Item Manager Framework (IMF)

Future Proofing Example with Dependency Injection

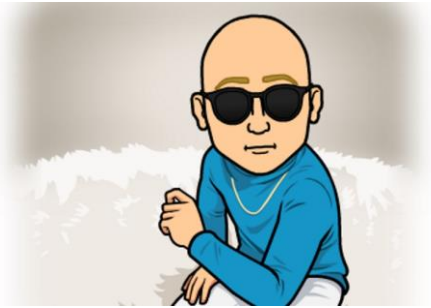
09/25/2019

Michael Klessens

Introduction

Michael Klessens

- Stress Automation Solutions lead at Intel Corporation (right now just leading myself :P)
- Certified LabVIEW Architect since 2006
- Presented CLA Summit 2012 on Dynamic Database Driven GUI
- And yes this is my real hair



Expected Outcome

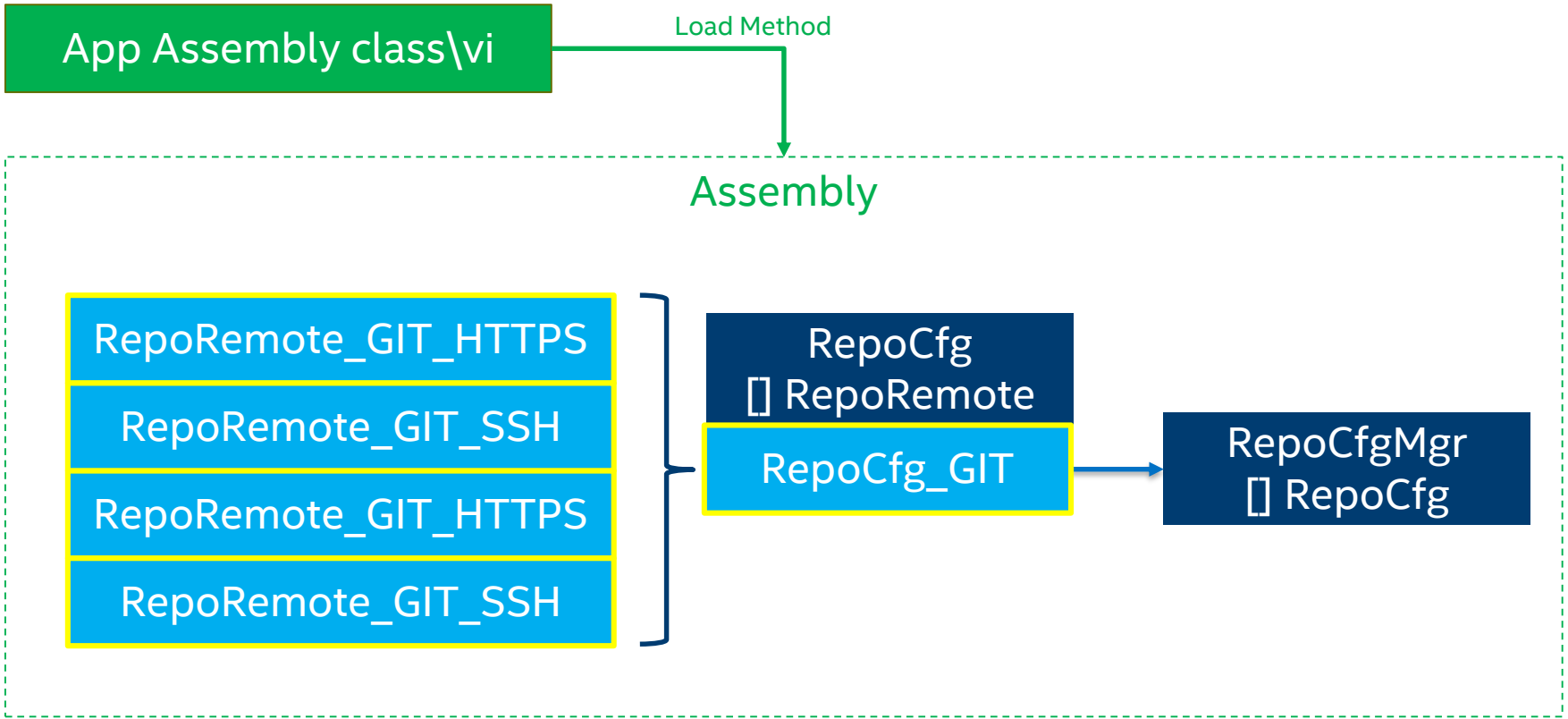
Dancing in the streets. Dogs and cats fist bumping

Problem Statement

We have a bunch of data that is being used in our application and we want to handle it in classes so we can implement methods to operate on said data

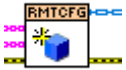
- One or more classes end up needing to be dynamically loaded based on some system or user configuration
- Need to be able to modify the data in the application
- Need to be able to load and save from different locations and format (throughout)
- Sounds generic – that's why it is a framework

