

The SAP Killer

*Build or Migrate
any DBMS or MIS
Up to a full blown ERP*

By Redhuan D. Oon (RED1)

Dated January 1-29, 2019
(Book in progress, 40% done)

What Can This Book Do?

This book shows you, without doubt, how to do anything related to creating a DBMS or MIS from scratch or migrate from a legacy to a brand new modern powerful open source suite called iDempiere.

The author has more than 36 years struggle in the industry with much travel and work experience from 40 countries facing formidable challenges which culminate in a set of best practice and his own novel approach that cuts through the hype and myth of software engineering.

Now, you can build your MIS or ERP in the quickest time imaginable compared to many options out there. You are guided in easy steps from practical real life scenarios.

Using the Ninja toolset, it simplifies the use of the best free heavy weight ERP, iDempiere, which has replaced many large ERP systems in many countries around the world. Now you can use this as a real SAP killer.

The first part of the book assumes you have setup of iDempiere from www.idempiere.com. The second or appendix part of the book will give you all the basics from setting up iDempiere either in-house or on the cloud or use our free cloud for the back-end, up to the fundamentals of iDempiere structure, features and functionality.

Further paid consulting support is available by contacting the author at red1@red1.org. Meanwhile, you may continue your DIY journey by accessing freely the technical notes at www.red1.org/adempiere.

Who The Heck Is SAP?

For those who doesn't know can actually Google exactly the above and there is a blog on it. SAP is actually the most expensive software in the world not due to its code but the know how and consulting or advisory services that goes with it. This is because it is usually used by the Deutsche Banks, Red Bulls and Petronases of the world. Its cost is thus sky high running into hundreds of millions of dollars and that is in licensing alone.

Does that means Open Source is free? No, it is free as in Freedom, not Free Lunch. Then how do you make money? Read here <https://www.gnu.org/philosophy/selling.en.html>.

So can I switch from SAP to iDempiere with Ninja? Not until you think about it by reading the 19th October, 2011 article down here <http://www.red1.org/pos.html>.

You may still be deeply confused, that is why I make life easy with philosophy.

