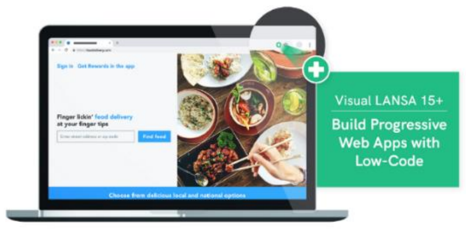
March 2021 | Issue 1 www.lansa.com

LANSA V15+

Progressive Web Apps Go Low-Code

INSIDE THIS ISSUE

The latest edition of Visual LANSA, V15+ (this is LANSA V15 plus EPC150010, EPC150020 and EPC150030), introduces the ability to easily create progressive web apps (PWAs). Now, developing PWAs is as easy as developing any app in Visual LANSA. There is no need to worry about creating a service worker or an app manifest, both of which are vital to PWAs. Visual LANSA V15+ does all that through the power of low-code. As amazing as launching just a PWA edition would have been, LANSA went above and beyond and packed even more features into the latest edition.



Progressive Web Apps Go Low-Code

Effortlessly Create Progressive Web Apps (PWAs) with Visual LANSA 15+

The highlight of V15+ is its ability to create PWAs effortlessly—meaning that Visual LANSA will now automatically generate a PWA when you build your new web app, giving users a seamless experience across all devices and platforms. With this one PWA, you will reach iOS, Android, Windows, Chromebook, and MacBook users without having to worry about multiple app stores and OS upgrades.

Visual LANSA v15+ Makes Your Web App Shareable

Progressive web apps take advantage of both web and browser technologies, making them shareable through a link. You can send the link through text messages or emails or share it directly on a webpage. It becomes simpler than ever to share your app and reach a new audience or enable easier access for your current audience. Because PWAs put the user and the user experience first, studies have shown that user engagement is much higher with PWAs than native apps.

It's All About the Cache

Cache plays a vital role in enabling PWAs to be so versatile. The ability to cache resources allows for greater app performance as well as reduce network load in areas of low connectivity or slow service. Visual LANSA V15+ makes it easy for developers to set their caching strategies and select their resources to be cached.

Developers simply select the cache strategy and then drag and drop assets they would like to have cached when end users visit the site or use the application. Any assets defined by the developer to be cached will be automatically handled by the service worker, making V15+ the easiest way to create PWAs.

When creating PWAs, developers will have the option to select from two popular caching strategies known as Cache First or Network First, with additional options being available in later releases. Additionally, developers will be able to have their current application cache resources seamlessly transitioned to utilizing the service workers and caching strategies.

The Magic of the Service Worker

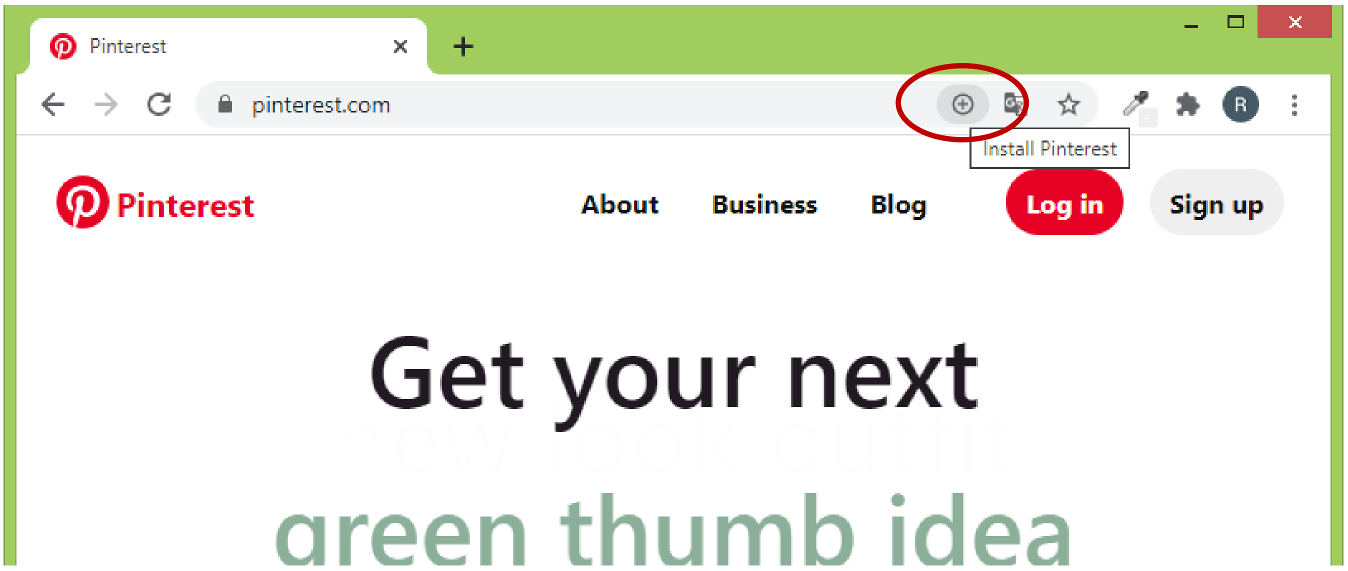
The service worker is another critical component in the creation of a PWA. And much like the app manifest, V15+ creates the service worker for you, so it is ready to be tweaked to your application. Service workers have been customized to ensure that requested resources are routed to the right cache storage. When possible, the service worker will serve content from the cache instead of allowing the network call to go through. If the requested resource is not in the cache, the request is sent to the server. When it comes back, the service worker goes back to work and clones the resource to cache for later use. The service worker is key to opening your app to the offline world.

Start Creating PWAs Now with Visual LANSA V15+

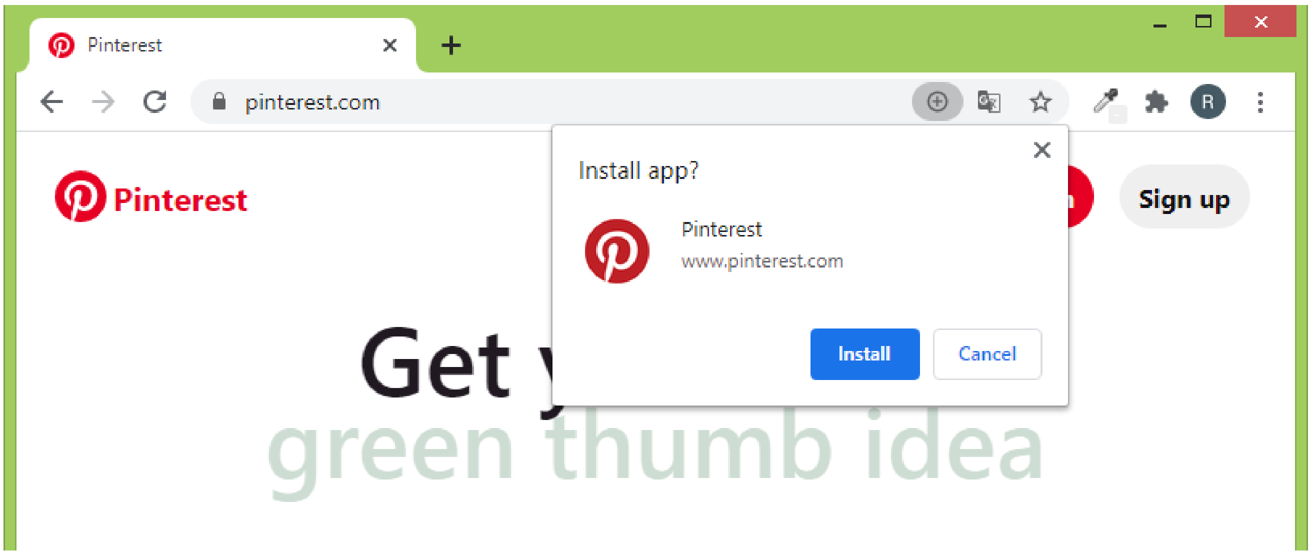
LANSA's hybrid low-code approach combines the efficiency of low-code with the versatility of traditional coding. With the ability to easily create PWAs, Visual LANSA strengthens its position as the most powerful enterprise low-code app builder available. Check out V15+ and find out for yourself how quickly you can have your first PWA ready for the world to consume.

Want to See a Progressive Web App in Action?

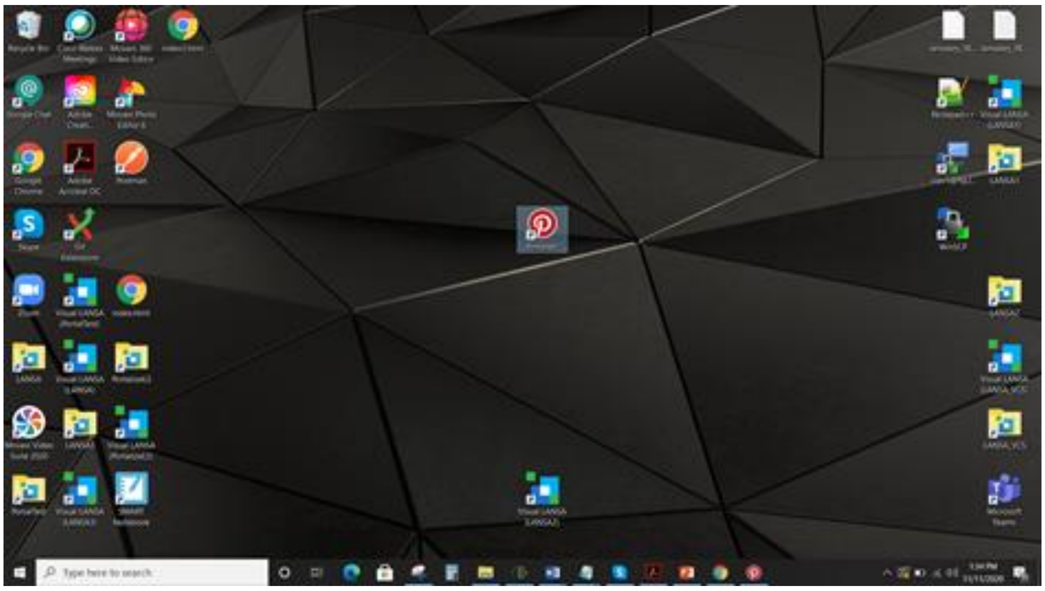
- Go to https://www.pinterest.com/
- Click the "plus" icon on the right of the address bar.



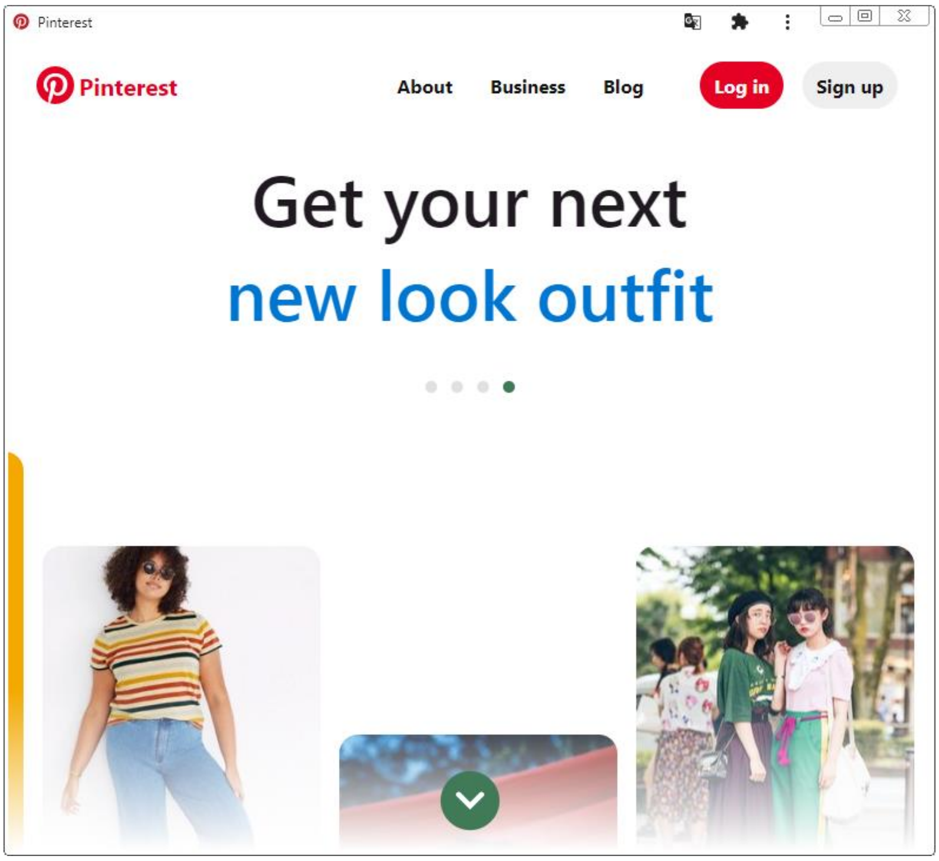
- Click the "install" button.



- The app will install on your desktop, mobile phone, or tablet.



- When you launch the app, notice the address bar is gone, and the app looks and feels like a native app.