Example Normal Approach



Item Manager Framework

- Assembler doesn't call children
- Parent Init method creates child



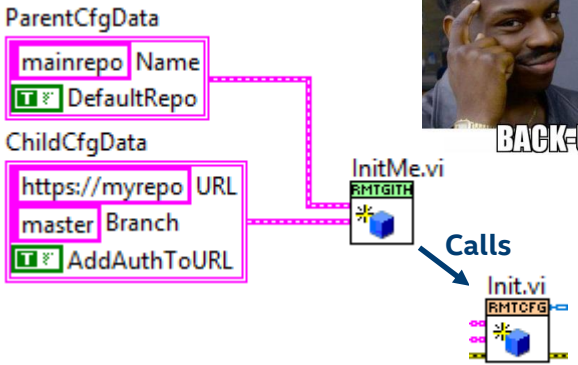
Item Manager Framework

RepoCfgFmt

RepoCfgFmt_File

RepoCfgFmt_DB

Load Method



Assembly

RepoRemote

RepoRemote_GIT_HTTPS

RepoRemote_GIT_SSH

RepoCfg
[ ] RepoRemote

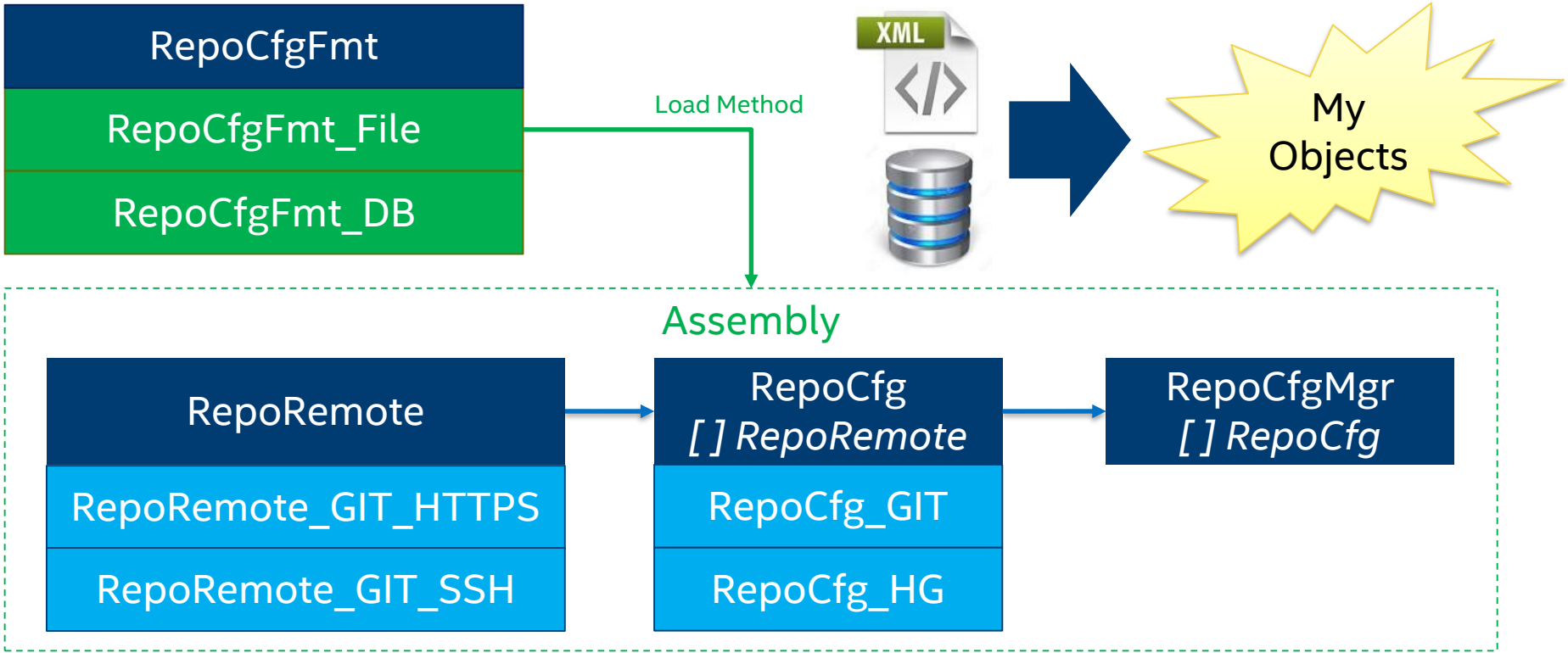
RepoCfg_GIT

RepoCfg_HG

RepoCfgMgr
[ ] RepoCfg

Item Manager Framework

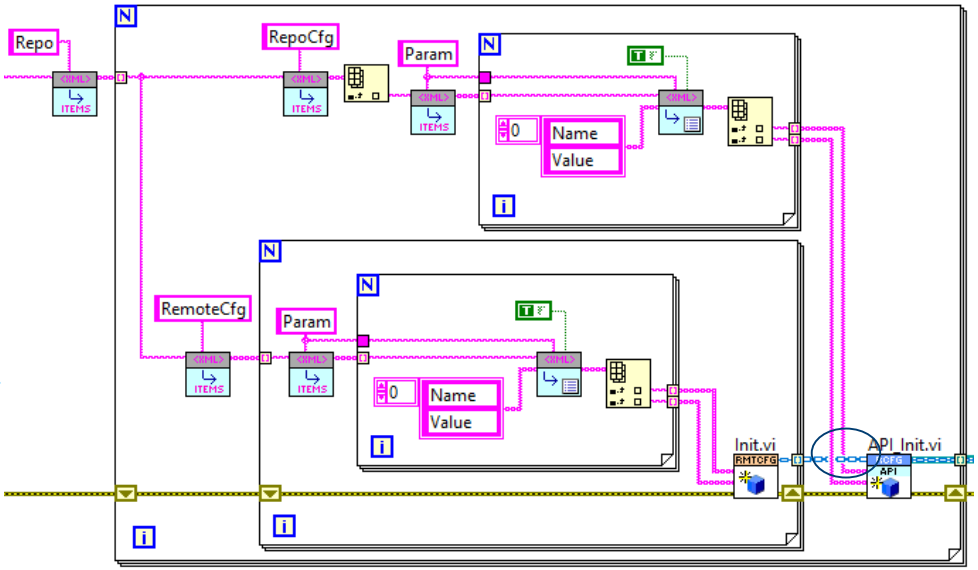
★ Assembler format is abstracted



Simple Assembler Example

```
<Repos FormatVer="1">
    <Repo>
        <RepoCfg>
            <Param Name="Name" Value="mainrepo"/>
            <Param Name="RepoType" Value="GIT"/>
            <Param Name="LocalPath" Value="C:\Projects\CLA"/>
            <Param Name="SubModule" Value="False"/>
            <Param Name="TrunkName" Value="master"/>
        </RepoCfg>
        <RemoteCfg>
            <Param Name="RepoType" Value="GIT"/>
            <Param Name="Name" Value="Origin"/>
            <Param Name="ConnectionType" Value="SSH"/>
            <Param Name="URL" Value="git@bitbucket.org:myproject/cla.git"/>
            <Param Name="DefaultRepo" Value="True"/>
            <Param Name="PrivateKey" Value="{programdata}\privatekey.key"/>
        </RemoteCfg>
        <RemoteCfg>
            <Param Name="RepoType" Value="GIT"/>
            <Param Name="Name" Value="Origin"/>
            <Param Name="ConnectionType" Value="HTTPS"/>
            <Param Name="URL" Value="https://bitbucket.org/myproject/cla.git"/>
            <Param Name="DefaultRepo" Value="False"/>
            <Param Name="UserName" Value="klessm1"/>
            <Param Name="Password" Value="ABCDEF12345"/>
        </RemoteCfg>
    </Repo>
</Repos>
```

- Different Keys Lookup Child Class
- Parameters can differ between children
- Can pass in composite\aggregate classes



Framework Classes

ItemBase and ItemBaseDVR

- Centralize code all items need
- Specifies required overrides
- ItemBase_DVR used with Mgr

