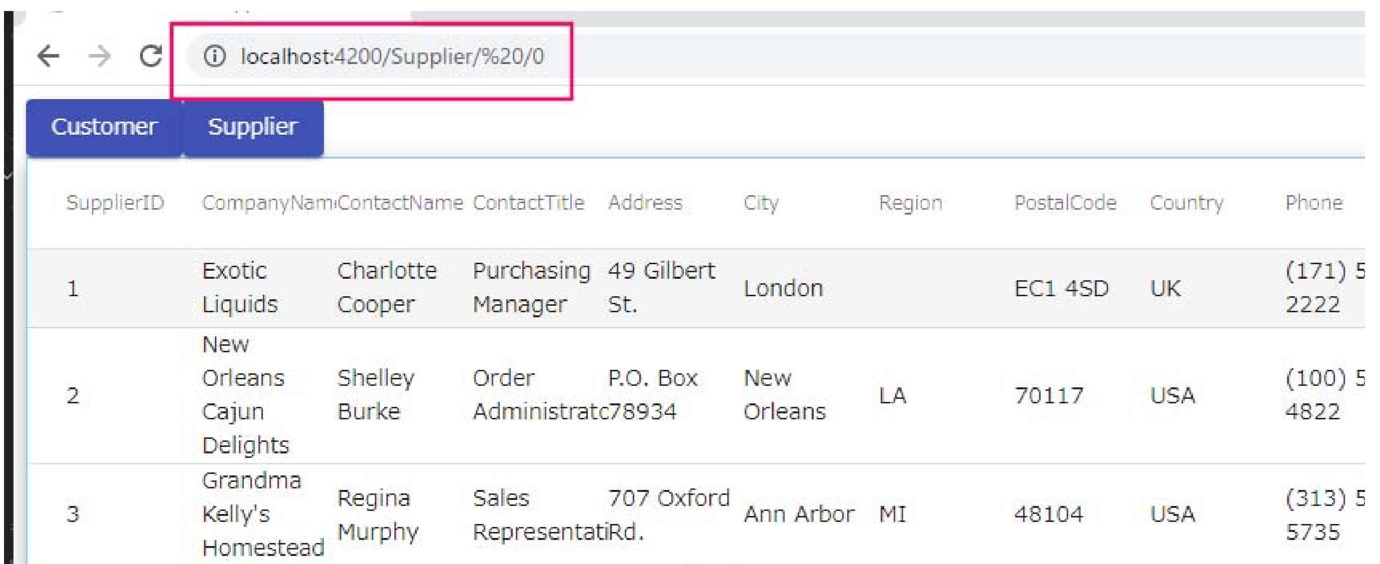


［ルート］イベントに対する［イベント］ロジックユニットを作成すると、2 つの読み取り専用のパラメータ項目が追加されます。

Customers ボタンをクリックして［ルート］イベントが発行されると、WebClientCustomer プログラムが呼び出され、顧客の詳細がルートタスクのアウトレットに表示されます。下の図のように、アドレスバーに「/Customer」と表示されていることに注意してください。これは、［コール］処理コマンドの［ルート名］特性で説明されているとおりです。Magic xpa エンジンは顧客の詳細のパスを覚えています。

同様に、Supplier ボタンをクリックすると WebClientSuppliers プログラムが呼び出され、ルートタスクのアウトレットにサプライヤの詳細が表示されます。アドレスバーの「/Supplier」に注目してください。





顧客の詳細をもう一度表示するには、［Customer］ボタンをクリックします。Magic xpa エンジンは、［Customer］ボタン用に記憶されていたパスに移動します。

ルーティングを使用すると、以前にアクセスしたすべてのパスに移動することができます。

注意：

- 複数のサブフォームがある場合は、ルータアウトレットにしたいコントロールの［デフォルトアウトレット］プロパティを「True」に設定します。
- 1 レベル下に移動することができます。この場合、ページ A からページ C へのルートを作成する場合は、ページ B を経由する必要があります。
- コピー、切取り、上書き、ショートカットキーの Ctrl + Shift キーによる移動などの操作は、非ルートのロジックユニットからの［ルート］イベントのロジックユニットには対して実行することはできません。

magic
OUTPERFORM THE FUTURE™

- ［ルート］イベントのロジックユニット内に入ったら、［コールプログラム／タスク］と［コメント］のみが利用できます。何らかのイベントを非ルートから［ルート］イベントに変更しようとすると、その［イベント］ロジックユニット内の処理コマンドが全て削除されます。
- ［イベント］ロジックユニット内のイベントは、すべて［ルート］イベントに変更できますが、その逆はできません。

オーバーレイウィンドウのカスタマイズ

ヒント :このトピックにはコードスニペットが含まれています。これはそのまま利用できます。

Magic プログラムの UI はカスタマイズ可能です。しかし、今までは、Magic はデフォルトのコンテナを提供していました。これで、オーバーレイコンテナの UI を完全に制御できるようになりました。

オーバーレイウィンドウは次の 2 つの部分から構成されています。

- コンテナ（タイトルバー、背景、境界線などを含む）
- クライアントエリア（Magic プログラムのフォームが表示されます）

オーバーレイウィンドウをカスタマイズするには、次の手順を実行する必要があります。

1. オーバーレイコンテナを用意する

MyOverlayContainer という Component を作成します。これは、Magic プログラムのオーバーレイウィンドウのコンテナとして使用されます。この MyOverlayContainer は、@magic-xpa/angular モジュールの BaseMagicOverlayContainer のサブクラスになる必要があります。

カレントを src¥app¥magic フォルダに移動して次のコマンドを実行し Angular コンポーネントを作成します。

```
ng g c MyOverlayContainer
```

my-overlay-container フォルダが作成されます。

入力パラメータ

MyOverlayContainer には、Magic プログラムのオーバーレイコンポーネントとその他のいくつかのパラメータが入力として提供されます。MyOverlayContainer の役割は、コンテナ内のコンポーネントを表示し、それによって受信されたパラメータを内部コンポーネントに渡すことです。

これを行うには以下のように修正します。

1. Magic プログラムのプレイスホルダとして HTML の <div> タグを定義します。

```
<div>
   <div #overlaybody>
   </div>
</div>
```

2. TS ファイル内で @ViewChild を使用して参照します。

```
@ViewChild('overlaybody',{read:ViewContainerRef})overlaybodyViewContainerRef;
```

3. ComponentFactoryResolver を使用して、このプレースホルダ内に Magic プログラムのコンポーネントを作成します。

```
const factory =
this.componentFactoryResolver.resolveComponentFactory(this.ModalComp);
this.componentRef = this.overlaybodyViewContainerRef.createComponent(factory);
...
Object.assign(this.componentRef.instance, this.ModalCompParameters);
```

このアプローチは、コンテナが（MagicModalInterface を使用して）Magic コンポーネントからいくつかのプロパティ（ShowTitleBar、ShouldCloseOnBackgroundClick など）を使用する必要がある場合に便利です。

例えば、

```
let magicModalInterface: MagicModalInterface = this.componentRef.instance as
MagicModalInterface;
```

これは、コード内で使用できるコンポーネント参照を取得するためです。

```
import {
   Component,
```


magic
OUTPERFORM THE FUTURE™

```
   OnInit,
   ViewChild,
   ViewContainerRef
   Input,
   Output,
   EventEmitter
} from "@angular/core";

import{MagicModalInterface} from '@magic-xpa/angular';
import{BaseMagicOverlayContainer} from '@angular/core';

@Component({
  selector: "app-my-overlay-container",
  template: `
  <div class="fullbg-overlay"></div>
  <div class="overlay-box1"[ngStyle]="getStyle()">
    <div *ngIf="showtitle">
      <h3
        align="center"
        style="display:flex-direction:row; background-color: #5C6276; color: white;"
      >
        {{Formname }}
      </h3>
    </div>
    <div>
      <div#overlaybody></div>
    </div>
  </div>
  `,
  styleUrls: ["./my-overlay-container.component.css"]
})
export class MyOverlayContainer extendsBaseMagicOverlayContainer implements OnInit {
  Formname: string;
  showtitle: boolean;
  formwidth: string;
  /*** content will be displayedin this placeholder */
  @ViewChild("overlaybody",{ read: ViewContainerRef, static: true })
  overlaybodyViewContainerRef;
  /****/
  @Input() ModalComp = null;
  /****/
  @Input() ModalCompParameters:any = {};
  /****/
  @Output() onClose = newEventEmitter();
  /****/
  private componentRef = null;
  /**** @param componentFactoryResolver*/
  constructor(private componentFactoryResolver:ComponentFactoryResolver) {
    super();
  }
  /****/
  ngOnInit() {
    const factory= this.componentFactoryResolver.resolveComponentFactory(
      this.ModalComp
    );
    this.componentRef= this.overlaybodyViewContainerRef.createComponent(
      factory
    );
    let magicModalInterface:MagicModalInterface = this.componentRef
      .instanceas MagicModalInterface;
    this.Formname= magicModalInterface.FormName();
    this.showtitle= magicModalInterface.ShowTitleBar();
```

```
      this.formwidth= magicModalInterface.Width();
      Object.assign(this.componentRef.instance,this.ModalCompParameters);
      //  modal.Width
    }
    getStyle() {
      let styles ={};
      let modal: MagicModalInterface= (
        this.componentRef.instance
      );
      styles["width"]= modal.Width();
      styles["height"]= modal.Height();
      styles["top"]= modal.X() + "vh";
      styles["left"]= modal.Y() + "vw";
      // if(modal.Width()!='100%'){
      //   styles['border']= '1px solid grey';
      // }
      return styles;
    }
    /**
     *
     */
    OnClose() {
      this.onClose.emit();
    }
  }
```

