Microsoft

PowerApps 캔버스 앱 코딩 표준 및 지침

백서

요약: 이 기술 백서는 엔터프라이즈의 Microsoft PowerApps 제작자를 대상으로 합니다. 개체, 컬렉션 및 변수의 명명 표준과 일관되고 성능이 좋으며 손쉽게 관리할 수 있는 앱 개발을 위한 지침이 담겨 있습니다.

작성자: Todd Baginski, Pat Dunn

기술 기여자: Mehdi Slaoui Andaloussi, Alex Belikov, Casey Burke, Ian Davis, Brian Dang, Rémi Delarboulas, Aniket J. Gaud, Nick Gill, Audrie Gordon, Erik Orum Hansen, Eric Mckinney, Santhosh Sudhakaran, Hubert Sui, Vanessa Welgemoed, Keith Whatling

목차

개요 4

이 백서의 목적 4

이 백서의 범위 4

계속 업데이트되는 문서 5

일반 명명 규칙 5

카멜 표기법 5

파스칼 표기법 5

개체 명명 규칙 5

화면 이름 6

컨트롤 이름 7

데이터 원본 이름 8

코드 명명 규칙 11

변수 이름 11

컬렉션 이름 12

개채 및 코드 구성하기 13

구성에 그룹 사용하기 13

텍스트 서식 지정 기능 13

생성 컨트롤 수 최소화하기 14

코드에 가장 적합한 위치 찾기 14

기타 구성 팁 20

일반 코딩 지침 21

대상 클릭 21

변수 및 컬렉션 21

중첩 22

성능 향상을 위한 최적화 23

OnStart 코드 23

Concurrent 함수 23

위임 가능한 호출 대 위임 불가능한 호출 24

로컬 컬렉션 사용하기 24
SQL 최적화 24
고비용 호출 26
패키지 크기 제한하기 27
앱을 주기적으로 다시 게시하기 28
고급 설정 28
앱 설계 29
부모/자식 관계를 이용한 상대적 스타일링 29
갤러리 29
양식 30
Common Data Service for Apps 31
여러 폼 팩터 31
구성 값 31
숨겨진 구성 화면 생성하기 31
Common Data Service for Apps 구성 값 저장하기 33
사용자 지정 API 사용하기 34
오류 처리/디버깅 34
오류 처리를 위한 토글 컨트롤 34
캔버스 컨트롤을 디버그 패널로 사용하기 35
앱 제작자에게 디버그 컨트롤 표시하기 35
문서화 36
코드 주석 36
문서화 화면 36

개요

Microsoft PowerApps는 높은 생산성을 자랑하는 Microsoft의 애플리케이션 개발 플랫폼입니다. Microsoft는 이 플랫폼을 사용하여 Microsoft Dynamics 365 for Sales, Microsoft Dynamics 365 for Service, Microsoft Dynamics 365 for Field Service, Microsoft Dynamics 365 for Marketing, Microsoft Dynamics 365 for Talent 내의 자사 애플리케이션을 빌드하고 있습니다. 엔터프라이즈 고객도 같은 플랫폼을 사용하여 LOB(기간 업무) 애플리케이션을 맞춤 빌드할 수 있습니다. 조직 내의 개인 사용자 및 팀들도 코드 작성을 적게 하거나 아예 하지 않고도 개인 또는 팀용 생산성 애플리케이션을 빌드할 수 있습니다.

이 백서의 목적

이 백서는 기업 또는 정부 기관에서 PowerApps 앱의 설계, 빌드, 테스트, 배포, 유지 관리를 담당하고 있는 엔터프라이즈 애플리케이션 제작자(개발자)를 대상으로 합니다. 이 백서는 Microsoft PowerApps 팀, Microsoft IT 부서 및 업계 전문가의 공동 작업으로 개발되었습니다. 물론 엔터프라이즈 고객들은 자유롭게 자체적인 표준과 모범 사례를 만들어 갈 수 있습니다. 그러나 이 백서의 지침에 따른다면 다음 영역에서 개발자에게 도움이 되리라고 봅니다.

- 간소화
- 가독성
- 지원 가능성
- 쉬운 배포 및 관리
- 성능
- 접근성

이 백서의 범위

별도의 설명이 없다면 이 백서에 나오는 모든 기능은 2018년 12월 이후로 이용 가능한 것들입니다. 다음 주제는 이 백서의 범위 밖입니다.

- 애플리케이션 빌드를 위한 PowerApps 기본 사항. 이 백서는 독자에게 PowerApps 앱 빌드 방법과 관련하여 전문 지식까지는 아니라도 어느 정도 실무적인 지식이 있다고 가정하고 작성되었습니다. 관련 블로그, 교육 리소스, 커뮤니티 지원은 https://docs.microsoft.com/en-us/powerapps/index를 참고하시길 바랍니다.
- Microsoft Power BI 및 광범위한 Microsoft Power 플랫폼의 다른 부분.
- Microsoft Azure App Service 및 Function App 등 PowerApps 외부의 코드.
- 일반적 거버넌스 및 ALM(애플리케이션 수명 관리).
- 환경 관리. 이 주제에 관해 알아보려면 Administering a PowerApps enterprise deployment(PowerApps 엔터프라이즈 배포 관리) 백서를 참고하세요.

계속 업데이트되는 문서

이 백서는 계속 업데이트되는 문서입니다. Microsoft Power 플랫폼 기능과 업계 표준이 변화함에 따라 이 백서도 함께 변경됩니다.

Microsoft Corporation은 고객의 피드백을 경청함으로써 Power 플랫폼을 끊임없이 진화시켜 여러분이 사용자를 위해 더 나은 앱을 빌드할 수 있도록 돕고 있습니다. 따라서 앞으로 새로운 기능이 적용되어 가장 효율적인 접근 방법이 달라지면서, 지금의 모범 사례가 구식이 될 수 있습니다. 최신 표준과 지침을 알아보기 위해서는 주기적으로 다시 확인해 주시길 바랍니다.

모든 전문가 여러분께서 공동으로 지침과 경험을 공유해 주신 덕분에 이 백서를 작성할 수 있어 ***감사드립니다***. 이제 본격적으로 지침을 소개하겠습니다.

일반 명명 규칙

이 섹션에서는 "카멜 표기법"과 "파스칼 표기법" 명명 규칙을 설명합니다. 이미 해당 용어에 익숙하시다면 이 부분을 건너뛰셔도 됩니다.

카멜 표기법

컨트롤 및 변수는 카멜 표기법으로 표기해야 합니다. 카멜 표기법 명명 규칙은 소문자 접두사로 시작하고, 개체나 변수 이름에 공백을 넣지 않으며, 첫 단어를 제외한 각 단어의 첫 글자는 대문자로 쓰는 것입니다. 예를 들어 텍스트 입력 컨트롤의 경우 txtUserEmailAddress라고 명명될 수 있습니다.

파스칼 표기법

데이터 원본은 파스칼 표기법으로 표기해야 합니다. 파스칼 표기법은 "대문자 카멜 표기법"이라고도 합니다. 공백을 전혀 넣지 않으며 각 단어의 첫 글자를 대문자로 쓰는 것은 카멜 표기법과 비슷하지만, 첫 단어의 첫 글자도 대문자로 쓴다는 점이 다릅니다. PowerApps의 공통 데이터 원본은 Microsoft Office 365 Users 커넥터로서 코드에는 Office365Users로 명명됩니다.

개체 명명 규칙

PowerApps 앱에서 개체를 만들 때, 화면, 컨트롤, 데이터 원본에 대하여 일관된 명명 규칙을 사용하는 것은 중요한 일입니다. 이러한 방법을 사용하면 앱의 유지 관리가 더 쉬워지고, 접근성이 개선되며, 코드에 해당 개체가 참조될 때 더 읽기 쉬워집니다.

참고: 이 백서를 준비하면서 조직마다 명명 규칙에 많은 차이가 있음을 알게 되었습니다. 어느 숙련된 제작자는 수식에 참조되는 경우에만 컨트롤 이름을 변경한다고 합니다. 컨트롤 이름에 붙이기를 선호하는 접두사가 제작자에 따라 다른 경우도 있습니다.

그래도 괜찮습니다! 이 백서의 개체 명명 규칙 섹션은 지침을 제시할 뿐이며 조직들은 자유롭게 자체적인 표준을 개발하면 됩니다. 가장 중요한 핵심은 일관성이 있어야 한다는 점입니다.

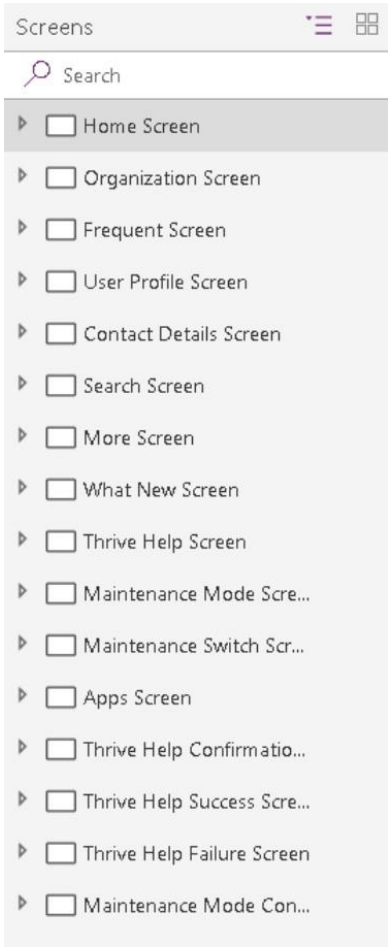
화면 이름

화면 이름은 화면의 목적을 나타냄으로써 PowerAppsStudio의 복잡한 앱들을 더 쉽게 탐색할 수 있도록 해야 합니다.

여기서 간과하기 쉬운 부분은 화면 읽기 프로그램이 화면 이름을 소리 내어 읽는다는 점입니다. 시각적 접근성 요구 사항이 있는 사람을 위한 것입니다. 그러므로 ***화면을 명명할 때는 반드시 쉬운 언어를 사용하고 공백을 넣으며 약어를 사용해서는 안 됩니다***. 또한 *화면 이름을 읽을 때 맥락을 이해할 수 있도록 끝부분에"화면"을 넣을 것을 권장합니다.*

좋은 예는 다음과 같습니다.

- Home Screen
- Thrive Help Screen



안 좋은 예는 다음과 같습니다.

- Home
- LoaderScreen
- EmpProfDetails
- Thrive Help

컨트롤 이름

캔버스의 모든 컨트롤 이름에는 카멜 표기법을 사용해야 합니다. 컨트롤 이름은 세 글자의 형식 설명자로 시작하고 그 뒤에 해당 컨트롤의 목적을 입력해야 합니다. 이러한 방법을 사용하면 컨트롤 유형 파악에 도움이 되고 수식 작성과 검색이 더 쉬워집니다.

좋은 예: lblUserName

다음 표에는 공통적인 컨트롤의 약어가 나와 있습니다.