Progressive Web Apps

Create a Universal Experience Across All Devices

Introduction

The web has changed tremendously since its static beginnings. As the face of an online business, a website is built to invite new and internal customers into the world of an organization. However, as time went on, the web lacked the functionality, design, and native features that organizations desired the most, phones became ubiquitous, and mobile applications became a game-changer. Imagine downloading an application to your mobile device and being able to access local device storage, cameras, and Near Field Communications (NFC). So, we moved to creating mobile apps, housed in app stores and sometimes limited to the mobile and desktop operating environments that we use on a daily basis. Fast forward to 2020, and a new game-changer has emerged, one whose technology has been around for quite some time, but the guidelines and realizations of possibilities were seldom fused together.

Welcome the progressive web application—a way to create a universal experience across mobile and desktop alike and to view and utilize the web in a completely new way while increasing your marketability.

What's the Goal of a Progressive Web Application?

The goals of progressive web applications are best described by the following three fundamentals:

Capable
Progressive web applications have a vast array of web APIs at their disposal to accomplish much of what a native application can provide. With technologies such as WebRTC, a progressive web application can be a chat client that connects via a peer-to-peer interface, stream video games down to the application, and enable live video chatting.

Reliable
Progressive web applications are fast, and the goal is to reach the next billion users who have limited network connectivity. End users need their applications to be a fast experience, have the most recent content, and be able to interact with their other applications with as little friction as possible.

Installable
A progressive web application can be installed to the desktop and mobile environment with its own unique icon. When you launch a progressive web Application, you are not viewing a web browser; you are viewing an application window that looks and acts like any other application you may have on your phone, tablet, or computer. Additionally, the user experience and interaction is seamless across environments—that is, end users will have the same experience across mobile and desktop environments.

Why Progressive Web Apps?

The progressive web app centres itself around core principles that enable the web to be more user friendly, performant, and interactive. The web has become a set of powerful standards that provide almost endless possibilities for your web applications. From interacting with native device features such as the camera all the way to being installable on desktop and mobile devices, progressive web apps provide a whole host of possibilities. However, what you should know is that many of the standards that make progressive web applications possible are not new, but with the emergence of key technologies such as the service worker, PWAs are more poised than ever to create scalable, reliable, and flexible experiences within the browse.

Twitter

- 65% increase in pages per session

- 75% increase in tweets sent

- 20% decrease in bounce rate

- Requires less than 3% of device storage compared to the native twitter for android.

Nikkei--publishing for more than 140 years, one of the most influential media businesses in Japan:

- 2x better speed index

- 14 seconds faster time-to-interactive

- 75% faster loading with prefetch

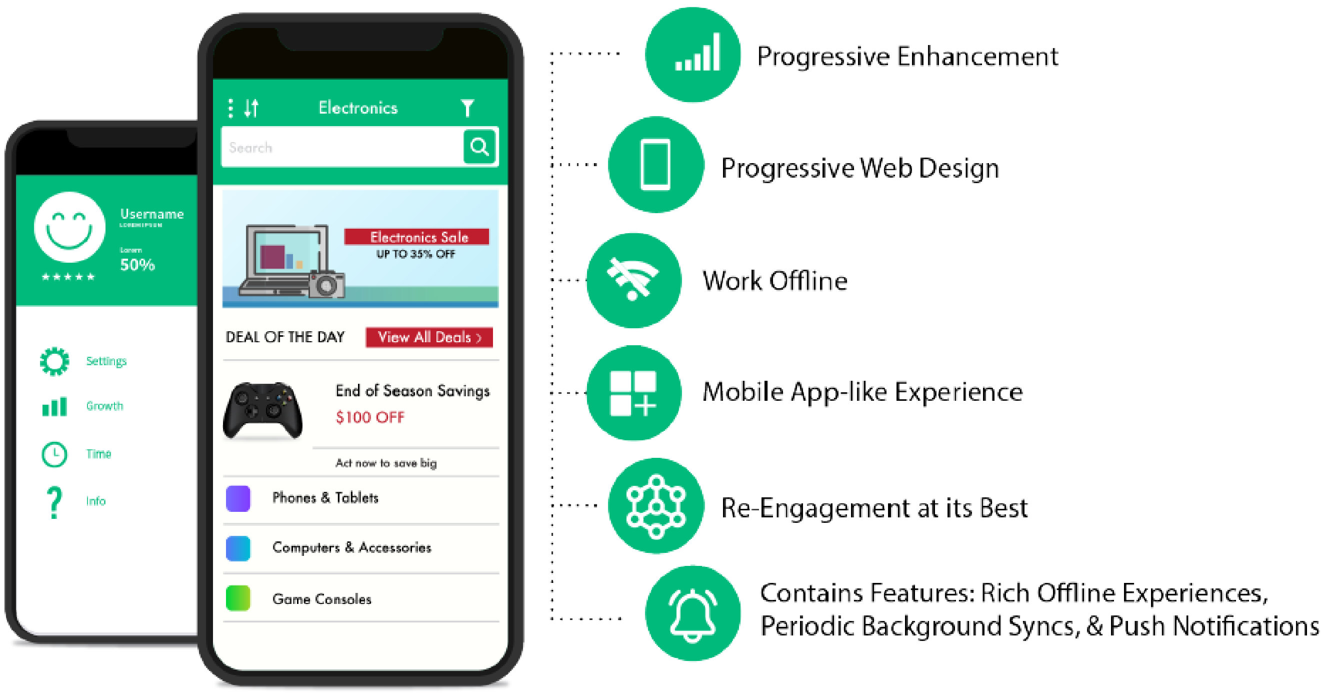
- 49% more daily active users

- 58% more conversions (subscriptions)

All of this sounds great, but what are the benefits of PWAs for the enterprise environment? That question can be answered readily by Pinterest, Twitter, Nikkei, and a host of other businesses with varying models that have found that providing a progressive web app increases retention and acquisition of new users, while decreasing the amount of data usage. These are only a few use cases for the utilization of progressive web apps.

Let's take a look at how PWAs can be used for your business practice:

- Crafting new user experiences for employees to access and query the information they need on any platform.
- Creating a web-based ticketing system for support agents, enabling work to be completed on a desktop or mobile environment.
- Utilizing Web NFC to communicate with IoT devices and retrieve information about manufacturing equipment performance.
- Creating a Content Management System (CMS) to deliver student content that can be available offline, enabling students with limited internet connectivity to view, edit, and complete assignments.
- E-commerce with the Web Payment API and Paypal integrations
- Mobile accident assessment applications with the ability to take photos and videos of an accident, use audio recording to record testimonies, and more.



How Do They Work?

Service Workers
At the core of a progressive web application is a wonderful piece of technology called the service worker. The service worker acts as a network proxy that greatly speeds up the performance of a PWA by interrupting a request for resources to the network to determine if the requested items have been cached first. If they have been cached, then the service worker will in most cases use the cached response first, decreasing the need for the overabundance of network requests we typically see. Remember, you have the flexibility to customize these actions of the service worker, enabling your application to mold to the best use case of the business.

The use of caching by the service worker allows your PWA to work offline and load reliably regardless of the end user’s network connection. For example, the service worker can be programmed to always serve content from the cache first and fall to the network. The added benefit of this strategy is that, if the content is not available in the cache, then what is served from the network can be cloned by the service worker and placed into the cache for future use.

In a way, service workers are akin to dark magic for the web. One of the magical parts of the service worker is that it creates a semblance of multithreading in JavaScript, meaning that while it fetches network requests, caches resources that you nominate to be available, and helps route requests to the right places, it is not blocking the UI of the application, allowing the end user to continue to interact with the application rather than seeing a page spinner and waiting for more content.

Web Storage
For the business environment, storage is an important part of the PWA experience. Previously, web applications have been limited to utilizing a small subset of storage within the browser. Frustratingly, the subset of storage had difficulty in persisting, required multiple page refreshes when updates were available, and sacrificed in the area of performance. With PWAs, we have been provided guidelines and a new means of interacting with storage that allows for dynamic persistence of data. Rather than use the application cache, we can use cache storage and create multiple stores for the necessary resources to load our applications and file-based content. In conjunction with cache storage, we are able to utilize IndexedDB to store other data, query the stored data, and much more. As you read this, you are probably asking, ”How much data can I store?”
The answer is:

LOTS!

For example, within the Chrome browser an application can use up to 60% of total disk space, Firefox allows up to 2 GB of storage, and Safari allows for up to 1 GB and will increase the limit in 200MB increments with permission from the end user.

Web App Manifests
The web app manifest is another part of the PWA that enables it to be installable on desktop and mobile devices. Basically, the manifest wraps relevant information about the application in JSON. Within the web app manifest, you can provide the resources for icons, versions, descriptions, related applications, splash screens, and much more. At the end of the day, having the web app manifest enables your application to install an icon on the device and launch as if it is a native application.

Building a Strategy around PWAs

Think offline first

When developing a PWA it is best to think in terms of the application being developed offline first. What this means is that the application should be developed with remote and limited network connectivity in mind. In considering how the application will work offline as you build out technical requirements, your team will find solutions that impact the vast majority of end users.

Examine your user base

Examine the data available about your user base. Question the types of actions that are necessary to them. What are the jobs that need to be completed? What are the pain points in your current application? How do your end users use the application today? Asking these questions will help formulate a foundation to determine how a PWA fits into the overall strategy for your organization.

Create the ideal workflow

When determining your organization PWA strategy, start from the beginning of the user experience. Ask questions such as, "What are the goals and jobs that need to be accomplished?" From there sketch out what the workflow would look like. How does an end user get from point A to point B? What type of data should be accessible offline for the end user? How do they retrieve that data? Should the end user be able to save information for use later? Thinking along these lines aids in considering both a design- and a development-driven approach to your organization's PWA strategy and helps to clarify how and why the organization should utilize a PWA.

Whiteboard technical requirements

Following the design of the ideal user workflow, it is necessary to consider the technical requirements necessary for developing a PWA. Engaging in a whiteboard session with the necessary stakeholders such as business analysts and IT departments will provide the necessary conversations, feedback, and requirements for laying out the flow of information, architecture requirements, and user requirements for the PWA.

Equally important, you must acknowledge that while the web is becoming more powerful every day, some areas are still in experimental stages, such as interacting with a device native file system through the web. Therefore when writing the technical requirements you may find that some functionality is only available through a native application. Even in those scenarios it may be beneficial to still create a PWA that links to the native application in the app store to download if users need to access more advanced functionality.

Understand the business impact

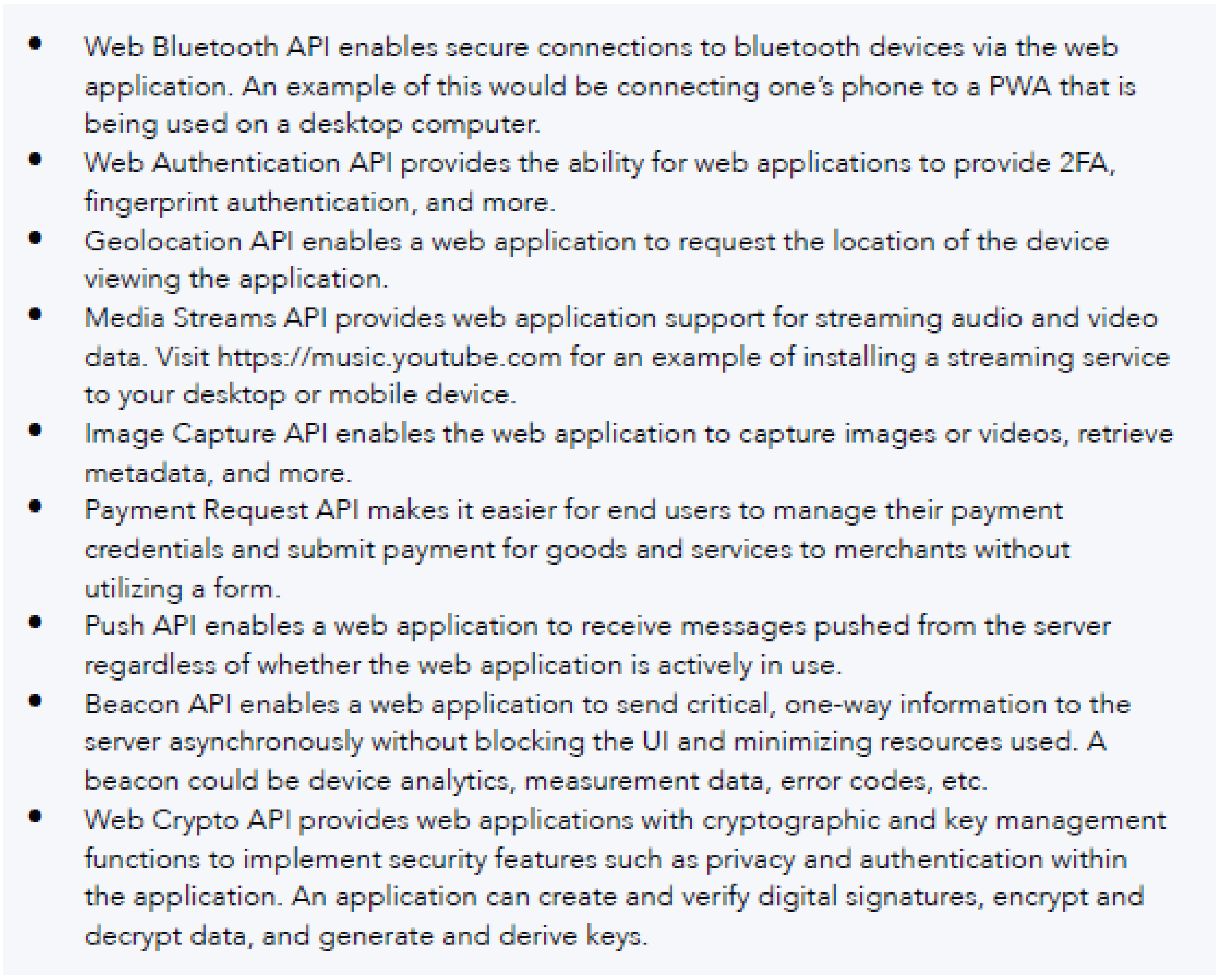
There are a number of ways to view the business impact of developing a PWA, and they are all positive. For businesses, utilizing a PWA provides an easier way of marketing solutions to end users. Since a PWA is linkable, it is easy to email, text, or instant message the link to users, enabling them to visit the site and install the application if they so choose. An added benefit is removing the necessity of entering an app store to access your application's features, resulting in more traffic and interactivity with your web application.