出力パラメータ

MyOverlayContainer は、ウィンドウを閉じる必要があるときに onClose イベントを発行する必要があります。ウィンドウがすでに閉じられているときはそうではありません（例：ngOnDestory）。その理由は、Magic は実際にウィンドウを閉じる前に、[レコード後]、[タスク後] などのタスクを実行する必要があるからです。いくつかのエラーが原因で終了が許可されないこともあります。

2. デフォルトの OverlayContainerMagicProvider の代わりに使用するプロバイダを指定します。

Magic は、OverlayContainerMagicProvider.getComponent() を使用してオーバーレイコンテナを照会しています。

このため、MyOverlayContainer を返す getComponent() を使用して、MyOverlayContainerProvider と言う新しいプロバイダを作成します（# 1 を参照)。

```
import { Component, Injectable} from'@angular/core';

import { TaskBaseMagicComponent, magicProviders}from "@magic-xpa/angular";
import { MyOverlayContainer } from './my-overlay-container.component';


// change this to correct path dependingon where you place MyOverlayContainer.ts
@Injectable()
  export class MyOverlayContainerProvider {
  getComponent() {
    // return MyOverlayContainer
    return MyOverlayContainer;
  }
}
```

3. app.module.ts に次の変更を加えます。

1. declarations セクションの Overlay コンテナ

magic
OUTPERFORM THE FUTURE™

```
declarations: [
   AppComponent,
   MyOverlayContainer
],
```

2.mports セクションの Overlay コンテナ

```
//For ComponentFactoryResolver add the following:
// import { MyOverlayContainer }

   entryComponents: [
   MyOverlayContainer,
],
entryComponents:[
   MyOverlayContainer
],
```

3.以下の providers セクションで、OverlayContainerMagicProvider の代わりに MyOverlayContainerProvider を使用するように変更します。

```
providers: [
  {
  provide: OverlayContainerMagicProvider,
  useClass: MyOverlayContainerProvider
  }
 ]
```

デフォルトのオーバーレイウィンドウとカスタマイズされたオーバーレイウィンドウ

これは、カスタマイズされたオーバーレイウィンドウがデフォルトのオーバーレイウィンドウと比べてどのように見えるかを示しています。

図 3-1 デフォルトのオーバーレイウィンドウ：

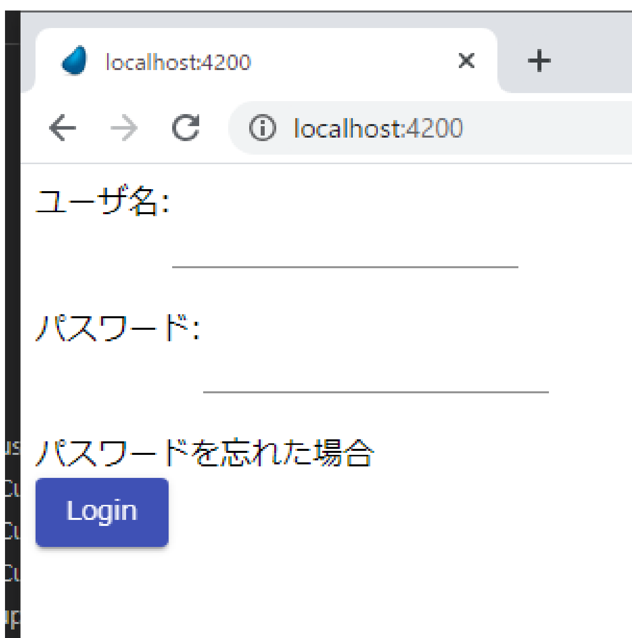


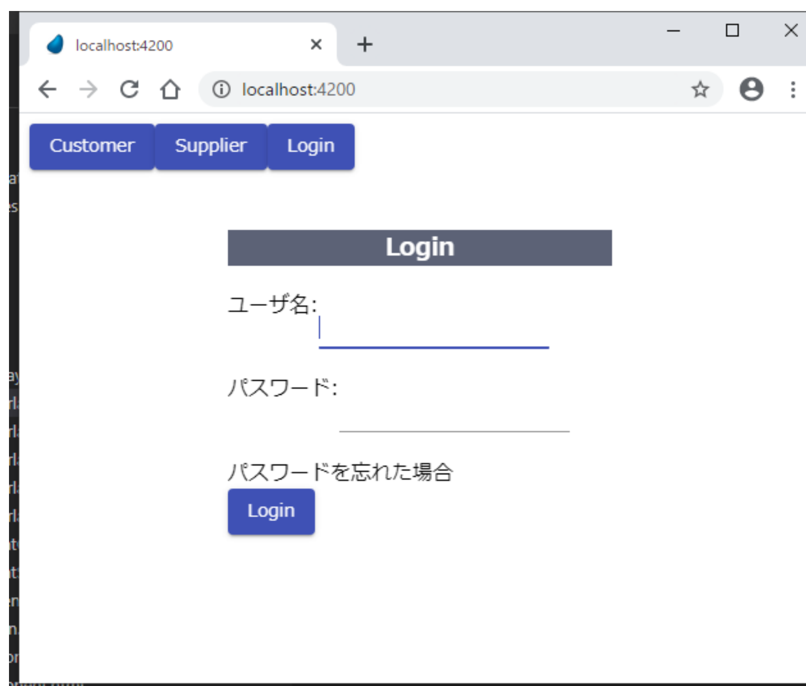
図 3-2 カスタマイズされたオーバーレイウィンドウ

4. スタイルシート（my-overlay-container.component.css）を定義します。

```
.fullbg-overlay {
  opacity: 0;
  background-color: aqua;
  width: 100%;
```

```
    height: 100%;
    position: fixed;
    top: 0;
    right: 0;
    bottom: 0;
    left: 0;
  }


  .overlay-box {
    position: absolute;
    top: 50%;
    left: 50%;
    width: 50%;
    height: 50%;
    transform: translate(-50%, -50%);
    color: #ffff00;
    background: #b22222;
    opacity: 0.8;
    z-index: 1000;
    border-radius: 20px;
  }
```

図 3-3 スタイルシートを反映させた場合



OUTPERFORM THE FUTURE™

ブラウザの履歴の操作

Web ブラウザは、あるページ（Web サイト）から別のページ（Web サイト）に移動している間に、アクセスしたすべてのページの記録を保持することができます。Angular と Java を使用することで、Web ブラウザで以前にアクセスしたページにアクセスすることができます。

ブラウザの履歴を操作するための Angular のメソッド

Angular は、保存された履歴ページを操作できるように Location サービス（$location）を提供します。

forward()

ブラウザの履歴を先に進めます。。

構文	forward(): void
パラメータ	null
戻り値：	void

back()

ブラウザの履歴を前に戻します。。

構文	back(): void
パラメータ	null
戻り値：	void

Web ページを更新するための Java メソッド

Angularal の $location サービスは URL だけを変更することができます。Web ページを再読み込みすることはできません。Java は、完全な Web ページをリロードまたは更新できるように Location オブジェクトを提供しています。

reload()

