



Qlik Sense- Qlik Deployment Framework

Getting Started Guide

December, 2016





Table of Contents

Why a Framework?	3
Standards	3
Qlik Deployment Framework	3
Qlik Deployment Framework resource containers	4
Benefits using Resource Containers	4
Global Variables	5
Get started Qlik overview	6
QDF Deploy Tool	6
Container Type Selector	6
Quick Install	7
Qlik Deployment Framework mandatory containers	7
Manage and add containers	7
Mapping Qlik Sense to QDF	8
Qlik Sense folder connections and QDF	8
Single LIB mount	8
Separate LIB mounts	11
Qlik Sense Server modifications	13
Optional settings	13
QDF Function Library	14
QlikView Components (QVC)	14
Function examples	14
Custom Global Variables	14
Default containers	15
Administrative Container (0.Administation)  0.Administration	15
Shared Folders Container (99.Shared_Folder)  99.Shared_folders	15
Resource Containers  1.Project_HR	15
Example Container (1.Example)	15

Why a Framework?

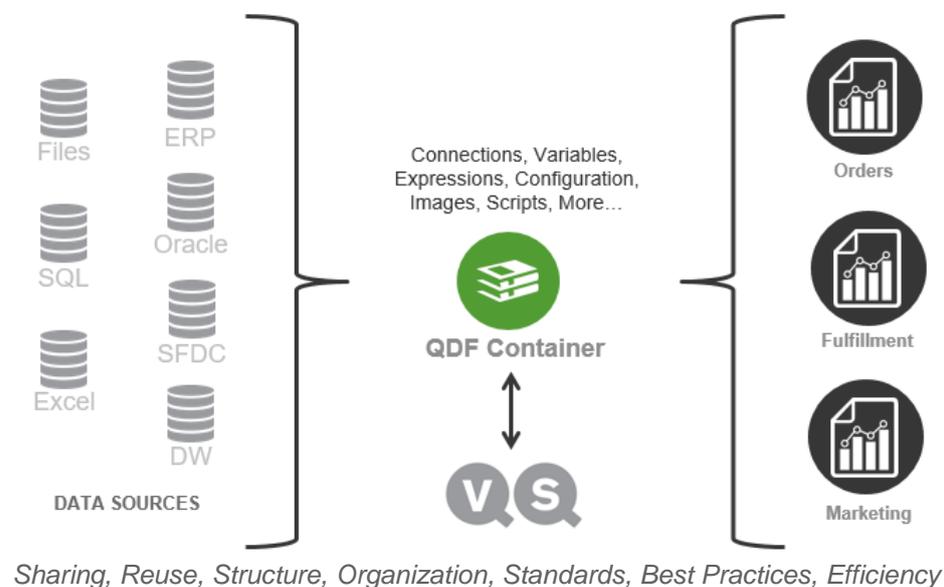
- **Do more in Less Time**
- **Improve Overall Quality**
- Develop in a way that others can understand
- Standards and Best Practice *“Don’t reinvent the wheel”*
- Rapid Development and Manageability in a Growing Deployment *“Scale up and out”*
- Multiple development teams

Standards

It’s important to have and use standards when developing and maintaining a development platform. There are many ways of getting the same result, but not all of them are efficient and understandable. QDF in combination with guidelines allows for a cohesive multi-development environment.