B2E app for company employees

A PWA for the enterprise application could begin with a portal for other partners or employees to enter and gain access to systems for ordering supplies or parts or updating manufacturing data across their desktop and mobile devices. Having this type of fluid installability removes barriers to entry. Additionally, since a PWA is browser-based, whitelisting and Mobile Device Management (MDM) solutions do not come into play. In other words, there is no need to deploy the application; employees can simply visit the website and download the application directly to their mobile or desktop environment, allowing data to be stored off the device and at the browser level.

Sample Web APIs Available for Use in PWAs

The APIs listed below are a drop in the bucket compared to the functionality that is available in the web today. From streaming media and video games to saving data offline and syncing back to the server when connectivity is regained, the web has grown tremendously. As an added bonus, there are more and more APIs being developed each day by Google, Microsoft, Mozilla, and more to become standards across the web. Such APIs provide the ability to exchange data through NFC tags, share links and information through the browser, and connect two different devices to share data—think Airdrop but for the web.



- Web Bluetooth API enables secure connections to bluetooth devices via the web application. An example of this would be connecting one's phone to a PWA that is being used on a desktop computer.
- Web Authentication API provides the ability for web applications to provide 2FA, fingerprint authentication, and more.
- Geolocation API enables a web application to request the location of the device viewing the application.
- Media Streams API provides web application support for streaming audio and video data. Visit https://music.youtube.com for an example of installing a streaming service to your desktop or mobile device.
- Image Capture API enables the web application to capture images or videos, retrieve metadata, and more.
- Payment Request API makes it easier for end users to manage their payment credentials and submit payment for goods and services to merchants without utilizing a form.
- Push API enables a web application to receive messages pushed from the server regardless of whether the web application is actively in use.
- Beacon API enables a web application to send critical, one-way information to the server asynchronously without blocking the UI and minimizing resources used. A beacon could be device analytics, measurement data, error codes, etc.
- Web Crypto API provides web applications with cryptographic and key management functions to implement security features such as privacy and authentication within the application. An application can create and verify digital signatures, encrypt and decrypt data, and generate and derive keys.

How Do You Build PWAs?

PWAs consist of HTML, CSS, JSON, and JavaScript and can be built using a variety of different methods. They are built nearly identical to regular web apps but with the addition of code needed to handle the service worker and manifest. Various libraries and frameworks are available to help accelerate the development process. However, even using these libraries and frameworks, a substantial amount of hand coding will still be needed to finish the app. Using these traditional methods will require extensive knowledge of service workers and how they need to interact with your app. Along with service worker knowledge, creating the tedious aspects of the user interface will also be a manual task. This approach really isn't all that difficult for small applications with limited functionality but can quickly become cumbersome and inefficient for enterprise PWAs.

Low-code PWA development

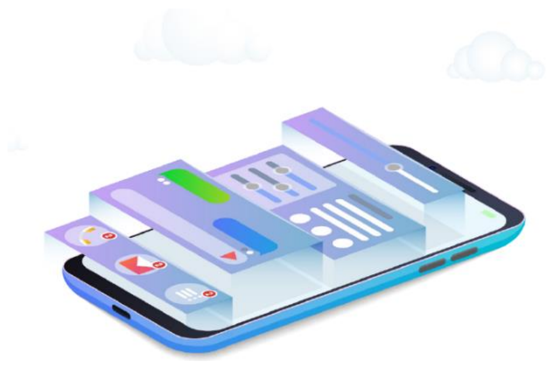
Low-code is the perfect use case for PWA development.

The user interface is created through drag-and-drop design, often with standards like Google Material Design built into the designer. In addition to saving time creating the visuals of your app, low-code will automatically handle the service worker and manifest, meaning the developer doesn't need to worry about the details of how the app will interface with the device. Much like other low-code principles, the developer focuses on creating solutions instead of spending time on coding functionality before coding to the actual business solution. Without having to know the intricacies of PWAs. As a result, low-code developers can quickly create enterprise-level PWAs and businesses will see high ROIs from their low-code investments.

Are PWAs Right for Your App?

PWAs are not the answer to every app being developed. There are still some cases in which creating a native app might be the best option. If you intend to solely focus on mobile, with in-app purchases, then you will want to go with developing a native mobile app. However, if you want your app to be used across all devices and be discoverable by SEO, then PWA is the way to go.
You also might see some misinformation surrounding PWAs. Don't let the myths deter you from seriously looking into developing them. Progressive web apps do, in fact, have access to nearly all aspects of a mobile device like native apps, including NFC. Also, PWAs do not use more battery power than a comparable native app, and they can be run on older hardware and browsers. Whether you need a customer-facing or internal app, a PWA should be given serious consideration for your development approach.

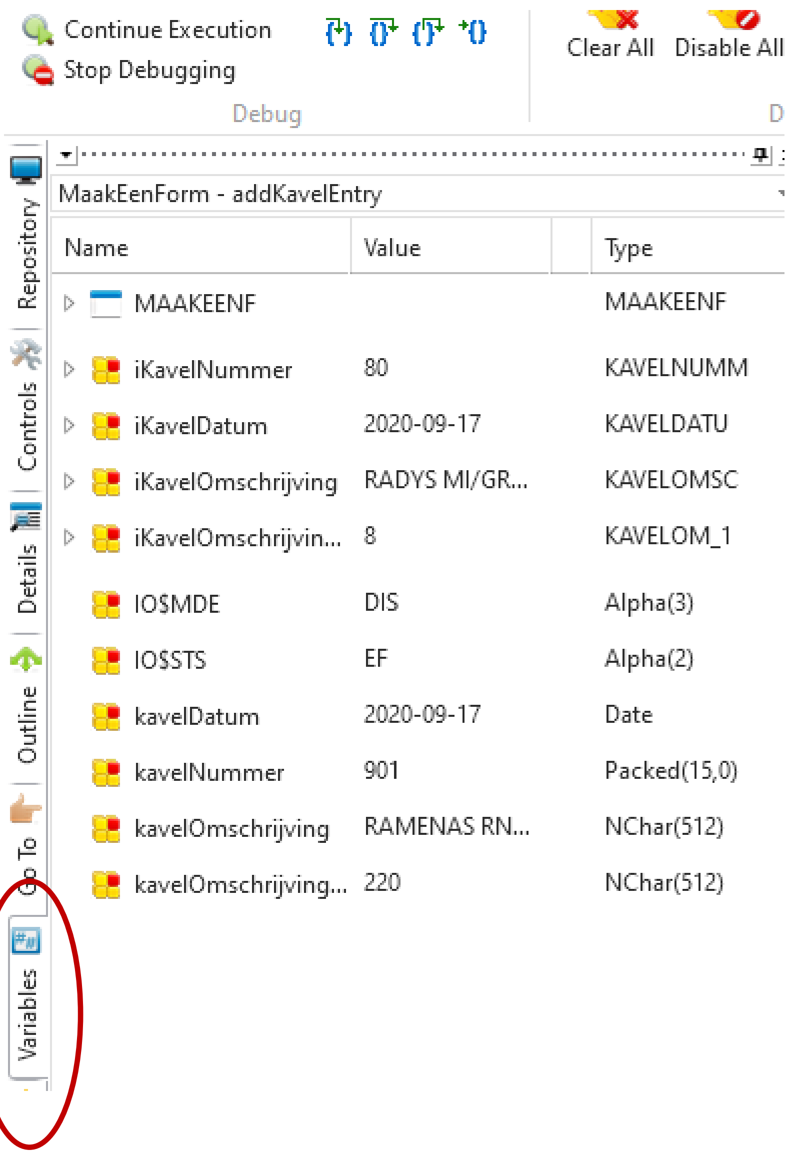
Be on the lookout for more information on how Visual LANSA is creating the next steps for low-code enterprise PWA.



Did you know?

Debug hint

If you run your Visual LANSA application in debug mode, you have access to the actual field values via the Variables tabsheet:

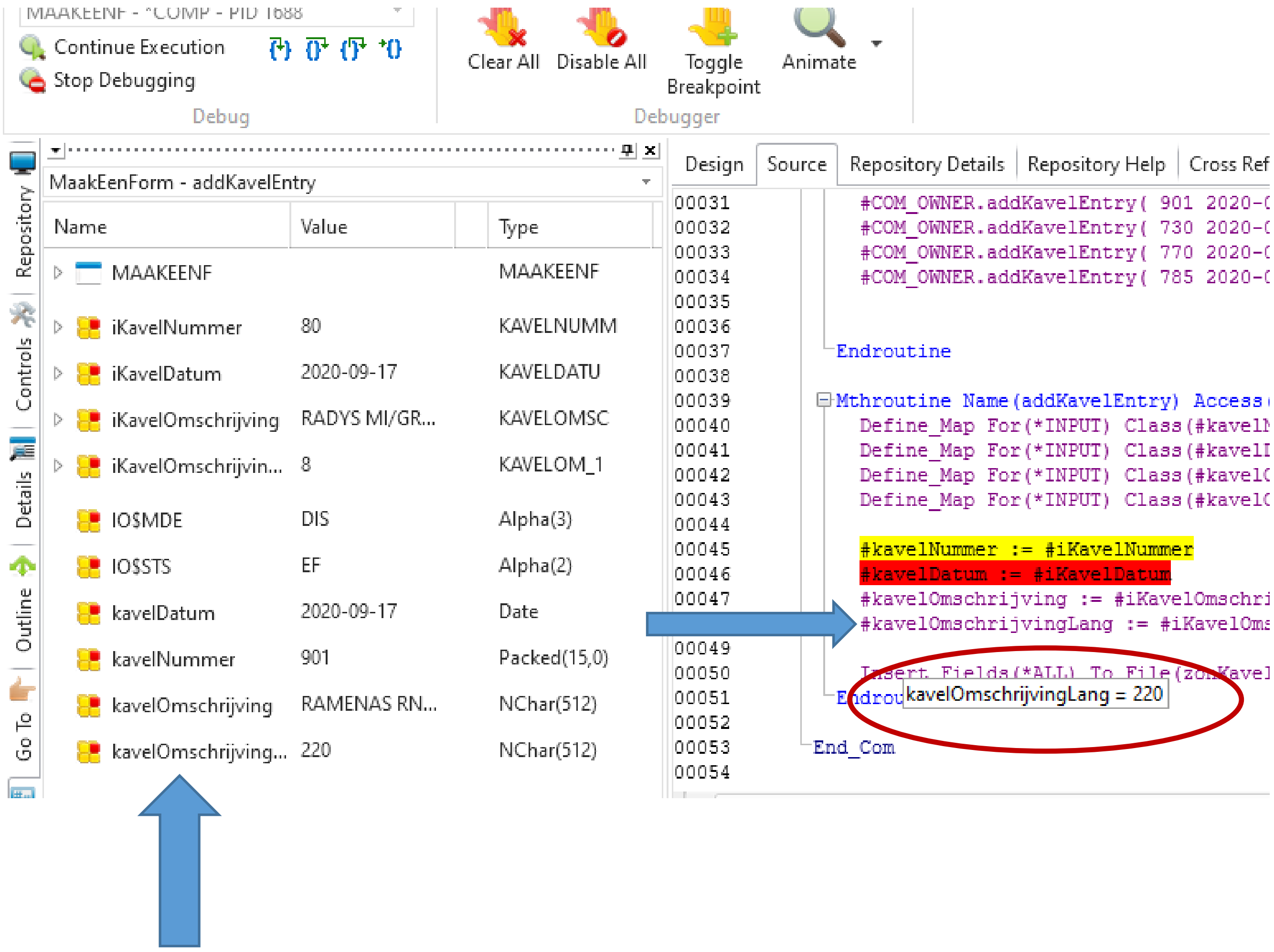


MaakEenForm - addKavelEntry

Name	Value	Type
MAAKEENF		MAAKEENF
iKavelNummer	80	KAVELNUMM
iKavelDatum	2020-09-17	KAVELDATU
iKavelOmschrijving	RADYS MI/GR...	KAVELOMSC
iKavelOmschrijvin...	8	KAVELOM_1
IO$MDE	DIS	Alpha(3)
IO$STS	EF	Alpha(2)
kavelDatum	2020-09-17	Date
kavelNummer	901	Packed(15,0)
kavelOmschrijving	RAMENAS RN...	NChar(512)
kavelOmschrijving...	220	NChar(512)

But did you know that ….