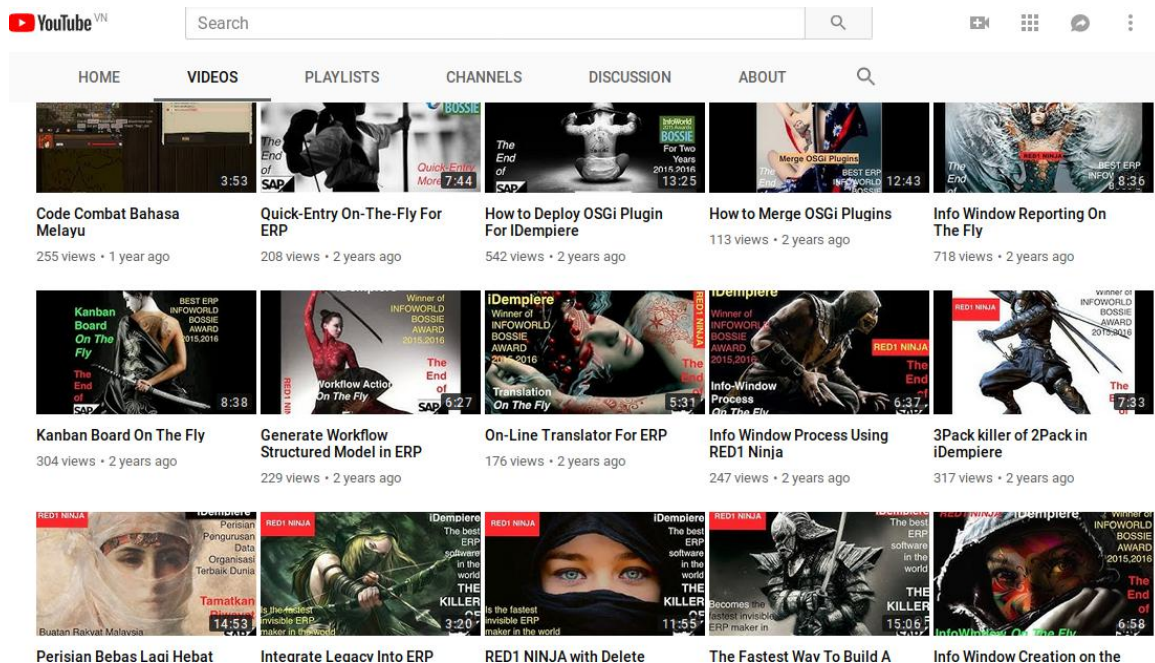
RED1's THREE LINE MANTRA

INFORMATION IS FREE, YOU HAVE TO KNOW

PEOPLE ARE NOT, YOU HAVE TO PAY

CONTRIBUTORS ARE PRICELESS, YOU HAVE TO BE

<https://www.youtube.com/user/redhuanoon/videos>



Before You Continue

Have you read my 19th October 2011 essays? If no, then you may use this series of questions to determine if you need to continue with this or look elsewhere. This is to save everyone's time.

1. Are you looking for freedom to have sovereignty over your system, the freedom to have it fit into your needs rather than the other way around? If Yes, proceed.
2. Are you hoping this will replace your SAP? If Yes, then be careful and tick all the advice given here.
3. Are you looking for more security, independence, inter-operability, and sustainability of your ERP system? If Yes, then this is certainly the direction to pursue.
4. Are you trying to have a free as in free lunch ERP system? If Yes, then this is not for you. It involves much more costs than you think. Not just your time, but the emotional burden and attitude towards this project may hurt. I have seen people from as far as Slovakia and Japan cursing this project for faults of their own making. So stop here, if you think this will save you money outright without homework.
5. Are you ready to be well prepared with a clearly defined budget and resources and expectations of time schedule of deliveries. At least someone like me has giving you a transfer of know how course? If Yes, then proceed.
6. Is your team ready to go through the sharp and short learning curve for 3 months? If yes, proceed.
7. Are you willing to accept best practice advice that maybe in contravention with your own best practice? If Yes, you may proceed.

Under part two there will be a chapter for you to understand why I put these in and the rest of my best practice that you can read through and decide if you can proceed on your own or you need me to come in to explain further usually in a paid Skype session or on site workshop.

Not For You

This iDempiere is not for you if you are looking for a standalone ERP system, handling a single POS. It will be overkill. It is like driving a Ferrari to go next door. It is better you use simple free download-able apps out there that probably cost below USD1,000. Or Excel, also below USD1,000.

In fact there is no such thing as a standalone ERP. The word enterprise in there means it is for a starship enterprise effort.

However the Ninja is making this ERP into a better faster easier MIS. So if you are looking for an MIS, which actually is bigger than a standalone. It has to be a multi-user, multi-organisation, multi-role, multi-user and multi module, then this will be worth your while.

What is iDempiere?

iDempiere is a fork of a fork of Compiere, the first Open Source ERP project to hit the world in 1999. In mid of 2006 Compiere was sold off. The community under the leadership of the author forked as ADempiere during the first week of September, 2006. Later the project evolved further migrating onto the Eclipse-like OSGi as iDempiere.

Compiere is originally meant for heavy duty serious ERP that covers accounting, inventory, supply chain, CRM, and document management. Like any other ERP software, it will try to be one size fits all, which no one can. There shall always be gaps where new modules or verticals have to be built.

What is powerful is it has within its design the Application Dictionary or AD which is actually an application within the ERP application. The AD uses metadata to quickly and easily configure Menus, Windows, Tabs, Tables and Columns as well as Print Formats and Processes and a host of many other things without the developer having to modify code. Thus it is not error prone, but versatile and stretchable. Even so, grappling with the AD is still always a challenge and this book lays out how best to meet this challenge.

iDempiere since its first fork of ADempiere from Compiere, has incorporated amazing features such as Composite Web-Services API, FitNesse testing engine, and Self-Help screens. You can refer to many of them here http://wiki.idempiere.org/en/Category:New_Features

Of course, the OSGi framework is what separated iDempiere from ADempiere. It allows separation of concern where all developers may add on without touching the core thus ensuring maintainability, always synching with upgrades of the core and easier testing. See the many plugins done by the world wide community here:

http://wiki.idempiere.org/en/Category:Available_Plugins.