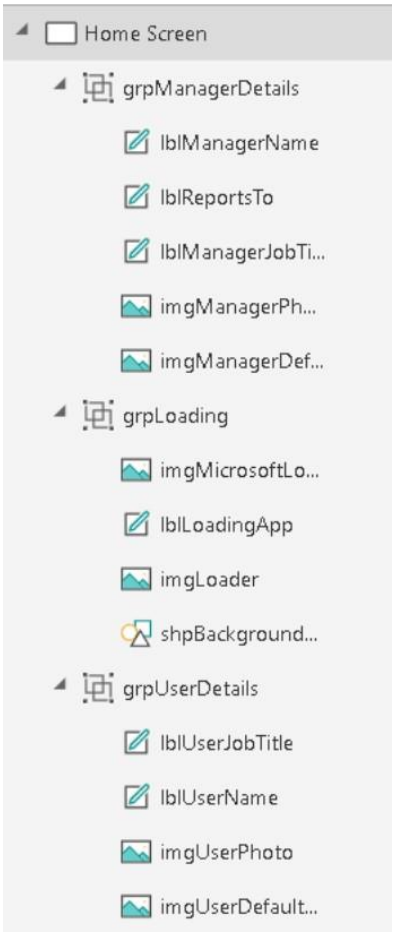
컨트롤 이름	**약어**
button	btn
camera control	cam
canvas	can
card	crd
collection	col
combo box	cmb
dates	dte
drop down	drp
form	frm
gallery	gal
group	gap
header page shape	hdr
html text	htm
icon	ico
image	imge
label	lbl
page section shape	sec
shapes(직사각형, 원 등)	shp
table data	tbl
text input	txt
timer	tim

컨트롤 이름은 애플리케이션 전체에서 고유하게 사용되어야 합니다. 컨트롤 이름을 여러 화면에서 다시 사용하는 경우 끝부분에 짧은 화면 이름을 접미사로 붙여야 합니다. 예를 들어 galBottomNavMenuHS에서 "HS"는 "Home Screen"입니다. 이러한 방법을 사용하면 화면 전체의 수식에서 컨트롤을 더 쉽게 참조할 수 있습니다.

안 좋은 예는 다음과 같습니다.

- zipcode
- Next

다음 이미지에 나타난 것과 같이 컨트롤을 일관성 있게 명명하면 탐색 뷰에서 앱이 훨씬 깔끔해지고 코드 역시 훨씬 깔끔해질 것입니다.



데이터 원본 이름

애플리케이션에 데이터 원본을 추가하면 PowerApps 앱에서 이름을 변경할 수 없습니다. 이름은 원본 커넥터 또는 연결에서 파생된 데이터 엔터티에서 상속됩니다.

다음은 몇 가지 예입니다.

- **원본 커넥터에서 상속된 이름:** 코드에서 Office 365 Users 커넥터의 이름은 Office365Users입니다.
- **연결에서 파생된 데이터 엔터티: Employees**라고 명명된 Microsoft SharePoint 목록이 SharePoint 커넥터에서 반환됩니다. 따라서 코드에서 데이터 원본 이름은 Employees가 됩니다. 동일한 PowerApps 앱에서 ***동일한 SharePoint 커넥터***를 사용해 Contractors라고 명명된 SharePoint 목록에 액세스할 수도 있습니다. 이 경우 코드에서 데이터 원본의 이름은 Contractors가 됩니다.

커넥터 및 연결에 관한 자세한 내용은 PowerApps 캔버스 앱용 커넥터 개요 문서를 참고하세요.

표준 작업 커넥터

LinkedIn처럼 함수를 노출하는 표준 작업 커넥터를 보면 데이터 원본 이름과 그 작업에 파스칼 표기법(즉, UpperUpperUpper)을 사용합니다. 예를 들어 LinkedIn 데이터 원본은 LinkedIn으로 명명되고 그 작업 중에는 ListCompanies가 있습니다.

```
ClearCollect(
    colCompanies,
    LinkedIn.ListCompanies()
)
```

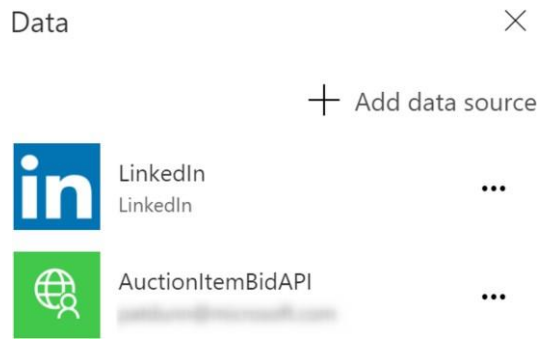
사용자 지정 커넥터

사용자 지정 커넥터는 여러분이 소속된 환경 안에 있는 모든 제작자가 만들 수 있습니다. 이러한 커넥터는 타사 서비스 또는 소속된 조직의 IT 부서가 만든 LOB(기간 업무) API 등의 사용자 지정 API(애플리케이션 프로그래밍 인터페이스)에 연결하는 데 사용됩니다. 데이터 원본 이름 및 그 작업에는 파스칼 표기법이 권장됩니다. 다만 사용자 지정 커넥터 이름과 그것이 PowerApps에 나타나는 방식은 다를 수 있다는 점에 유의하세요.

예를 들어 다음은 **MS Auction Item Bid API**라고 명명된 사용자 지정 커넥터입니다.

MS Auction Item Bid API
MS CSE PowerApps

그러나 이 커넥터로 연결을 만들고 PowerApps 앱에 데이터 원본으로 추가하면, **AuctionItemBidAPI**로 나타납니다.



이유를 알기 위해서는 OpenAPI 파일 안을 보면 됩니다. 파일 안에 보면 제목 특성에 Auction Item Bid API라는 텍스트가 보입니다.

```
"info": {
  "version": "v1",
  "title": "Auction Item Bid API"
},
```

PowerApps는 이 특성 값에서 모든 공백을 제거한 후, 이를 데이터 원본 이름으로 사용합니다. 이러한 특성 값을 AuctionItemBidAPI와 같이 파스칼 표기법으로 바꾼 후에 사용자 지정 연결 이름으로 사용할 것을 권장합니다. 이렇게 하면 혼란이 없습니다. 사용자 지정 커넥터를 만들기 위해 OpenAPI 파일을 가져오기 전에 이 값을 변경하세요.

참고: 기존 OpenAPI 파일을 가져오는 대신 **Create from blank(빈 페이지에서 만들기)** 옵션을 사용하는 경우 PowerApps에 사용자 지정 커넥터 이름을 입력하라는 메시지가 표시됩니다. 이 이름은 사용자 지정 커넥터 이름은 물론 OpenAPI 파일 내의 제목 특성 값으로도 사용됩니다. 다시 말씀드리지만 파스칼 표기법으로 쓴 AuctionItemBidAPI 와 같은 이름이면 됩니다.

Excel DataTables

PowerApps는 Microsoft Excel의 DataTable을 사용하여 Excel 워크시트 데이터에 연결합니다. 데이터 원본으로 Excel 문서를 만들 때는 다음 사항에 유의하세요.

- DataTable에 설명이 포함된 이름을 지정합니다. DataTable에 연결하기 위해 코드를 작성할 때 PowerApps 앱에 해당 이름이 표시됩니다. 워크시트당 하나의 DataTable을 사용합니다.
- DataTable과 워크시트에 같은 이름을 지정합니다.
- DataTable 내에 설명이 포함된 열 이름을 지정합니다.
- 파스칼 표기법을 사용합니다. DataTable 이름의 각 단어는 대문자로 시작해야 합니다(예: EmployeeLeaveRequests).

코드 명명 규칙

PowerApps 앱에 코드가 추가됨에 따라 변수와 컬렉션에 일관된 명명 규칙을 적용하는 일은 점점 더 중요해집니다. 변수를 올바르게 명명했다면 각 변수의 *종류*, *목적*, *범위*를 빠르게 파악할 수 있습니다.

백서를 작성하면서, 조직마다 코드 및 개체 명명 규칙에 많은 차이가 있음을 알게 되었습니다. 예를 들어 어떤 팀은 데이터 종류를 변수의 접두사로 사용했고(예: 문자열임을 나타내기 위한 strUserName), 어떤 팀은 밑줄(_)을 모든 변수의 접두사로 사용하여 IntelliSense 상에서 그룹화되도록 했습니다. 또한 전역 변수와 컨텍스트 변수를 나타내는 방법도 팀마다 차이가 있었습니다.

여기에도 같은 지침이 적용됩니다. *팀에 맞는 패턴을 만들고 일관성 있게 사용합니다.*

변수 이름

- 변수의 기능에 대한 설명을 포함합니다. 변수가 무엇과 바인딩되어 있는지, 어떻게 사용되는지를 생각해 본 후 그에 맞게 명명하세요.
- 전역 변수와 컨텍스트 변수에 다른 접두사를 사용합니다.

현명하게 대처하세요! PowerApps에서는 컨텍스트 변수와 전역 변수가 같은 이름을 공유할 수 있습니다. 이 때문에 혼란이 발생할 수 있는데요, 명확성 연산자를 사용하지 않는 한 수식이 컨텍스트 변수를 기본값으로 적용하기 때문입니다. 다음 규칙에 따름으로써 이런 상황을 방지할 수 있습니다.

- 컨텍스트 변수에는 접두사 loc를 사용합니다.
- 전역 변수에는 접두사 gbl을 사용합니다.
- 접두사 뒤의 이름은 해당 변수의 의도/목적을 나타냅니다. 여러 단어를 사용할 수 있으며, 각 단어의 첫 글자를 대문자로 썼다면 특수 문자(공백 또는 밑줄 등)로 구분할 필요가 없습니다.
- 카멜 표기법을 사용합니다. 모두 소문자인 접두사로 변수 이름을 시작한 후, 뒤에 나오는 각 단어의 첫 글자는 대문자로 습니다(즉, lowerUppperUpper).

좋은 예는 다음과 같습니다.

- **전역 변수:** gblFocusedBorderColor
- **컨텍스트 변수:** locSuccessMessage

안 좋은 예는 다음과 같습니다.

- dSub
- rstFlds

- hideNxtBtn
- ttlOppCt
- cFV
- cQId

EID와 같이 짧고 이해하기 어려운 변수 이름은 피하시고, 대신 EmployeeId와 같은 이름을 사용합니다.

참고: 앱에 변수가 많은 경우, 수식 입력줄에 접두사만 입력하면 사용 가능한 변수 목록이 표시됩니다. 이 문서의 지침에 따라 변수를 명명한다면 앱을 개발할 때 수식 입력줄에서 아주 쉽게 변수를 찾을 수 있습니다. 이러한 방법은 궁극적으로 더 빠른 앱 개발로 이어집니다.

컬렉션 이름

- 컬렉션의 내용에 대한 설명을 포함합니다. 컬렉션에 무엇이 담겨 있는지 및/또는 그것이 어떻게 사용되는지를 생각해 본 후 그에 맞게 명명하세요.
- 컬렉션에는 접두사 col을 사용합니다.
- 접두사 뒤의 이름은 해당 컬렉션의 의도 또는 목적을 나타냅니다. 여러 단어를 사용할 수 있으며, 각 단어의 첫 글자를 대문자로 썼다면 공백 또는 밑줄 등으로 구분할 필요가 없습니다.
- 카멜 표기법을 사용합니다. 소문자로 된 접두사 col 로 컬렉션 이름을 시작한 후, 다음에 나오는 각 단어의 첫 글자는 대문자로 습니다(즉, colUpperUpper).

좋은 예는 다음과 같습니다.

- colMenuItems
- colThriveApps

안 좋은 예는 다음과 같습니다.