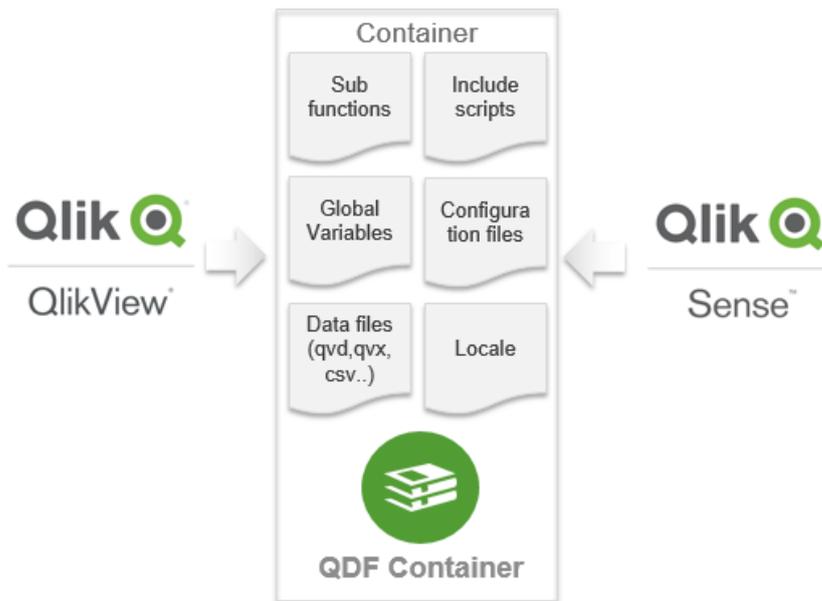
Qlik Deployment Framework

With the introduction of Qlik Deployment Framework building and managing Qlik deployments becomes easier and more efficient, for not only IT but professional services as well. QDF is a set of guidelines and utilities that enable: Resource Sharing, Reuse, Organization, Structure and Standards providing an effective and efficient deployment including both QlikView and Qlik Sense. The Qlik Deployment Framework (QDF) design and documentation is based on knowledge gathered from real life scenarios.



Qlik Deployment Framework resource containers

QDF is based on the concept of **resource containers**. This is building blocks and security separators that is aligned to fit the current needs. when demand changes its easy to reorganize and add additional containers. QlikView and/or Qlik Sense applications are hooked into the containers in which all needed resources are resided. Each container has identical file structures and contains predefined script functions.



Benefits using Resource Containers

- The built-in function library will simplify development and management.
- Containers represents security boundaries separating access based on roles.
- The container content is seamlessly reused between QlikView and Qlik Sense
- Containers can be created for different purpose, like common data and expressions (shared resources), departmental data, source data and many more...
- A container can be specialized for a single purpose, like SAP data source that IT want to control.
- Consolidation is easy when using container architecture. Just move containers from different locations to one single place.
- QDF is designed for application lifecycle management, to simply promote content and applications from development to production
- It is easy to share data between containers and still have the data separation.
- Scale from a small one container setup to a big 10 container setup.
- Metadata, the framework provides metadata regarding containers, data and variables.

Global Variables

Qlik applications utilizing QDF need to execute an initiation script during reload. The initiation will dynamically auto-generated **Global Variables**, these are associated to the applications “home container” during reload. When moving applications across environments the initiation script will identify the new location and regenerate the Global Variables to another “home container”, making load scripts run independently on location.

Container Naming Convention

The folders inside the containers are simplified to fit as many languages and companies as possible. Each container folder starts with a sequential number, this to make it easy to identify and document the containers and its content. Global variables are global as they are reused between applications and environments. The name standard is `vG.xxx` (**Variable.Global**).

Container Variables

Container folders points to a corresponding **Global Variable**, referencing content placed in the containers using **Global Variable** (in *Qlik Load script*) makes it easy to move code and applications between containers and environments without any script modifications, this also makes it possible to seamlessly share scripts between QlikView and Qlik Sense.

Sense only container example

Container folders	Global Variables	Description
0.Template		Folder used for examples and templates. <i>Only exists in the 0.Administration Container.</i>
1.QVD	<code>vG.QVDPath</code>	QlikView Data files are stored in subfolders under 2.QVD
1.Extract	<code>vG.ExtractPath</code>	Extract path for QVD files
2.Transform	<code>vG.TransformPath</code>	Transform path for QVD files
3.Load	<code>vG.LoadPath</code>	Load path for QVD files
2.Config	<code>vG.ConfigPath</code>	Configuration and language files like Excel and txt. This folders could be shared to make configuration changes easier
3.Include	<code>vG.IncludePath</code>	Folder where QlikView Include files are stored. These are script parts that are called from the main QlikView script.
1.BaseVariable	<code>vG.BaseVariablePath</code>	Stores all the variables needed to use the framework, like paths inside the container
2.Locale	<code>vG.LocalePath</code>	Locale for different regions, used for easy migration between regions
3.Custom	<code>vG.CustomPath</code>	Store for custom include scripts
4.Sub	<code>vG.SubPath</code>	Store for sub routines, this is a way to reuse code between applications
4.Export	<code>vG.ExportPath</code>	Folder used to store from QlikView exported data, probably txt or qvx
5.Import	<code>vG.ImportPath</code>	Folder used to store import data from external systems
6.Misc		To store documentation, extension and other things related to this container
Info.txt		Information files describing the folder purpose and Path variable. There are Info files in every folder.
Version.xx.txt		Version Revision list

