Manual Version 2.50

Preface
- Goals
- Assumptions
- Long Term Benefits
- How our business is working
- Simplified Workflow description

CAPP Structure
- CAPP ®
- Overview
- Apache Velocity
- Telosys
- Stay connected
- Sample System

CAPP Generator
- Definitions
- How to control the generated Code
- Template
- Global System Parameter
- Local Module Parameter
- Module Variables
- Field Parameter
- Field Attributes
- Embeds
- Data Storage

Meta
Version History

Document Revision 2.50 © Dettmer Informatik GmbH & Co. KG June 2020

Version 2.5

- Application View uses a treeview now
- Licensing was changed and made easier
- Embed view now has a source editor with syntax highlighting
- Free to use edition with only very few limitations

Plugins used for Version 2.5 do not allow to offer CAPP as Open Source in complete any more. Finally the existing Community Edition shows the principles of work in general and runs completely without any embed in pure XOJO.

Version 2.5 is a licensed product, but it is and will be free to use for everyone.

Version 2.43

First commercial version

Community Edition 4.3

First and only Open Source Version (for Mac only)

Preface

Using templates is a common shortcut for any developer already. Designing database applications makes this principle even more important. CAPP is a tool based on Apache Velocity and its command line interface TeloSys to create a database application from any existing data model without coding. This works with templates which are completely transparent and adaptable to your needs. CAPP is a registered trademark and an abbreviation of the term "Computer Aided Program Production".

Goals

- CAPP wants to automate the process of coding as much as possible
- A database will provide the basic structure and attributes
- A framework will provide nearly 100% of the required code
- Only project specific code will be entered manually
- This code can be inserted in the generated app as so called embeds
- These embeds can be edited in the destination system either
- The changes are given back to CAPP
- Code modifications made this way will be used in the next generator loop
- The output code can be deleted at any time completely
- CAPP is open and usable for any destination language
- CAPP itself can be used on Mac OS and Windows PCs

Assumptions
Creating programs nowadays is like a Lego® story: Setting up a new system always requires a set of different components, which are building something new if put together. But it is not like Lego® because the pins are missing or incompatible. Every language and each environment has its own pinning and setting.

Additionally most of the real life applications are somehow database centric. This leads to boring repeated tasks with regard to tables, fields, relations and variables.

If a bug is found and the problem is solved, the solution has to be implemented in a lot of modules - in the worst case for each customer and for each system.

Due to the lack of structure and modularity it is often difficult to maintain systems written by other developers.

These facts are common to all programming languages and development environments and cost a lot of motivation, time and money. We designed CAPP as a system to improve all features of an environment in a way that minimizes the above mentioned inconveniences as much as possible.

**Long Term Benefits**

- Advantage of mature and bug free generated code
- Very good maintainability due to structure and modularity
- No stupid simple database coding any more
- Improvents can be propagated to all running systems automatically
- Keeping the freedom of editing any code directly at any time

**How our business is working**

As described a developer works in some very simple and always repeated steps:

- Ask the customer for his specification
- Look for similar systems you already developed
- The more similarity the less work for you
- Make an offer with having taken in account the "always there" part - or not
- Being competitive is decisive weather you will get a contract - or not
- If you got the contract - may be it is because you have already prepared something
- Make a demo with a prototype based on the existing framework
- At least now you should get the contract or order
- Ask for further requirements
- Start the serious and final work on the system based on your framework
- Deliver the product

This implies some rules like:

- The more systems you already have made the better your company will work
- The more structure and systematics you use the easier the adaptions will be
- The more work is done inside the framework the less work is left

**Simplified Workflow description**
- Take any system currently not under development in text form as **foundation**
- **Copy** this system as a template
- Make a "**dry run**" without any changes to check
- Introduce all parts to be modified as **Velocity variables** (i.e. fields, tables, menus)
- Insert Velocity **statements and commands** into the template
- Connect to **data input**
- **Run** generator
- **Check** and **compile** code again
- Place **embeds** into templates where needed
- Fill embeds with the **specific** code
- **Compile** code again
- Make **changes** in the generated system inside the embed "gaps"
- **Get back** the changes into CAPP
- **Change data structure** and template(s) and run again including changes on the destination system

**CAPP Structure**

**CAPP ®**

The Xojo Application to build Xojo code (or code in any other language like C, PHP etc.), CAPP is the abbreviation for "Computer Aided Program Production". There is an Open Source Edition available for the community (Mac OS only). A subscription model for professional or enterprise usage is available as well. A template repository with free templates will help any developer in any language to have a starter pod.