ItemFormatBase

- Assembler (Load, Save)
- Acts as “Interface”
- Used by Mgr

ItemBase.lvclass

ItemBase_DVR.lvclass

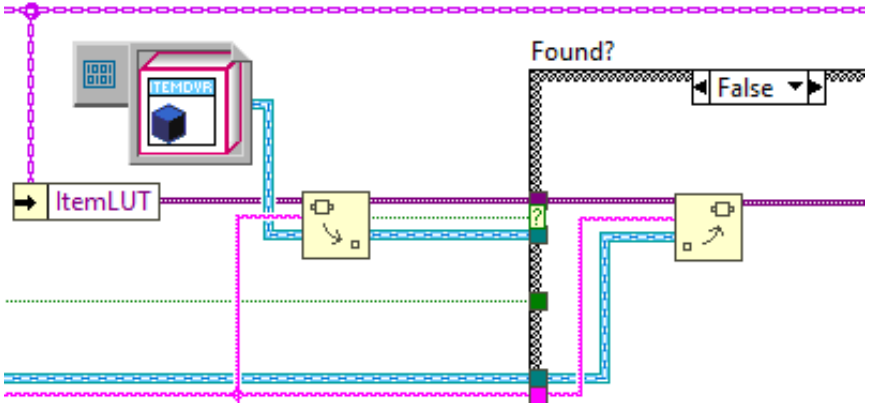
ItemMgrBase.lvclass

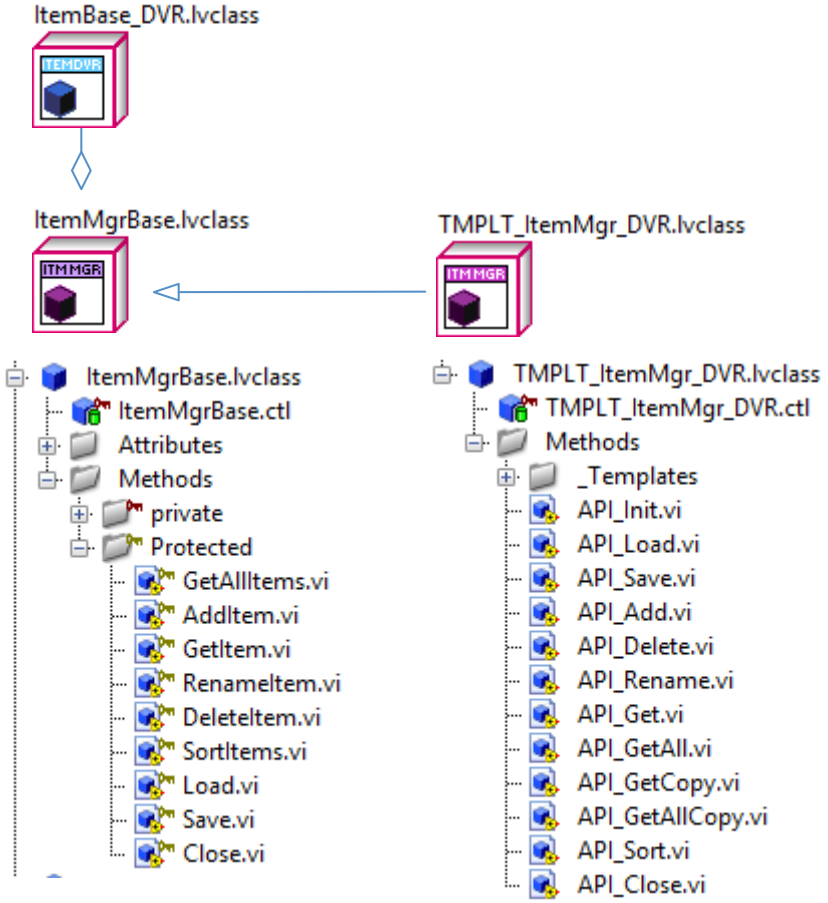
ItemFormatBase.lvclass

Item Manager Class

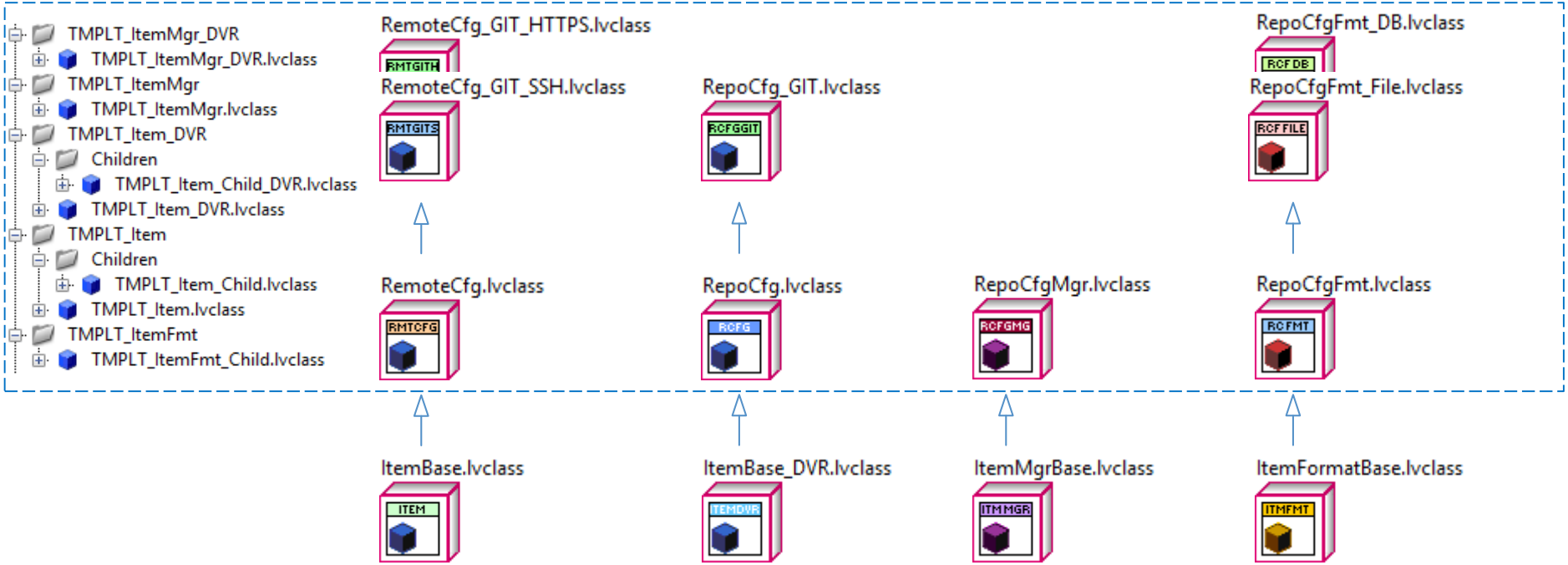
ItemMgrBase “manages” ItemBaseDVRs

- Uses ItemLUT to hold DVRs by “ID”
- Multiple methods for managing
- Template adds copy capability





Template is used to create children of framework



50/136 BL Classes in project inherited from these: 28 ItemBase_DVR

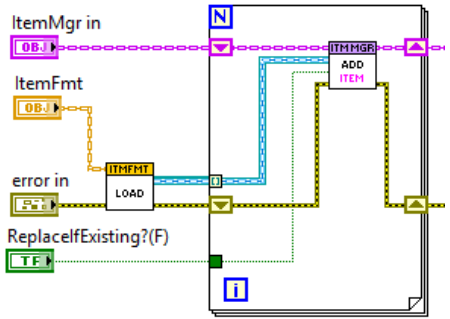
Item Format Class (Assembler class)

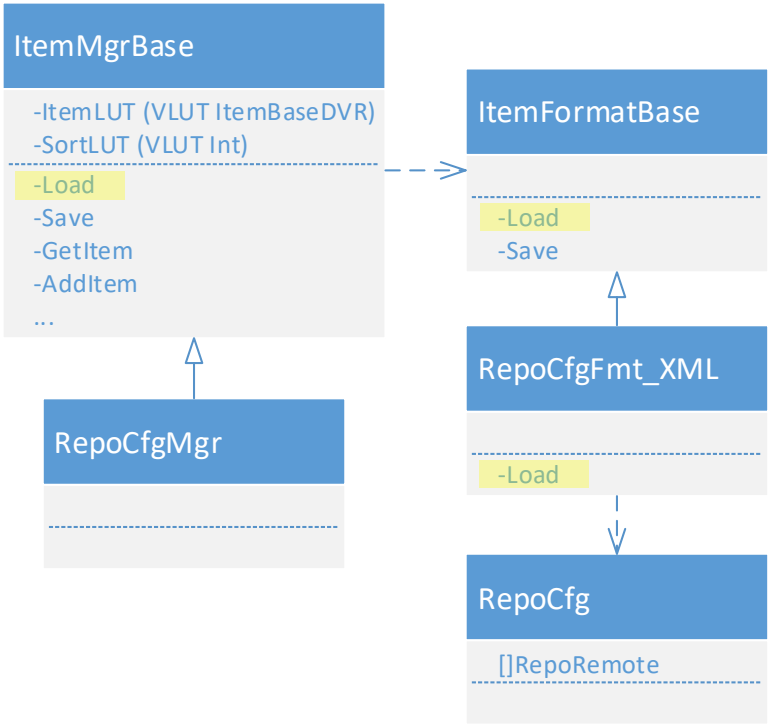
Item Manager “can” utilize ItemFormatBase