Since LANSA V14, the LANSA debugger in Windows shows the actual field value as a hint in the IDE:



MaakEenForm - addKavelEntry

Name	Value	Type
MAAKEENF		MAAKEENF
iKavelNummer	80	KAVELNUMM
iKavelDatum	2020-09-17	KAVELDATU
iKavelOmschrijving	RADYS MI/GR...	KAVELOMSC
iKavelOmschrijvin...	8	KAVELOM_1
IO$MDE	DIS	Alpha(3)
IO$STS	EF	Alpha(2)
kavelDatum	2020-09-17	Date
kavelNummer	901	Packed(15,0)
kavelOmschrijving	RAMENAS RN...	NChar(512)
kavelOmschrijving...	220	NChar(512)

```
     #COM_OWNER.addKavelEntry( 901 2020-(
     #COM_OWNER.addKavelEntry( 730 2020-(
     #COM_OWNER.addKavelEntry( 770 2020-(
     #COM_OWNER.addKavelEntry( 785 2020-(


  Endroutine

  Mthroutine Name(addKavelEntry) Access(
     Define_Map For(*INPUT) Class(#kavelN
     Define_Map For(*INPUT) Class(#kavelI
     Define_Map For(*INPUT) Class(#kavelO
     Define_Map For(*INPUT) Class(#kavelO

     #kavelNummer := #iKavelNummer
     #kavelDatum := #iKavelDatum
     #kavelOmschrijving := #iKavelOmschri
     #kavelOmschrijvingLang := #iKavelOms

     Insert Fields(*ALL) To File(zonKavel
  Endrou kavelOmschrijvingLang = 220

End_Com
```

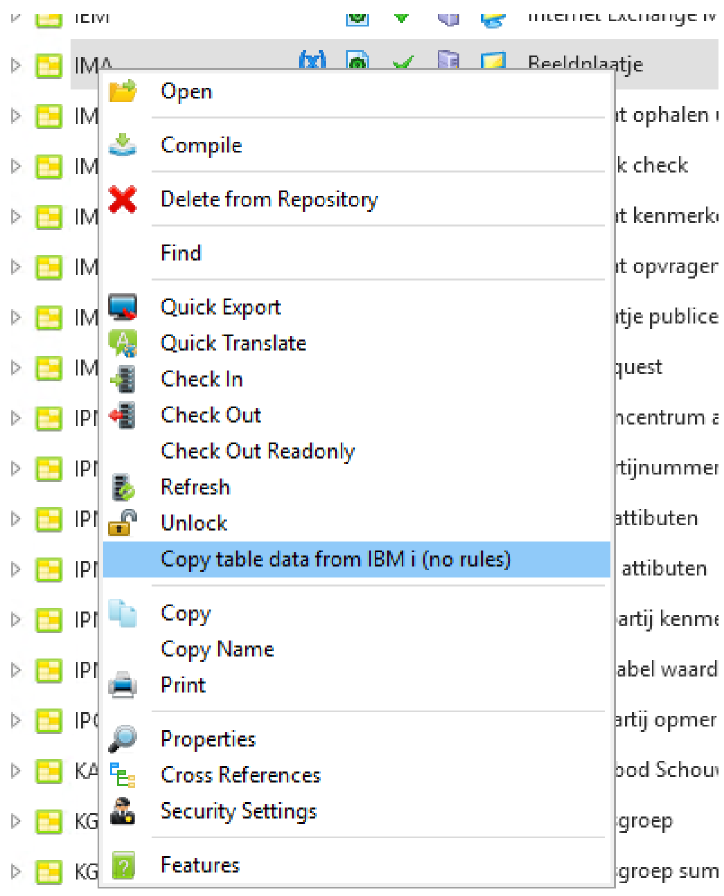
Copy table data from IBM i

Did you know that ….

File data can be copied from master IBM i files to the corresponding file in the Visual LANSA database.

- Rules and triggers are not applied during the file data copy.
- Files on IBM i and Visual LANSA must have equivalent definitions.
- Files must be compiled with appropriate database tables generated.
- Files must be enabled for RDMLX.

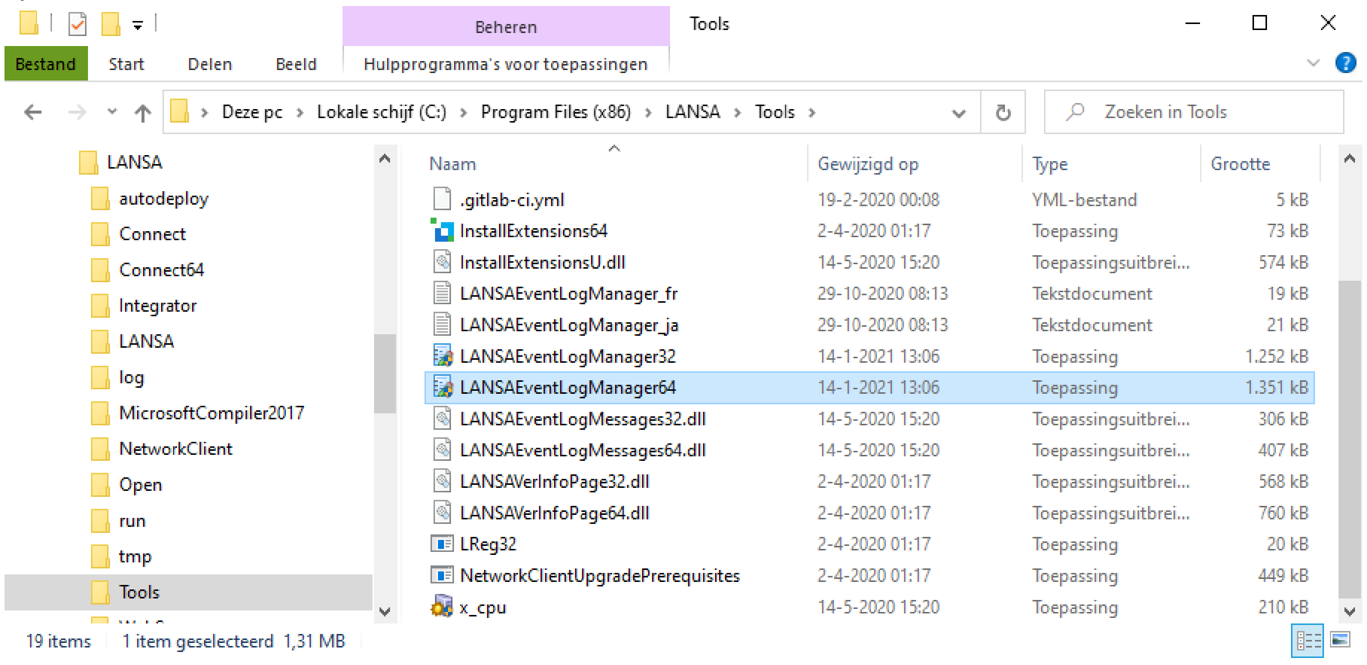
So, this feature is only available for RDMLX files, which have been compiled on the IBM i (so that the file definition is the same as on Windows):



Used LANSA ports in Windows

The LANSA Event Log Manager collects and centralizes detailed event and error information for all Visual LANSA related products, thus eliminating the need to manually search for log and trace files.

To start the Error Log Manager, locate and execute LANSAEventLogManager32 or LANSAEventLogManager64 (for 32 or 64 bit operating systems) in the Tools directory of your LANSA system:

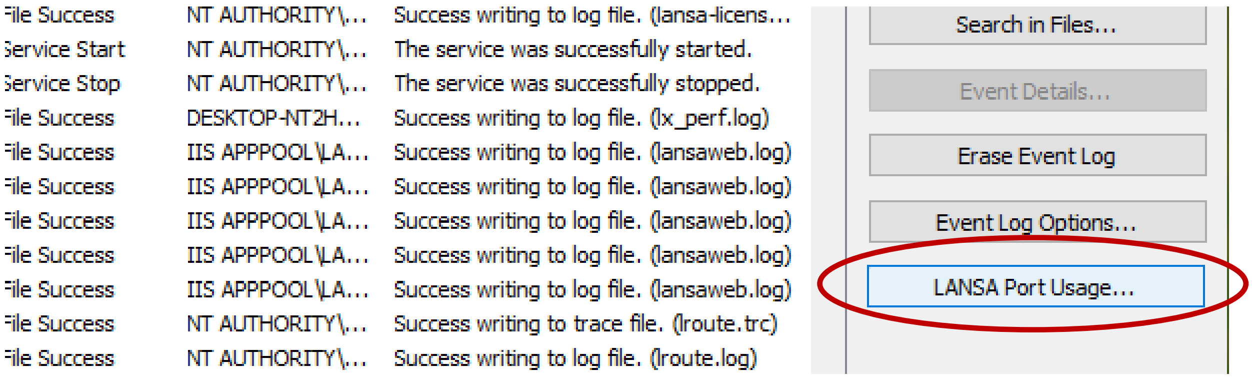


Naam	Gewijzigd op	Type	Grootte
.gitlab-ci.yml	19-2-2020 00:08	YML-bestand	5 kB
InstallExtensions64	2-4-2020 01:17	Toepassing	73 kB
InstallExtensionsU.dll	14-5-2020 15:20	Toepassingsuitbrei...	574 kB
LANSAEventLogManager_fr	29-10-2020 08:13	Tekstdocument	19 kB
LANSAEventLogManager_ja	29-10-2020 08:13	Tekstdocument	21 kB
LANSAEventLogManager32	14-1-2021 13:06	Toepassing	1.252 kB
LANSAEventLogManager64	14-1-2021 13:06	Toepassing	1.351 kB
LANSAEventLogMessages32.dll	14-5-2020 15:20	Toepassingsuitbrei...	306 kB
LANSAEventLogMessages64.dll	14-5-2020 15:20	Toepassingsuitbrei...	407 kB
LANSAVerInfoPage32.dll	2-4-2020 01:17	Toepassingsuitbrei...	568 kB
LANSAVerInfoPage64.dll	2-4-2020 01:17	Toepassingsuitbrei...	760 kB
LReg32	2-4-2020 01:17	Toepassing	20 kB
NetworkClientUpgradePrerequisites	2-4-2020 01:17	Toepassing	449 kB
x_cpu	14-5-2020 15:20	Toepassing	210 kB

19 items 1 item geselecteerd 1,31 MB

Did you know that ....

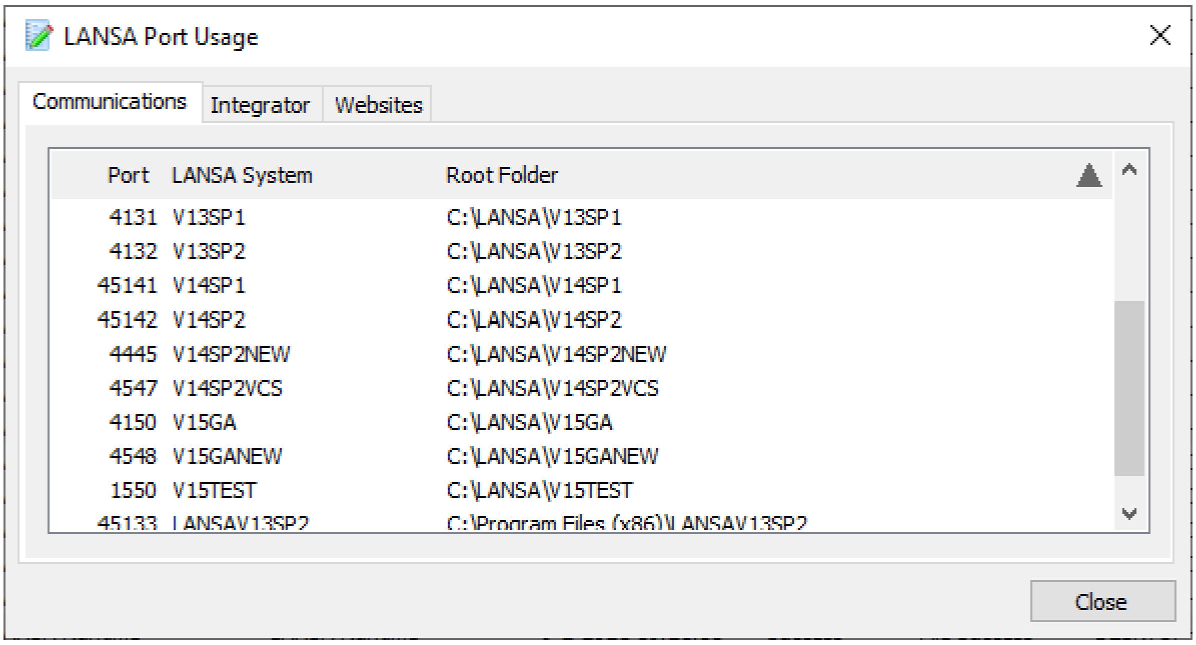
It also contains (since EPC150030) and option to view all Windows ports in use by the different LANSA products:



```
ile Success      NT AUTHORITY\...   Success writing to log file. (lansa-licens...
ervice Start     NT AUTHORITY\...   The service was successfully started.
ervice Stop      NT AUTHORITY\...   The service was successfully stopped.
ile Success      DESKTOP-NT2H...    Success writing to log file. (lx_perf.log)
ile Success      IIS APPPOOL\LA...  Success writing to log file. (lansaweb.log)
ile Success      IIS APPPOOL\LA...  Success writing to log file. (lansaweb.log)
ile Success      IIS APPPOOL\LA...  Success writing to log file. (lansaweb.log)
ile Success      IIS APPPOOL\LA...  Success writing to log file. (lansaweb.log)
ile Success      IIS APPPOOL\LA...  Success writing to log file. (lansaweb.log)
ile Success      NT AUTHORITY\...   Success writing to trace file. (lroute.trc)
ile Success      NT AUTHORITY\...   Success writing to log file. (lroute.log)
```

The different ports can be viewed based on:

- Communications.
- Integrator.
- Websites.