- orderscoll
- tempCollection

참고: 앱에 컬렉션이 많은 경우, 수식 입력줄에 접두사만 입력하면 사용 가능한 컬렉션 목록이 표시됩니다. 변수의 경우처럼 컬렉션도 이 문서의 지침에 따라 명명한다면 앱을 개발할 때 수식 입력줄에서 아주 쉽게 찾을 수 있습니다. 이러한 방법은 궁극적으로 더 빠른 앱 개발로 이어집니다.

개체 및 코드 구성하기

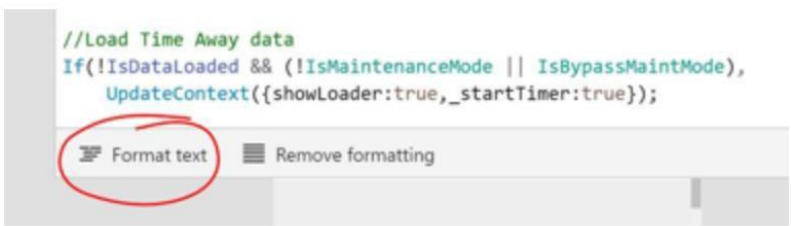
구성에 그룹 사용하기

화면에 있는 모든 컨트롤은 그룹에 속해야 합니다. 손쉽게 컨트롤의 목적을 파악하고, 화면 안에서와 화면 사이로 컨트롤을 이동시키고, 축소하여 뷰를 간소화하기 위해서입니다. 갤러리, 양식, 캔버스 컨트롤은 이미 자체 그룹을 형성하고 있지만, 선택적으로 다른 그룹에도 속하게 하여 구성을 향상할 수 있습니다.

선택적으로 실험 단계의 기능인 [향상된 그룹 컨트롤]을 이용할 수 있습니다. 이 기능을 사용하면 그룹 중첩, 그룹 수준 설정, 키보드 탐색 등이 가능합니다.

텍스트 서식 지정 기능

수식이 복잡해질수록 가독성과 유지 관리가 어려워질 수 있습니다. 여러 줄 함수가 포함된 긴 코드 블록은 읽기가 매우 어려울 수 있습니다. 텍스트 **서식 지정** 기능은 줄 바꿈과 들여쓰기를 하여 수식을 읽기 쉽게 만들어 줍니다. 코드 주석의 경우, 클라이언트에 다운로드된 앱 패키지에서 추가 공백이 제거됩니다. 따라서 앱을 게시하기 전에 **서식 제거** 기능을 사용할 필요가 없습니다.



생성 컨트롤 수 최소화하기

복잡성을 최소화하려면 앱의 컨트롤 수를 최대한 제한해야 합니다. 예를 들어 Visible 속성 설정이 서로 다른 네 개의 이미지 컨트롤을 겹쳐서 사용하는 대신, Image 속성에 서로 다른 이미지를 표시하는 로직이 포함된 하나의 이미지를 사용합니다.

코드에 가장 적합한 위치 찾기

PowerApps 앱이 복잡해질수록 애플리케이션을 디버그할 시기가 되었을 때 코드를 찾는 일이 더 어려워질 수 있습니다. 일관된 패턴이 있다면 이러한 어려움이 줄어듭니다. 이 섹션이 자세한 내용을 모두 담고 있지는 않지만, 코드에 가장 적합한 위치를 찾는 일에 어느 정도의 지침을 제시합니다.

일반적인 지침은 향후에 더 쉽게 찾을 수 있도록 코드를 최대한 "최상위"로 이동하는 것입니다. 일부 제작자는 코드를 OnStart 속성에 입력하는 것을 선호합니다. 이 방법도 괜찮지만 OnStart 속성의 한계와 앱 성능에 미칠 수 있는 인지된 영향을 이해하고 있어야 합니다. 다른 제작자들은 코드를 OnVisible 속성에 입력하는 것을 선호합니다. 코드를 찾기 쉽고 화면이 표시될 때마다 코드가 안정적으로 실행되기 때문입니다.

코드 캡슐화

가능하면 코드를 여러 화면에 분산시키지 않고 모든 코드가 한 화면에 있도록 해야 합니다. 예를 들어 어떤 제작자가 조직의 계층 구조를 갤러리 형태로 보여 주는 인물 브라우저 앱을 만들었다고 합시다. 사용자가 이름을 클릭하면 앱은 새 화면으로 이동해 직원 프로필을 보여 줍니다. 그런데 이 경우 제작자는 갤러리의 OnSelect 속성에 프로필을 로드하는 로직을 입력하지 않았습니다. 대신 앱은 다음 화면에 필요한 모든 변수를 Navigate 함수의 컨텍스트 변수로 전달합니다. **사용자 프로필** 화면이 사용자 프로필을 로드하기 위한 모든 작업을 수행합니다.

다음은 이 예시에 나오는 갤러리의 OnSelect 속성 안에 있는 Navigate 함수입니다.

```
Navigate(
    'User Profile Screen',
    Cover,
    {
        locSelectedEmployeeID: ThisItem.id,
        locSelectedEmployeeName: ThisItem.displayName,
        locSelectedEmployeeJobTitle: ThisItem.jobTitle,
        locProfileFetchComplete: false,
        locDirectReportsFetchComplete: false,
        locMgrHierarchyFetchComplete: false,
        locPeersFetchComplete:false,
        locSelectedTab:"Profile"
    }
)
```

그런 다음, **사용자 프로필** 화면의 OnVisible 속성에서, 이전 화면에서 수신한 사용자 ID를 사용하여 Office365Users.UserProfileV2를 호출합니다. 이후 코드는 전달된 다른 컨텍스트 변수를 사용합니다.

```
ClearCollect(colUserProfile, Office365Users.UserProfileV2(locSelectedEmployeeID))
```

참고: 이전 예시에서는 이전 화면의 Selected 속성을 다음 화면에서 참조하도록 하는 대신 ThisItem 값을 컨텍스트 변수로 전달했습니다. 의도적으로 이런 방법을 사용한 것입니다. 이 앱에는 갤러리가 포함된 다른 화면에서 **사용자 프로필** 화면으로 이어지는 여러 경로가 있기 때문입니다. 이제 화면이 캡슐화되어 이 앱과 다른 앱에서 쉽게 재사용할 수 있습니다.

OnStart 속성

일반적으로 OnStart 속성에 입력하는 코드는 제한하는 것이 좋습니다. 디버그하기 어렵기 때문입니다. 여기서 디버그하려면 저장하고 닫은 후에 PowerApps Studio에서 PowerApps 앱을 다시 열어 코드를 다시 실행해야 합니다. 이 속성에서는 컨텍스트 변수를 만들 수 없습니다. 화면이 표시되기 전에 한 번만 실행되는 Application.OnStart라고 생각하면 됩니다.

권장되는 OnStart의 사용법은 다음과 같습니다.

- **화면 라우팅:** OnVisible 속성과 달리 OnStart 속성에서는 Navigate 함수를 사용할 수 있습니다. 따라서 라우팅 결정을 하기에 편리할 수 있습니다. 예를 들어 mode라는 이름의 매개 변수를 평가하여 어느 화면을 표시할지 결정할 수 있습니다.

```
Navigate(
    Switch(
        Param("mode"),
        "new",
        'New Order Screen',
        "edit",
        'Edit Order Screen',
        "history",
        'Order History Screen',
        'Dashboard Screen'
    ),
    ScreenTransition.None
)
```

- **가장 또는 디버그 권한:** OnStart 속성 안에 코드를 작성하여 현재 사용자가 전자 메일 주소 목록에 있는지 확인하고, 목록에 있다면 숨겨진 화면과 텍스트 입력 컨트롤을 표시해 주는 디버그 모드를 켤 수 있습니다.

```
Set(
    gblAllowDebug,
    If(
        User().Email in [
            "bob@contoso.com",
            "susan@contoso.com",
            "rajesh@contoso.com"
        ],
        true,
        false
    )
)
```

참고: AAD(Azure Active Directory) 그룹 멤버십을 확인하여 보안 설정을 앱에 적용할 수도 있습니다.

- **정적 전역 변수:** OnStart 속성을 사용하여 오류 메시지 컬렉션을 만들거나 컨트롤 색, 테두리 두께 등의 전역 스타일 변수를 설정합니다. 다른 방법을 알아보려면 이 백서 뒷부분에 나오는 숨겨진 구성 화면 만들기 섹션을 참고하세요.
- **"한 번 실행" 코드**: OnStart 속성에 입력한 코드는 그 정의에 따라, 앱이 시작되는 동안에 그러나 첫 화면이 표시되기 전에 단 한 번만 실행됩니다. 반면 OnVisible 속성에 있는 코드는 사용자가 해당 화면으로 갈 때마다 실행됩니다. 따라서 코드를 한 번만 실행해야 한다면 OnStart 속성에 입력할 것을 고려해야 합니다.
- **빠르게 실행되는 코드:** OnStart 속성에 대한 구체적 지침은 이 백서의 뒷부분에 나오는 성능 향상을 위한 최적화 섹션을 참고하세요.

OnStart 및 OnVisible 속성에 대한 자세한 정보는 Todd Baginski의 PowerApps OnStartand OnVisible Development Tricks(PowerApps의 OnStart 및 OnVisible 개발 팁) 동영상을 참고하세요.

OnVisible 속성

OnVisible 속성는 사용자가 화면에 갈 때마다 실행되어야 할 코드를 입력하는 곳입니다. 이 속성에 코드를 입력할 때 주의하세요. 가능하면 PowerApps 앱의 첫 화면에서는 OnVisible 속성에 로직을 입력하지 않는 것이 좋습니다. 대신 컨트롤 속성 안에 인라인 표현식을 사용하도록 합니다.

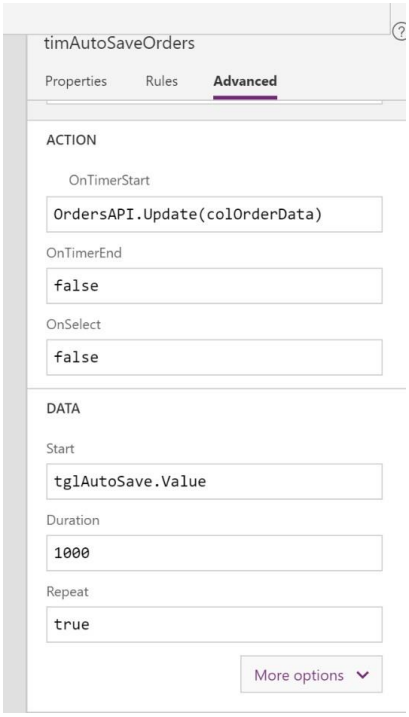
OnVisible 속성은 전역 변수 또는 컨텍스트 변수를 입력하기에 좋습니다. 그러나 이러한 변수를 설정하기 위한 호출에 유의해야 합니다. Office365Users.Profile에 대한 호출이나 컨트롤에서 재사용할 정적 색을 설정하기 위한 호출 등의 빠른 호출은 괜찮습니다. 그러나 실행이 오래 걸리는 복잡한 로직과 코드는 피해야 합니다.

OnVisible 속성과 연관된 성능 문제를 자세히 알아보려면 이 백서의 뒷부분에 나오는 고비용 호출 섹션을 참고하세요.