- Only if the manager needs to assemble
- Usually only in very high level managers

ItemMgrBase:Load calls ItemFormatBase:Load

- Then adds\replaces ItemBaseDVRs

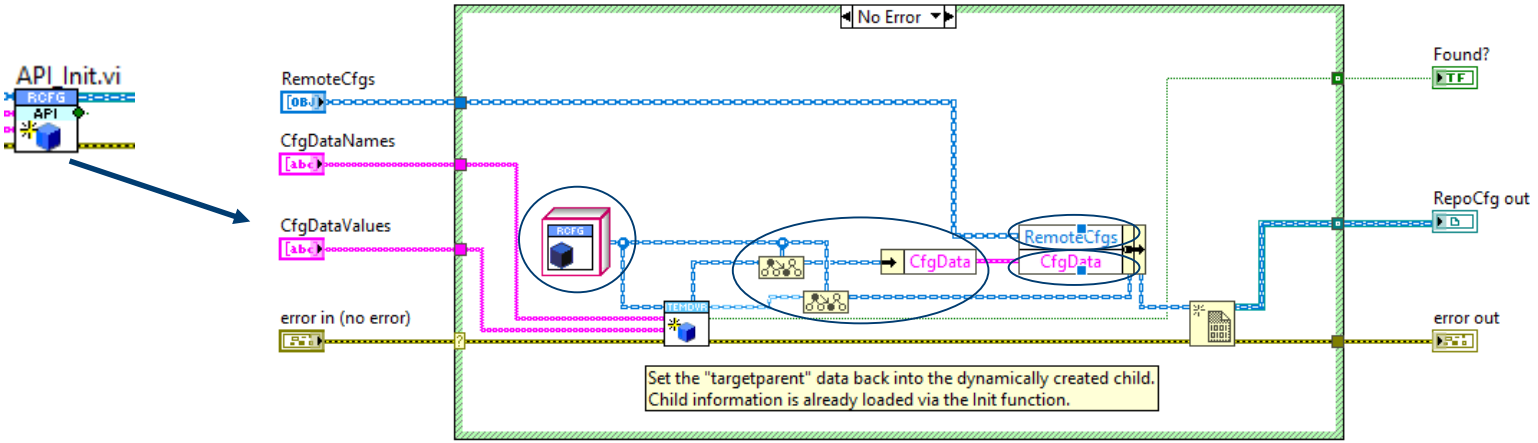




Loading Item Data

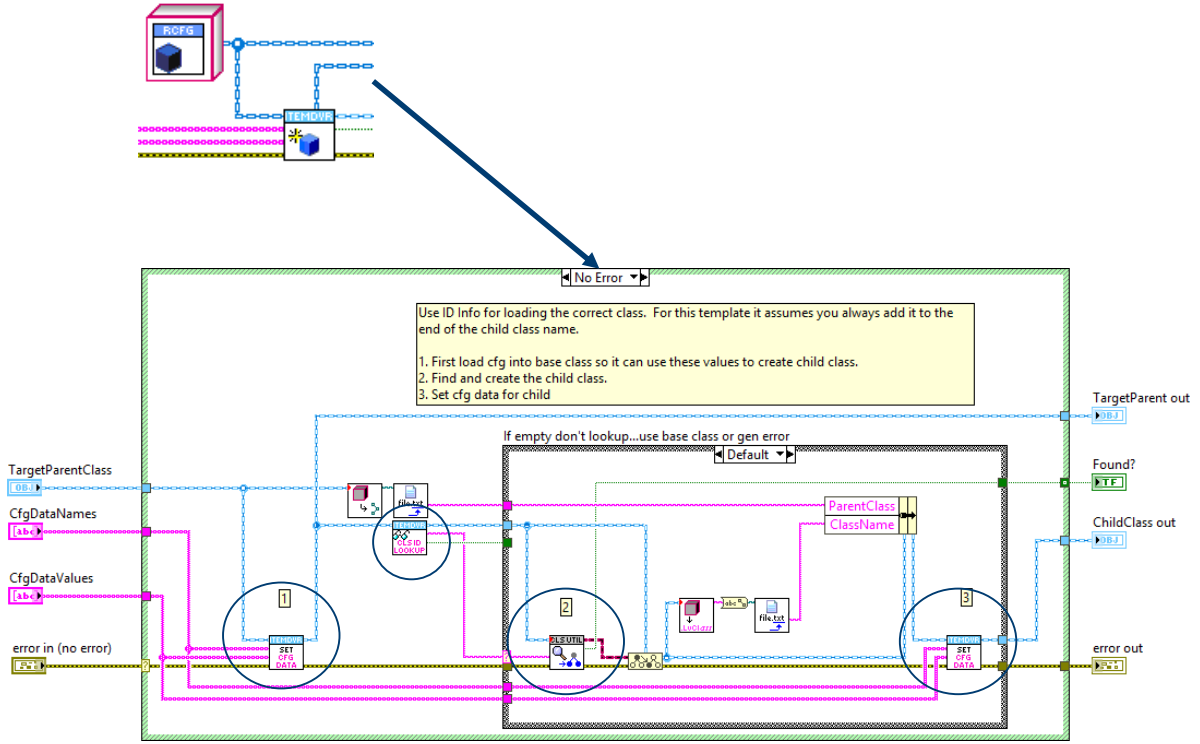
Item data loaded as name value pair to cluster by ItemBase or ItemBaseDVR

- Parent target class is passed in which is used to find the correct child
- Data is loaded into the parent as well as child: parent is set (is in template)
- Other data that is part of class can be set as well

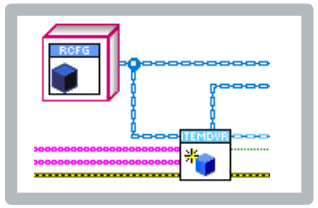


Loading Item Data – ItemBaseDVR Init Method

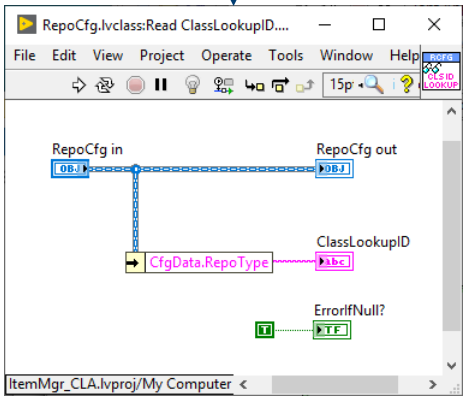
- Load CfgData into base class (e.g. RepoCfg)
- Get the class lookup info and use to find the child class on disk relative to the parent
- Find and load the child class
- Set the cfg data for the child class