現在の Web ページを再読み込みします。

構文	location.reload(forceGet)
パラメータ	forceGet（Bool） これは、再読込の方法を指定するためのオプションのパラメータです。forceGet のデフォルト値は False です。この場合、Web ブラウザは現在のページをキャッシュから再読込します。forceGet が True に設定されている場合、Web ブラウザは現在のページをサーバから再読込します。 location.reload(true) 1 つのコンポーネントだけを再読み込みする必要がある場合は、Magic の内部イベント［サブフォームリフレッシュ］を使用します。
戻り値：	void

例

ブラウザの履歴を通じて操作を実装するコードスニペットは以下のようになります。

```
import {Component, ChangeDetectorRef} from '@angular/core';


import { FormGroup } from "@angular/forms";
import { MgFormControlsAccessor, MgControlName, MgCustomProperties } from "./
form1.mg.controls.g";


import {Location} from '@angular/common';

import {TaskBaseMagicComponent,magicProviders,MagicServices} from "@magic-xpa/
angular";
import { Router } from '@angular/router';


@Component({selector: 'mga-form1',
      providers: [...magicProviders, Location],
      templateUrl: './form1.component.html'

})

export class form1 extends TaskBaseMagicComponent {
    mgc = MgControlName;
    mgcp = MgCustomProperties;
    mgfc: MgFormControlsAccessor;


    createFormControlsAccessor(formGroup: FormGroup) {
        this.mgfc = new MgFormControlsAccessor(formGroup, this.magicServices);
    }


   constructor ( ref: ChangeDetectorRef,magicServices: MagicServices,
       private locationService: Location, private router: Router) {
       super(ref, magicServices);


    forward() {
      this.locationService.forward();
    }

    back() {
      this.locationService.back();
    }

    reload() {
      location.reload();
    }
}
```

Component.html

```
<div
    novalidate
    [formGroup]="screenFormGroup"
>

    <div
        style="display: flex; flex-direction: column"
        [magic]="mgc.form1"
    >

        <div style="display: flex; flex-direction: row">
           <button
```


magic
OUTPERFORM THE FUTURE™

```
            mat-raised-button
            color="primary"
            (click)="back()"> Back
        </button>
        <button
           mat-raised-button
           color="primary"
           (click)="forward()"> Forward
        </button>
        <button
            mat-raised-button
            color="primary"
            (click)="reload()"> Reload
       </button>
```

......

警告と確認メッセージのカスタマイズ

Magic xpa には、Web Client アプリケーション用に警告と確認メッセージをカスタマイズできる機能があります。このトピックは、この機能が重要である理由、および警告と確認メッセージをカスタマイズする方法を理解するためのものです。

この機能の目的

この機能の目的は、Magic xpa が提供する Angular 用コンポーネントを使用して、デフォルトの JavaScript および確認メッセージダイアログをカスタム実装で置き換えることができるようにすることです。これによってシステム上にカスタムテーマやザインを実装することが可能になります。

Web Client アプリケーションの開発中に、アプリケーション実行の流れを予測することができます。開発者は、関連メッセージでユーザにメッセージを表示する必要があるさまざまな状況を想定することができます。そのような要件が生じたときに、表示するメッセージを変更することによって、ユーザが次に何をすべきかを知ることができます。

Magic xpa は独自のコンポーネントを開発する方法を提供し、見栄えの良い警告や確認メッセージを設計することができるようになります。

ConfirmationComponentMagicProvider

置き換えを可能にするメインのサービスは、ConfirmationComponentsMagicProvider です。このサービスはデフォルトで @magic/angular で提供されており、JavaScript メッセージボックスを提供するようにデフォルトで設定されています。そのため、メッセージをカスタマイズしたい場合は、次のように独自の ConfirmationComponentsMagicProvider を提供する必要があります。

```
import { Injectable } from "@angular/core";
import { MagicAlertComponent } from '../ui/components/magic-alert.component';
import { MagicConfirmationBoxComponent } from "../ui/components/magic-confirmation-
box.component";


export class ConfirmationComponentsMagicProvider {
// Returns true when use default javascript alert and confirmation or return false to
provide custom components
showDefaultConfirmations() {
return true;
}


// Returns component that will replace javascript alert. The component will be used
only if showDefaultConfirmations = false
getAlertComponent() {
return MagicAlertComponent;
}



// Returns component that will replace javascript confirmation box. The component
will be used only if showDefaultConfirmations = false


getConfirmtionComponent() {
return MagicConfirmationBoxComponent;
}
}


ConfirmationComponentsMagicProvider.decorators = [
{ type: Injectable },
];
```

ここでは、以下の 2 つのコンポーネントを使用しています。

- BaseMagicAlertComponent

magic
OUTPERFORM THE FUTURE™

警告と確認メッセージをカスタマイズする手順

警告と確認メッセージをカスタマイズする手順は次のとおりです。

1. Angukar コンポーネントの作成

次のコマンドを使用して Angular コンポーネントを作成します。

```
ng g c <コンポーネント名>
```

例：カレントを src\app\magic フォルダに移動して以下を実行します。

```
ng g c NewAlert
```

これにより、"new-alert" というフォルダが作成されます。

2. 基本クラスの拡張

コンポーネント用に作成された .ts ファイルを開き、以下に示すように更新して、BaseMagicAlertComponent または BaseMagicConfirmComponent を拡張します。

src¥app¥magic¥new-alert¥new-alert.component.ts の場合：

```
import{ Component} from '@angular/core';
import { BaseMagicAlertComponent} from "@magic-xpa/angular";
@Component({
selector:'app-new-alert',
templateUrl:'./new-alert.component.html',
styleUrls:['./new-alert.component.css']
})
export class NewAlertComponent ?extends BaseMagicAlertComponent {}
```

3. HTML ファイルの編集

タイトルを表示するには {title} を、メッセージを表示するには {{message}} を使用します。デフォルトでフォーカスするボタンを定義するには、magicFocus ディレクティブを使用します。

次の .html テンプレートを使用してください。

```
<div class="mg-message-background">
   <h2>{{title}}</h2>
   <p>{{message}}</p>
   <button ?magicFocus(click)="OnClose()">OK</button>
</div>
</div>`,
```

src¥app¥magic¥custome-alert¥custome-alert.component.html の場合は以下のようになります。

```
<div class="mg-background">
   Customer Alert Component
   <h2 mat-dialog-title>{{title}}</h2>
   <mat-dialog-content>{{message}}</mat-dialog-content>
   <br>
   <br>
   <mat-dialog-actions>
      <button magicFocus ?mat-button(click)="OnClose()">OK</button>
   </mat-dialog-actions>
</div>
```

4. CSS ファイルの編集

このファイルでメッセージの外観をカスタマイズすることができます。たとえば、コンポーネント用に作成された .css ファイルを開き、Look & Feel を次のように更新します。

src¥app¥magic¥custome-alert¥custome-alert.component.css の場合：

```
.mg-background{
/*semi-transparent black */
background-color:lightblue;
/*//background-color: white;*/
text-align:center;
width:40%;
font-family:'Open Sans', 'Helvetica Neue', Helvetica, Arial, sans-serif;
padding:17px;
border-radius:5px;
text-align:center;margin-top: 30% ;
margin-left:auto;
margin-right:auto;
}
```

5. magic.gen.lib.module.ts ファイルの編集

このファイルに以下の記述があるかどうかを確認してください。

この機能の目的は、Magic xpa が提供する Angular 用コンポーネントを使用して、デフォルトの JavaScript および確認メッセージダイアログをカスタム実装で置き換えることができるようにすることです。これによってシステム上にカスタムテーマやザインを実装することが可能になります。

1. ng g... コマンドによって作成された新しいコンポーネントへの参照。これらの行が magic.gen.lib.module.ts にあることを確認し、@magic-xpa/angular の import ステートメントに ConfirmationComponentMagicProvider を追加してください。

```
import { ComponentListMagicService,MagicModule, ExitMagicService,
ConfirmationComponentsMagicProvider} from "@magic-xpa/angular";
import {CustomeAlertComponent}from "./custome-alert/custome-alert.component"
import {MyConfirmationMagicProviderService} from './
myCconfirmationMagicProviderService';
```

※ MyConfirmationMagicProviderService は、後で作成します。

2. 以下に示すように、NewAlertcomponent と NewCnfcomponent の参照が @NgModule に追加されています。

```
@NgModule({
  declarations:[
    ...magicGenComponents,
              CustomeeAlertComponent
               ],
```

3. DynamicModule.withComponents にコンポーネント参照を追加します。

```
MagicModule,
DynamicModule.withComponents([...magicGenComponents,CustomeAlertComponent]),
```