Now it is the Application Dictionary or AD that I want everyone to pay attention to. I realised that this AD is under-marketed as the focus and lingo surrounding this project has been mostly on the ERP which is a complex subject matter. This lesser affair of simple module or new application building is overlooked and not realised by non ERP users. This lesser realm actually solves the needs of the world of MIS or DBMS (Management Information System or Database Management System).

So I intend to promote that useful nature and I even come out with a wizard suite called Ninja which focus on letting the AD be unleashed via script strings which generate the metadata AD on the fly.

I also have a tool call the Wukong Menu Controller which can hide all the ERP sub menus from the main menu tree giving it the appearance of a simple empty shell to be sculptured quickly to any MIS or DBMS system. The Wukong tool can then reactivate back any ERP module and this gives the appearance that iDempiere can start as an MIS all the way to a full fledged ERP.

Now, another overlooked killer app outside all this, is the spreadsheet. In my case I used to use MacBook Numbers but now switched to Linux based operating system for speed, using Libre Calc. Now it is this wondrous spreadsheet that I build my scripts from. You shall see in later chapters how I progressively exploit the functions within the cells to produce script formats in a more visible and easy form, to avoid redundant typo and human errors.

What Is Red1's Ninja?

Ninja is meant to be a killer wizard that I took the time since early 2016 to craft which helps you overcome the complexity of building any form of database application and implementation, working alongside the best free ERP software in the world, iDempiere.

The Ninja takes an Excel spreadsheet template to generate most of an iDempiere module from the ground up instantly on the fly. It can also do so in tandem with the migration process of legacy data. The artifacts it can generate or handle within a single blow are:

1. Each data model and its columns with data type settings
2. 7 mandatory columns automatically inserted
3. Master-Detail relationship with choice of parent key column
4. Menu item on Menu tree with Window and Process
5. Window Tab with attached Process stub
6. Process java stub with defined parameters
7. Workflow linking a series of windows as a ready user guide
8. Info-Window creation from View of multiple models and link on main panel
9. Info-Window additional displayed columns
10. Info-Window Report output from the selected rows.
11. Info-Window with attached Process stub
12. Translation of menu and window tabs and fields into Translation tab
13. Master-Detail Print-Format reflecting displayed layout, with Corporate image and window description as headline.
14. Document Type with Action/Status Model
15. Kanban Board with 12 column status for cards
16. ORM Classes generation
17. Model Factory classes
18. OSGi XML definitions
19. Callout JSR 223 generation
20. PackOut format
21. New blank plugin for the newly generated module
22. Migration of master data in a single zip, generating any new data models.

All the above are handled from a single spreadsheet which any dumb user can understand instead of confronting the complex bulky parts of any database configuration framework. The idea is to give high level single reference visibility.

What is left for a the programmer is to just code the logic of:

- A. Process
- B. Model class events

Sample Excel template and blank OSGi plugin are provided. Also great data model design ideas from me for some common user cases. No haggling with the setup, scaffolding, and boiler plate code. All this is contributed freely by me to make iDempiere more popular and easy to use. Now, lets get going to build a brand new ERP module or domain vertical in record time.

CHAPTER 0 - A Crash Course in MIS

My first job in 1982 was a COBOL Programmer for a HR Payroll system. My second job in 1985 was given the title EDP Supervisor Programmer. EDP stands for Electronic Data Processing. Today it has evolved into DBMS (Database Management System), MIS (Management Information System), and ERP (Enterprise Resource Planning).