**Overview**

These parts are used to setup CAPP as a tool:

- Apache Velocity Template Generator
- Telosys Command Line Interface for Velocity
- CAPP - a XOJO App to handle the system
- Germ Application(s) as foundation
- Templates for several destination languages

CAPP is available as an Open Source system, a Professional Version and an Enterprise Version. The Open Source Version is based on the open tools Velocity ad Telosys and is available on GitHub:

[https://github.com/MickMaster/CAPP-CE](https://github.com/MickMaster/CAPP-CE)

All germs and templates for any language will be Open Source as well.

**Apache Velocity**
Velocity is an Apache Project for a Template Generator. It is Open Source, written in Java and brings its own template language for generating code with repetitive content, i.e. databases and their fields. A short overview can be found at WikiPedia:


The Project itself is an "Apache Featured Project" and was last updated October 2018. The description about what a template engine is about can be found here:


The schema how a template engine works like is as follows (Image Copyright WikiMedia):

There are four **artefacts** used in Velocity (left to right and top down):

- Input files (also called **context** in Velocity) in different formats
  - Direct SQL connection to nearly all common systems
  - XML definitions as flat files
  - ASCII flat files
- Prepared Templates in the destination language
- Template Engine generating output code
- Result Documents can be generated in any language and syntax

The generating process is based on either **text files**, which contain the database information for the template engine or Velocity is **connected directly to a database**. CAPP uses the direct database connection only yet.

**Telosys**

Telosys is an Open Source **Command Line Interface** (CLI) for Apache Velocity. It is written in Java and can be used to initiate Velocity with a very small footprint.
Velocity is plain Java code and has "no interface". It is usually working as a plugin for an IDE like Eclipse. Telosys was developed to help with a kind of wrapper to be able to control Velocity with a command line. It includes Velocity Version 1.7 already, so there is no need to install Velocity on top.

https://www.telosys.org/index.html

Velocity and Telosys are both written in Java, but this does not mean that Java is involved here or you have to learn Java now. Only the installation of the Java Runtime Environment JRE is required. This can be downloaded here.


Please take care to respect the new Java License conditions. Everything will work also with the Java Open Source Edition which is available at

https://openjdk.java.net/

You will have to install Java manually if you want to use this license model.

**CAPP** takes care about the complete environment and control of Telosys and Velocity. It writes all the required files and calls up Telosys in a shell. No further work to do than giving the required information about database, paths, variables and the structure of your application.

The Velocity template language is used to automate the coding for Xojo. This language is easy to use and has only very few commands - which are quite powerful if used properly. A complete reference with quickstart and reference manual can be found at the Apache Foundation:

http://velocity.apache.org/

The User Guide for Version 1.7 can be found here:

http://velocity.apache.org/engine/1.7/user-guide.html

**Stay connected**

Most important for you: If you do not want to work with CAPP any longer, you can use your templates and work i.e. with Eclipse, the famous IDE for Java. So nothing will be lost at all, ever. What you will miss is the ease and comfort CAPP functionality.

**Sample System**

For the current template set an application was used named **CAPPgerm**, which is also delivered as open source. This application contains the "germs" for the later generated output files. At the moment there are three major templates in use:

- Table Views (TABLE) generates 1 to n listbox windows
- Record Views (RECORD) generates 1 to n masks for editing single records
- Menue (MAIN) copies ressources and generates a menu structure

The sample system is available also as a subtree of CAPP:

https://github.com/MickMaster/CAPP-Germ
This germ is a simple code with no plugins or add-ons - it is only pure XOJO code dealing with a SQL connection. This allows to generate very lean systems with no need for any additional purchase.

**CAPP Generator**

**Definitions**

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>System</td>
<td>The main application</td>
</tr>
<tr>
<td>Module</td>
<td>All the smaller items used to build the system</td>
</tr>
<tr>
<td>Template</td>
<td>A Velocity macro used to generate code from</td>
</tr>
<tr>
<td>Parameter</td>
<td>Value given to a template to control generating through variables</td>
</tr>
<tr>
<td>Variable</td>
<td>Name used in Velocity (like $test) to store a parameter/value/attribute</td>
</tr>
<tr>
<td>Attribute</td>
<td>Values of a variable assigned to a field</td>
</tr>
<tr>
<td>Embed</td>
<td>Code snippet to be pasted at a defined location</td>
</tr>
</tbody>
</table>

**How to control the generated Code**

There are **seven** ways to control the generating process at the moment:

1. Templates as a framework
2. System Parameter
3. Module Parameter
4. Module Variables
5. Field Parameter
6. Field Attributes
7. Embeds

The higher the number is the less is the impact on the generated system. A template change (1) can influence the **complete** system - while an embed is very dedicated to a combination of template, parameters, variables and the assigned **module** (7).

**Template**

Write or change the template file (with the postfix .vm in the /template/language/ directory). Sample: If you want to create an entry mask, take your existing code and save it as "mask.vm" into this directory. Replace all variable fields with Velocity variables like "$test". See the samples and the demo code.