4. 次のようにプロバイダを追加します。

```
providers:[ExitMagicService, {
provide: ConfirmationComponentsMagicProvider,
useClass: MyConfirmationMagicProviderService}
],
```

5. src¥app¥magic¥magic.gen.lib.module.ts ファイルは以下のようになります。

```
import {NgModule, NgModuleRef}  from '@angular/core';
import { RouterModule }          from '@angular/router';
import { ReactiveFormsModule }  from '@angular/forms';
```


magic
OUTPERFORM THE FUTURE™

```
import { CommonModule }          from "@angular/common";
...


import {magicGenComponents, magicGenCmpsHash, title} from './component-list.g';
import { MagicAngularMaterialModule } from "@magic-xpa/angular-material-core";
import { CustomAlertComponent } from './custom-alert/custom-alert.component';
import { MyConfirmationMagicProviderService } from './
myConfirmationMagicProviderService';


@NgModule({
   declarations:[
   ...magicGenComponents,
      CustomAlertComponent
      ],
   exports: [
   ...magicGenComponents,
   MagicModule
   ],
   imports:[
   ....
   RouterModule,
   //Magic Modules
   MagicModule,
   DynamicModule.withComponents([...magicGenComponents, CustomAlertComponent]),
   InfiniteScrollModule,
   //Material Modules

   ....
   MagicAngularMaterialModule
],
providers: [ExitMagicService,{provide: ConfirmationComponentsMagicProvider, use-
Class: MyConfirmationMagicProviderService} ],


})
export class MagicGenLibModule{
   .....
}
```

6. src¥app¥magic フォルダに myConfirmationMagicProviderService.ts というファイルを作成します。内容は以下の通りです。

```
import{ Injectable } from '@angular/core';
import { ConfirmationComponentsMagicProvider} from "@magic-xpa/angular";
import { CustomeAlertComponent } from "./custome-alert/custome-alert.component";

@Injectable()
export class MyConfirmationMagicProviderService  extends ConfirmationComponentsMag-
icProvider{
  getAlertComponent()  {
      return  CustomeAlertComponent;
  }
  showDefaultConfirmations(): boolean {
    return  false;
  }
}
```

7. カスタマイズの代わりにサンプル画面を使用するだけの場合は、getAlertComponent() をコメントアウトし、showDefaultConfirmations() のみを False として使用します。

magic
OUTPERFORM THE FUTURE™

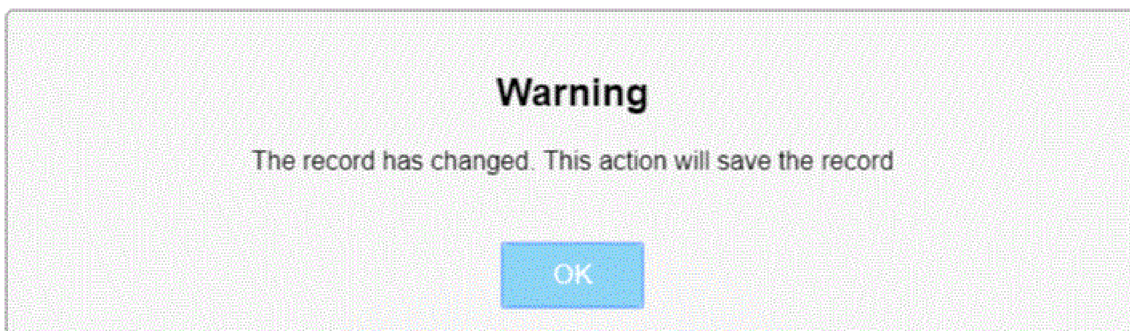
```
import{ Injectable } from '@angular/core';
import { ConfirmationComponentsMagicProvider} from "@magic-xpa/angular";
import { CustomeAlertComponent } from "./custome-alert/custome-alert.component";

@Injectable()
export class MyConfirmationMagicProviderService  extends
ConfirmationComponentsMagicProvider{

//getAlertComponent() {
//  return NewAlertComponent;
// }
  showDefaultConfirmations() : boolean  {
    return false;
  }
//getConfirmtionComponent()  {
//  return NewcnfComponent;
//  }
}
```

注意 :確認と警告メッセージをカスタマイズするには、showDefaultConfirmations() を False のままにする必要があります。それが True であれば、そこから古い警告と確認メッセージボックスが表示されます。

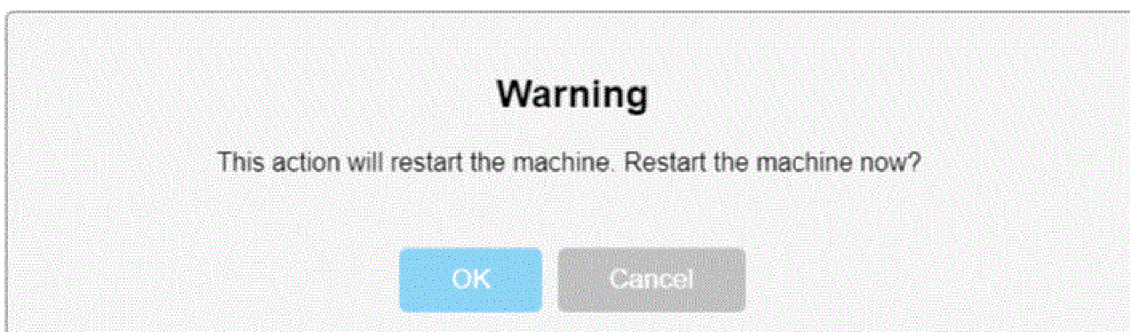
カスタマイズされた警告および確認メッセージの結果

これはカスタマイズによって警告と確認メッセージがどのように表示されるかを示したものです。

警告メッセージ



確認メッセージ



magic
OUTPERFORM THE FUTURE™

ログインとログアウトの概念

どの Web アプリケーションでも、ユーザ認証とアクセス制御を処理する必要があります。エンタープライズ Web アプリケーションには、ログインやログアウトなどの基本認証機能が必要です。認証とともに、ユーザ権限を管理する機能を提供する必要があります。

ユーザ認証は Magic xpa の Logon（）関数を使用して行われます。また、Magic には、ユーザ権限に基づいてビルドされた組み込みのセキュリティシステムもあります。特定の権限でログインしている場合は、任意の式で Rights（）関数を使用して、必要なロジックを実行することができます。権利は、あらゆる Web Client プログラムの実行権として使用することができます。

Magic xpa は、現在以下の機能をサポートしています。

- Web Client に機能を実装することができます。
- ログインステータスの維持 - ユーザがログインしているかどうかにかかわらず、機能を使用して HTML に出力することができます（Magic と Angular）。
- ユーザがログインしているときにルーティングするために Angular の CanActivate ガードを使用できるようにします。

上記の機能をサポートするために、Magic xpa は以下の関数を追加しました。

Magic の関数

IsLoggedIn()

この関数は、ユーザがログインしているかどうかを確認することができます。この関数は現在のセッションを確認し、ユーザがログインしている場合は True を返します。そうでなければ Falser を返します。

Logout()

この関数は、現在のセッションからユーザをログアウトさせます。この関数は、現在のセッション内のユーザをチェックし、そのユーザをログアウトさせます。ユーザが正常にログアウトした場合、ログアウト関数は True を返し、そうでない場合は False を返します。Logout 関数はセッションを閉じませんが、ユーザのすべての権限をクリアし、成功したログアウトに対して True を返します。

Angular 関数

mg.isLoggedIn()

このクライアント側の Angular 関数は、ユーザがログインしているかどうかを確認するために追加されています。この関数は現在のセッションをチェックし、ユーザがログインしていれば True を返します。そうでなければ False を返します。

getIsLoggedIn()

サービス EngineMagicService 上でのこの関数は、すべての TypeScript（TS）ファイルから使用されます。使用する TS に EngineMagicService を Inject する必要があります。その後、この関数をすべてのコードで使用できます。

注意：
mg.IsLoggedIn 関数は、コンポーネントに属する HTML および Typescript から使用することができますが、コンポーネントに属していない TypeScript から mg.IsLoggedIn を使用することはできません。したがって、必要に応じてすべての TS で使用できるようにするために、新しい関数 getIsLoggedIn（）を追加しました。