OnTimerStart 속성

타이머는 이벤트 기반 코드 실행에 흥미로운 가능성을 제시합니다. 제작자는 보통 타이머 컨트롤을 숨기고 Start 속성이 부울 변수나 컨트롤 상태를 감시하도록 합니다.

예를 들어 사용자가 자동 저장 기능을 켜고 끌 수 있는 양식을 만들기 위해 토글 컨트롤을 tglAutoSave라는 이름으로 만들 수 있습니다. 그러면 화면에 있는 타이머의 Start 속성이 tglAutoSave.Value로 설정되고 OnTimerStart 속성에 있는 코드가 데이터를 저장할 수 있습니다.



OnTimerStart 속성에는 ClearCollect 함수를 사용한 코드를 입력하여, 지정된 새로 고침 간격에 따라 데이터를 다시 로드하도록 할 수도 있습니다.

OnTimerStart 속성은 Navigate 함수도 지원합니다. 이 함수를 이용하여 특정 조건이 충족되었을 때 다른 화면으로 이동하도록 할 수 있습니다. 예를 들어 로더 화면에서 모든 데이터가 로드되면 부울 컨텍스트 변수가 설정되고, 타이머가 데이터 표시 화면으로 이동하도록 할 수 있습니다. 또는 이 속성을 이용하여 일정 시간 동안 활동이 없으면 "세션 타임아웃" 메시지 화면으로 이동하게 할 수 있습니다.

이러한 방식에는 두 가지 유의 사항이 따릅니다.

- PowerApps Studio에서 앱을 편집할 때는 타이머가 실행되지 않습니다. AutoStart가 true로 설정되어 있거나 Start 속성에서 표현식이 true로 평가되어도 OnTimerStart 속성은 트리거되지 않습니다. 그러나 미리 보기 모드(F5)로 전환하면 트리거됩니다.
- Navigate 함수가 실행되기 전에, 화면에서 추가 코드가 실행될 때까지 충분한 지연이 발생할 수 있습니다.
  예를 들어 로더 화면에 타이머 컨트롤이 있다면, 컨트롤의 Start 속성을 부울 컨텍스트 변수인 locRedirect로 설정하고 다음의 탐색 코드를 OnTimerStart 속성에 입력합니다.

```
If(
    locIsError,
    Navigate(
        'Error Screen',
        None,
        {locStatusMessage: locStatusMessage}
    ),
    Navigate(
        'Confirmation Screen',
        None,
        {locStatusMessage: locStatusMessage}
    )
)
```

  로더 화면의 OnVisible 속성은 직원 ID를 검색하고 ID가 숫자가 아닌 경우 locRedirect를 false로 설정합니다 (숫자가 아닌 직원 ID는 오류 조건이기 때문입니다).

```
//Get the employee ID from PeopleData
UpdateContext({locUserID: Text(PeopleData.PersonnelNumber)});
If(
    !IsNumeric(PeopleData.PersonnelNumber),
    //Couldn't get personnel number from basic profile call. error out
    UpdateContext(
        {
            locIsError: true,
            locStatusMessage: "Unable to retrieve user information",
            locRedirect: true //this will cause our redirect timer to fire
        }
    )
);
```

locRedirect가 true로 설정되면 타이머 컨트롤의 OnStart 코드가 실행되지만 약간의 지연이 발생하며, 그동안 OnVisible 속성은 계속 실행됩니다. 따라서 다음 몇 줄의 코드에 대하여 오류 검사를 추가로 수행합니다.

```
//there is a small delay when a timer fires. Change the status message only if there is no error
If(
    !locIsError,
    UpdateContext({locStatusMessage: "Authorizing User..."})
);
//Get token
UpdateContext(
    {
        locSuccessFactorToken: LDService.GetSfToken(
            {
                scope: {
```

OnSelect 속성

컨트롤의 OnSelect 속성에 입력된 코드는 개체를 선택할 때마다 실행됩니다. 개체의 선택은 버튼을 클릭하거나 텍스트 입력 컨트롤을 선택하는 등, 사용자의 상호 작용을 통해 이루어집니다. 여기서 실행되는 코드는 양식 데이터의 유효성을 검사하여 유효성 검사 메시지나 힌트 텍스트를 표시할 수 있습니다. 또는 데이터 원본을 읽어 들이고 데이터 원본에 기록할 수 있습니다.

참고: OnSelect 속성에는 장기 실행 코드를 입력하지 않는 것이 좋습니다. 장기 실행 코드는애플리케이션이 응답하지 않는 것 같은 인상을 주기 때문입니다. 자세한 정보는 이 백서의 뒷부분에 나오는 성능 향상을 위한 최적화와 고비용 호출 섹션을 참고하세요. 또한 느리다는 인식을 방지하는 데 도움이 되는 로드 중 표시 또는 상태 메시지도 고려할 수 있습니다.

OnSelect 속성에 입력된 코드는 Select 함수를 사용하여 컨트롤이 선택되었을 때도 실행됩니다. 로더 화면을 애플리케이션의 첫 화면으로 삼는 것은 유용한 방식입니다. 레이블 컨트롤은 "앱 데이터 로드 중"과 같은 메시지를 표시할 수 있습니다. 이 레이블 컨트롤의 OnSelect 속성은 데이터 원본을 호출하고 변수를 초기화한 후 애플리케이션의 홈 화면으로 이동할 수 있습니다. 그런 다음, 애플리케이션의 OnStart 속성에서 Select 함수를 호출하여 레이블 컨트롤을 선택합니다.

초기화 코드는 OnStart 또는 OnVisible 속성에도 있을 수 있지만 이러한 방법에는 다음과 같은 이점이 있습니다.

- OnVisible 코드는 탐색을 허용하지 않습니다. 따라서 타이머 등의 컨트롤에 탐색 코드를 추가해야 합니다.
- OnStart 코드는 시작 화면이 길어지게 합니다. 또는 **비차단 OnStart 규칙 사용** 미리 보기 기능이 켜져 있는 경우, 예기치 않은 결과를 초래할 수 있습니다.

- 레이블 컨트롤이 프로그래밍 방식으로 선택된 경우, 시각 장애가 있는 사용자는 데이터가 로드됨에 따라 화면 읽기 프로그램이 레이블 컨트롤의 텍스트 읽는 소리를 듣게 됩니다. 이런 방식으로 하면 사용자는 "화면 로드 중", "앱 데이터 로드 중", "홈 화면" 등의 소리를 들으며 좋은 환경을 누릴 수 있습니다.

주의: OnVisible 속성의 Select 함수를 사용하여 컨트롤을 선택한 다음, 이어서 해당 컨트롤이 Navigate 함수를 사용하여 다른 화면으로 이동하는 경우, 화면 편집이 불가능할 수 있습니다. 이러한 상황을 방지하기 위해서는 앱의 숨겨진 설정 화면에서 토글 컨트롤을 사용합니다. 그리고 Navigate 함수를 호출하기 전에 OnSelect 속성에서 이 토글의 상태를 확인합니다.

기타 구성 팁

- 초기 문 뒤에 If를 명시적으로 사용하여 보조 논리 테스트를 "중첩"하지 않습니다.

```
If(
    One = 1,
        UpdateContext({Nothing: false}),
    If(One = 2,
        UpdateContext({Nothing: true}),
    If(One = 3,
        UpdateContext({All: true})
)))
```

- 보조 논리 테스트를 작성하려면 일반적인 논리 테스트를 작성하되 명시적으로 If를 사용하지 않습니다.

```
If(One = 1,
      UpdateContext({Nothing: false}),
   One = 2,
      UpdateContext({Nothing: true}),
   One = 3,
      UpdateContext({All: true})
)
```

- 가능하면 긴 표현식은 피합니다.
- 수동으로 코드의 서식을 지정하려면 다음 지침에 따릅니다.
    - 각각의 세미콜론은 줄 바꿈을 나타내도록 합니다.

```
ClearCollect(AwesomeCollection, {AwesomeStuff: "Awesome"});
UpdateContext({TemplatesGood: true});
Set(GlobalVariable, "On")
```

- 긴 한 줄 수식의 경우 괄호, 쉼표, 콜론 앞뒤 등 적절한 위치에 줄 바꿈을 삽입합니다.

일반 코딩 지침

대상 클릭

어떤 컨트롤 그룹을 클릭했을 때 작업이 수행되어야 하는 경우, 세 가지 방법을 사용할 수 습니다.

- 가장 간단한 방법은 컨트롤을 그룹화한 다음, 클릭 이벤트를 그룹의 OnSelect 속성에 할당하는 것입니다.
- 그룹의 컨트롤 중 하나(가장 중요한 컨트롤)에 코드를 입력한 다음, Select(*controlWithLogic*)를 그룹 내 다른 모든 컨트롤의 OnSelect 속성에 추가합니다. 첫 번째 방법의 경우 추가적인 컨트롤이 필요하지 않으며 편집기에서 쉽게 컨트롤을 선택할 수 있습니다.
- 그룹 위에 투명한 직사각형을 배치하고 직사각형의 OnSelect 속성을 사용합니다.

여기서는 세 번째 방법을 권장합니다. 그룹의 컨트롤이 변경되더라도 크게 영향을 받지 않기 때문입니다. 이 방법을 사용하면 제작자는 클릭 가능한 영역의 모양에 대해 유연성을 확보할 수 있습니다. 직사각형 내부의 컨트롤은 화면에서 직접 선택하기가 더 어렵지만 편집기 왼쪽에 있는 **Screens(화면)** 창에서 개별적으로 선택할 수 있습니다.

이 방법에 관해 자세히 알아보려면 Todd Baginski의 HOW TO: Use Transparent Rectangles Effectively In a PowerApp(투명 직사각형 효과적으로 사용하기)을 참고하세요.

변수 및 컬렉션

컨텍스트 변수

컨텍스트 변수 사용을 제한합니다. 절대적으로 필요한 경우에만 사용하도록 노력합니다.

컨텍스트 변수와 전역 변수를 언제 사용해야 하는지 파악합니다. 모든 화면에서 사용할 수 있어야 하는 경우 전역 변수를 사용합니다. 변수의 범위를 하나의 화면으로 제한하려면 컨텍스트 변수를 사용합니다.

전역 변수를 사용하는 것이 더 적절하고 디버그하기도 훨씬 쉬울 수 있는 경우에는 화면 간에 컨텍스트 변수를 전달하지 않는 것이 좋습니다.

필요한 모든 컨텍스트 변수를 한 번의 UpdateContext 호출로 업데이트하도록 합니다. 이렇게 하면 코드를 더 효율적이고 읽기 쉽게 만들 수 있습니다.

예를 들어 다음의 호출을 사용하여 여러 컨텍스트 변수를 업데이트할 수 있습니다.

```
UpdateContext({All: true, Nothing: false, One: 1, Two: "two"})
```

다음과 같이 개별적인 호출을 사용하지 않습니다.

```
UpdateContext({All: true});
UpdateContext({Nothing: false});
UpdateContext({One: 1});
UpdateContext({Two: "two"})
```

전역 변수

하나의 변수만 사용할 수 있는데 여러 변수를 사용하지 않습니다. 여러 변수의 예는 다음과 같습니다.

```
Set(SelectedMeetingId,First(Filter(AllFutureMeetings,isCurrent = true)).Id);
Set(SelectedMeetingName,First(Filter(AllFutureMeetings,isCurrent = true)).Subject);
Set(SelectedMeetingStartTime,First(Filter(AllFutureMeetings,isCurrent = true)).Start);
Set(SelectedMeetingEndTime,First(Filter(AllFutureMeetings,isCurrent = true)).End);
Set(SelectedMeetingHours,DateDiff(SelectedMeetingStartTime,SelectedMeetingEndTime,Hours));
```

대신 다음과 같이 하나의 변수만 사용할 수 있습니다.

```
Set(SelectedMeeting, ThisItem);

SelectedMeeting.Id;
SelectedMeeting.Subject;
SelectedMeeting.Start;
SelectedMeeting.End;
SelectedMeeting.Start;
DateDiff(SelectedMeeting.Start, SelectedMeeting.End, Hours)
```

컬렉션

컬렉션 사용을 제한합니다. 절대적으로 필요한 경우에만 사용하도록 노력합니다.