You can add any editor as default for calling up the template.vm files. See also CAPP reference.
Global System Parameter

Variables which are set on the system level and are available throughout the whole system in all templates and modules. Sample: The name of the system is stored in the Parameter SystemName or you could code the name of your customer into a parameter called CustomerName. These will be generated to Velocity variables named $SystemName and $CustomerName like

```
#set ( $SystemName = "Testsistem" )
#set ( $CustomerName = "Mister Geek" )
```

System parameters can be set in the CAPP system settings.

Local Module Parameter

Variables which are set on the module level and are available in the specified module only. They are generated like the System Parameters above from the CAPP database.

Local Module Parameter can be set in the module view of CAPP.

Module Variables

Variables are set in the templates. They are set dependent on the parameters valid for this module.

```
$cols = 8  \' Number of columns for tables or masks
```

They can be added in the template code - and have to find their counterpart in the Local Parameter of the module, set in the Module View.

Field Parameter

Variables which are set in the comment field in a table. They are to be defined in the template before as Module Variables. If you put the string

```
$skip=true;$label="Field"
```

into the comment column of a field in the table definition, the assigned field will be skipped and get the label named "Field". (Not all DBMS and not all Database-Editors allow using these comments). Assignments are separated by ";" - except at the end.

**Sample:** You want to hide some content from being used in the templates or you want to set a field to read only:

Define these two variables at the top of your field-template:

```
#set ( $hide = true )  \' set and ignored until overwritten by field content
#set ( $ro = false )  \' same ...
```
Then use these two variables to control your template output:

```
#if ( ${hide} == "false" )
   This text will appear only if "hide" is false!
#end
#if ( ${ro} == "true" )
   display field in read/only
#else
   allow editing field code
#end
```

So these variables will get the value assigned in the first two lines above. If there are assignments in the field comment of the table, the **values used there will be taken**. Last not least the decision made in the second code piece will generate the appropriate code.

These variables are like the famous **Baron Münchhausen** who jumped from one cannonball to the other to keep track of the enemies positions in a battle. He travelled back the same way.

So these variables **start as text** in the **comment field** of the database table, jump into the **template** where they are **executed** and influence the finally generated code - which will be executed.
Caution: What you enter here is executed AS CODE in the template which uses it. If the Velocity syntax is not right, the production of the code will fail.

Field Attributes

Field attributes consist out of a variable and the assigned value. They have to be declared (#set) before use. These attributes are defined already in the delivered templates:

```latex
$format' (string) sets the format attribute, default empty
$mask' (string) sets the entry mask, default empty
$just' (char) Justification (l)eft/(r)ight/(c)enter, default default ;-
$hide' (boolean) Generated but not visible true/false, default false
$ro' (boolean) Field visible but not writable false/true, default false
$skip' (boolean) Ignored in generation process false/true, default false
$width' (integer) Width in pixels, default default
$height' (integer) Height in pixels, default default
$fgcol' (hex) Foreground color, default default
$bgcol' (hex) Background color, default default
$label' (string) Descriptive text as label, default <field name>
```

Not yet implemented:

```latex
$tpl' (string) Template specific selection flags
```

Embeds

Parameters can be used to set variables to make decisions about using a piece of the template or not. Now we are going to put completely manual code in our module. This can be established by using the so called "Embeds". An embed is a code snippet, which can be placed at any defined location in the code.

Embed locations are to be defined inside a template like:

```latex
#embed( "TABLE After setting fields" )
```

This one belongs to the template "TABLE" and is located after the generated code which sets the fields. It is important that all embeds are named unique - best practice is to start with the name of the template it contains in upper case.

If a template is assigned to a module, the template is scanned and all embeds included are inserted into the CAPP embed database table for this module. If the template is changed, the data is kept in the database, but marked as "orphaned" in the embeds list.

Data Storage
All CAPP system data is stored in an **unencrypted SQLite database** (CAPP.sqlite). This file can be modified and damaged however. The CAPP database exists as often as systems are generated plus one. The origin of all CAPP system databases is located in the library directory of the user (or in the "documents" folder). There the list of **systems in use** is stored.

If a system is created, this origin directory tree is copied to the systems path. It contains the complete tree with all the TeloSys code inside. It also contains also the Sqlite file for this specific system which contains the application structure.

This allows to run systems with different TeloSys versions and different templates.

If CAPP is installed from scratch the system database is empty. If you already have systems on your disc, you can insert them by selecting the directory.

**Meta**

- All ideas and information are a matter of ongoing changes.
- All rights of other Trademarks are fully respected
- No rights granted to anyone for the open source part (also templates)
- All rights reserved by Dettmer Informatik GmbH & Co. KG.