The programmer label is now developer, supposedly doing everything from requirements analysis through programming through testing. In 2010 I received a new title CTFL (Certified Testing Foundation Level). So now, I am going to debunk many myths by going back to only 2 things you need to know.

1. GIGO - Garbage In Garbage Out. Whatever goes into the system, remains as such and you won't find anything else. That also means if data has gone in, do not worry. It is there.
 - a) This leads to the sub law called non-redundancy. It means once recorded, it is there. You need not repeat it. It may not be well formatted, or raw, but again, do not worry. What is most important is that it has been captured. It is somewhere. We shall get it.
 - b) This thinking is important because later and in fact quite right away in the next chapter you see my use of the Excel spreadsheet where I capture basic data model, and like magic it becomes a full fledged MIS or ERP system. How? It is because of this sub law of non redundancy, which its mother law is GIGO. Read it 15 times. Every day. Until the day you attain Nirvana.
2. Input, Process, Output. This simple three word is your only mantra to good systems design. There is only and only three parts to any system. Again, note how it follows the first law of GIGO. Input is what goes in. Output is what comes out. But there is something in between to rearrange or set in order the data within. That is where programming supposed to be.
 - a) Both Input and Output is the same. GIGO. Thus that is why a single spreadsheet can represent both. Once you design well this stage of data model, you need not worry where to fetch the data for the output.
 - b) Input are all formats used to capture data. They can be input screen, mobile input, sensors, keyboard, mouse or voice. They are usually found in the form of input formats. That can be put into a spreadsheet. As your data design.
 - c) Output are all required data demanded of a system, mostly reporting. Even charts and analysis is a mesh of such output. Again, once input is in, output is derivable. Now, even migration, which was hell, is now a breeze. Once I got the data, I got it.
 - d) Process is what to do with this data bit or that data bit. Usually it is taking a piece of data, looking up another related to it, applying a calculation if needed, and writing the result back or in a new model. Now this seems to break the first GIGO law, that suddenly a new model exists that maybe non-garbage. No. It is its non-redundancy derived data. It didn't come from somewhere else. It came from what has gone in. Thus, you have to read all these and meditate on it. Daily.

What I actually did, is using another modern rule called, Separation of Concern. You can call this the 3rd law but it is a modern software design principle. I am a traditional fundamentalist. I do not wish your mind to be clouded. But this 3rd law is actually in the first 2 just that it is made more subtle. By using the easy and universal Excel, I am separating the hard work and the Input/Output Data Model part. I am also putting the grunt work of Process, which are the code stubs scaffolding there too. What is left is just pure coding work of business logic. No grappling with redundant repetitive code preparation, as you shall see later in advance chapters.

Now, the journey shall be painful. It is a matter of degree of pain, depending on how smart, or experienced or trained you are. I maybe a guru but that is not important. There are no bad students but only bad gurus. My task is to guide you in a way that is most effective and at times it maybe non-conventional. I know many things, but they are part of the problem. I am putting finally the solution.

CHAPTER 1 - The Excel Template

This is a clean module setup template. Later we shall look at the legacy data migration template. You can fetch this first template from

https://sourceforge.net/projects/red1/files/ModuleCreator/Ninja_Excel.xlsx/download

Open it and see in the first sheet called **1_RO_ModelHeader**

	A	B	C
1	Name	Version	
2	NewMenu	1	
3			
4			
5			
6			
7			
8			
9			
10			

This defines the Menu Tree item. In this case it is NewMenu. That shall appear as the new item in the iDempiere's menu tree. Next we move to **2_RO_ModelMaker**.

A	B	C	D	E	F	G	H
RO_ModelHeader_ID	Seq No	WorkflowStructure	Kanban Board	Master	Name	Help	ColumnSet
NewMenu	1	N	N	C_BPartner	MY_Model	Workflow Structure will generate Doc Mgmt Structure with Process. Use with KanbanBoard in later Ninja model.	Description,IsActive

Here a lot of actions happens. So we start from the left. Look at the header row. The **RO_ModelHeader_ID** means this is a lookup field. The Excel already set that its value is referencing the first sheet's value. So you need not refill such.