Clear;Collect 대신 ClearCollect를 사용합니다.

```
//Use this pattern
ClearCollect( colErrors, { Text: gblErrorText, Code: gblErrorCode } );

//Not this pattern
Clear(colErrors);
Collect( colErrors, { Text: gblErrorText, Code: gblErrorCode } )
```

로컬 컬렉션의 레코드 수를 계산하려면 Count(Filter()) 대신 CountIf를 사용합니다.

중첩

불필요한 DataCard와 캔버스는 특히 갤러리가 중첩된 경우 더욱 사용하지 않도록 합니다. ForAll 함수와 같은 다른 연산자에서도 중첩을 피합니다.

```
ClearCollect(FollowUpMeetingAttendees,ForAll(ForAll(Distinct(AttendeesList,EmailAddress.Address),LookUp(Attend
```

참고: 이 문서의 이전 버전에서는 중첩 갤러리가 더 이상 사용되지 않을 것이라고 명시했습니다. 이는 잘못된 정보이며 이 점에 대해 사과드립니다.

성능 향상을 위한 최적화

OnStart 코드

OnStart 속성은 앱 초기화에 필요한 일회성 호출에 매우 유용합니다. 그래서 데이터 초기화 호출도 이 속성에 입력하고 싶을 수 있습니다. 그러나 OnStart 코드가 실행되는 동안 사용자는 계속해서 앱 시작 화면과 "데이터 가져오는 중"이라는 메시지를 보며 로드 시간을 더 길게 인식하게 됩니다.

더 나은 사용자 환경을 위해서는 홈 화면에 이중 대시(--)와 같은 데이터 자리 표시자 를 표시하는 것이 좋습니다. 그런 다음, 검색된 데이터로 자리 표시자를 채웁니다. 이렇게 하면 사용자가 홈 화면에서 콘텐츠를 읽거나 데이터에 의존하지 않는 컨트롤과 상호 작용할 수 있습니다. 예를 들어 사용자는 **정보** 화면을 열 수 있습니다.

Concurrent 함수

PowerApps는 모듈에서 데이터 원본 호출을 위에서 아래로 실행합니다. 여러 호출이 있다면 이러한 선형 실행은 앱 성능에 부정적인 영향을 줄 수 있습니다. 지금까지 이 문제에 대한 한 가지 해결 방법은 타이머 컨트롤을 사용하여 데이터 호출을 동시에 실행하는 것이었습니다. 그러나 이 방법은 특히 일부 타이머가 다른 타이머에 의존하는 경우 더욱 유지 관리와 디버그하기가 어렵습니다.

Concurrent 함수를 사용하면 타이머 컨트롤로 여러 데이터 호출을 동시에 수행할 필요가 없습니다. 다음 코드 조각은 앱 타이머 컨트롤의 OnTimerStart 속성에 있던 여러 API 호출을 대체합니다. 이 방법을 사용하면 유지 관리가 훨씬 쉽습니다.

```
    )
);
Navigate('Home Screen',Fade);

Concurrent(
    //Stores Manager details in CurrentManagerHierarchy collection
    ClearCollect(
        colCurrentManagerHierarchy,
        PeopleApi.PeopleGetMyManagerHierarchy({includePhoto: false})
    );
    //Stores peers of the Manager in CurrentPeers collection
    ClearCollect(
        colCurrentPeers,
        PeopleApi.PeopleGetUserDirectReports(
            Last(colCurrentManagerHierarchy).UserPrincipalName,
            {includePhoto: false}
        )
    );
    //Removes the manager from the CurrentPeers collection
    Remove(
        colCurrentPeers,
        Filter(
            colCurrentPeers,
            UserPrincipalName = locCurrentUser
        )
    ),
    //Collects direct reports in CurrentDirectReports collection
```

Format text Remove formatting

이러한 호출을 실행하려면 OnVisible 속성에 입력하면 됩니다. 이 방법이 너무 복잡해졌다면 타이머 컨트롤에 호출을 입력하고 타이머의 Start 속성에서 참조하는 변수를 OnVisible 속성 또는 숨겨진 컨트롤의 OnSelect 속성에서 설정할 수 있습니다. 또한 타이머를 다른 컨트롤과 결합하여 OnVisible 속성의 코드가 실행되는 동안 로드 중 메시지를 표시할 수 있습니다. 이러한 방법을 통해 사용자에게 앱이 작동되고 있음을 알릴 수 있습니다. 자세한 내용은 이 백서의 앞부분에 나오는코드에 가장 적합한 위치 찾기 섹션을 참고하세요.

참고: 코드를 더 쉽게 유지 관리하기 위해서는 OnVisible 속성을 사용하는 것이 좋습니다. 그러나OnVisible 속성을 사용하면 Navigate 함수를 사용할 수 없습니다.

Concurrent 함수의 실제 사용 사례를 보려면 Todd Baginski의 동영상 HOW TO: Use the Concurrent Function To Make Your PowerApps Perform Better(Concurrent 함수를 사용하여 PowerApps 성능 향상하기)를 참고하세요.

위임 가능한 호출 대 위임 불가능한 호출

데이터 원본을 호출할 때는 위임 가능한 함수도 있지만 그렇지 않은 함수도 있다는 점에 유의해야 합니다. 위임 가능한 함수는 서버에서 평가할 수 있으며 성능이 더 좋습니다. 위임 불가능한 함수는 데이터를 클라이언트에 다운로드하여 로컬로 평가해야 합니다. 이러한 프로세스는 위임 가능한 호출에 비해 더 느리고 데이터 집약적입니다.

자세한 내용은 캔버스 앱의 위임 이해 문서를 참고하세요.

로컬 컬렉션 사용하기

소규모 데이터 세트의 경우 특히 잦은 액세스가 문제일 때 데이터 세트를 먼저 로컬 컬렉션에 로드하는 것이 좋습니다. 그런 다음, 해당 컬렉션에서 함수를 수행하거나 해당 컬렉션에 컨트롤을 바인딩하면 됩니다. 이 방법은 위임 불가능한 호출을 자주 하는 경우에 특히 유익합니다. 다만 데이터를 검색해야 하고 반환되는 레코드 수에 제한이 있기 때문에 초기에는 성능에 영향을 미칠 수 있다는 점에 유의해야 합니다. 자세한 내용은 Slaoui Andaloussi의 훌륭한 블로그 게시물 Performance considerations with PowerApps(PowerApps의 성능 고려 사항)를 참고하세요.

SQL 최적화

소속된 조직에서 데이터 백 엔드용으로 Microsoft Azure SQL Database를 사용하여 풍부한 관리 기능과 상호 운용성을 활용하고 있을지도 모르겠습니다. 그러나 설계가 제대로 구현되지 않으면 동시성을 처리할 수 없으므로 DTU(데이터 트랜잭션 단위) 크기를 늘려야 할 수 있으며, 따라서 비용도 증가할 수 있습니다.

예를 들어 Microsoft IT 부서는 1,700명이 참석한 내부 컨퍼런스를 위해 Thrive Conference 앱을 개발했습니다. 백 엔드는 SQL 데이터베이스 인스턴스 100DTU였습니다. 성능 테스트를 할 때

Microsoft는 운영 센터 직원 120명에게 동시에 앱을 열어달라고 요청했습니다. 앱이 응답을 멈췄습니다. 네트워크 추적 결과 PowerApps 연결 개체에서 HTTP 500 오류가 발생한 것으로 나타났습니다. SQL 로그에 따르면 서버가 완전 가동 중이며 호출이 시간 초과되고 있었습니다.

컨퍼런스 전까지 앱을 다시 개발할 시간은 없었기 때문에 Microsoft IT 부서에서는 규모를 4,000DTU로 늘려 동시성 요구 사항을 충족하도록 했습니다. 따라서 원래 예산이 책정된 100DTU 서버보다 비용이 훨씬 많이 들게 되었습니다. 그 이후로 Microsoft IT 부서는 여기 설명한 방식으로 설계를 최적화했습니다. 이제 100DTU 서버로도 충분히 부하를 처리할 수 있으며 SQL 호출도 훨씬 빨라졌습니다.

SQL용 위임 가능한 함수

이전 섹션에서 위임에 관한 개요를 살펴보셨다면 데이터 소스 및 지원되는 위임 목록을 참고하여 최상위 함수와 Filter 및 Lookup 함수의 조건자 가운데 지원되는 것이 무엇인지 알아보세요. 이 정보는 모바일 디바이스에서 PowerApps 앱의 성능을 크게 변화시킵니다. 데이터 세트 전체를 클라이언트에 다운로드하여 평가할 필요가 없기 때문입니다.

표 대신 뷰 사용하기

데이터를 가져오기 위해 표 사이를 이동하는 대신 조인이 수행된 뷰를 노출합니다. 표를 올바르게 인덱싱했다면 이 방법은 매우 빠를 것이며, 위임 가능한 함수를 사용하여 결과를 제한한다면 훨씬 더 빠를 것입니다.

성능을 위해 흐름을 통한 저장 프로시저 사용하기

Microsoft SQL Server를 사용하는 PowerApps 앱의 성능을 가장 크게 향상할 수 있는 방법은 Microsoft Flow(즉, 흐름)의 구현에서 저장 프로시저를 호출하는 것입니다. 이 방법은 PowerApps 앱에서 데이터베이스 설계를 분리해 준다는 추가적인 이점도 있습니다. 그래서 앱에 영향을 주지 않고 기본 표 구조를 변경할 수 있습니다. 앞으로 설명하겠지만 이 방법은 더욱 안전하기도 합니다.

이미 SQL Server 커넥터를 사용하는 PowerApps 앱에서 이 방법을 사용하려면 먼저 앱에서 SQL Server 커넥터를 완전히 제거해야 합니다. 그런 다음, 데이터베이스의 저장 프로시저에서 EXECUTE 권한으로 제한된 SQL 로그인을 사용하는 새 SQL Server 커넥터를 만듭니다. 마지막으로 흐름에서 저장 프로시저를 호출하고 PowerApps 앱에서 매개 변수를 전달합니다.

흐름을 만들고 PowerApps에 결과를 반환하는 방법에 대한 자세한 설명은 Brian Dang의 게시물 Return an array from a SQL Stored Procedure to PowerApps(Split Method)(SQL 저장 프로시저에서 PowerApps로 배열 반환하기(분할 메서드))를 참고하세요.

이 방법을 사용하면 다음과 같은 성능상의 이점이 있습니다.