Port	LANSA System	Root Folder
4131	V13SP1	C:\LANSA\V13SP1
4132	V13SP2	C:\LANSA\V13SP2
45141	V14SP1	C:\LANSA\V14SP1
45142	V14SP2	C:\LANSA\V14SP2
4445	V14SP2NEW	C:\LANSA\V14SP2NEW
4547	V14SP2VCS	C:\LANSA\V14SP2VCS
4150	V15GA	C:\LANSA\V15GA
4548	V15GANEW	C:\LANSA\V15GANEW
1550	V15TEST	C:\LANSA\V15TEST
45133	LANSAV13SP2	C:\Program Files (x86)\LANSAV13SP2

EPC 150030 – Routing and Caching

With EPC150030 Visual LANSA is delivering two enhancements:

- **Routing** – the ability to use the path component of a URL to move a Visual LANSA Single Page Application through different visual states, and
- **Caching** – utilize a generated script to leverage the functionality of a service worker to support system and application resource caching.

The following article will first explore the Routing enhancement before looking into the Caching enhancement.

Routing

The Routing option allows you to use the path component of a URL to move a Visual LANSA single page application through different visual states.

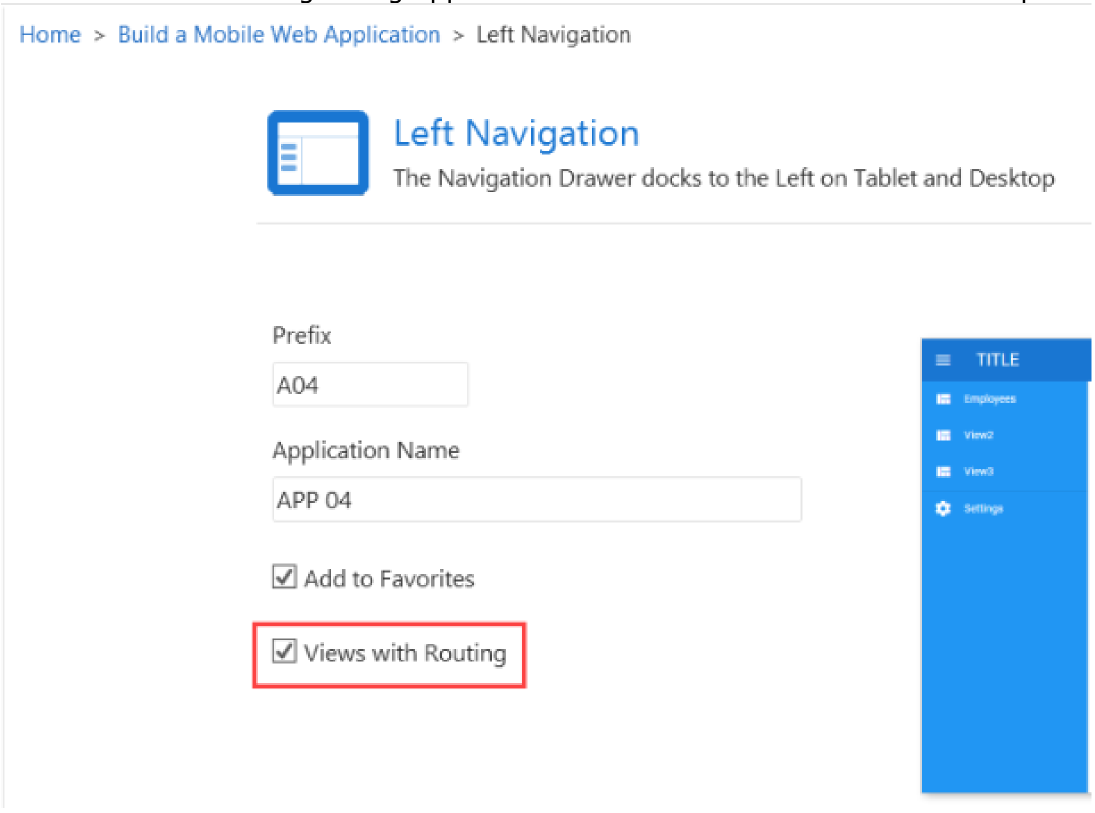
Views with Routing

When creating a new Web Page, you can select the Views with Routing option to create a Web Page and Web Views with routing features for application navigation.

To select Views with Routing option:

1. Go to LANSA IDE Home Page.
2. Select Build a Mobile Application template.
3. Click on Left Navigation and check this option.

The Views with Routing dialog appears with the Router initialized with a couple of routes.


Home > Build a Mobile Web Application > Left Navigation

Left Navigation

The Navigation Drawer docks to the Left on Tablet and Desktop

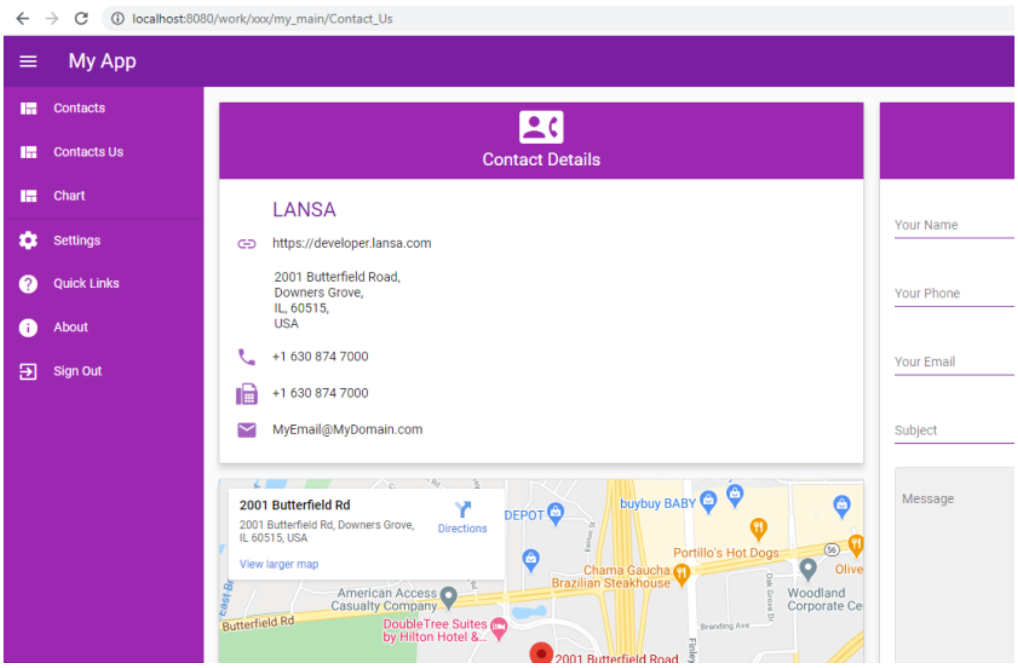
Prefix: A04

Application Name: APP 04

☑ Add to Favorites

☑ Views with Routing

When you run this template, the following Web Page is generated with its navigation.



localhost:8080/work/xxx/my_main/Contact_Us

My App

Contacts
Contacts Us
Chart
Settings
Quick Links
About
Sign Out

Contact Details

LANSA
https://developer.lansa.com
2001 Butterfield Road,
Downers Grove,
IL 60515,
USA
+1 630 874 7000
+1 630 874 7000
MyEmail@MyDomain.com

Your Name
Your Phone
Your Email
Subject
Message

2001 Butterfield Rd
2001 Butterfield Rd, Downers Grove, IL 60515, USA
Directions
View larger map

The Build a Mobile Web Application template generates the following RDMLX code:

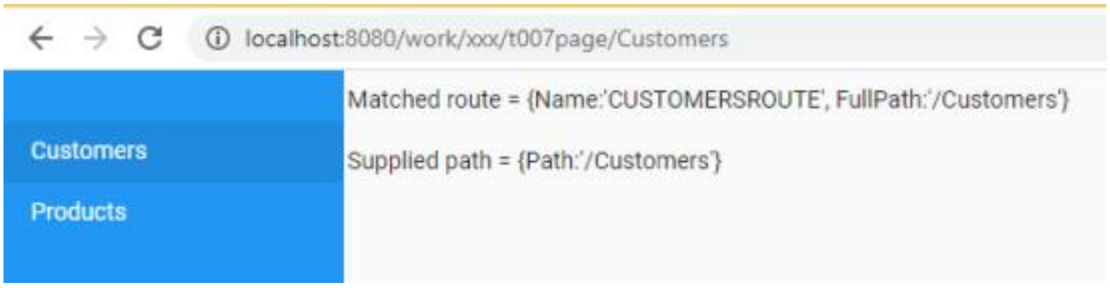
```
 * Views
 Define_Com Class(#PRIM_MD.ViewContainer) Name(#ViewContainer) Displayposition(3) Left(220)
     Parent(#COM_OWNER) Tabposition(3) Top(96) Height(704) Width(980) Router(#Router)

Define_Com Class(#prim_web.Router) Name(#Router)
 Define_Com Class(#prim_web.Route) Name(#Home) Path('/') Parent(#Router)
 Define_Com Class(#prim_web.Route) Name(#ContactsRoute) Path('/Contacts') Parent(#Router)
     View(#MY_Contacts) Access(Protected)
 Define_Com Class(#prim_web.Route) Name(#ContactUsRoute) Path('/Contact_Us') Parent(#Router)
     View(#MY_Contacts_Us)
 Define_Com Class(#prim_web.Route) Name(#ChartRoute) Path('/Chart') Parent(#Router) View(#MY_Chart
     Access(Protected)
 Define_Com Class(#prim_web.Route) Name(#SettingsRoute) Path('/Settings') Parent(#Router)
     View(#MY_Settings) Access(Protected)
 Define_Com Class(#prim_web.Route) Name(#QuickLinksRoute) Path('/QuickLinks') Parent(#Router)
     View(#MY_QuickLinks)
 Define_Com Class(#prim_web.Route) Name(#SignInRoute) Path('/SignIn') Parent(#Router)

 * SignIn Dialog
 Define_Com Class(#MY_SignIn) Name(#SignIn)
 * About Dialog
 Define_Com Class(#MY_About) Name(#About)
```

Visual LANSA Web View is associated with a route component. This route component identifies the Web view class that is used to populate the View Container when the path of the associated route is requested.

The Access property of **#PRIM_WEB.Route** class is used to flag protected routes. Visual LANSA applies no meaning to a protected route. The application code is responsible that the application takes the appropriate action to ensure the integrity of the route. Usually, this involves triggering a Sign In route if the user is not signed in at the time the route is accessed.

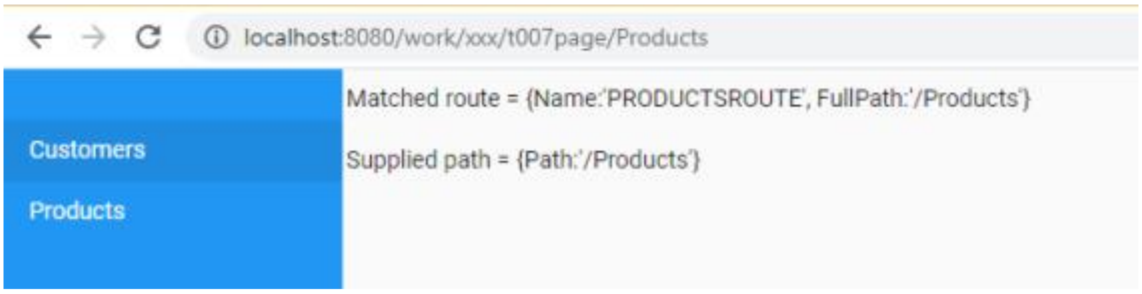
URL Changes

The following images illustrate the URL changes to control the new routing features:


localhost:8080/work/xxx/t007page/Customers

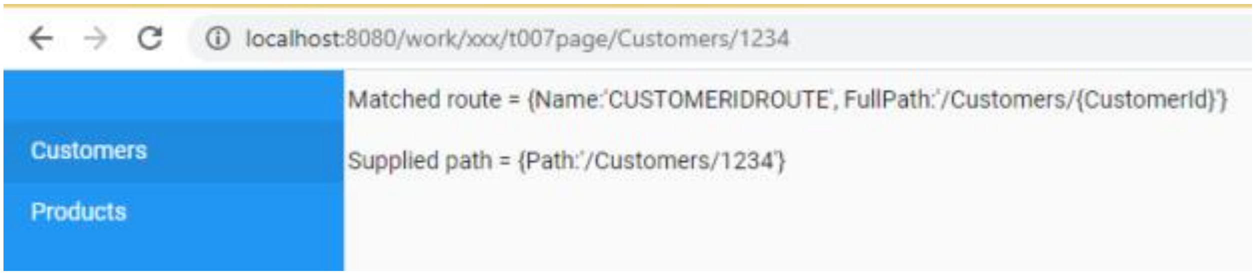
Matched route = {Name:'CUSTOMERSROUTE', FullPath:'/Customers'}