結果として、ログインについての情報を取得するために 2 つの方法を持つことになります。

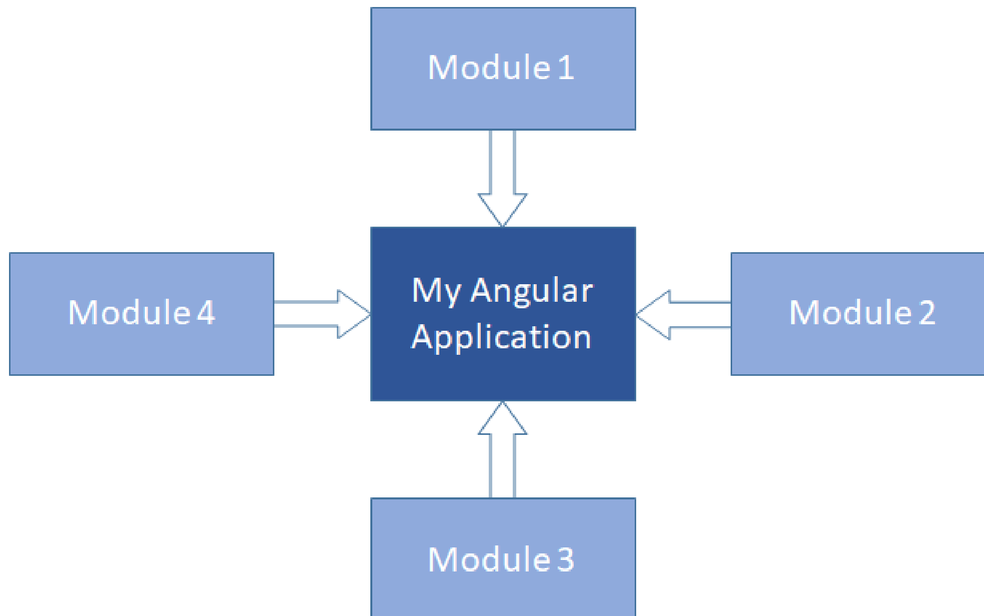
mg.IsLoggedIn（）関数の使用とサービス EngineMagicService.getIsLoggedIn（）の使用です。

Web モジュールについて

このトピックでは、Magic xpa による Web モジュールを紹介し、それらの使用方法について説明します。

ngModule の導入における考え方

アプリケーションがますます複雑になるにつれて、すべてのアプリケーションをモジュールに分割する必要があることが明白なりました。各モジュールはそれ自体が小さなミニアプリケーションです。そして、大きなアプリケーションを作成するために、これらのミニアプリケーションをバンドルすることができます。



Angular におけるモジュール作成は、@NgModule です。これは、モジュールを作成するためのタグです。Angular のモジュールは、コンポーネントまたは他のモジュールのコンポーネントで構成されています。

NgModule はインジェクターとコンパイラを設定し関連するものをまとめます。

NgModule は、@NgModule デコレーターが付与されたクラスのことです。@NgModule は、コンポーネントのテンプレートをコンパイルする方法と実行時にインジェクタを作成する方法が記述されたメタデータオブジェクトを引数として受け取ります。モジュール自身のコンポーネント、ディレクティブやパイプを識別し、

この中のいくつかを exports プロパティを通して公開し、外部コンポーネントから使用できるようにすることができます。@NgModule はアプリケーションの依存性インジェクタにサービスプロバイダーを追加することもできます。

NgModule は、コンポーネント、ディレクティブ、およびパイプをまとまった機能ブロックに統合します。それぞれの機能ブロックは、アプリケーションのビジネスドメイン、ワークフロー、または共通のユーティリティのコレクションなどの機能の領域に焦点を当てています。

- モジュールはアプリケーションを整理し、外部ライブラリの機能を使用して拡張するための素晴らしい方法です。
- モジュールはアプリケーションにサービスを追加することもできます。 そのようなサービスは内部的に使用されることになるかもしれません。例えば、自身で開発したものや Angular router や HTTP クライアントのような外部ソースなどです。
- モジュールはルータによって非同期に遅延ロードすることができます。

Magic xpa の Web モジュール

Magic xpa の Web モジュールは、Angular の NgModule のアイデアから派生しています。

Web モジュールは、モジュールとも呼ばれる特定のフォルダのすべての Magic コンポーネントを含む論理ユニットです。アプリを拡張している間は、特定の機能に関連するコードをフォルダにまとめることができます。これは機能に対して明確な境界を適用します。Web モジュールを使用すると、特定の機能または機能に関連するコードを他のコードとは別にすることができます。アプリの領域を明確にすることは、開発者とチーム間のコラボレーション、ディレクティブの分離、およびルートモジュールのサイズ管理に役立ちます。

magic
OUTPERFORM THE FUTURE™

Magic xpa は、Web クライアントアプリケーションの Magic フォルダごとに独立した Web モジュールを作成する機能を提供します。これにより、フォルダが別の論理ユニットとして識別されるようにフォルダを扱うことができます。この論理ユニットは必要に応じてロードすることができます。コンピューティングリソースを節約して目的のフォルダのみを読み込むことができるため、この機能強化により Web アプリケーションのパフォーマンスが向上します。

ルートモジュールと Web モジュール

ルートモジュールとフィーチャーモジュール の概念を継承することで、Magic xpa では 2 種類のモジュールがあります。

- ルートモジュール …… アプリの入り口です。技術的には、アプリ内のすべてのものを 1 つのモジュールにまとめることができます。
- フィーチャーモジュール…… 開発中に懸案事項を分離離すためだけでなく、アプリの一部の遅延ロードなどのためにも使用されます。これは大規模なアプリケーションにとってはより大きな問題ですが、早めに行うことで確実に正しく設定できるようになります。

Web モジュールは、ユーザのワークフロー、ルーティング、フォームなど、特定のアプリケーションのニーズに焦点を合わせてまとまった機能を提供します。ルートモジュール内ですべてを実行できますが、Web モジュールを使用すると、アプリを焦点を絞った領域に分割することができます。Web モジュールは、提供するサービス、および共有するすべてのコンポーネント、ディレクティブ、およびパイプを介して、ルートモジュールおよび他のモジュールと連携します。

Angular と Magic のフォルダ構造と相関

AngularCLI によって生成されたファイルは、デフォルトですべて src\app ディレクトリに配置されます。Angular プロジェクトは、スタンドアロンアプリケーション、ライブラリ、または一連のエンドツーエンド（e2e）テストを構成する一連のファイルです。Angular ワークスペースには、1 つ以上のプロジェクトのファイルが含まれています。同様に、Magic の Web Client プロジェクトが生成されると、すべてが Projects\<project_name> フォルダに配置されます。Magic の　Web Client プロジェクトのフォルダの構造は、Angular プロジェクトのフォルダと同じです。

Is-Web-Module プロパティを使用する場合と使用しない場合の Magic フォルダ

現在、Magic xpa のルートプログラムにフォルダが含まれている場合、作成された \magic フォルダの component-list.g.ts ファイルには、そのフォルダの外部に存在するすべてのプログラムのエントリが含まれています。

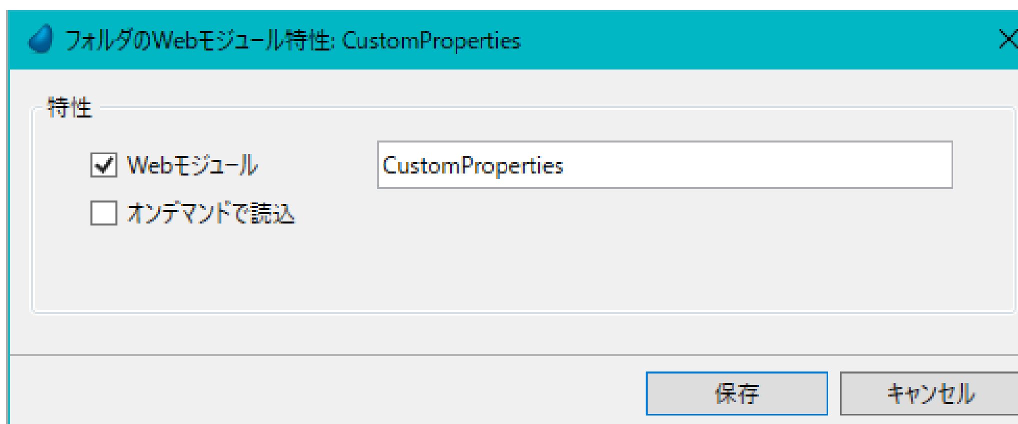
ルートプログラムに Web モジュール対応のフォルダが含まれている場合は、フォルダ内に作成された 2 つの ts ファイルが作成されます。

- component-list.g.ts …… 特定の Web モジュール対応フォルダに存在するプログラムのリストが含まれています。
- Magic.gen.lib.module.ts