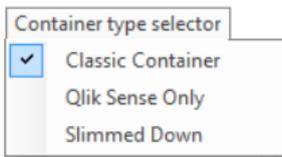
Get started Qlik overview

QDF Deploy Tool

Containers are managed using the QDF Deploy tool that is available for download on [Qlik community](#). First read the *Qlik Deployment Framework-ReadMe.pdf* document that is attached with the Deploy tool on how to use the tool.

Container Type Selector

From QDF version 1.7.0 you have the possibility to select between three container layouts. This means that the QDF Deploy tool extracts different container layouts (folders) depending on selections in the *Container Type Selector*. The different settings and correlating layouts are presented below.

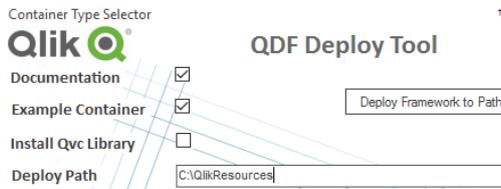


- **Classic Container (default)** has the same container layout as traditional QDF (from version 1.0)
- **Qlik Sense Only** Here we have removed everything related to QlikView, for example Application and mart folders. Extract, Transform, Load folders have also been added under the QVD structure.
- **Slimmed Down** is intended for smaller deployments and several folders have been removed. Extract, Transform, Load folders have also been added under the QVD structure.

Classic Container	Qlik Sense Only	Slimmed Down
<ul style="list-style-type: none"> ▼ 1.Example <ul style="list-style-type: none"> > 1.Application 2.QVD > 3.Include > 4.Mart > 5.Config > 6.Script 7.Export 8.Import > 9.Misc 	<ul style="list-style-type: none"> ▼ 1.Example <ul style="list-style-type: none"> ▼ 1.QVD <ul style="list-style-type: none"> 1.Extract 2.Transform 3.Load > 2.Config > 3.Include 4.Export 5.Import > 6.Misc 	<ul style="list-style-type: none"> ▼ 1.Example <ul style="list-style-type: none"> > 1.Application ▼ 2.QVD <ul style="list-style-type: none"> 1.Extract 2.Transform 3.Load > 3.Include > 4.Config 5.Import

Quick Install

After changing *Container Type* add a location in the **Deploy path** or double click on the box to brows folders on disk. This location is where the framework will be extracted to and press deploy framework to path.



Qlik Deployment Framework mandatory containers

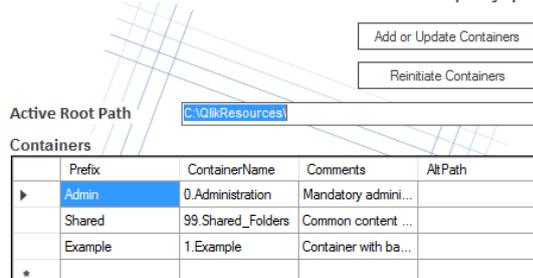
After the Deploy Tool have been executed these default containers have been extracted (in addition to your personal containers). Read more on this later in this document.

- **Administration** container it's from here additional containers are created and managed.
- **Shared folders.** A fresh QDF installation always contains a shared container, this is a repository to store scripts, configuration and data files that are shared and reusable by all applications.
- **Example** This is a container containing some examples and QVD files, this is used during the exercise documentation.

Manage and add containers

Use the deploy tool to add containers. Active frameworks are identified automatically when typing a valid URL. Deploy tool will store last used URL to make it easy to continue.