Supplied path = {Path:'/Customers'}


localhost:8080/work/xxx/t007page/Products

Matched route = {Name:'PRODUCTSROUTE', FullPath:'/Products'}

Supplied path = {Path:'/Products'}


localhost:8080/work/xxx/t007page/Customers/1234

Matched route = {Name:'CUSTOMERIDROUTE', FullPath:'/Customers/{CustomerId}'}

Supplied path = {Path:'/Customers/1234'}

The Views with Routing option generates the following RDMLX sample code:

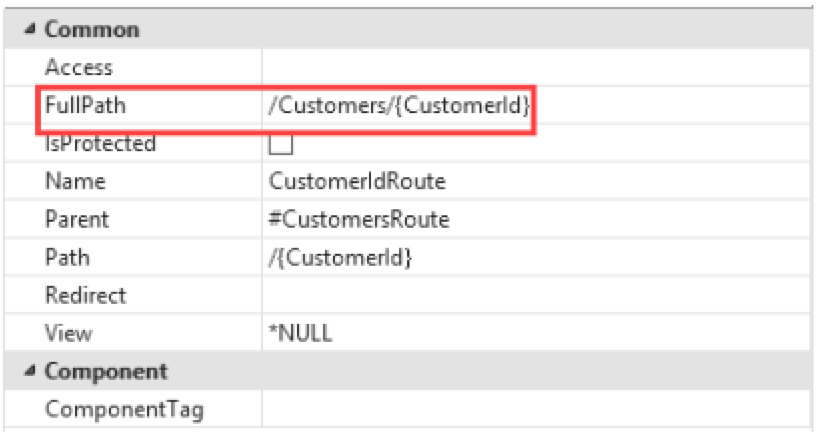
```
Define_Com Class(#PRIM_MD.ViewContainer) Name(#ViewContainer) Displayposition(1) Left(200)
    Parent(#COM_OWNER) Tabposition(1) Height(800) Width(1000) Router(#Router)

Define_Com Class(#PRIM_MD.AppDrawer) Name(#AppDrawer) Displayposition(2) Parent(#COM_OWNER)
    Tabposition(2) Height(800) Themedrawstyle('MediumTitle') Width(200)
    Layoutmanager(#LayoutAppDrawer)
Define_Com Class(#PRIM_MD.MenuItem) Name(#Customers) Caption('Customers') Displayposition(1)
    Parent(#AppDrawer) Tabposition(1) Link('/Customers') Width(200) Left(0) Top(30)
Define_Com Class(#PRIM_MD.MenuItem) Name(#Products) Caption('Products') Displayposition(2) Left(0)
    Parent(#AppDrawer) Tabposition(2) Top(66) Link('/Products') Width(200)

Define_Com Class(#prim_web.Router) Name(#Router)

Define_Com Class(#prim_web.Route) Name(#Default) Path('/') Parent(#Router) Redirect('/Home')
Define_Com Class(#prim_web.Route) Name(#Home) Path('/Home') Parent(#Router)

Define_Com Class(#prim_web.Route) Name(#CustomersRoute) Parent(#Router) Path('/Customers')
Define_Com Class(#prim_web.Route) Name(#CustomerIdRoute) Parent(#CustomersRoute)
    Path('/{CustomerId}')

Define_Com Class(#prim_web.Route) Name(#ProductsRoute) Parent(#Router) Path('/Products')
Define_Com Class(#prim_web.Route) Name(#ProductIdRoute) Parent(#ProductsRoute) Path('/{ProductId}')
```

Where:

#PRIM_WEB.Router
Acts as the root of the routes tree.

#PRIM_WEB.Route
Associates a path string to a Visual LANSA web view component. It is displayed when the path string matches with the current URL path component.

The parent property of the #PRIM_WEB.Route class forms a tree of routes where the FullPath of a route is derived from the FullPath of its parent and the route owns the Path string as highlighted below:



Common	
Access	
FullPath	/Customers/{CustomerId}
IsProtected	☐
Name	CustomerIdRoute
Parent	#CustomersRoute
Path	/{CustomerId}
Redirect	
View	*NULL
Component	
ComponentTag	

The Link #PRIM_MD.MenuItem class enables a menu item to trigger a change to the URL path component when clicked.

Generated Template

Router(#Router)
The router property on the #PRIM_MD.ViewContainer class links a router and its set of Routes to a view container.

Path('/')
The path property on #PRIM_WEB.Route provides the location of the component that should be matched.

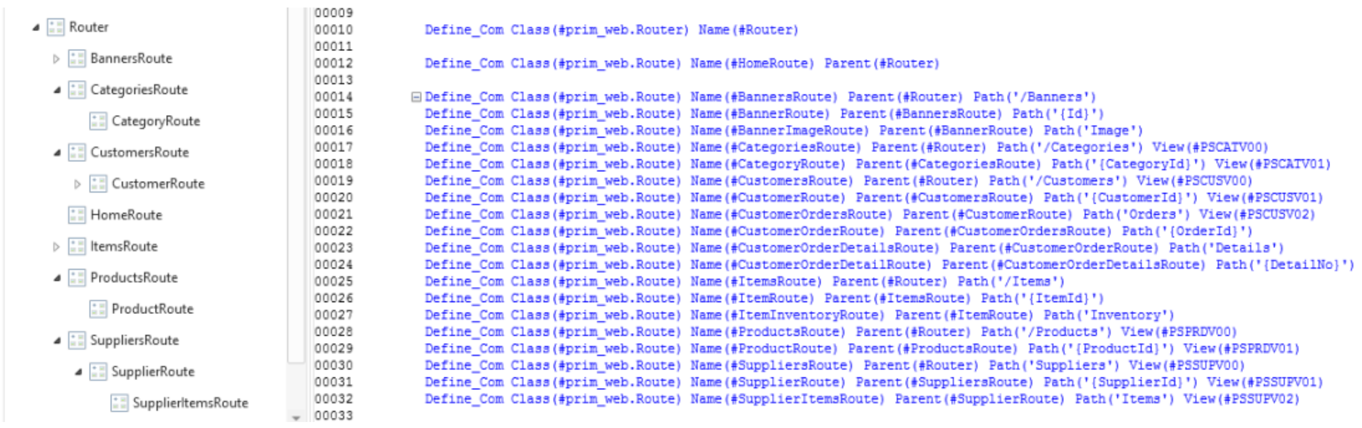
View(#MY_Contacts) and View(#Contact_Us)
As highlighted with red in the sample code above, the views are now associated with a route component. The route component will identify the web view class that will be used to populate the View Container when the Path('/Contacts') or Path('/Contact_us') of the route is requested.

Access(Protected)
The new Access property on #PRIM_WEB.Route is used to flag routes that are protected such as those that require authentication prior to generating the view. Developers have access to the following values (Private, Public, and *SQLNULL)

Routes

When you select the Routing option, you have the following routes available:

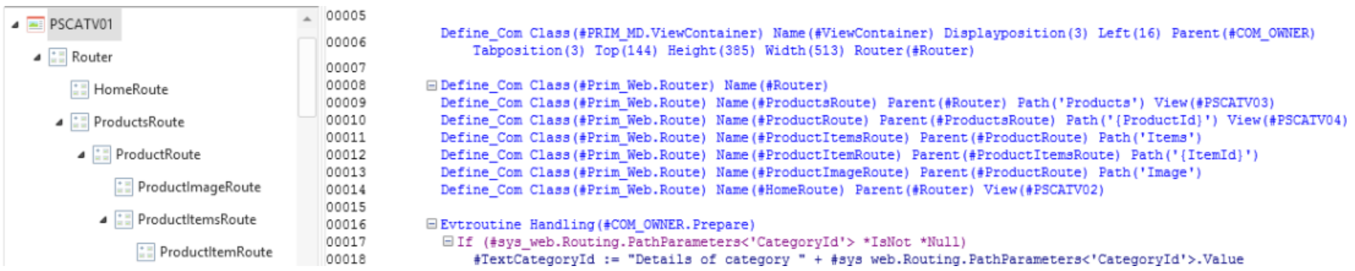
Child Routes
A child route is instantiated by adding the Parent(#ExampleRoute) to the #PRIM_WEB.Route class. The purpose of a child route is to build a path hierarchy. For example, a #CustomersRoute may have child routes #CustomerContact, #CustomerMap, and #Customer Profile.



```
Define_Com Class(#prim_web.Router) Name(#Router)

Define_Com Class(#prim_web.Route) Name(#HomeRoute) Parent(#Router)

Define_Com Class(#prim_web.Route) Name(#BannersRoute) Parent(#Router) Path('/Banners')
Define_Com Class(#prim_web.Route) Name(#BannerRoute) Parent(#BannersRoute) Path('{Id}')
Define_Com Class(#prim_web.Route) Name(#BannerImageRoute) Parent(#BannerRoute) Path('Image')
Define_Com Class(#prim_web.Route) Name(#CategoriesRoute) Parent(#Router) Path('/Categories') View(#PSCATV00)
Define_Com Class(#prim_web.Route) Name(#CategoryRoute) Parent(#CategoriesRoute) Path('{CategoryId}') View(#PSCATV01)
Define_Com Class(#prim_web.Route) Name(#CustomersRoute) Parent(#Router) Path('/Customers') View(#PSCUSV00)
Define_Com Class(#prim_web.Route) Name(#CustomerRoute) Parent(#CustomersRoute) Path('{CustomerId}') View(#PSCUSV01)
Define_Com Class(#prim_web.Route) Name(#CustomerOrdersRoute) Parent(#CustomerRoute) Path('Orders') View(#PSCUSV02)
Define_Com Class(#prim_web.Route) Name(#CustomerOrderRoute) Parent(#CustomerOrdersRoute) Path('{OrderId}')
Define_Com Class(#prim_web.Route) Name(#CustomerOrderDetailsRoute) Parent(#CustomerOrderRoute) Path('Details')
Define_Com Class(#prim_web.Route) Name(#CustomerOrderDetailRoute) Parent(#CustomerOrderDetailsRoute) Path('{DetailNo}')
Define_Com Class(#prim_web.Route) Name(#ItemsRoute) Parent(#Router) Path('/Items')
Define_Com Class(#prim_web.Route) Name(#ItemRoute) Parent(#ItemsRoute) Path('{ItemId}')
Define_Com Class(#prim_web.Route) Name(#ItemInventoryRoute) Parent(#ItemRoute) Path('Inventory')
Define_Com Class(#prim_web.Route) Name(#ProductsRoute) Parent(#Router) Path('/Products') View(#PSPRDV00)
Define_Com Class(#prim_web.Route) Name(#ProductRoute) Parent(#ProductsRoute) Path('{ProductId}') View(#PSPRDV01)
Define_Com Class(#prim_web.Route) Name(#SuppliersRoute) Parent(#Router) Path('Suppliers') View(#PSSUPV00)
Define_Com Class(#prim_web.Route) Name(#SupplierRoute) Parent(#SuppliersRoute) Path('{SupplierId}') View(#PSSUPV01)
Define_Com Class(#prim_web.Route) Name(#SupplierItemsRoute) Parent(#SupplierRoute) Path('Items') View(#PSSUPV02)
```

Nested Routes
Nested routes are routes that are attached to their own View Container and Router. Within a nested route, a developer can have a distinct tree of routes that are associated with additional views within its own view container.

The following snippet from the view (PSCATV01) associated with the component name #CategoryRoute in the previous image has its own router that allows the nesting of additional views inside its own view container. Thus, an active path like /Categories/Fluffy/Products/P123 uses the path segment /Categories/Fluffy to identify PSCATV01 and when PSCATV01 is created, the route component named ProductItemRoute is identified using the Products/P123.



```
Define_Com Class(#PRIM_MD.ViewContainer) Name(#ViewContainer) Displayposition(3) Left(16) Parent(#COM_OWNER)
    Tabposition(3) Top(144) Height(385) Width(513) Router(#Router)

Define_Com Class(#Prim_Web.Router) Name(#Router)
Define_Com Class(#Prim_Web.Route) Name(#ProductsRoute) Parent(#Router) Path('Products') View(#PSCATV03)
Define_Com Class(#Prim_Web.Route) Name(#ProductRoute) Parent(#ProductsRoute) Path('{ProductId}') View(#PSCATV04)
Define_Com Class(#Prim_Web.Route) Name(#ProductItemsRoute) Parent(#ProductRoute) Path('Items')
Define_Com Class(#Prim_Web.Route) Name(#ProductItemRoute) Parent(#ProductItemsRoute) Path('{ItemId}')
Define_Com Class(#Prim_Web.Route) Name(#ProductImageRoute) Parent(#ProductRoute) Path('Image')
Define_Com Class(#Prim_Web.Route) Name(#HomeRoute) Parent(#Router) View(#PSCATV02)

Evtroutine Handling(#COM_OWNER.Prepare)
 If (#sys_web.Routing.PathParameters<'CategoryId'> *IsNot *Null)
    #TextCategoryId := "Details of category " + #sys web.Routing.PathParameters<'CategoryId'>.Value
```

Protected Routes
Protected routes are used by Web Applications to trigger special handling that includes authentication and authorization.

Wildcards
Wildcards are used in paths. The path string of a route can end with the asterisk character and trigger a match with this route. No exact or partial matches can be found.