Web モジュール特性

Web モジュール特性を使用すると、Magic フォルダ用の独立した Web モジュールを作成することができます。

たとえば、[プログラム] リポジトリ上の特定のフォルダの Web モジュールを生成する必要があるとします。目的のフォルダ上で右クリックして、[特性] オプション（Alt+Enter）を選択すると、次のようなダイアログボックスが開きます。



Web モジュール特性に表示される特性は次のとおりです。

特性	説明
Web モジュール	このフィールドをチェックすると、そのフォルダに対して独立した Web モジュールが作成されます。
名前	フォルダ名が入力されています。フォルダ名は 30 文字です。Web クライアントのプロジェクト内でユニークです。
オンデマンドで読込	このフィールドをチェックすると、Web モジュールフォルダがオンデマンドで作成されます。このモジュール内のプログラムはルーティングを通して呼び出す必要があります。 モジュール内で個々のプログラムを実行することはできません。

Web モジュール生成プロセス

［プログラム］リポジトリ内のフォルダに対して Web モジュールを生成するには、Web モジュールのプロパティを設定する必要があります。

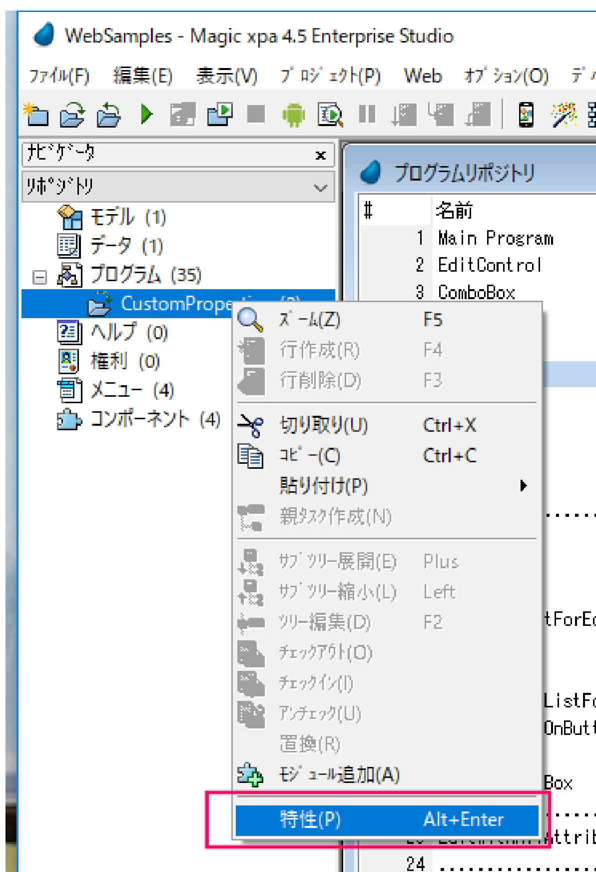
Web モジュールの生成手順

1.［プログラム］リポジトリに移動します。

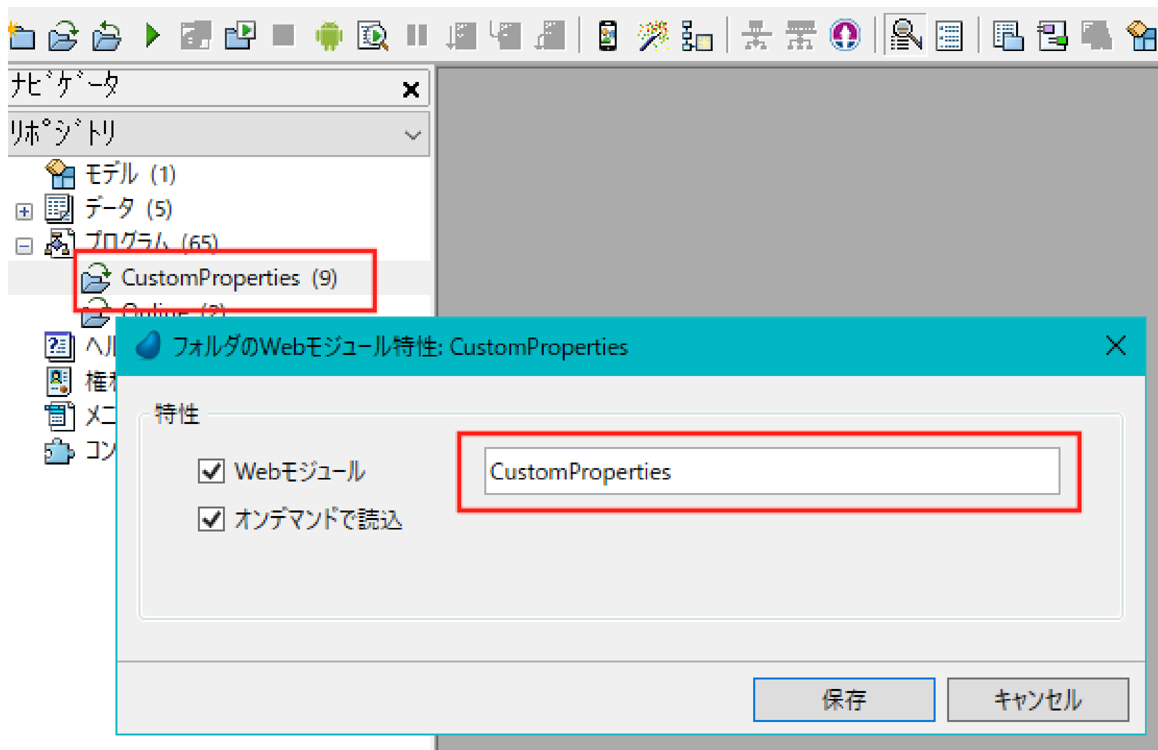
2. Web モジュールを作成するフォルダを決定します。

3. 目的のフォルダ上で右クリックします。

4.［特性］オプションを選択します（Alt + Enter）。



次のように［Web モジュール特性］ダイアログボックスが開きます。



デフォルトでは、タイトルバーと最初のチェックボックスの隣にフォルダ名が表示されます。

5.［Web モジュール］のチェックボックスをチェックします。Magic は、フォルダ名から継承するデフォルト名を付加します。

6. Web モジュールの新しい名前を入力するか、デフォルトの名前のままにします。

注意 : フォルダ名と Web モジュール名は同じでも異なっていてもかまいません。名前に特殊文字を入力したり、以前に使用した名前と同じフォルダ名を使用しないでください。

7. Web クライアントプロジェクトを生成すると、［Web モジュール］チェックボックスをオンにしたフォルダに以下の 2 つのファイルが作成されます。

- component-list.g.ts
- magic.gen.lib.module.ts

8.［オンデマンドで読込］のチェックボックスをチェックします。この場合、以下の 3 つのファイルが作成されます。

- component-list.g.ts
- magic.gen.lib.module.ts
- app.routes.ts

注意 : ロードオンデマンドフォルダーにあるプログラム :

- ルーティングを通じてのみアクセスできます。
- フォルダ外から呼び出されているサブフォームやオーバーレイウィンドウとして使用することはできません。

Web モジュールの生成シナリオ

たとえば、次のような状況を考えてみましょう。

［プログラム］リポジトリには、HR、Admin、および RandD という 3 つのフォルダがあり、それぞれにいくつかの Web Client タスクがあります。

