CAPP Version 2.30 Manual

Abstract

Preface
Assumptions
Long Term Benefits
How it works
Simplified Workflow description

CAPP Structure
CAPP ®
Overview
Apache Velocity
Telosys
Sample System

Installation

CAPP Generator
Definition of Terms
How to control the generated Code
Template
Global System Parameter
Local Module Parameter
Module Variables
Field Parameter
Field Attributes
Embeds
Inserting a new Embed
Removing an Embed
Changing the Template
Embed Structure

Data Storage

CAPP Reference
General Behaviour
Main Window
Environment Window
System Window
Language
Connection
Database
Startmodule
Velocity
Templates
Parameter
Application Window

© Dettmer Informatik GmbH & Co. KG, 2019 - www.dettmer-informatik.de
Abstract

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".

This is the manual for CAPP in general, containing all Information for the Community Edition (CE) and the Professional Edition (PE).

Preface

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 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
- Improvements can be propagated to all running systems automatically
- Keeping the freedom of editing anything at any time

### How it works

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

- Ask 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 whether 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

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 adaptations 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
- Copy this system as a template
- Make a "dry run" without any changes
- 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 Community Open Source Edition available. A subscription model for professional or enterprise usage is already or will be available as well. A template repository with free templates will help any developer in any language to have a starter pod.

CAPP is a development tool to control code production by using Apache Velocity in an usable way. It keeps you away from configuration issues and control stuff needed by Java and Velocity by default.

**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.

[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:

https://en.wikipedia.org/wiki/Apache_Velocity

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:

https://en.wikipedia.org/wiki/Template_processor

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 different formats
  - Direct SQL connection to nearly all common systems
  - XML definitions as flat files
  - ASCII flat files
- Prepared Template 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.

**Telosys**

Telosys is an Open Source Command Line Interface 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](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.

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

Sample System

For the current template set an application was used named **CAPPgem**, 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 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 first.

Installation

The installation procedure is described in the Quickstart section. CAPP will be installed preferrably into the /User directory ("all users") or directly in the users directory /User/Username.

**CAUTION**: CAPP will not be installed into the default Application Directory. This is caused by the current structure which has to serve TeloSys and Velocity all together.
Below the CAPP main directory the four directories above can be found. They contain what they are named for. Most important one is the ./ctl directory - it contains Telosys and Velocity and the required templates below in the directory ./ctl/TelosysTools/templates/xojo. In the templates directory may be also other languages than XOJO later on.

CAPP Generator

Definition of Terms

<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 system or module 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, i.e $height=&quot;100&quot;</td>
</tr>
<tr>
<td>Embed</td>
<td>Code snippet to be pasted at a defined location</td>
</tr>
<tr>
<td>Context</td>
<td>Database Information delivered via Variables containing Attributes</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 (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.

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

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

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.

Module Variables
$cols  ' Number of columns for tables or masks

Field Parameter

Variables which are set in the comment field in a table. They are to be defined in the template before. Sample: You want to exclude some fields from being used in the templates or you want to set a field to read only:

Put the string $skip=true;$label="Field" into the Comment of a field in the table. (Not all DBMS and not all Database-Editors allow using these comments). Assignments are separated by ";".

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

```python
#set ( $height = "100" )
#set ( $ro = false )
```

Then use these two variables to control your template output:

```python
#if ( $height != "0" )
#set ( $somevariable = "Height was set to ${height}!" )
FieldName.Height = $height
System.Debug $somevariable
#end

#if ( $ro = true )
FieldName.enable = false
#end
```

These variables are like the famous Baron Münchhausen who jumped from one cannonball to the other to have a better overview about the battlefield. When he saw that he will come down behind the enemies lines he simply jumped on one of the enemies bullets to fly back. So these variables start as variables in the template - go into the comment field of the database and jump back into the template and from there they were used for generating the final code which is executed. Münchhausen won the battle of course.

Field Attributes

Field attributes consist out of a variable and the assigned value. They have to be declared (#set) before use.