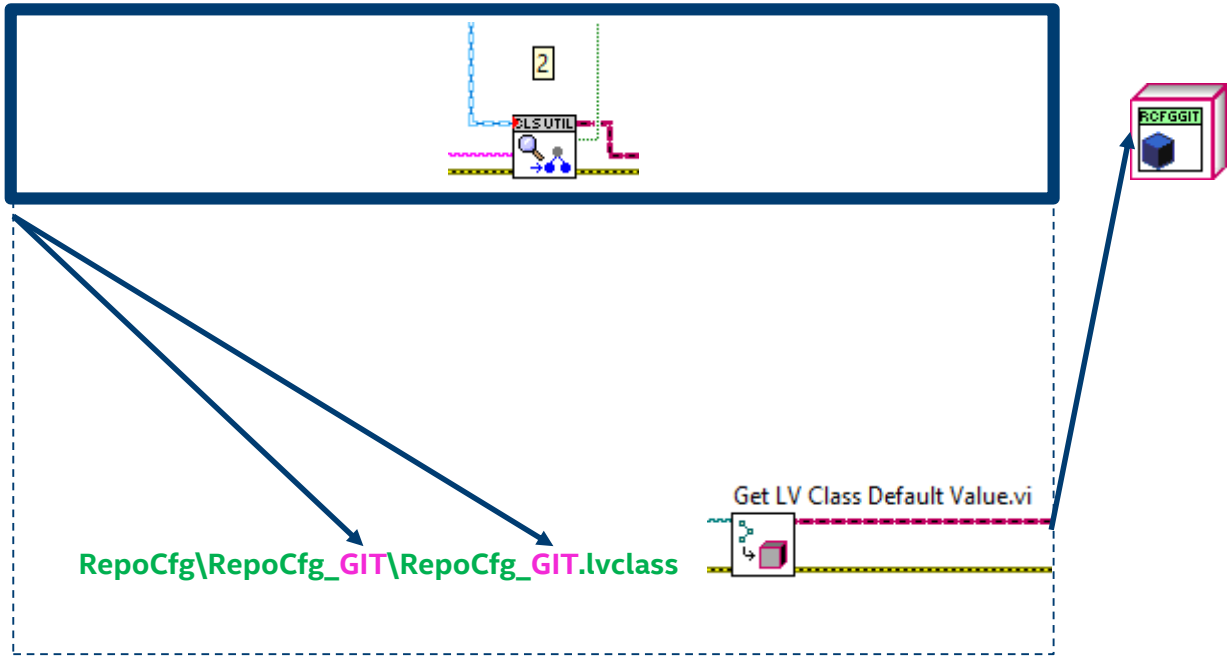


Loading Item Data – Looking up Child Class



Output -> GIT

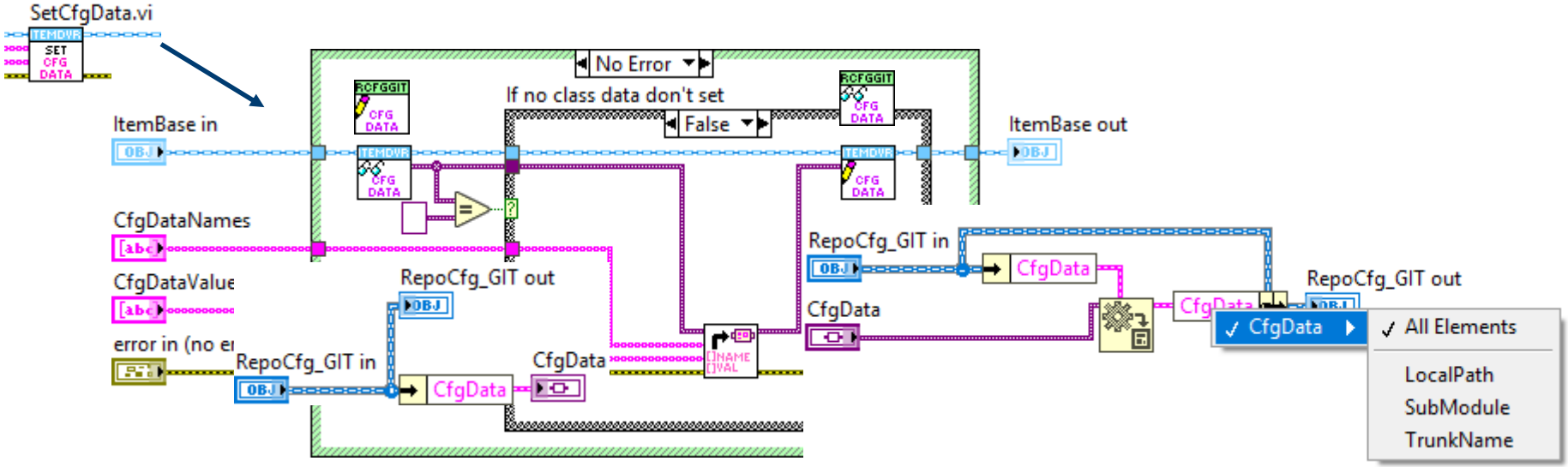




Loading Item Data – Setting Class private data

ItemBase class takes care of loading the data into the private data

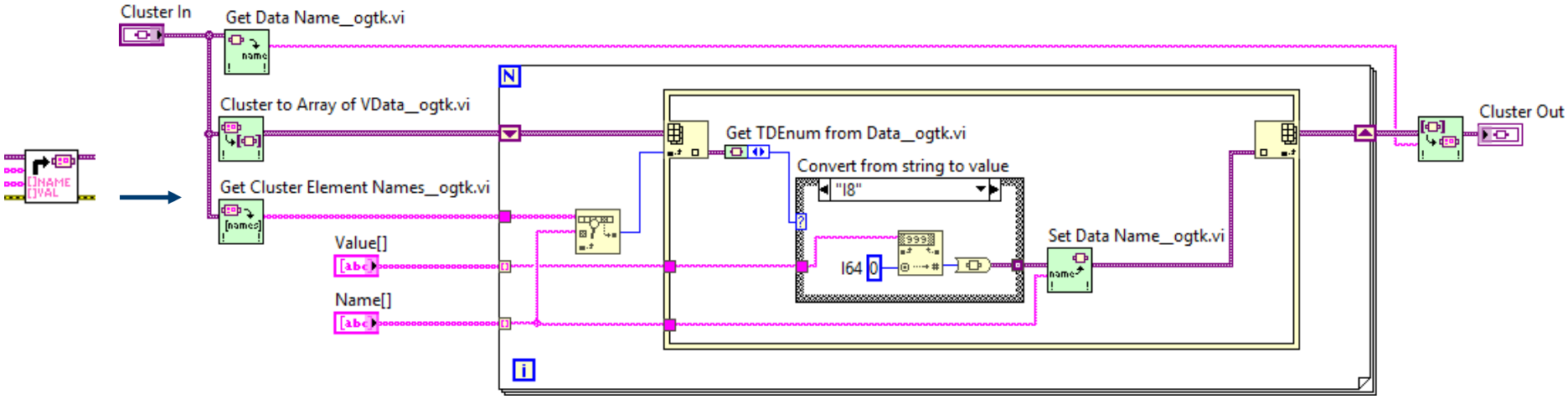
- Protected methods GetCfgData and SetCfgData (required override)
- Cluster is used for the data store for the “load\save” class data