以下のようにフォルダ毎に異なるプロパティで設定されているとします。

フォルダ名	Web モジュール	オンデマンドで読込
Admin	アンチェック	アンチェック
RandD	チェック	チェック
HR	チェック	チェック

アプリケーションを生成すると、与えられた 3 つのフォルダに対して次のファイルが作成されます。

Admin フォルダ

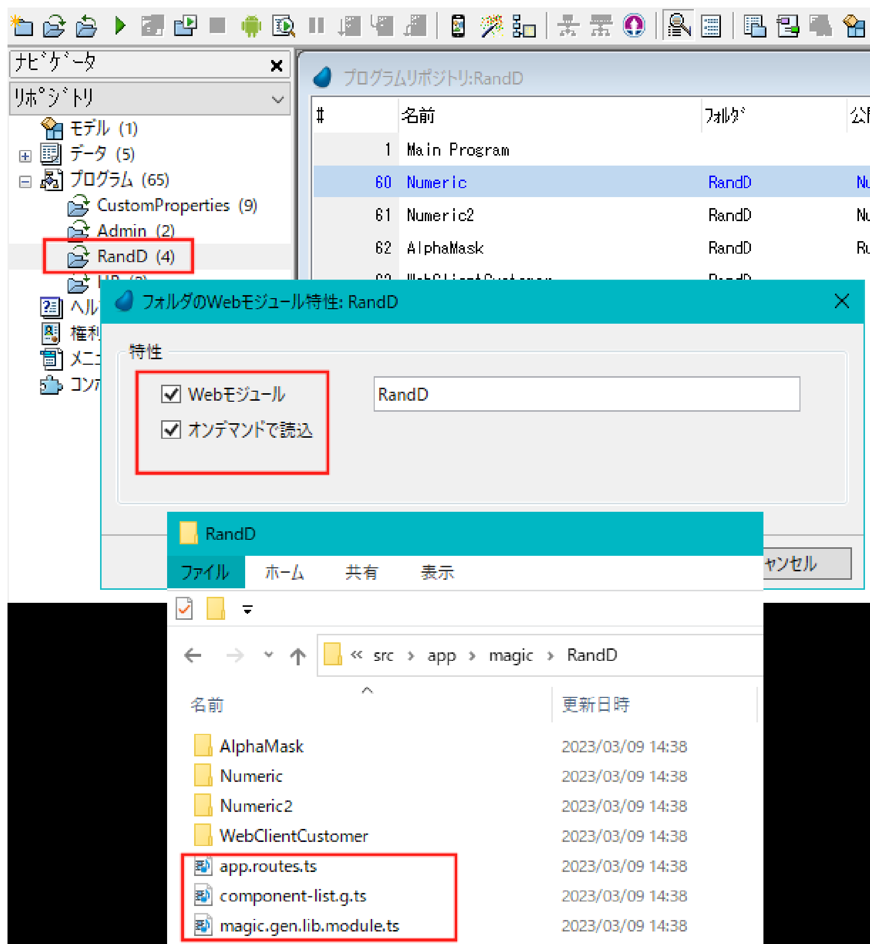
ここでは、Web モジュールを作成することも、オンデマンドで読み込むことも選択されていません。動作に変更はありません。

¥magic フォルダ内に以前に作成された TS ファイルが使用されることを意味します。

RandD フォルダ

両方が選択されました。Web モジュールを作成し、オンデマンドで読み込む。この場合、¥RandD フォルダに次の 3 つのファイルが作成されます。

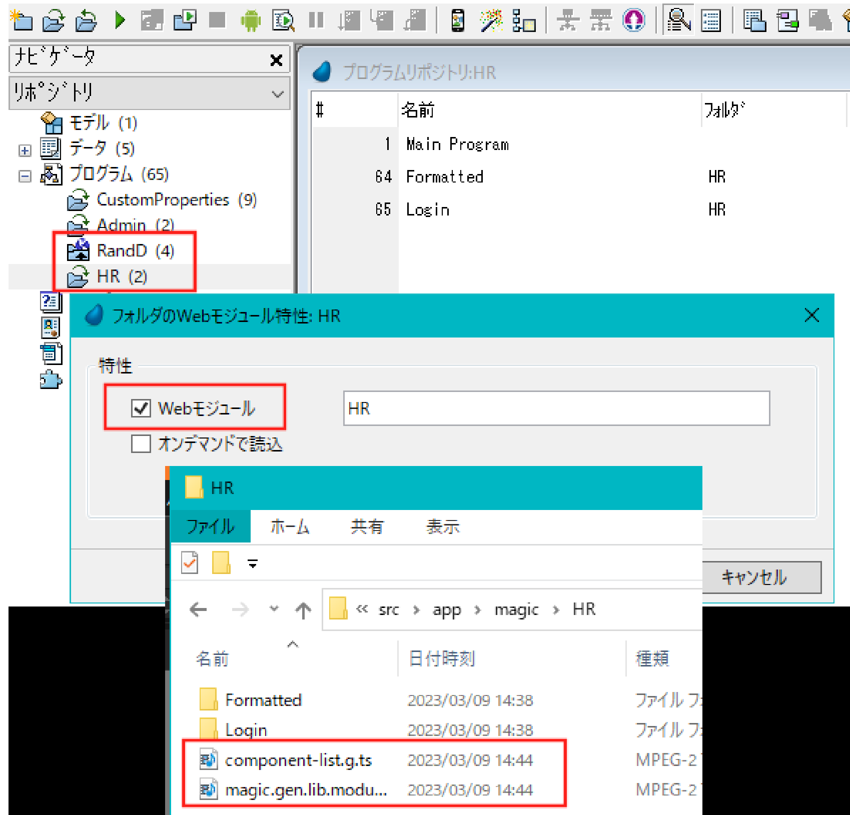
- component-list.g.ts
- magic.gen.lib.module.ts
- app.routes



HR フォルダ

［Web モジュール］がチェックされただけです。デフォルトでは、\magic フォルダに作成されたファイルに加えて、2 つのファイルが生成されます。

- component-list.g.ts
- magic.gen.lib.module.ts



Load on Demand フォルダからオーバレイウィンドウにアクセスする

Load on demand フォルダ内にあるオーバレイウィンドウ内のプログラムにルーティングなしでアクセスできるようになりました。（サポートバージョン：4.7）

Load on Demand フォルダ内にある オーバレイ ウィンドウ内の番組にアクセスするには、以下の手順で行います。

1. Magic xpa は component-list.g.ts に LazyLoadModulesMap を生成し、サービスに設定します。

   例えば、RandD モジュールのオーバーレイプログラムがオンデマンドでロードされるように設定されている場合、マップは以下のようになります。

   ```
   export const LazyLoadModulesMap = {
   Prg1_Prg1 : { moduleName : 'MagicRandDModule',
                 modulePath : 'src/app/magic/RandD/magic.gen.lib.module'}
   }
   ```

   これは、コンポーネント Prg1_Prg1 の moduleName が MagicRandDModule で、modulePath が 'src¥app¥magic¥RandD¥magic.gen.lib.module' です。

   これらのマップには、Load on Demand として設定されているすべてのオーバレイプログラムのマップが含まれています。

   生成された LazyModulesMap は、Magic.gen.lib.module の ComponentListMagicService に以下のように設定する必要があります。

   ```
   componentList.lazyLoadModulesMap = LazyLoadModulesMap
   ```

2. バージョン 8 以降の Angular では、ルーティングを使わずにオンデマンドでモジュールをロードする機能が提供されています。この機能を利用するには、angular.json の lazyModules[] でモジュールを指定します。

   Magic xpa では、オーバーレイが Load on Demand の場合、生成処理で angular.json の lazyModules[] に追加されます。

   同じ例では、angular.json は以下のように変更されます。

```
"lazyModules": [
   "src/app/magic/RandD/magic.gen.lib.module"

  ]
```

angular.json の lazyModules のパスは LazyLoadModulesMap で指定されたパスと同じでなければなりません。

magic
OUTPERFORM THE FUTURE™

Web Client アプリケーションでの Magic コンポーネントの使用

このトピックでは、Magic コンポーネントを使用して Web Client アプリケーションを作成および生成する方法について説明します。

はじめに

Magic xpa は、Magic コンポーネントが非 Web Client の Magic アプリケーションの場合と同じ方法で、Web Client アプリケーションで Magic コンポーネントを使用することができます。

コンポーネントの作成方法や、ecf の生成方法、またはビルダの呼び出し方法に違いはありません。ただし、Angular アプリケーションは、Magic コンポーネントとして Angular コンポーネントを使用できるようにする必要があります。

このトピックでは、Angular コンポーネントを Magic コンポーネントとして使用する方法について説明します。

用語

コンポーネントという用語は Magic と Angular の両方で使用されます。

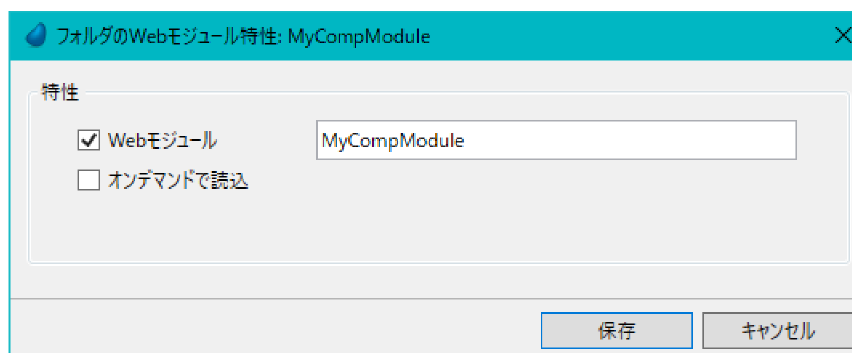
- Magic コンポーネント …… Magic アプリケーションで作成されたコンポーネント。
- Angular コンポーネント …… Angular アプリケーションで作成されたコンポーネント。