1. If an active framework is found the Deploy path text will change to **Active Root Path**.



2. Within the grid new *Container Names* with related *Prefix* can be typed in
3. Container properties:
 - **Prefix** (Mandatory) this is the Container *identifier* this is a **unique** short name for company department or project. This is used when mounting containers within the Qlik Scripts. Do not use long names or spaces
 - **Container Name** (Mandatory) this is the physical container name shown on disk, could be company department or project. This could also include subfolders within the framework, examples:
 - **1.AcmeSales**
 - **100.DataSources\1.SQL-DB**
 - **Comments** (optional) is used as descriptive meta-data.
 - **AltPath** (optional) is used when creating a container in a different location. Example *C:\QlikTempContainers*
4. To add the new containers press **Add or Update Containers**.

Mapping Qlik Sense to QDF

After Deploy Tool extracted QDF and created the container structure we need to map these into Qlik Sense Desktop or Enterprise. In Enterprise version mapping the containers only need to be done once as the connections are reused across applications.

Getting started exercises are available in the *Qlik Deployment Framework-Qlik Sense Exercises.pdf* document that will be installed using the *Deploy Tool*.

Qlik Sense folder connections and QDF

Qlik Sense has a folder access method called *folder connection*. Adding a *folder connection* will create a **LIB** URL to the repository folder. The traditional QlikView way to accessing files (*c:\xxx\yyy* or *\\ServerName\yyy*) is in Sense called *legacy mode* and disabled by default while accessing files using LIB's is called *native mode* and enabled by default. Qlik Deployment Framework works in native mode and utilizes the LIB url's.

Qlik Sense integration can be done in two different ways. First is to mount the whole container structure (*Root*) under on single LIB (*Singe LIB mounts*) and the second is to mount each container individually as its own LIB folder connection (*Separate LIB mounts*). The modes can be mixed from app to app in the same setup, for example depending on security level.

Example:

QlikView Global Variable	Qlik Sense Global Variable	
<code>vG.BasePath=\\Server\QDF\1.Example\</code>	<code>vG.BasePath=LIB://Example</code>	<i>Separate Mounts</i>
	<code>vG.BasePath=LIB://Root\1.Example</code>	<i>Single Mount</i>

The next sections will explain the difference in setting up single mount and separate mounts.

Single LIB mount

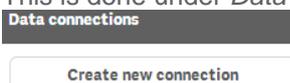
When using single mount, QDF locates containers in the same way as QlikView does by use of the container map.

For single LIB mount add a Sense folder connection (LIB) named *Root* pointing to the framework starting point

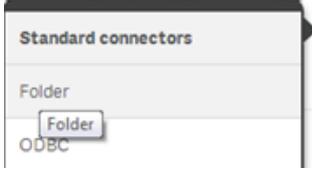


In this example a lib named Root pointing to QDF's root folder.

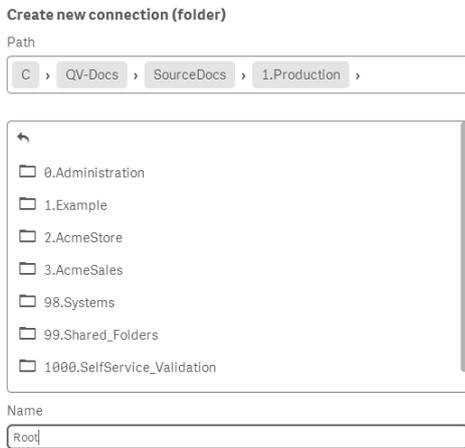
1. To create the **LIB://Root** described above we first need to create a new *Folder Connection* in Qlik Sense. This is done under *Data connections* selecting *Create new connections*.



2. Select the *Folder Connection* alternative.



3. Point to the root location of QDF container architecture and give it the name **Root**. Type in UNC or drive path here.



The *Folder Connection* name is case sensitive.