$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
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:

```
#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 which are included are inserted into the CAPP embed database table for this module. There are some side effects of this system which you will have to know:

**Inserting a new Embed**

Simply edit the template .vm file and insert the line above into the template at the place where you want it to go. If you call up the module the next time, the embed will show up in the embeds list automatically. Please take care about the naming of the embed. It is proven as good practice that you take the name of the template as the first word (above: TABLE). The second word should say if the template is **before** or **after** something else. And the rest describes the position inside the code.

**Removing an Embed**

If you want to remove an embed from the template, the entry in the CAPP database will still remain existing. Due to the fact that it can not be found in the .vm code anywhere, the entry in the database is marked as **ORP** for orphaned. This has no effect except that your code is not deleted. The ORP marker is for information only and has no effect on the generation process.

**Changing the Template**

Not yet implemented:

```
$tpl   (string) Template specific selection flags
```
A very similar effect occurs, if you for example used the template TABLE first, but then change this to RECORD. Any embed which is not empty and belongs to TABLE will be orphaned then. If you switch back vice versa. Sounds strange but is very useful - you will see.

**Embed Structure**

Each embed is terminated with the string "-+-" (minus plus minus) in the destination code. In XOJO this string is preceded by the comment character. So if you edit the destination code take care about writing it between the "Embed-Statement" and "+-"!

If you do not do so, CAPP will not have a chance to bring back your code into the embed database and from there back to the newly generated code.

**Data Storage**

All system data is stored in an unencrypted SQLite database (CAPP.sqlite). You can look into this structure for yourself in the Community Edition CAPP-CE. There are four very simple tables:

- Systems
- Modules
- Parameters
- Embeds

Sounds very similar to the above mentioned seven ways to manipulate code, does it? **Systems** contains the overall system data for the generated app. Finally thes contains the "1" for the first and only system in the community edition. If you want to work with multiple systems in CE you may have to make one copy of the database per system.

**Modules** contains one line for each Velocity call. This may end in one or more pieces of destination code. This is the table which is edited in the Modules window and which contains all of the data needed by TeloSys and Velocity.

**Parameters** contain the name of a parameter, the content and where it belongs. This is not to be confused with the variables used in the templates! A parameter can become a variable and it’s content will be the content of the variable, but it does not have to. The login for the database is a parameter which will never show up in the code - except you want to have it. Then it is becoming a variable.

The **Embeds** are pieces of code which are stored in the Embeds table. Always together with the System- and Module- Numbers.

Please take care editing the database directly. You may cause inconsistencies which will prevent a correct behaviour of CAPP. If you encounter strange effects, the simpliest way to find out is to go into the wizard and restore the database.
Choose “Step 1” to restore the original setting and generate again. Your previous database is not deleted but saved with a random number in the name of the SQLite file.

**CAPP Reference**

**General Behaviour**

Some aspects are used throughout the complete CAPP system. All windows can be opened by clicking on the icon or by typing the **first capital letter** of the option. For example pressing "E" on the main window will call up the **Environment Window**.

If a window is open, the **ESC** key will terminate the window - or even CAPP completely. CAPP is designed to be used with the keyboard mainly. **ESC** will terminate a window and its entries. **ENTER** will terminate and save all entered data.

The Only exception is the **About Window** which can be called by pressing "B".

On each window will be a help line in the lower part of the window. If implemented it will show a short tip for each field you move on - or even other helpful information.

If there is a table to be edited, the "I" key and the "D" key are important. They **Insert** and **Delete** records in the table. **Caution**: Deletions cannot be undone!
Main Window

The Main Window allows to setup the working **Environment**, configuration of the **System**, setting up the structure of an **Application**, starting the **Generate** process and getting back the embeds with the **Loopback** function. On the right side of the window you find the wizard described later and the About-Box, containing all the valuable Open Source contributions used for CAPP-CE.

Here you see the screen of the Professional Version (PE), which contains the Loopback-Button. The Community Edition does not show that because it is not open source. In the upper right corner you can see which edition of CAPP you are working with. In the lower left corner you can see, that this is the PE in Demo mode - and still 23 days left.

In the top menu bar you find an entry for "License" where you can either enter the license key or remove the license from this computer. At the moment you can use the license on one computer at a time.

Environment Window

By default you should not have to take care about the environment - the wizard sets it for you.
Press the small buttons on the right to set an application for a given purpose.

**System Window**

The System Window sets all necessary values for the complete system:
It contains a menu bar which allows to control further settings:

**Language**

This option is not active in the CE version. It allows the selection of another template set:

Of course the template set has to be stored at the Template directory of TeloSys/Velocity (see above).

**Connection**

How CAPP connects to the database is set in this window. Up to now the database driver and URL are fixed in the CE Version. DBMS is always MySQL yet.
The database to be used can be set and also the user/password combination of your database.

**HINT:** If you use the newer, not community driven version of MySQL it may be that you are using the stronger password encryption of Version 8. This will not work. Revert to the weaker encryption or use the Community Edition of MySQL or MariaDB instead.

**Database**

This button calls up the database editor which you entered in the Environment Window. It is just a shortcut for you that you do not have to leave CAPP for calling up your favourite editor for the database.

Most often the MySQL Workbench is called here, but there is a lot of other helpful tools out there.

**Startmodule**

At one point you will have to tell CAPP which will be your Main- or Start-Module. This is here. In our Demo-Situation the Name will always be **CappTest**.
Velocity

If you encounter errors while you generate code you may wish to talk to Telosys and Velocity directly and not "through" CAPP. This can be done by using the "V"elocity Key:

You will wonder about what you can do here? In short terms: Generate the **LAST** item in your module list. Not more. There are three important commands which you could or should use here:

```
  h .
```

This is the "home" command - and it sets the "home" directory of TeloSys to its own directory.
Shows the current environment of TeloSys, including paths and context.

Generates all templates for all entities. In our case it generates just the last one.

**Templates**

This option calls up simply the templates directory in Finder. If you have assigned any Text Editor to the ".vm" Postix then you can open any template from here with a click:

Parameter

If you want to use a variable **system wide** yo can set it here. In this sample we set the value "XojoTest" to the variable "test". This will result in a variable named $test, which can be used in any template. It will be generated into the template called "fieldvars.vm".

© Dettmer Informatik GmbH & Co. KG, 2019 - www.dettmer-informatik.de
Of course you will have to distinguish this variable from the variables on module level. These can be set in the Module Window later on.

Application Window

If you set up a system, this system has a given structure. This structure can be maintained in the Application Window:

In our sample case the system has 10 modules, numbered from 0 to 9. The first column shows simply the current ID in the SQLite database file. The second one defines the order or structure the modules are in. This may have for example influence on a menu in the system.

The third column can contain these values, related to TeloSys:

- Resource - Copies files from the /resource subdirectory to the Output destination
- Once - Runs the template for ONE instance of the context only
- Multiple - Runs the template for ALL instances of the context
- Ignore - skips this module while the generation process - for debugging

If you encounter an error you can switch all modules **after** the erroneous module to ignore - so you can use "S"ystem and "V"elocity to check.

The sample above shows already the output of one generation run. you can see that it says "1 file(s) generated". No errors. If an error occurs, the string "ERROR" is detected in the output and the module is marked with a small red "ERR" in the leftmost column of the module list.

Here another example of the Output of the **records.vm** template:

![Output of records.vm template](image)

As you can see here, the "records.vm" template generated 23 files - one for each table or view of the Sakila test database.

There is one "hidden" function in the Application Window - you can call the Embeds Window directly from here without entering the Module Window - simply by pressing "E"

**Module Window**

The Module Window is the most important part of CAPP. It shows the parameters for each module of a system in detail:
Input

In this case the Main Menu of the generated app is shown. Let us start with the "Input-Area":

- Input is a database structure, or also called "Context" in Velocity. Here a "**" means a wildcard, so all of the tables and views are included into this generation. If you want to generate only one table, you can enter it here. Telosys allows also multiple entities, separated with semikolon.
- The second field contains the assigned Velocity template. The small button on the side allows selection from the template directory. In this case the menu.vm template generates all the buttons and menu items on the main screen for the selection of the table views. Here a snippet generating the menu:

```plaintext
Begin MenuItem FileMenu
    SpecialMenu = 0
    Text = "Tables"
    Index = -2147483648
    AutoEnable = True
    Visible = True
    #foreach($entity in $selectedEntities)
        Begin MenuItem Data_${entity.name}_View
            SpecialMenu = 0
```

© Dettmer Informatik GmbH & Co. KG, 2019 - www.dettmer-informatik.de
The Volocity commands will disappear in the generated code and the variable $selectedEntities sets the multiplication of the block between the #foreach and the #end. The variable $entity.name is replaced by the name of the current entity - as the name says. Please refer to the Velocity documentation about further information on Velocity and Telosys.

**Control**

The Control Area contains all items needed for controlling the generation process:

- **Name** - contains the name of the Module - which will or can be used inside the template. This is useful if you want to label a window or a menu item.
- **The "Structure" field is used to position the module inside the system or also the menus. If you want to move an item inside the structure you can rename the structure item. It will be sorted alphabetically. A good practice is to use 6.1 or 7.3.1 which inserts a kind of level inside the structure. This value can be accessed in the Velocity macro later on by the use of the variable $modules.**
- **Type** - contains the type of step this module will go through:
  - Resource - Copies files from the /resource subdirectory to the Output destination
  - Once - Runs the template for ONE instance of the context only
  - Multiple - Runs the template for ALL instances of the context
  - Ignore - skips this module while the generation process - for debugging
- **Parameters** - the small button opens the above explained parameter window - and assigns this parameter to this module exclusively. This allows to have for example two modules with the same template and the same input - but different parameters.

**Embeds**

This area simply shows the embeds which are in use for this module - also exclusively again. A press on the small button opens the Embed Window for this specific module and the assigned template.

**Output**

The output area controls - as the name says - the output. This is simply the name of the file that will be generated. As you can see in the sample screens there are two special characters, which will be replaced by variables at the generation:

- *(Star) will be replaced by the Entity Name of then selected context
● # (Hash) will be replaced by the Module Name

Here is another example for the Module Window, containing a "multiple" entry and one output for each entity - it calls the record.vm template:

As described above already Esc terminates the entry and Enter saves everything

**Embeds Window**

The Embeds window can be called from the little button in the Embeds Area of the Module Window or from the selected module in the Module Window by pressing "E".
In the left pane you see the embed points which are available in the `record.vm` template. As already mentioned above it is smart to have a good naming structure for the names of the embeds.

It is proven as good practice that you take the name of the template as the first word (above: `TABLE`). The second word should say if the template is `before` or `after` something else. And the rest describes the position inside the code.

**HINT:** Inside the database the apostrophe used as a comment marker inside XOJO is replaced by a HEX value of 127. So if you are editing directly in the database take care.

**Generate**

"G"enerate calls up the Generator Window, which starts TeloSys several times (as many modules as are there). The output of each generation process is stored in each related module, so that you can see problems or errors, which occur while the generation process.
There will be no error message - If there is any problem the Gerate-Icon in the CAPP Main Window gets coloured red. If you look in the Application Window a small "ERR" will appear in the leftmost column.

**Loopback**

The **Loopback** command will look through the generated source code and get back all enbeds into the embed database table. This is only available in the Professional Edition CAPP-PE.

**This function is one of the most important parts of the CAPP concept.** It allows you to generate an application and to modify it later on in XOJO. If you change **ONLY** code inside the embed markers, all your changes will be preserved.

You can **delete all files** in the output path - and restart generating all modules new. Your changes you made in the code are **given back to the new code** inside the embeds. If you want to make some changes outside an embed simply create a new one by inserting a line like this anywhere you want:

```
#embed( "RECORD Before anywhere you want" )
```

The Embed Window will scan the template when you open it - and show your new embed. On the richt side of the window you can edit the code.

**Wizard Window**

The Wizard will help you to fix things if someting goes wrong:
At the first start of CAPP simply go through all steps. Here is what each step does:

- **Step 1** - Copies a default system from a stored database to the current CAPP database. The current database is saved with a random number as a backup. If you entered a structure already it seems lost if you all Step 1. But it is not - simply copy the randomnumber file back to CAPP.sqlite.
- **Step 2** - CAPP and the TeloSys-Velocity combination is very sensitive about the path settings. If you modified those and nothing is running any more - use Step 2 to fix it.
- **Step 3** - Setting the current database and the SQL user/password combination is always necessary.

### About Window

CAPP is not a free software. There is an Open Source Community Edition (CAPP-CE) available. This edition does not make use of any plugins or libraries outside the default functionality of XOJO. All parts which are used from other developers are mentioned in this About Window:
Clicking the links brings you directly to the contributing development pages. We respect all the rights of the involved parties and are grateful for the great word they provided to make CAPP possible.

### About us

Dettmer Informatik GmbH & Co. KG is a specialist for individualized software, which means business software for purposes, where no standard product would fit. Our customers run their businesses in printing, wholesaling, health, trade and many other areas.

See our website at [http://www.dettmer-informatik.de](http://www.dettmer-informatik.de)
Meta

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