Loading Item Data – NVP to Cluster

Reuse VI makes use of OpenG functions to set cluster data from NVP

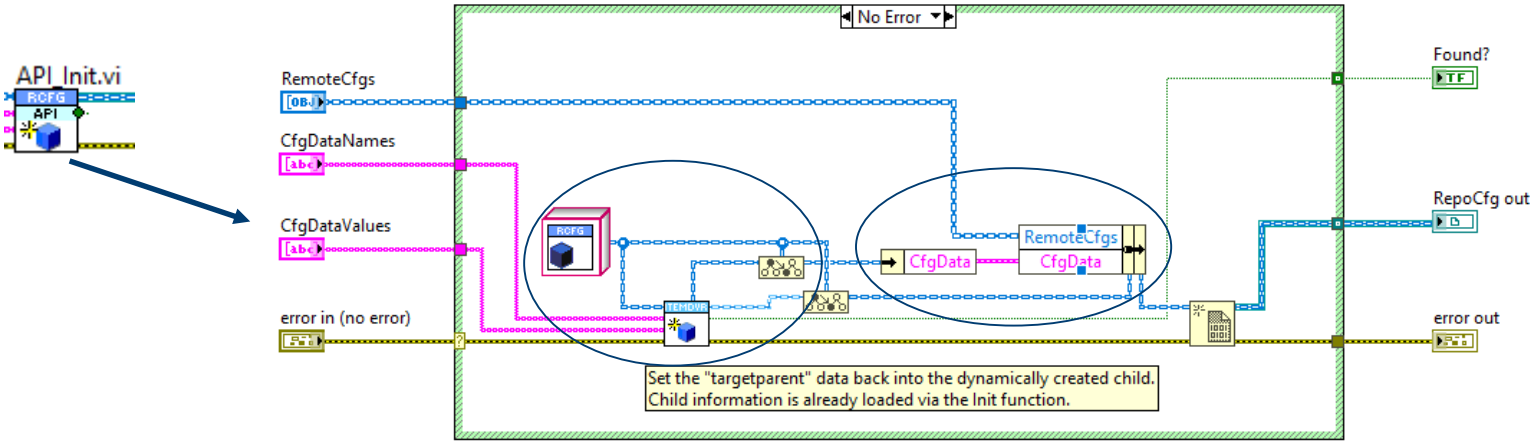
- Over simplification of the vi but the essence is there (checking removed)
- Nice as it does not require all matching elements and handles extras
- Most data types supported including enum, datetime, but not arrays



Loading Item Data - Summary

Creating the correct class and loading the data is mostly handled by ItemBase

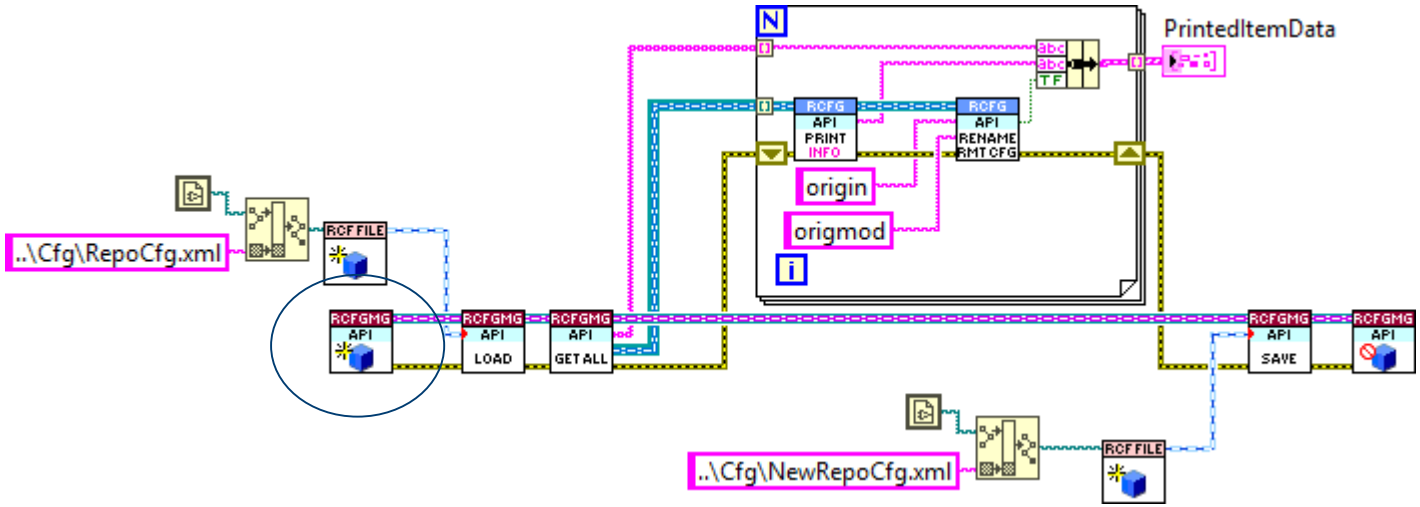
- Specific class is just responsible for setting its CfgData (in template)
- Also for initializing other composite\aggregate classes



Example – Loading · Displaying · Modifying · Saving

Create the class that will be managing your set of data: in our case RepoCfgMgr

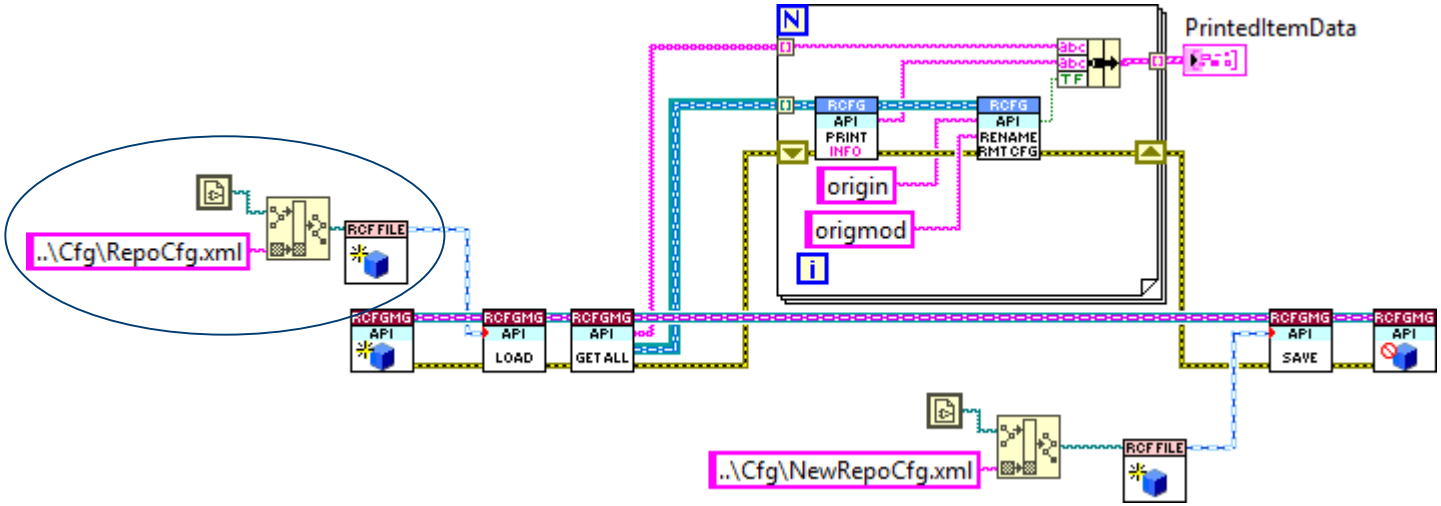
- In this example we are creating RepoCfgMgr to manage RepoCfg Items