4. Last add the initiation script first in the load editor :

```
//SET vG.BasePath=; //Uncomment this line to reset cache
$(Include=lib://Root\InitLink.qvs);
```

5. Run the script, a similar text like below should be presented stating that single LIB mount have been identified.

```
Started loading data

'### DF InitLink Started, trying to link to 1.Init.qvs
script'

'### DF Info, identified Sense root path lib://Root/
(single LIB mount) '

'### DF Info, identified Sense home container
lib://Root/0.Administration\'
```

6. Also Global Variables (*vG.xxxPath*) pointing to *lib://root* should be available in the variable editor.

Variables		Create new
Name	Definition	
vG.BasePath	lib://Root/0.Administration\	
vG.BaseVariablePath	lib://Root/0.Administration\3.Include\1.BaseVariable\	
vG.ColorSchemePath	lib://Root/0.Administration\3.Include\5.ColorScheme\	
vG.ConfigPath	lib://Root/0.Administration\5.Config\	
vG.ConnStringPath	lib://Root/0.Administration\3.Include\3.ConnString\	
vG.CustomPath	lib://Root/0.Administration\3.Include\6.Custom\	
vG.ExportPath	lib://Root/0.Administration\7.Export\	

Optional LIB name

To change default **Root** folder connection (LIB) name add **SET** `vG.RootContainer='LIB name'` before QDF Initiation, the value should match the folder connection name pointing to **root**, see more under *Optional settings*. This setting only applies in single mount mode.

Home container

Home container is the place where the generated global variables will be pointing to (ex. `vG.QVDPATH`) and the location where custom global variables are read from. The default home container will be identified automatically (usually `0.Administration`) to change this behavior add **SET** `vG.HomeContainer='Physical container name'` in the script beginning, see more under *Optional settings*.

Notes

- When using single mount and a container is not under the root (called *Alt Root Path* in QDF container Map) a Separate LIB mount need to be created. This mount should have the same name as the *Variable Prefix* in the Container Map.
- Home container identification is only done automatically directly under the root LIB, to use specific container always use `vG.HomeContainer` variable before of 1.Init initiation.
Example **SET** `vG.HomeContainer= '98.Systems\1.Oracle'`

Separate LIB mounts

In this mode each folder connection (LIB) is pointing to an physical container. In other words, add a folder connection for each container you want to use in your application. Remember to use same folder connections name (LIB) and *Prefix* in Deploy Tool. In *Qlik Sense Enterprise* LIB's can be managed by the system administrator thereby **restricting file access** (read more below).

The screenshot shows the Qlik Sense interface with three folder connections listed on the left: AcmeStore, Example, and Shared. To the right, a table titled 'Containers' lists the mapping between Prefix and ContainerName. Further right, a list of containers is shown with icons and names.

Prefix	ContainerName
Admin	0.Administration
Shared	99.Shared_Folders
Example	1.Example
AcmeStore	2.AcmeStore
AcmeSales	3.AcmeSales

Containers list:

- 0.Administration
- 1.Example
- 2.AcmeStore
- 3.AcmeSales
- 98.Systems
- 99.Shared_Folders

Set home container

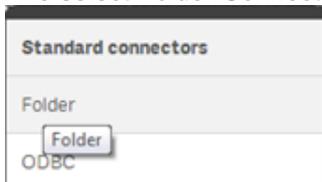
Home container is the place where the generated global path variables will be pointing to and the location where sub functions and global custom variables are read from (ex `vG.QVDPATH`). A Sense folder connection (LIB) need to be created. Just add `SET vG.HomeContainer='LIB-Name'` statement in the script beginning before framework initiation

Separate lib mounts step by step

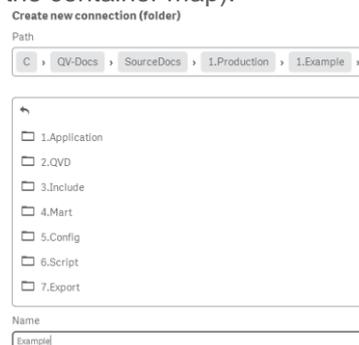
1. To create `LIB://Example` shown above we first need to create a new *Folder Connection* in Qlik Sense. This is done by under *Data connections* selecting *Create new connections*.



2. And select *Folder Connection*



3. Point to the location of 1.Example container path and give it the name Example (same name as used in the container map).