Note: Routing handles partial matches.

Component Features

#PRIM_WEB.Application

The Web Application object is a globally scoped singleton object providing access to web runtime specific features such as the browser Console, Local Storage and the current URL.

At runtime, the #SYS_WEB variable provides programmatic access to its features.

This existing component class has been enhanced with the addition of four features.

1. #PRIM_WEB.Application.Routing

The property provides read only access to an instance of the #PRIM_WEB.Routing class. This provides programmatic access to the current state of the Routing framework.

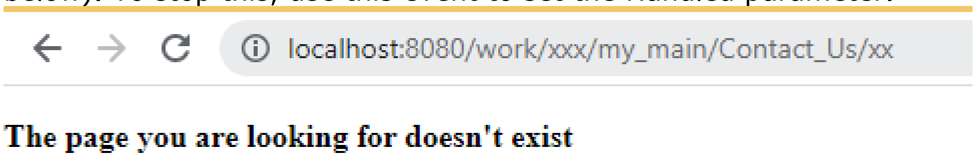
2. #PRIM_WEB.Application.RoutingChanged

This event is signalled once the routing framework has successfully matched a route with the current URL path component and has prepared the appropriate views in the appropriate view containers. The following code snippet shows the handling of the RoutingChanged event, specifically spotting the use of the "/SignIn" path in order to trigger the Sign-in dialog.

```
Evtroutine Handling(#SYS_WEB.RoutingChanged)

  #sys_appln.CloseAllDialogs

  If (#sys_web.Routing.ActivePath = #SignInRoute.Path)
    #IsSignedIn := false
    #ViewContainer.Clear
    #SignIn.ShowForSignIn
  Endif
Endroutine
```

3. #PRIM_WEB.Application.PageNotFound

This event is signalled when the routing framework fails to match a route to the current URL path component. Should this event not be handled, the default Visual LANSA display will appear (shown below). To stop this, use this event to set the Handled parameter.



localhost:8080/work/xxx/my_main/Contact_Us/xx

The page you are looking for doesn't exist

4. #PRIM_WEB.Application.ReportPageNotFound

This method is used to signal the PageNotFound event and if not handled, show the default Visual LANSA display. This method can be used to trigger application 404 (page not found) handling where appropriate – like a path variable that should identify the primary key of a resource raising a not found in the database.

#PRIM_WEB.Router

This component class is a simple component that acts as the root of a tree of routes. It is associated with a #PRIM_MD.ViewContainer instance to activate routing on view container.
It inherits from #PRIM_WEB.Routable so that it can be the parent of #PRIM_Web.Route instances.

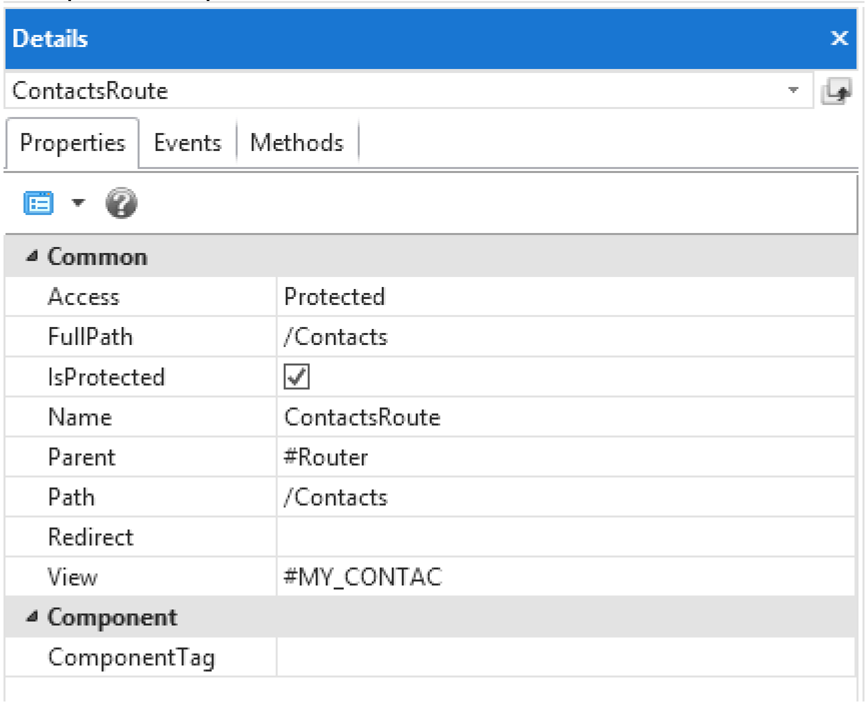
#PRIM_WEB.Router.Match

This event is signalled once the routing framework has successfully matched a route with the current URL path component and is provided for the application to check its state and determine if the matched route is valid to continue. The following code snippet shows the handling of the Match event, specifically spotting the use of the protected route and should the user not be signed-in yet, triggering a redirect to the "/SignIn" path.

```
Evtroutine Handling(#Router.Match) Route(#Route) Allow(#Allow) Redirect(#Redirect)

  If (#Route.IsProtected)
    If (#IsSignedIn = true)
      #Allow := true
    Else
      #Redirect := #SignInRoute.Path
      #RedirectOnSignIn := #Route.FullPath
    Endif
  Endif
Endroutine
```

#PRIM_WEB.Route

This component class is a simple component that associates a path string with the Visual LANSA Web View class that is to be displayed when the path string matches the current URL path component.



Details

ContactsRoute

Properties | Events | Methods

Common	
Access	Protected
FullPath	/Contacts
IsProtected	✓
Name	ContactsRoute
Parent	#Router
Path	/Contacts
Redirect	
View	#MY_CONTAC
Component	
ComponentTag	

#PRIM_WEB.Route.Access

This property supports three values - *SQLNULL, Public and Protected. The default is *SQLNULL. When the routing framework attempts to resolve a route's Access, *SQLNULL is used to indicate that the Access property of the route's parent should be used. When the top of the tree is reached, Public is used.

#PRIM_WEB.Route.FullPath

A read-only property that provides a path string that includes the FullPath of the route's parent.

#PRIM_WEB.Route.OnCloseAction

This property supports four values - *SQLNULL, DropInstance, KeepInstance, KeepRealized. The default is *SQLNULL. DropInstance is used to reset the reference to the view being closed which means that a new instance will be created the next time this route is activated. KeepInstance retains the component instance but does not keep it realized. KeepRealized keeps the component instance in its realized state.

When the routing framework attempts to resolve a route's OnCloseAction, *SQLNULL is used to indicate that the OnCloseAction property of the route's parent should be used. When the top of the tree is reached, DropInstance is used.

When this property is changed after the associated view is closed, the selected OnCloseAction is applied to the component instance.

#PRIM_WEB.Route.Parent

A property that controls where in the path hierarchy this route belongs. Use a Router as the parent when the route defines a topmost path, or another route component that will qualify the route's path.

#PRIM_WEB.Route.Path

A string that defines the values to be used to match against the current URL path component.

Use the "{name}" format for a path segment should the segment need to identify a variable.

Use the form "/name/partialname*" for a wildcard path. The asterisk must be at the end of the path segment.

#PRIM_WEB.Route.Redirect

A string that defines a new path should the current route match the current URL path component.

Make sure the value starts with a forward slash ("/") if the new path is an absolute path and is to be append to the current application's URL root path. When the string does not begin with a forward slash it is treated as a relative path and is appended to the current URL's path component.

#PRIM_WEB.Route.View

Name of the web view repository component that will be displayed when the route's path matches the current URL path component.

The files that implement the web view will not be downloaded to the browser until the view is required. It will be downloaded asynchronously.

#PRIM_WEB.Routing

This component class defines the features of a helper component that is accessed using a new Routing property of the #PRIM_WEB.Application component class. Use the Routing class instance to access the state of the current routing request.

#PRIM_WEB.Routing.ActivePath

A read-only property that supplies a string containing that piece of the current URL path component that has been processed by the routing framework.

#PRIM_WEB.Routing.FullPath

A read-only property that supplies a string containing that the current URL path component. It is a concatenation of the RootPath and the ActivePath properties.

#PRIM_WEB.Routing.RootPath

A read-only property that supplies a string containing that piece of the current URL path component that identifies the current web page.

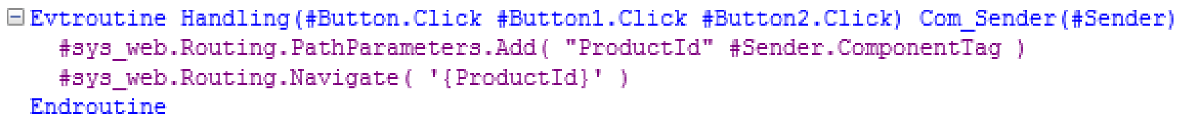
#PRIM_WEB.Routing.PathParameters

This read-only property provides access to an instance of the class #PRIM_WEB.RoutingPathParameters which is a read-write collection of #PRIM_WEB.RoutingPathParameter components.
Each instance of class #PRIM_WEB.RoutingPathParameter contains a Name/Value pair with the name corresponding to the name assigned to the path variable and the value corresponding to the actual value for the variable in the current URL path component.

For example, the path "/Categories/Fluffy/Products/P123" for the path "Categories/{CategoryId}/Products/{ProductId}" would supply the value "Fluffy" for the parameter named CategoryId and "P123" for the parameter named ProductId.

#PRIM_WEB.Routing.Navigate

Use this method to request that the application navigate to the path supplied as a parameter.
The following snippet shows how to establish path parameter values before calling the navigate method where the supplied path requires path variable substitution.



```
Evtroutine Handling(#Button.Click #Button1.Click #Button2.Click) Com_Sender(#Sender)
  #sys_web.Routing.PathParameters.Add( "ProductId" #Sender.ComponentTag )
  #sys_web.Routing.Navigate( '{ProductId}' )
Endroutine
```

#PRIM_WEB.Routing.Replace

Use this method to request that the application navigate to the path supplied as a parameter and replace the current item in the browser history.

#PRIM_MD.ViewContainer

This material design component class is not new but has been enhanced with a new property called Router. This new property is used to associate a view container with a tree of routes that use the current URL path component to determine what web view needs to be presented to the user.

#PRIM_MD.ViewContainer.Router

Property to associate a Router and the tree of routes with the container that will parent the views identified by the processing of the current URL's path component.

#PRIM_MD.MenuItem
#PRIM_MD.FlatButton

This material design component class is not new but has been enhanced with a new property called Link.

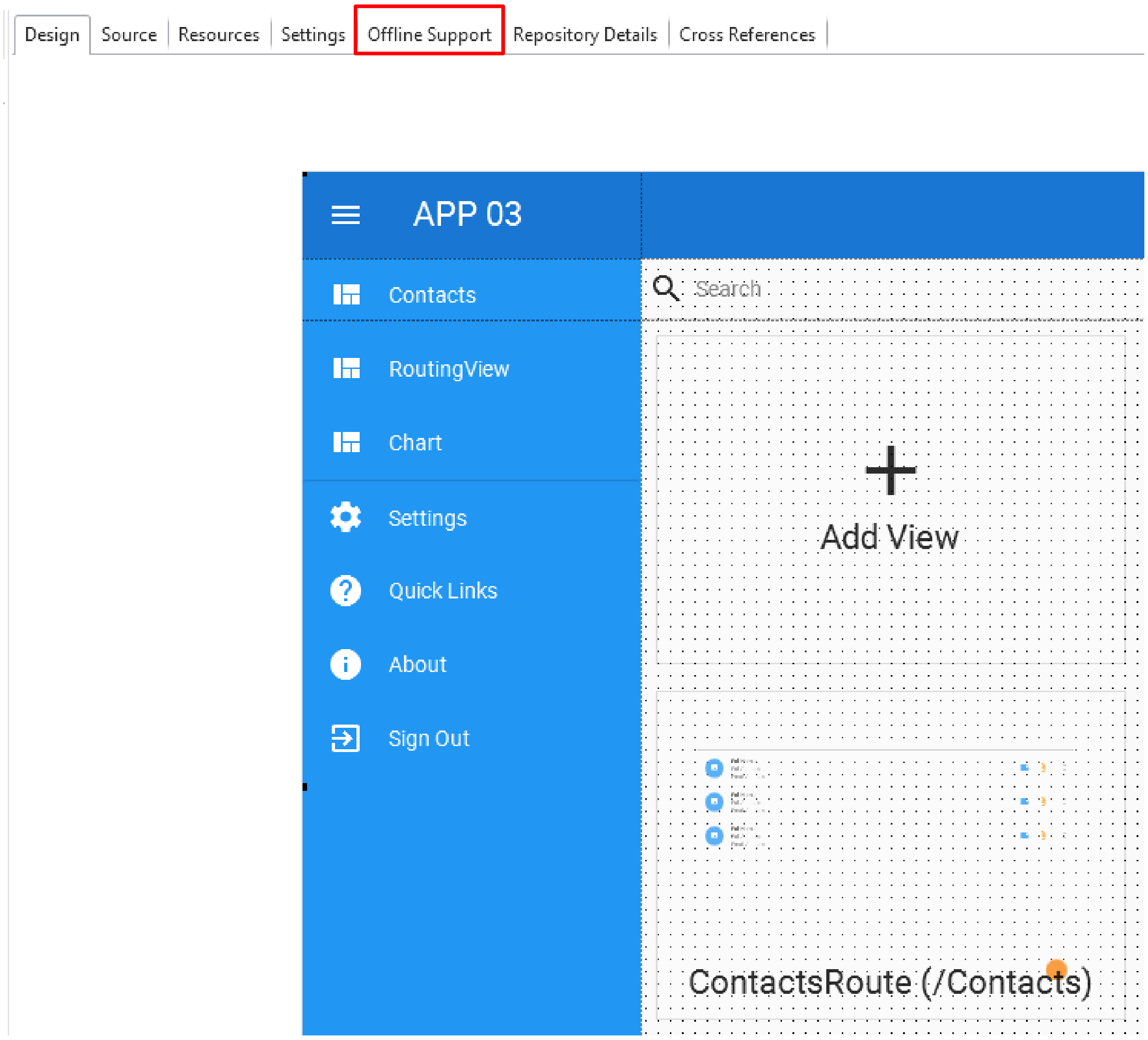
#PRIM_MD.MenuItem.Link

String property that defines a path to apply to the web application should the menu item be clicked. Make sure the link starts with a forward slash ("/") if the link is an absolute link and is to be append to the current application's URL root path. When the string does not begin with a forward slash it is treated as a relative link and is appended to the current URL's path component.