Example – Loading · Displaying · Modifying · Saving

Choose the format and location of data we want to load

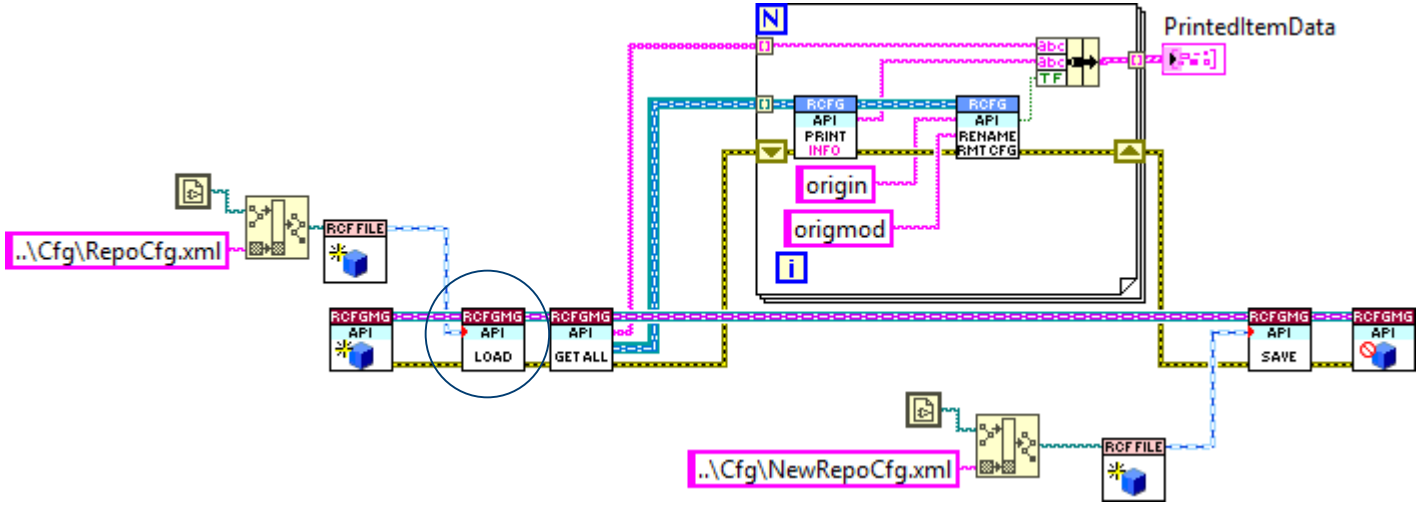
- This many times could be a configuration for the system
- Or through a GUI the user has a choice to pick (e.g. open from DB or file)



Example – Loading · Displaying · Modifying · Saving

Load the data into the manager from the data source

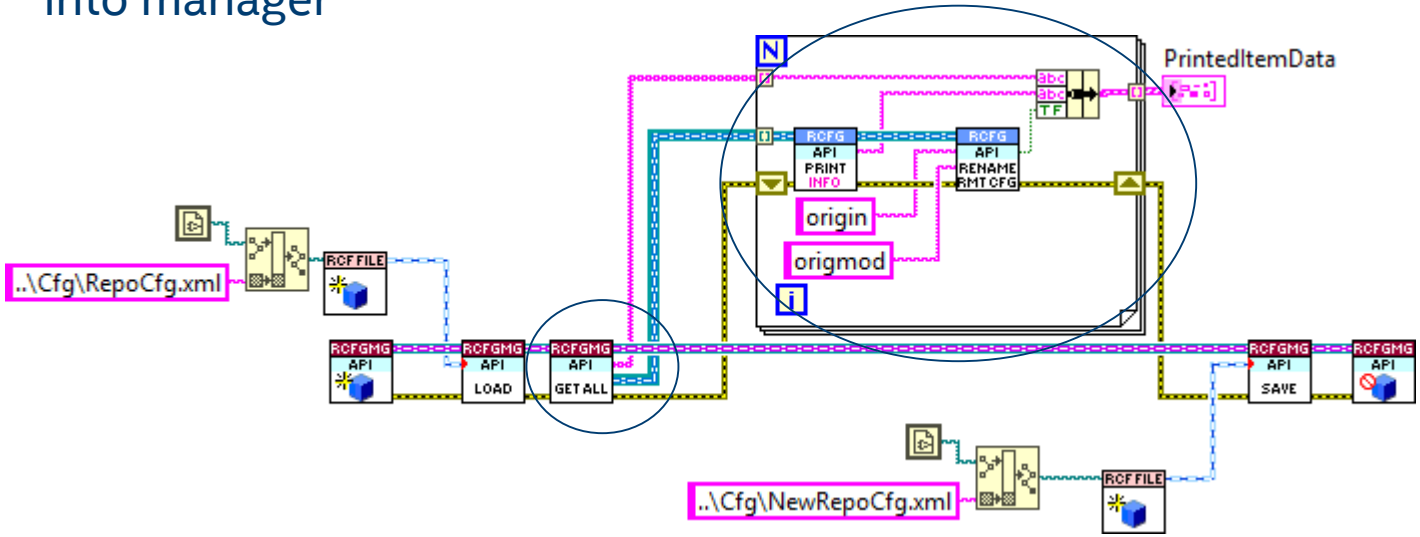
- Allows the manager to expose a load\save method but not contain code for the specific format of that data being loaded.
- Can load and save on the fly in application and be SOLID compliant