4. Now you need to initiate QDF in the load script. To use `Example` as home container add `SET vG.HomeContainer`. Easiest is to use statement below.

```
//SET vG.BasePath=; //Uncomment this line to reset cache
SET vG.HomeContainer='lib://Example';
$(Include=$(vG.HomeContainer)\InitLink.qvs);
```

- Run the script, a similar text like below should be presented stating that separate LIB mount have been identified.

```
Started loading data

'### DF InitLink Started, trying to link to 1.Init.qvs
script'

'### DF Info, identified Sense home container
lib://Example/ (Separate LIB mounts) '

'### DF 1.Init.qvs Started'
```

- Last validate that Global Variables are available (*vG.xxxPath*)

Variables Create new	
Name	Definition
 vG.ApplicationPath	lib://ASales/1.Application\
 vG.BasePath	lib://ASales/
 vG.BaseVariablePath	lib://ASales/3.Include\1.BaseVariable\
 vG.ColorSchemePath	lib://ASales/3.Include\5.ColorScheme\
 vG.ConfigPath	lib://ASales/5.Config\
 vG.ConnStringPath	lib://ASales/3.Include\3.ConnString\
 vG.CustomPath	lib://ASales/3.Include\6.Custom\
 vG.ExportPath	lib://ASales/7.Export\
 vG.HomeContainer	lib://ASales
 vG.ImportPath	lib://ASales/Import\
 vG.IncludePath	lib://ASales/3.Include\
 vG.LocalePath	lib://ASales/3.Include\2.Locale\

Note. When using Separate LIB mounts the global variable *vG.RootPath* (pointing at the framework root path) is removed as LIB's does not have any actual root path, different LIB's can point on different file systems.

★ vG.RootPath <NULL>

Qlik Sense Server modifications

When creating Lib's in Qlik Sense Server (during app development) the Lib name also contains concatenated *ComputerName + UID* this to make Lib's personal by default. When developing using QDF we do not want personal LIB's, instead the QDF folder LIB's should be common for all developers assigned by security rules. In the QMC *Data Connections* tab we can easily modify the name and apply one or more security rules depending on developer access rights.

Our Example Lib have (*ComputerName + UID*) added to the name

Example (sense1dot1_qlikservice) ✕
Folder



We need to change this in the QMC Data connections Tab

Data connections

Remove unwanted additional text and press apply

Data connection edit

Data connection edit

IDENTIFICATION

Name

Example (sense1dot1_qlikservice)

IDENTIFICATION

Name

Example

An *Associated Security rule* can also be added to the Data connection, so that only authorized developers have access to the container (In our case Example Container).

+ Create associated rule

Optional settings

Single LIB Mount

`SET vG.HomeContainer='1.Example'` Physical container name before 1.Init initiation script will use 1.Example container as Home container instead of the default 0.Administration container. **Remember**, here you need to type the physical container folder name instead of LIB name.

`SET vG.RootContainer='QDF'` before 1.Init initiation script will use QDF LIB Folder as Root instead of the default Folder connection name (**Root**)

Separate LIB Mounts

`SET vG.HomeContainer='Example'` before 1.Init initiation script will use Example Folder connection as Home container instead of the default Folder connection name (**Home**).

`SET vG.HomeContainer='lib://Example'` before 1.Init initiation script will use Example Folder connection as Home container instead of the default Folder connection name (**Home**). This will also work with variables in the Include script as shown below:

```
SET vG.HomeContainer='lib://Example';
```

```
$(Include=$(vG.HomeContainer)\UnitLink.qvs);
```

QDF Function Library

Qlik have the possibility of reusing scripts and functions by using the Sub and Call commands. The Framework contains library of nice to have functions. All sub functions are stored under the 3.Include\4.Sub folder and are initiated during QDF initiation. Use Call function_name('Input parameters or variables') command to execute the preloaded function.