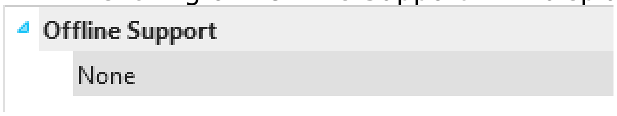
Caching

Caching has always been around. With the movement toward Progressive Web Applications and service workers, the browser cache has been extended as a storage option beyond local storage and session storage. Utilizing Cache Storage enables aspects of your LANSA Progressive Applications to be available in offline situations, speed up the availability of resources, and decrease the need to make network calls back to the server.

When viewing the canvas of the application being built, a tab is available entitled “Offline Support.” The purpose of this tab is to enable developers to select whether a service worker is to be included within the application, manipulate the web app manifest, and nominate resources to be cached.



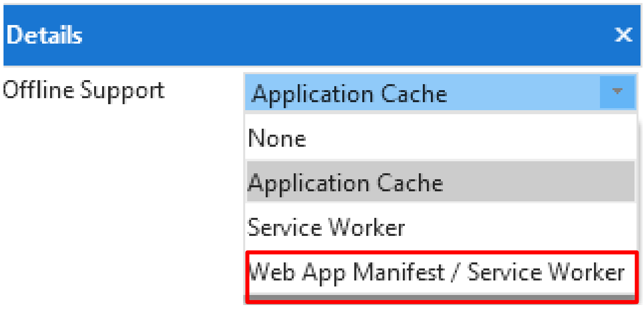
1. Clicking on "Offline Support" will display the following:



Offline Support

None

2. Double click "None" and the details tab will display a dropdown of available options:

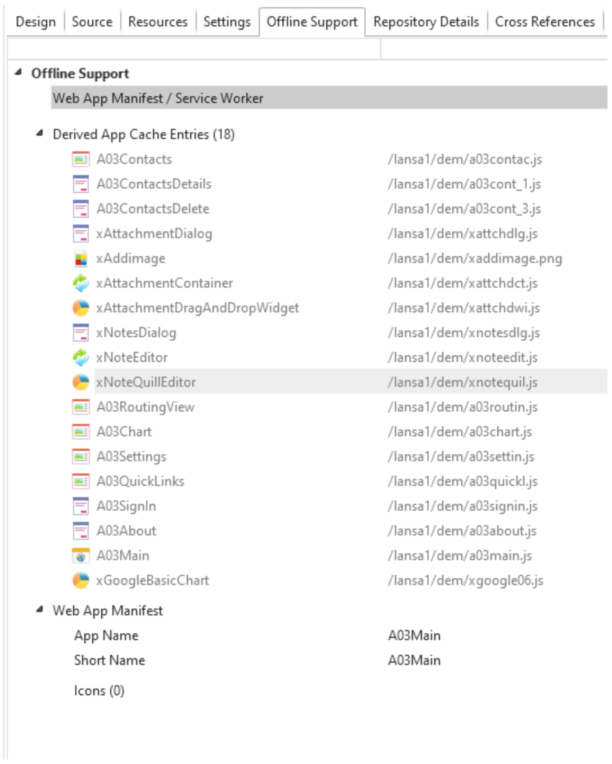


Details

Offline Support: Application Cache

3. Selecting "Service Worker" creates the functionality in the background and enables a developer to drag-and-drop Reusable Parts, Views, and other resources into a cache storage bucket.
    a. Selecting "Web App Manifest/Service Worker" enables full Progressive Web Application functionality i.e., the ability for the application to be installed on a desktop or mobile device without utilizing the app store.

Once either of those selections are made a new view is provided:



Offline Support

Web App Manifest / Service Worker

Derived App Cache Entries (18)

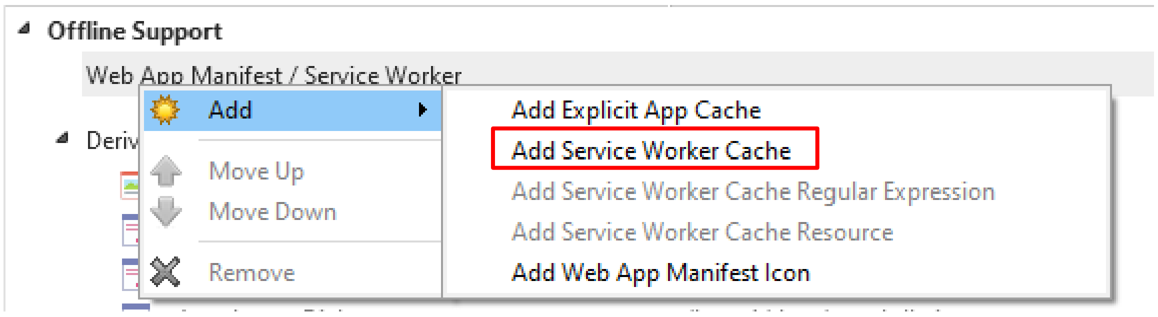
A03Contacts	/lansa1/dem/a03contac.js
A03ContactsDetails	/lansa1/dem/a03cont_1.js
A03ContactsDelete	/lansa1/dem/a03cont_3.js
xAttachmentDialog	/lansa1/dem/xattchdlg.js
xAddimage	/lansa1/dem/xaddimage.png
xAttachmentContainer	/lansa1/dem/xattchdct.js
xAttachmentDragAndDropWidget	/lansa1/dem/xattchdwi.js
xNotesDialog	/lansa1/dem/xnotesdlg.js
xNoteEditor	/lansa1/dem/xnoteedit.js
xNoteQuillEditor	/lansa1/dem/xnotequil.js
A03RoutingView	/lansa1/dem/a03routin.js
A03Chart	/lansa1/dem/a03chart.js
A03Settings	/lansa1/dem/a03settin.js
A03QuickLinks	/lansa1/dem/a03quickl.js
A03SignIn	/lansa1/dem/a03signin.js
A03About	/lansa1/dem/a03about.js
A03Main	/lansa1/dem/a03main.js
xGoogleBasicChart	/lansa1/dem/xgoogle06.js

Web App Manifest

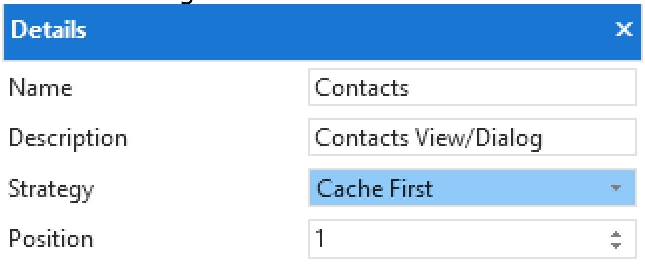
App Name	A03Main
Short Name	A03Main
Icons (0)	

4. Next, right-click the subheading “Web App Manifest/Service Worker” which will display a context menu with additional options.

5. Select “Add Service Worker Cache” to begin nominating objects and resources to be placed in cache storage.


Offline Support
Web App Manifest / Service Worker
Deriv
Add
Move Up
Move Down
Remove
Add Explicit App Cache
Add Service Worker Cache
Add Service Worker Cache Regular Expression
Add Service Worker Cache Resource
Add Web App Manifest Icon

6. After selecting “Add Service Worker Cache” the Details tab will display the following:

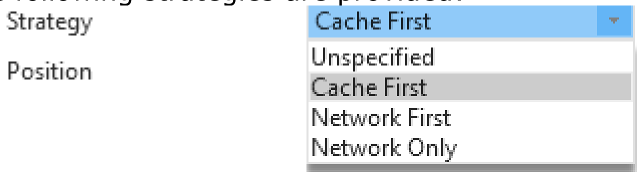

Details

Name	Contacts
Description	Contacts View/Dialog
Strategy	Cache First
Position	1

7. Input the details such as the Name, Description, and Strategy.

8. The name chosen for the cache storage will be viewable in the browser under developer tools.

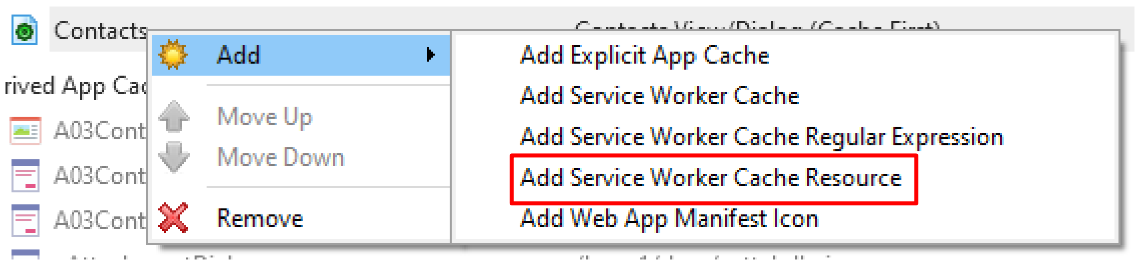
9. The following strategies are provided:



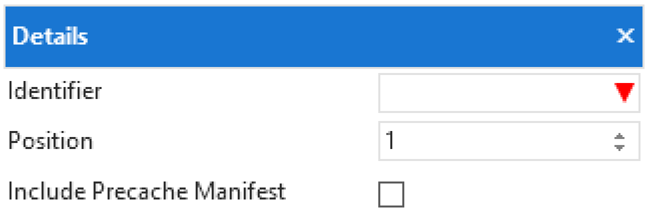
   a) **Cache First:** Signals to the service worker to check the cache first before making a network call. If the resource is not within the cache, then fall back to the network.
   b) **Network First:** Signals to the service worker to allow the call for a resource to be made over the network first, then clone it, and place it within the cache for future use.
   c) **Network Only:** Signals to the service worker that the resource requested should only come from the network. Network Only is a great strategy for when a specific resource is constantly changing, and the end-user needs the most up-to-date information.
   d) **Unspecified:** Selecting unspecified will utilize a Cache-First strategy by default.

10. Now that the important details have been filled in, click the “add” button or right click to bring up the context menu on the name of the cache storage bucket, and select “Add Service Worker Cache Resource.”

In this example, we show a 'Contacts' storage bucket:



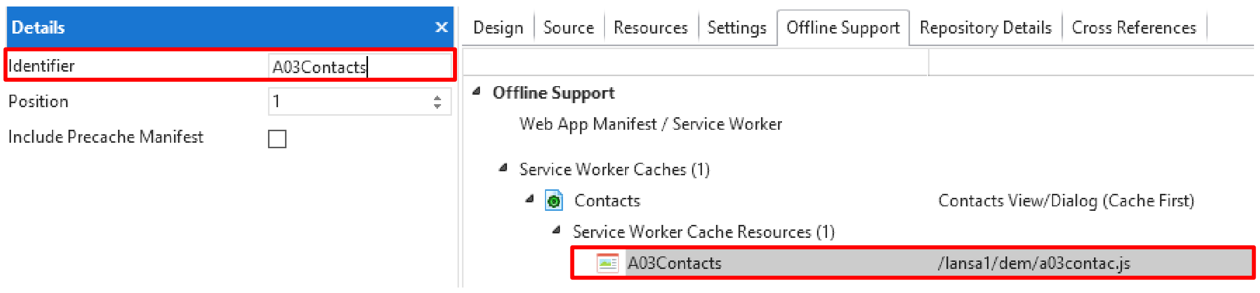
11. Using the "Add Service Worker Cache Resource" option enables developers to specify the identifier of a specific resource such as A03Contacts for the Contacts View.



Identifier	
Position	1
Include Precache Manifest	☐

   a. **Precache Manifest:** Including a resource in the Precache Manifest ensures that the resource is placed within the cache storage on the first load of the application.

Note:
As a developer starts typing in the name of the resource LANSA does a simultaneous lookup to determine which resource the developer needs prior to them completing the name of the identifier.



Identifier	A03Contacts
Position	1
Include Precache Manifest	☐

Offline Support
Web App Manifest / Service Worker
Service Worker Caches (1)
Contacts — Contacts View/Dialog (Cache First)
Service Worker Cache Resources (1)
A03Contacts — /lansa1/dem/a03contac.js

12. If a developer does not off-hand know the specific identifier, the developer can drag a resource from the repository view to the specified Cache bucket name.