- 저장 프로시저가 쿼리 실행 계획을 통해 최적화됩니다. 따라서 데이터가 더 빨리 반환됩니다.
- 저장 프로시저가 관련 데이터만 읽거나 쓰도록 최적화되어 위임 가능한 호출의 문제가 줄어듭니다.
- 최적화된 흐름은 이제 재사용 가능한 구성 요소가 되었습니다. 따라서 같은 환경에 있는 다른 제작자가 일반적인 읽기 및 쓰기를 할 수 있습니다.

고비용 호출

일부 데이터 또는 API 호출은 비용이 많이 들거나 시간이 오래 걸립니다. 긴 실행 시간은 성능에 대한 인식에 영향을 줍니다. 다음은 몇 가지 팁입니다.

- 다음 페이지가 열리기 전에 고비용 호출을 하지 않습니다. 다음 페이지가 즉시 로드되도록 합니다. 그리고 다음 페이지로 가서 OnVisible 속성에서의 호출을 백그라운드에서 수행합니다.
- 로드 중 메시지 상자나 회전자를 사용하여 백그라운드에서 로드가 진행 중임을 사용자에게 알립니다.
- Concurrent 함수는 호출을 병렬 실행하는 좋은 방법입니다. 그러나 이 방법을 사용하면 장기 실행 호출이 후속 코드 실행을 차단할 수 있습니다.

다음 페이지로 이동하는 OnSelect 속성의 안 좋은 예는 다음과 같습니다.

```
Set(ShowExportConfirmDialog,false);
If(CheckPlanner,
    ForAll(Tasks,
        Planner.CreateTask(SelectedPlanId,Name,{bucketId:SelectedBucketId,dueDateTime:DueTime,assignments:AssignToId})
    )
);

ClearCollect(Indexes,{Index:-1});

Navigate(ExportConfirm,None);
```

다음은 더 좋은 예입니다. 우선 OnSelect 속성의 코드입니다.

```
Navigate(ExportConfirm,None)
```

그리고 다음은 다음 페이지 OnVisible 속성의 코드입니다.

```
If(ExportConfirmed,
    Set(Loading, true);

        If(CheckPlanner.Value,
            ForAll(
                Tasks,Planner.CreateTask(
                SelectedPlan.id,Name,
                {
                    bucketId:SelectedBucket.id,
                    dueDateTime:AssnTaskDueDate,
                   assignments:AssignToUser.Id
                }
               )
            )
        );

    Set(Loading, false)
)
```

패키지 크기 제한하기

PowerApps는 앱 로드 최적화를 위해 많은 작업을 하지만, 자체적으로도 앱 설치 공간을 줄이기 위한 조치를 취할 수 있습니다. 설치 공간을 줄이는 일은 구형 디바이스 사용자 또는 대기 시간이 길거나 대역폭이 적은 로캘의 사용자에게 특히 중요합니다.

- 앱에 포함된 미디어를 평가하여 사용되지 않는 경우 삭제합니다.
- 포함된 이미지가 너무 클 수 있습니다. PNG 파일 대신 SVG 이미지를 사용할 수 있는지 확인합니다. 그러나 SVG 이미지에 텍스트를 사용할 때는 클라이언트에 사용 글꼴을 설치해야 하므로 주의해야 합니다. 텍스트를 표시할 때 좋은 해결책은 텍스트 레이블을 이미지 위에 겹쳐서 표시하는 것입니다.
- 해상도가 폼 팩터에 적합한지 평가합니다. 모바일 앱용 해상도는 데스크톱 앱용 해상도만큼 높을 필요가 없습니다. 이미지 품질과 크기가 적절한 균형을 이루도록 실험해 보는 것이 좋습니다.
- 사용하지 않는 화면이 있다면 삭제합니다. 단, 앱 제작자 또는 관리자 전용의 숨겨진 화면은 삭제하지 않도록 주의합니다.
- 하나의 앱에 너무 많은 워크플로를 넣으려 하는 것은 아닌지 평가합니다. 예를 들어 동일한 앱에 관리 화면과 클라이언트 화면이 모두 있지는 않은가요? 그렇다면 개별 앱으로 분리하는 것을 고려해 보도록 합니다. 이러한 방법을 사용하면 여러 사람이 동시에 앱에서 작업하기가 더 쉬워지며, 앱 변경 시 전체 테스트를 통과해야 할 때 "블라스트 반경"(테스트양)을 제한할 수 있습니다.

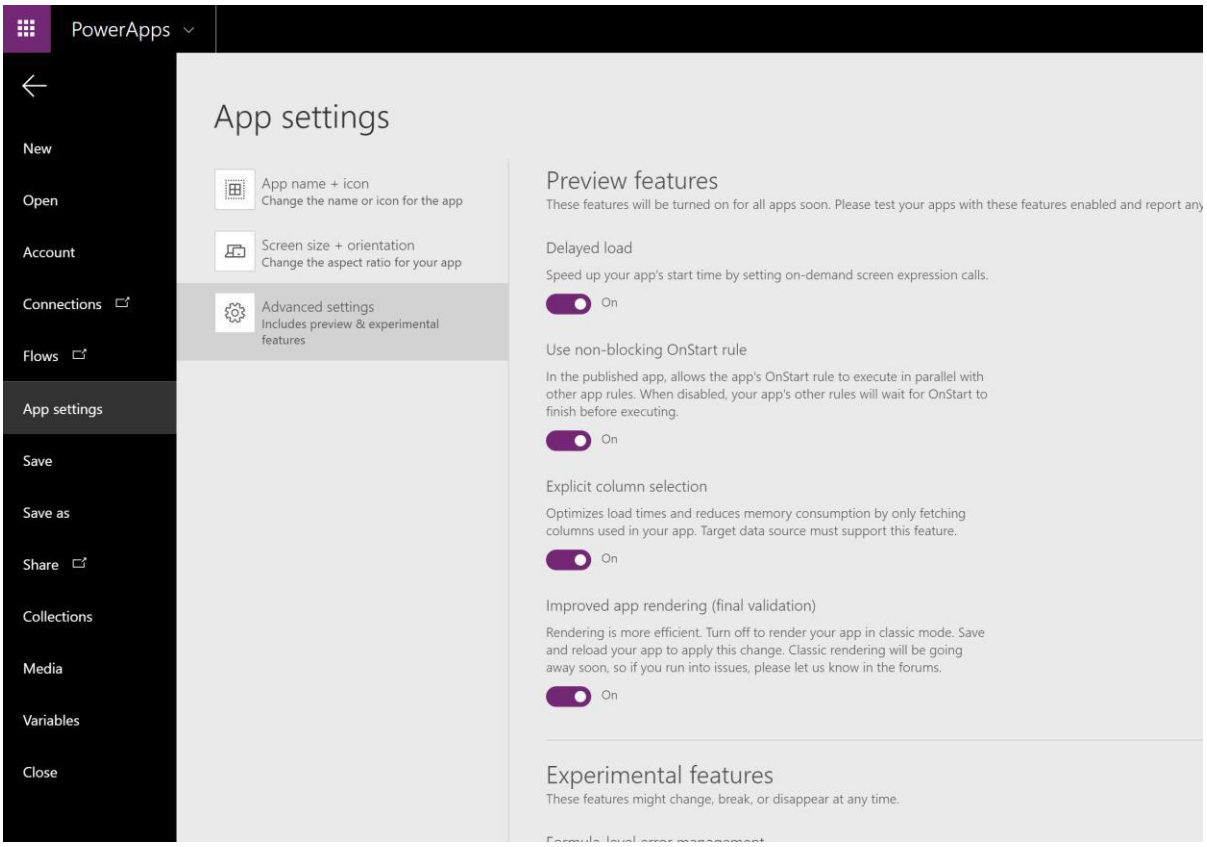
앱을 주기적으로 다시 게시하기

PowerApps 제품 팀은 계속해서 Power 플랫폼을 최적화하고 있습니다. 때로는 이전 버전과의 호환성을 위해 이러한 최적화가 특정 버전 이상을 사용하여 게시한 앱에만 적용되는 경우가 있습니다. 따라서 이러한 최적화를 활용하려면 주기적으로 앱을 다시 게시하시기를 권장합니다.

고급 설정

PowerApps 제품 그룹에는 제작자가 애플리케이션에서 선택적으로 활성화할 수 있는 미리 보기 기능이 있습니다. 이러한 기능 중 일부는 앱 성능을 크게 향상할 수 있습니다. 예를 들어 **지연된 로드** 기능은 애플리케이션에서 지연 로드 기능을 활성화합니다. 그러면 초기 로드 중에 런타임은 첫 화면을 표시하는 데 필요한 화면과 코드만 로드하게 됩니다.

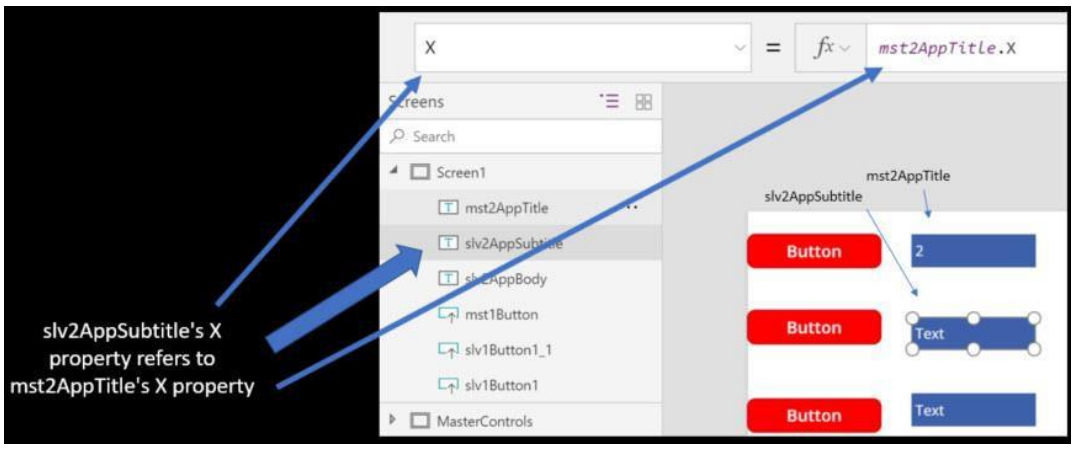
이 기능을 사용하여 발생하는 모든 위험에 대한 책임은 사용자에게 있으며, 이러한 기능을 실험할 때는 앱을 철저하게 테스트해 보시길 바랍니다.



앱 설계

부모/자식 관계를 이용한 상대적 스타일링

한 컨트롤의 스타일을 다른 컨트롤의 스타일 지정 기준으로 사용하는 것이 좋습니다. 일반적으로 색, 채우기, x, y, 너비, 높이 속성에 상대적 스타링을 사용합니다.