Read more in ***Qlik Deployment Framework-Function Reference Guide.pdf***

QlikView Components (QVC)

Optionally, the open source library QlikView Components (QVC) can be installed using the deploy tool in addition to the built-in QDF library. If QVC been installed it can be disabled in the script by adding this line before QDF initiation:

```
set vG.QVCDisable='true';
```

Function examples

After QDF initiation you can start using the built-in sub function library. Sub Function example, *vL.FileExist* variable will return true or false depending on if the file exists

```
Call vL.FileExist ('$(vG.QVDPath)\SOE.qvd');
```

Load Container Global Variables (LCGV)

By using *LCGV* function it's possible to create Global Variable links (mounts) between containers (security access needed). The benefit is to create generic, reusable and movable code as QDF validates and regenerates the links every time the script runs.

Example: *call LCGV('AcmeTravel');* Will create all Global Variables linking to AcmeTravel container. Variables created will have similar name as home container but with the additional *AcmeTravel* prefix, like *vG.AcmeTravelQVDPath* for QVD path to AcmeTravel container

call LCGV('Oracle','QVD;Include'); Will create two Global Variable links to different resources in Oracle container, by using an additional switch and ';' separator creates Global Variables *vG.OracleQVDPath* and *vG.OracleIncludePath* (instead of linking all folders as in the first example).

Custom Global Variables

Global variables are used when sharing variables across multiple applications within a Container. Custom Global Variables is additional variables created manually by the developers and generated during initiation. Add variables by modifying the *CustomVariables.csv*, that exists within each container under *3.Include\1.BaseVariable*.

Universal variables

Universal Variables is used when sharing variables between all applications within a site regardless of home container, creating a "single point of truth" across all applications. Add universal variables by modifying *CustomVariables.csv* file stored under **shared folders container** (*3.Include\1.BaseVariable*), and loaded during the framework initiation process.

Default containers

As mentioned earlier the Deploy Tool creates some default containers during install:

Administrative Container (0.Administration) 0.Administration

The Administrative container is a mandatory pre-installed Blue container. Administration is used for administrative duties and for the Qlik platform maintenance. Only system administrators need to have access to this container.

The folder *0.Templates* exists only in 0.Administration container. This folder stores templates, the most important is the container template that new containers are copied from.

Shared Folders Container (99.Shared_Folder) 99.Shared_folders

A fresh QDF installation always contains a shared container, this is a repository to store scripts, configuration, data files and meta-data that are shared and reusable by all applications. This is a “*single point of truth repository*” between all containers.

During QDF initiation global variables pointing to shared folders are generated in the applications. The variables are similar to the ordinary Global Variable name except for a Shared prefix (*vG.SharedxxxPath*).

Shared Sense only container

Container folders	Global Variables	Description
1.QVD	<i>vG.SharedQVDPath</i>	QlikView Data files are stored in subfolders under 2.QVD
1.Extract	<i>vG.SharedExtractPath</i>	Extract path for QVD files
2.Transform	<i>vG.SharedTransformPath</i>	Transform path for QVD files
3.Load	<i>vG.SharedLoadPath</i>	Load path for QVD files
2.Config	<i>vG.SharedConfigPath</i>	Configuration and language files like Excel and txt. This folders could be shared to make configuration changes easier
3.Include	<i>vG.SharedIncludePath</i>	Folder where QlikView Include files are stored. These are script parts that are called from the main QlikView script.
1.BaseVariable	<i>vG.SharedBaseVariablePath</i>	Stores all the variables needed to use the framework, like paths inside the container
2.Locale	<i>vG.SharedLocalePath</i>	Locale for different regions, used for easy migration between regions
3.Custom	<i>vG.SharedCustomPath</i>	Store for custom include scripts
4.Sub	<i>vG.SharedSubPath</i>	Store for sub routines, this is a way to reuse code between applications
4.Export	<i>vG.SharedExportPath</i>	Folder used to store from QlikView exported data, probably txt or qvx
5.Import	<i>vG.SharedImportPath</i>	Folder used to store import data from external systems

Resource Containers 1.Project_HR

All containers that have a green container icon are Resource Containers. Resource containers are created by the *Deploy Tool*. These are the primary containers for an organization and can be utilized in many different areas.

Example Container (1.Example)

This is an optional container that is used for Qlik Sense QDF exercises that are available in the *Qlik Deployment Framework-Qlik Sense Exercises.pdf* document that will be installed using the *Deploy Tool*.