Example – Loading · Displaying · Modifying · Saving

Manager allows you to “manage” data by getting items to call methods on

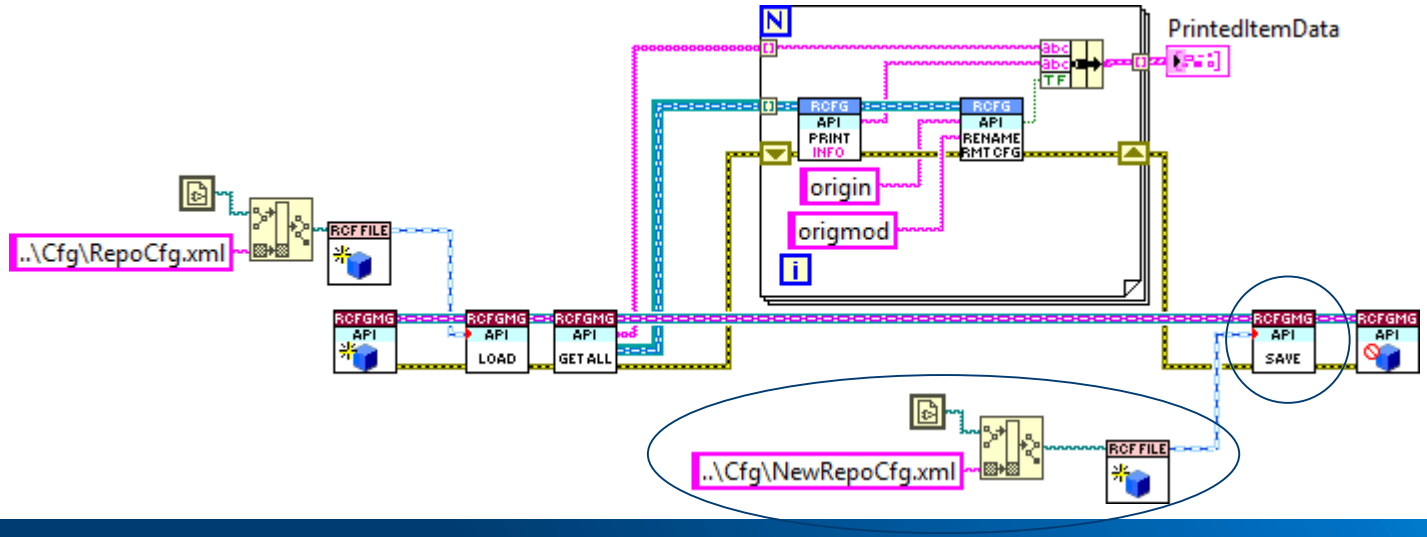
- Here we are getting all items, printing the info, and renaming a RemoteCfg
- The items are DVR so we can get by ID and modify without setting back into manager



Example – Loading · Displaying · Modifying · Saving

Save the modified data to any format

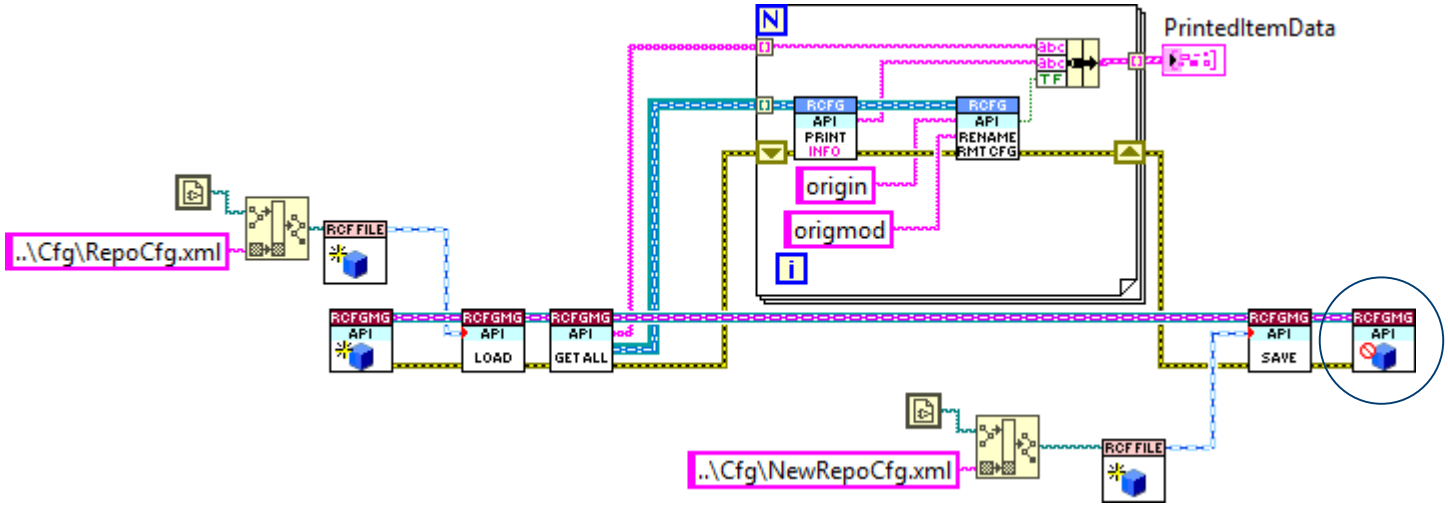
- "De-assembler" handled by the same format class
- Here we are saving to the same format, but this could be a database or ini



Example – Loading · Displaying · Modifying · Saving

Close will destroy all the item objects in the manager class (ItemBase handles)

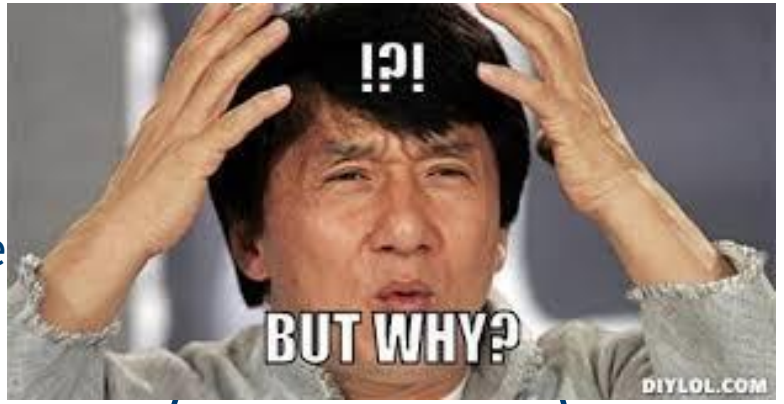
- Technically the manager didn't create the item classes (assembler did)
- But it is still responsible for the lifetime of the objects so we destroy them

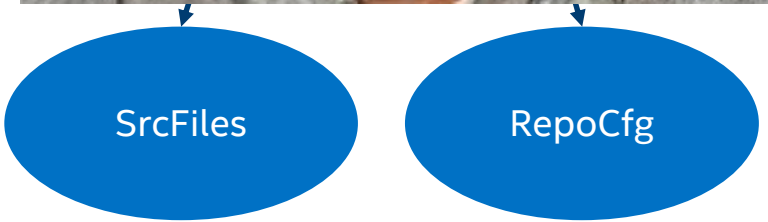


Example - Demo

Why use these Manager classes?

- Application with hierarchy of actors
  - In my case nested subpanels
- Can pass Manager or Item DVRs via message
  - Transfer is light
  - Can modify data without passing back
  - Initial try with non-DVR required sync





Pros

- Data loading into private class data made easy!
- Add\Delete private data with no parsing changes required
- I can use my private data in my class methods
- No format specific parsers where they shouldn't belong
  - RepoCfg.lvclass:LoadFromXML, RepoCfg.lvclass:LoadFromDB
- Simple overrides enables migration file -> DB based app
- Adding children for lookup is easy
  - Save RepoCfg_HG.lvclass -> RepoCfg\RepoCfg_HG

Cons

- Without a wizard new ItemMgr group time consuming
  - And Michael didn't make one before this conference
- DVR is nice in some instances but…
  - Too many "API" methods required in middle classes
  - Try copying deep aggregated classes (Mgrs in Mgrs)
- It is a little file heavy for the all components
  - Mgr(13), Fmt(4), Item(13), ItemChild(6)
- Missing some monitoring and debug helpers
- Not as fast as other data loading approaches

Next Steps

- Investigate a better way without class DVRs and possibly utilize maps
  - As well as not duplicating DVR\NonDVR base classes
- Could standardize on data format like JSON instead of name value pairs
- Get the code in bitbucket or github pending Intel approval
- Links: google.com search for memes


intel®
experience
what's inside™