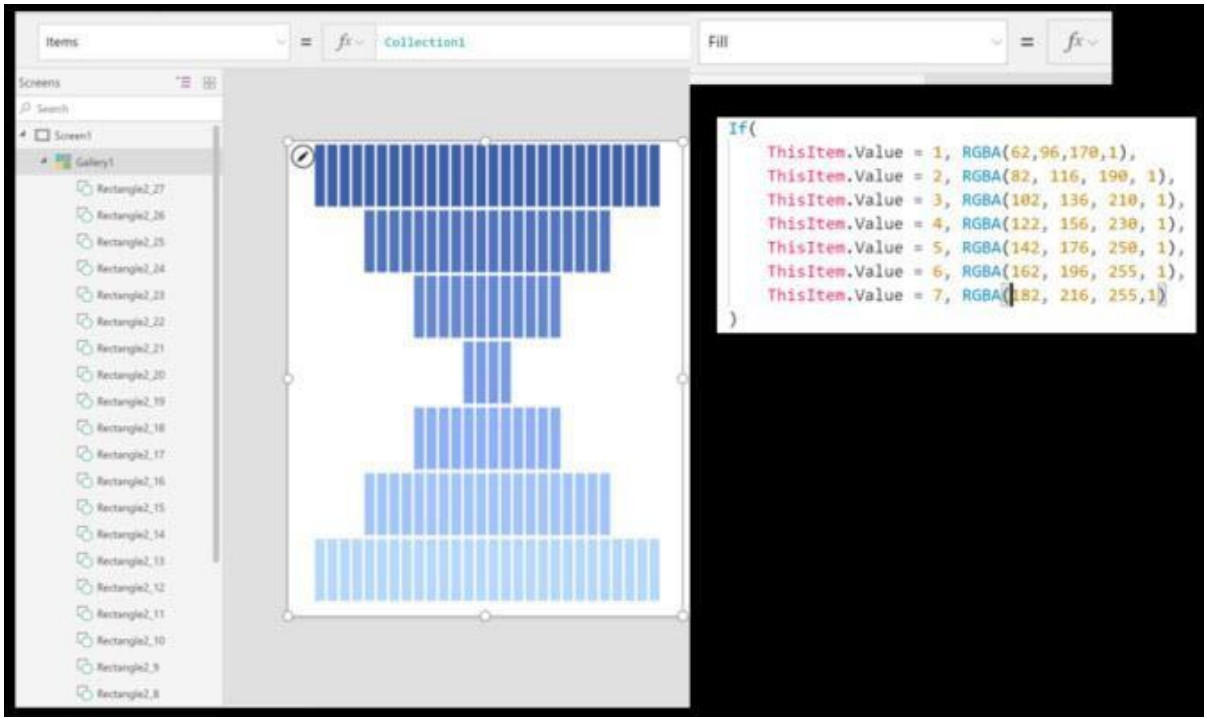


갤러리

반복적이고 선형적인 거의 모든 것에 갤러리를 사용합니다.

처음에는"무차별 대입"(여러 컨트롤을 수동으로 배치) 방법이 빠를 수도 있지만, 나중에 수정하려면 시간이 아주 많이 걸립니다.

반복적으로 보이는 일련의 정보나 컨트롤을 제시해야 하는 경우, 갤러리를 사용하여 내부 컬렉션을 만들 수 있는지 항상 확인해 보도록 합니다.



갤러리 컨트롤을 표시 양식 컨트롤이 아닌 보기 양식으로 사용하는 것도 유용합니다.

예를 들어 세 화면으로 된 "데이터 기반 앱"이 있다고 합시다. 이 앱의 **사용자** 데이터 원본에는 사용자 이름, 직함, 전화번호가 포함되어 있습니다.

첫 화면인 **사용자 목록** 화면에는 galUsers라고 명명된 컨트롤이 있습니다. 이 컨트롤은 모든 사용자를 나열합니다.

두 번째 화면인 **사용자 세부 정보** 화면에는 galUserDetails라고 명명된 갤러리 컨트롤만 있습니다. 이 컨트롤의 Items 속성은 다음과 같이 설정되어 있습니다.

```
Table(
        {Title: "User Name", Value: galUsers.Selected.DisplayName},
        {Title: "Job Title", Value: galUsers.Selected.JobTitle},
        {Title: "Phone Number", Value: galUsers.Selected.PhoneNumber}
)
```

이 메서드는 표시 양식에서 세 개의 개별 데이터 카드를 수정하는 것보다 훨씬 빠릅니다.

양식

반복적인 데이터 입력 필드에는 양식을 사용합니다.

양식을 사용하면 여러 텍스트 상자를 사용하는 대신 여러 필드를 빠르게 그룹화할 수도 있습니다.

또한 양식을 사용하면 부모/자식 관계를 사용하여 상대적인 스타일링을 구현할 수 있으므로 개별 텍스트 상자보다 훨씬 쉽게 작업할 수 있습니다.

Common Data Service for Apps

편집/삽입 작업 처리에 하나의 화면을 사용하는 것이 좋습니다.

가능하면 데이터 업데이트를 처리하기 위해 Patch 함수에서 개별 컨트롤을 참조하는 대신 CardGallery 컨트롤을 사용하는 것이 좋습니다.

컨텍스트 변수를 명명할 때는 해당 변수가 어떤 레코드와 연결되어 있는지 표시합니다.

여러 폼 팩터

동일한 PowerApps 앱이 휴대폰과 태블릿 레이아웃 모두를 대상으로 하는 경우, 먼저 앱의 한 버전을 만들고 테스트를 하고 마무리합니다. 그런 다음, 레이아웃과 화면을 수정하기 전에 다른 버전으로 변환합니다. 이 방법을 사용하면 표현식, 컨트롤, 변수, 데이터 원본 등의 이름을 동일하게 하는 데 도움이 됩니다. 따라서 앱을 지원하고 개발하기가 **훨씬** 쉬워집니다. 하나의 폼 팩터를 다른 폼 팩터로 변환하는 방법을 알아보려면 Todd Baginski의 블로그 게시물 How to convert a PowerAppfrom one layout to another(PowerApp을 하나의 레이아웃에서 다른 레이아웃으로 변환하는 방법)를 참고하세요.

구성 값

모바일 앱에 사용자 정의 설정을 저장하기 위해 SaveData 및 LoadData를 사용할 수 있습니다. 이들을 사용하면 데이터를 편리하게 캐싱할 수 있습니다.

참고: SaveData 및 LoadData는 *PowerApps 플레이어 클라이언트 앱 내에서만 작동합니다*. 앱을 설계할 때 이러한 제한 사항에 유의해야 합니다. 이러한 함수는 PowerApps가 웹 브라우저에 로드될 때는 작동하지 않기 때문입니다.

앱에는 색 구성표, 다른 앱으로 연결되는 URL, 디버그 컨트롤을 앱 화면에 표시할지 여부 등을 정의하는 설정 등 한 곳에서 쉽게 변경할 수 있는 애플리케이션 설정이 필요할 것입니다. 그러면 애플리케이션을 배포하는 사람도 이러한 값을 빠르게 설정할 수 있고, 배포 중에 코드가 엉망이 될 위험도 줄어듭니다. 이러한 설정은 ASP.NET web.config 파일과 비슷하다고 생각하면 됩니다.

다음은 구성 값을 저장하는 몇 가지 방법입니다. 난도가 낮은 순서대로 정렬했습니다.

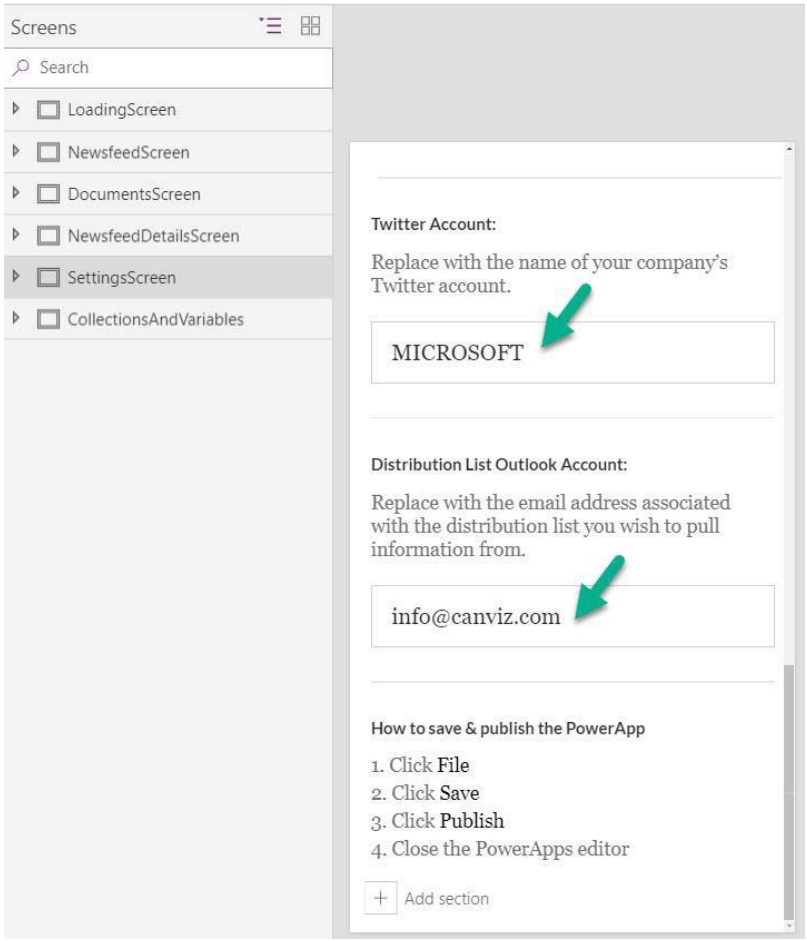
숨겨진 구성 화면 생성하기

구성 값을 설정하는 놀랍도록 쉬운 방법은 숨겨진 화면을 만들고 텍스트 입력 컨트롤에 구성 값을 넣는 것입니다. 이렇게 하면 코드를 편집하지 않고도 애플리케이션 설정을 변경할 수 있습니다. 이 방법을 사용하려면 다음 단계에 따르세요.

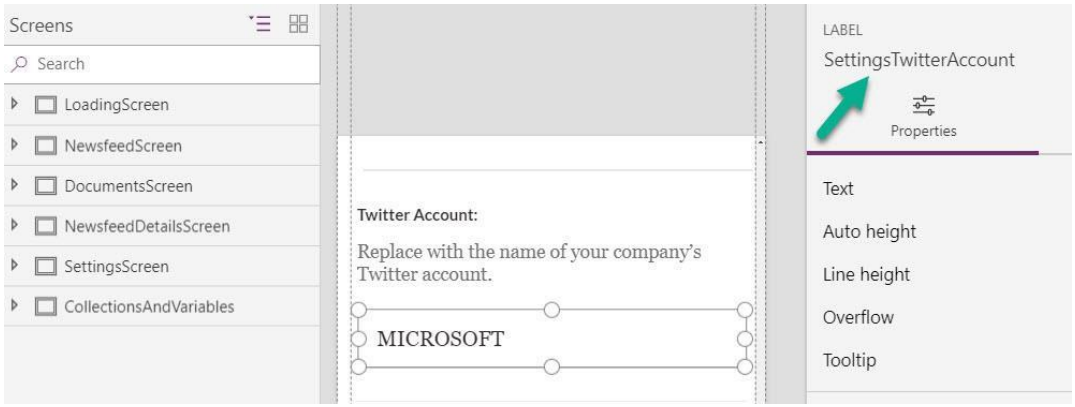
1. 구성 화면이 애플리케이션의 첫 화면이 되지 않도록 합니다. 첫 화면 외에 다른 순서에 배치하세요. 여기서는 쉽게 찾을 수 있도록 마지막 화면에 배치하기를 권장합니다.
2. 사용자는 해당 화면으로 이동할 수 없도록 합니다.