Angular モジュールでコンポーネントの使用

［ルート］イベントのロジックユニット以外で起動させる場合に利用します。

以下の手順に従って、Angular モジュールでコンポーネントを使用します。

1. Magic でコンポーネントアプリケーション（mycomp.xml）とホストアプリケーション（usecompProject.xml）を作成します。通常の方法で、Magic コンポーネントのホストプログラムからプログラムを呼び出すことができます。
2. Magic コンポーネントで次の操作を行います。
   a. Magic フォルダを作成します（フォルダがなくても可能ですが、このトピックでは、フォルダを考慮することで簡潔に説明しています）。
   b. Angular として使用するプログラムをこのフォルダに移動します。
   c. フォルダ上で右クリックして、［フォルダの Web モジュール特性］ダイアログを開きます。
   d. ［Web モジュール］チェックボックスをチェックします。
   e. Web モジュールに名前を付けます。例：MyCompModule。



   f. このプロジェクトで Angular アプリケーションを作成します。これにより、Angular アプリケーション内にフォルダ MyCompModule が作成されます。
3. ホストコンポーネントで以下を実行します。
   a. Magic コンポーネントを含めます。つまり、ホストアプリケーションの［コンポーネント］リポジトリにコンポーネントの .ecf パスを設定し、通常どおり Magic コンポーネントの Web Client プログラムを呼び出します。

b. 通常どおり、ホストアプリケーション側で Angular アプリケーションを作成します。

c. Angular アプリケーションで以下を実行します。

i. 作成されたコンポーネント アプリケーションの Angular モジュールフォルダ（この例では mycomp¥src¥app¥magic¥MyCompModule）をコピーします。

ii. コピーしたフォルダを作成されたアプリケーションの ¥magic フォルダ（usecompProject\src\app\magic）に貼り付けます。

iii. モジュールを参照するには、ホストモジュールファイル（usecompProject¥src¥app¥app.module.ts）にモジュールを追加します

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppComponent } from './app.component';
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
import { ReactiveFormsModule } from '@angular/forms';
import { MagicModule } from '@magic-xpa/angular';
import { MagicAngularMaterialModule } from '@magic-xpa/angular-material-core';
import { MagicGenLibModule } from './magic/magic.gen.lib.module';
import { MagicRoutingModule } from './app.routes';
import { MagicMySimpleModuleModule } from "./MySimpleModule/magic.gen.lib.module";
 declarations: [
  AppComponent
 ],

 imports: [
  BrowserModule,
  BrowserAnimationsModule,
  ReactiveFormsModule,
  MagicModule.forRoot(),
  MagicAngularMaterialModule,
  MagicGenLibModule,
  MagicRoutingModule,
  MagicMySimpleModuleModule
 ],

 providers: [],
 bootstrap: [AppComponent]
})
export class AppModule { }
```

ルーティングを使用したロードオンデマンドによるコンポーネントの使用

［ルート］イベントのロジックユニットで起動させる場合に利用します。

以下の手順に従って、ルーティングを使用したロードオンデマンドでコンポーネントを使用します。

1. Magic xpa でコンポーネントアプリケーション（mycomp.xml）とホストアプリケーション（usecompProject.xml）を作成します。ルーティングを使用した通常の方法で ホストプログラムから Magic コンポーネントのプログラムを呼び出すことができます。
2. Magic コンポーネントで次の操作を行います。

i. Magic フォルダを作成します（フォルダがなくても可能ですが、このドキュメントでは、フォルダを考慮することで簡潔に説明しています)。

ii. Angular で使用するプログラムをこのフォルダに移動します。

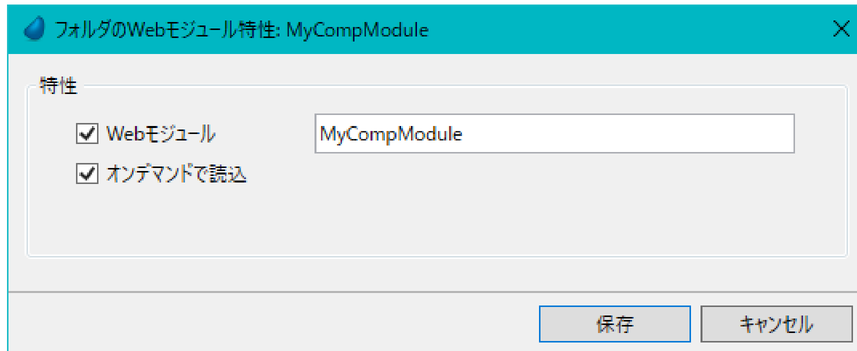
magic
OUTPERFORM THE FUTURE™

iii. フォルダ上で右クリックして、［フォルダの Web モジュール特性］ダイアログを開きます。

iv. ［Web モジュール］チェックボックスをチェックします。

v. Web モジュールに名前を付けます。たとえば、MyCompModule。

vi. ［オンデマンドで読込］チェックボックスをチェックします。



i. このプロジェクトで Angular アプリケーションをを作成します。これにより、Angular アプリケーションに新しいフォルダ MyCompModule が作成されます。

3. ホストコンポーネントで以下を実行します。
   a. Magic コンポーネントを含めます。つまり、ホストアプリケーションの［コンポーネント］リポジトリでコンポーネント .ecf パスを設定し、Magic コンポーネントの Web Client プログラムを通常の方法で呼び出します。

   b. 通常どおり、ホストアプリケーションで Angular アプリケーションを作成します。

   c. Angular アプリケーションで以下を実行します。

      i. 作成されたコンポーネント アプリケーションからモジュールフォルダ（この例では、mycomp¥src¥app¥magic¥MyCompModule）をコピーします。

      ii. 生成された Angular アプリケーションの \magic フォルダ（usecompProject¥src¥app¥magic）に貼り付けます。

      iii. モジュールを参照するには、ホストモジュールファイル（Project¥src¥app¥magic¥app.routes.ts）にルーティングパスを追加します。

```
import {Routes,RouterModule} from '@angular/router';
import {RouterContainerMagicComponent} from "@magic-xpa/angular";
import {CommonModule} from ;
import {NgModule} '@angular/core';

export const routes: Routes = [
  {

    path: 'callOnroute',

    component: RouterContainerMagicComponent,
    loadChildren: `./magic/MyCompModule/magic.gen.lib.module#MagicMyCompModuleModule`

@NgModule({
    imports: [CommonModule,
    RouterModule.forRoot(routes)
    ],
    exports: [RouterModule]
)
export class MagicRoutingModule {}
```

OUTPERFORM THE FUTURE™

カスタムプロパティの値が変更されると JavaScript コードをトリガする

Magic xpa では［カスタムプロパティ］プロパティの実効値が変更された場合、JS コードをクライアント側でトリガすることができます。

以下のメソッドを使用して実行する JS コードを実行するスクリプトを定義する必要があります。

PropertyChanged

このメソッドの目的は、(与えられたフォーム内の) カスタムプロパティの値が変更されると、クライアント側で JS コードを実装できるようにすることです。

このメソッドは、変更され Web Client アプリケーションでの Magic コンポーネントの使用た各カスタムプロパティのためにトリガされ、メソッドに送信される関連パラメータと一緒に変更されます。

このメソッドは、コンポーネントコントローラファイル（.ts ファイル）に追加する必要があります。

構文：　PropertyChanged(プロパティ名 : string, rowId: number, 値 : any): void {switchLogic}

パラメータ：　プロパティ名：変更されたカスタムプロパティの名前

書式は次の通りです。

< コントロール名 >~< プロパティ名 > または

this.mgcp.< コントロール名 >_< プロパティ名 >

rowId：［テーブル］コントロール内のコントロールの更新されたプロパティの行番号（テーブル外のコントロールの場合は 0)。

値：カスタム プロパティの新しい値

戻り値：　void

SwithLogic：　ここでは、プロパティ名のパラメータに基づいて実行条件を追加する必要があります。

例：

```
PropertyChanged(propertyName: string, rowID: number, propertyValue: any): void{
    if (propertyName == '<controlname>~<propertyname>')
        < 実行する JS code>;
    else if (propertyName == this.mgcp.<controlname>_<propertyname>')
        < 実行する JS code>;
}
```