3. 자신이 해당 화면으로 이동할 수 있는 방법을 마련합니다. 가장 쉬운 방법은 앱을 편집할 때만 화면에 액세스할 수 있도록 설정한 다음, 수동으로 화면에 이동하는 것입니다. 앱의 홈 화면에 앱 제작자와 관리자에게만 표시되는 숨겨진 버튼을 만들어 구성 화면으로 이동하게 할 수도 있습니다. 사용자가 앱 제작자 또는 관리자인지 여부는 전자 메일 주소(예: User().Email 확인), AAD 그룹 멤버십, 또는 제작자용 PowerApps 커넥터를 사용해 확인할 수 있습니다.

다음 예시는 Microsoft PowerApps Company Pulse 샘플 템플릿에서 가져왔습니다. 템플릿을 보면 PowerApps 관리자가 애플리케이션 설정 값을 구성할 수 있는 텍스트 입력 컨트롤이 있습니다.



다음에서는 Twitter 계정 설정 값을 저장하는 컨트롤의 이름을 볼 수 있습니다.



다음에서는 Twitter 커넥터에서 Twitter 계정의 트윗을 반환하기 위해 값이 사용된 부분을 볼 수 있습니다.

이는 값을 변경하는 가장 쉬운 방법이긴 하지만 단점도 있습니다.

- 값을 변경하고 이를 유지하려면 앱을 다시 게시해야 합니다.
- 값이 앱 내부에서 유지되기 때문에, 다른 환경으로 이동할 것에 대비하여 앱을 내보내기 전에 이러한 값을 업데이트하는 프로세스를 마련해야 합니다.

Common Data Service for Apps 구성 값 저장하기

또는 새 Common Data Service for Apps 엔터티를 만들어서 그곳에 구성 값을 저장할 수도 있습니다. 값이 앱 외부에서 유지되기 때문에 앱을 재배포하지 않고도 언제든지 수정할 수 있습니다. Common Data Service for Apps 엔터티는 환경별로 고유한 값을 가질 수 있습니다. 예를 들어 URL이 사전 프로덕션과 프로덕션 환경에서 다를 수 있습니다.

이는 구성 값을 유지 관리하는 좋은 방법이지만 단점도 있습니다.

- 텍스트 입력 컨트롤 방식과 달리 이 방법에는 Common Data Service for Apps에 대한 콜백이 필요합니다. 따라서 성능에 약간의 영향이 있을 수 있으며 사용자가 모바일 디바이스를 사용하다가 연결이 끊어지는 등의 이유로 Common Data Service for Apps를 사용할 수 없는 경우 앱이 올바르게 표시되지 않을 수 있습니다.
- 캐싱이 없어 앱을 열 때마다 새 호출이 이루어집니다.
- 호출에 실패해도 모니터링이 되지 않습니다. 사용자가 알려줘야만 앱에서 오류가 발생한 것을 파악할 수 있습니다.

사용자 지정 API 사용하기

구현적인 방법은 가장 어렵지만 Microsoft IT 부서에서는 Azure 테이블 저장소에 구성 값을 이름/값 쌍으로 저장하는 Azure 앱 서비스로 성공을 거뒀습니다.

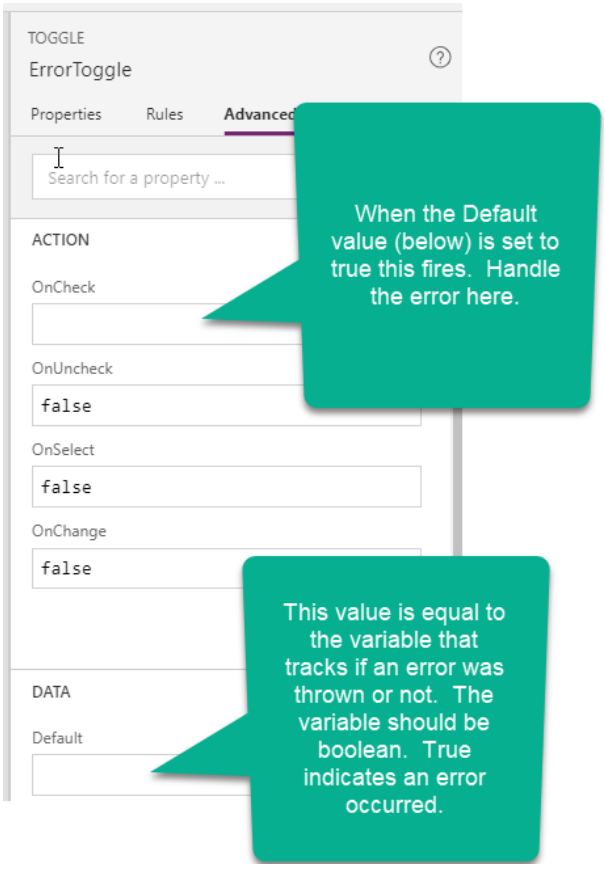
OAuth로 보안 설정된 사용자 지정 커넥터는 구성 값을 가져오고 출력 캐싱(값 변경 시 캐시 무효화 포함)을 통해 성능이 향상됩니다. Azure Application Insights는 문제가 발생할 경우 경고와 함께 알림을 보내며 사용자 세션의 문제를 훨씬 쉽게 해결할 수 있도록 합니다.

오류 처리/디버깅

오류 처리를 위한 토글 컨트롤

OnTimerStart 속성 섹션에서는 타이머 컨트롤로 오류를 처리하는 한 가지 방식을 보여 드렸습니다. 오류를 처리하는 또 다른 방식은 토글 컨트롤과 관련이 있습니다.

다음은 이 방법을 잘 보여 줍니다.



이 방법에서 유효성 검사 및/또는 오류 처리를 위한 로직은 하나의 컨트롤 내에 캡슐화할 수 있습니다. 토글 컨트롤은 복잡한 조건을 평가하여 true 또는 false 값을 반환할 수 있습니다. 그러면 다른 컨트롤이 해당 값을 참조하여 오류 메시지 표시/숨김, 글꼴 또는 테두리 색 변경, 버튼

이용할 수 없게 하기, Application Insights에 로그 등의 작업을 수행할 수 있습니다. 이 컨트롤이 표시되고 편집 가능하도록 설정하면 앱 제작자는 오류 조건을 켜고 끄면서 UI(사용자 인터페이스)가 반응하는 것을 볼 수 있습니다. 이 방법을 사용하면 앱을 개발 또는 디버그할 때 시간과 노력을 절약할 수 있습니다.

캔버스 컨트롤을 디버그 패널로 사용하기

앱을 개발하고 테스트할 때 캔버스 컨트롤을 사용하여 화면 상단에 표시되는 반투명 디버깅 패널을 만들 수 있습니다. 이 패널에는 필요에 따라 편집 가능한 필드, 토글, 기타 컨트롤 등을 추가할 수 있어 앱이 플레이 모드에 있는 동안 변수를 변경할 수 있습니다.

단계별로 설명된 동영상이 필요한 경우 Brian Dang의 PowerApps - Best Practices: Debug Panel(모범 사례: 디버그 패널)을 참고하세요.

앱 제작자에게 디버그 컨트롤 표시하기

모든 사용자에게 디버그 컨트롤을 표시해서는 안 됩니다. 따라서 디버그 패널의 Visible 속성을 수동으로 토글하거나(PowerApps Studio의 경우) 특정 사용자에 대한 컨트롤 표시 여부를 자동으로 토글해야 합니다.

훌륭한 방법 하나는 제작자용 PowerApps 커넥터를 추가하는 것입니다. 이름과는 달리 이 커넥터는 제작자가 아닌 사용자도 읽기 전용 호출에 사용할 수 있습니다. 그런 다음, GetAppRoleAssignments 함수를 호출하여 로그인한 사용자가 현재 앱의 제작자인지 여부를 확인합니다.

```
Set(gloCurrentUserEmail,User().Email);

ClearCollect(Makers,
    ForAll(PowerAppsforAppMakers.GetAppRoleAssignment("Your App ID Goes Here").value,
    {Email:properties.principal.email,Role:properties.roleName})
);

Set(gloIsMaker,
    And(gloCurrentUserEmail exactin Makers.Email,
    Not(LookUp(Makers,Email=gloCurrentUserEmail).Role="CanView"))
);
```

이 예시에서는 다음으로 디버그 컨트롤의 Visible 속성을 gloIsMaker로 설정하여 해당 컨트롤이 제작자 권한이 있는 앱 사용자에게만 표시되도록 합니다.

이 방법의 장점은 구성 테이블을 사용하여 특별한 디버그 권한을 지정할 필요가 없다는 것입니다.

또한 앱 제작자 또는 관리자의 전자 메일 주소(예: User().Email) 또는 AAD 그룹 멤버십을 확인하여 앱 제작자 또는 관리자만을 위해 디버그 컨트롤을 표시하거나 숨길 수도 있습니다.

문서화

코드 주석

2018년 6월부터 코드에 주석을 추가할 수 있습니다. 애플리케이션에서 코드를 작성할 때는 주석을 많이 달아야 합니다. 주석은 몇 달 후 애플리케이션을 다시 열었을 때 도움이 되며, 앱 작업을 맡게 되는 다음 개발자에게도 큰 도움이 됩니다.

주석에는 두 가지 종류가 있습니다.

- **라인 주석:** 코드 줄에 이중 슬래시(//)를 추가하면 PowerApps는I 줄의 나머지 부분(// 포함)을 주석으로 처리합니다. 라인 주석을 사용하여 다음에 일어날 일을 설명할 수 있습니다. 또한 코드 줄을 삭제하지 않고 일시적으로 비활성화할 때도 사용할 수 있어 테스트에 유용합니다.
- **블록 주석:** /* 및 */로 묶인 모든 텍스트는 주석으로 처리됩니다. 라인 주석은 한 줄에만 주석을 달지만 블록 주석은 여러 줄에 달 수 있습니다. 따라서 코드 모듈 헤더와 같이 여러 줄로 된 주석을 다는 데 유용합니다. 테스트나 디버그할 때 일시적으로 여러 줄의 코드를 비활성화하기 위해서도 사용할 수 있습니다.

특히 주석이 코드 블록 앞에 있는 경우, **텍스트 서식** 기능을 사용한 **후에** 주석을 추가하는 것이 좋습니다. **텍스트 서식 지정** 기능은 기존 주석에 대해 다음 로직을 사용합니다.

1. 어떤 속성이 블록 주석으로 시작하면 다음 코드 줄이 끝에 추가됩니다.
2. 어떤 속성이 라인 주석으로 시작하면 다음 코드 줄이 끝에 추가되지 않습니다. 추가되는 경우 해당 코드는 주석 처리됩니다.
3. 속성의 다른 모든 부분에 있는 라인 및 블록 주석은 이전 코드 줄 끝에 추가됩니다.

주석이 너무 많거나 너무 길다고 걱정할 필요는 없습니다. PowerApps가 클라이언트 앱 패키지를 만들 때 모든 주석이 제거됩니다. 따라서 주석이 패키지 크기에 영향을 미치거나 앱 다운로드 또는 로딩 시간을 지연시키지 않습니다.

문서화 화면

PowerApps 앱에서 사용되는 컬렉션 및 변수를 문서화하는 화면을 만드는 것이 좋습니다. 앱의 다른 화면을 이 화면으로 연결하지 마세요. 이 화면은 앱이 편집 모드로 열려 있을 때만 표시됩니다.

다음은 Microsoft PowerApps Company Pulse 샘플 템플릿입니다.