This is the power of Excel and the main reason it is used here. I could easily setup a sample template with many rules and formulas already set. The lay user also is more familiar with Excel than anything else.

Whenever you encounter a header value with a suffix **_ID**, it means the content below it is the Name column of the record referred.

Now the next header. It is **Seq. No**. This has to be a running number as Ninja will sort the records and execute them according to their sequence numbering. In this case we only has one record. You may proceed to try more after this. For now, we just proceed very slowly so as to get all the concepts grasped tightly first.

Next is **Workflow Structure**. This when set to Y will generate the model as a DocAction aware type. It will have extra fields to handle Document workflow actions and status..

Next is **KanbanBoard**. It will create a Kanban Board display with 12 status columns automatically for the DocAction aware model. But it must be set to N first and allow Workflow Structure to be Y first. After the first run of the Ninja Generate Module, the Kanban Board flag can be set to Y.

Next is **Master**. This is usually blank. If it has an entry, it means there is going to be a Master-Detail model relationship. However, whatever is stated here must have been defined prior. It has to be created first before being placed as a Master. So in this case C_Partner is a core table which already existed.

Now, the main item, **Name**. This is simply the name of the Table or Window Tab that shall be created. As stated, if the Master is blank, this will be a single Table. Don't worry if you are puzzled by anything above. After going through this round, we shall examine a more elaborate example 'Hospital' already imported into iDempiere Ninja to follow and learn quicker.

Next is the **Help** column. Whatever is put here shall appear in the Help section of the Tab created. By having this, will make life easier for all, as the designer, user and implementer has a common reference spreadsheet to refer to and understand the purpose of the design.

Finally, the **ColumnSet**, containing 2 fields only. These two, *Description* and *IsActive* shall be appearing in the intended Tab panel.

So in conclusion there shall be a sub tab to the Business Partner tab, called MY_Model. Actually the MY_ prefix would not appear in the sub window tab, and only be used in the underlying database table creation. This sub tab shall have two fields Description and Active. The 'Is' part is also hidden and it shall appear as a checkbox.

At the time of this writing, but before updating this spreadsheet, I just improved the ModelMaker with a new column checkbox, **PrintFormat**, which works in conjunction with Master-Detail definition to generate a Print Format with Sub Print Format and attach itself to the new document. It has a special way to do it, which is amazing as you can rearrange its layout from the Tab Editor of the Window Tab. This checkbox is in play only after you have imported your master-detail model and gone thru its layout in the Tab Editor.

The CodeMaker

Now we go to the **3_RO_CodeMaker** sheet.

A	B	C	D	E	F	G	H	I
RO_Model Header_ID	Seq No	Name	PluginLocation	Generate Model	Generate ModelFactory	GenerateProcess	Description	ProcessParameters
NewMenu	1	MY_Model	/usr/local/org.red1.blank/src/	Y	Y	MY_Process		IsActive

This is used to generate OSGi XMLs and code stubs and the model ORM classes automatically for the defined model in the previous sheet.

PluginLocation refers to the location of your new plugin. You may take my free sample plugin from <https://bitbucket.org/red1/org.red1.blank/>. However if you are not coding further logic to your new module, you need not have the CodeMaker involved. The ModelMaker should be in theory be sufficient to let you fly off with a simple database system, accepting data input, and exporting data to external parties.





GenerateModel and **GenerateModelFactory** are usually checked Y together to generate your new model I_ and X_ classes, as well as the Factory and Model classes.

GenerateProcess will specify a new process java class stub be generated and attached to the model's window tab at the same time. If you specify this on a new line and leave the ModelMaker_ID blank, then it will be attached under the new menu item tree.

The advantage of been attached with the model is that you can fetch its active record and process it.

(The following is now deprecated by the next chapter which directly send your sheets to the Ninja!)

With the above done, all we have to do now, is just to export the sheets as CSVs and zipped them up. Then attach them to the Ninja to GenerateModule and get the results within few seconds.

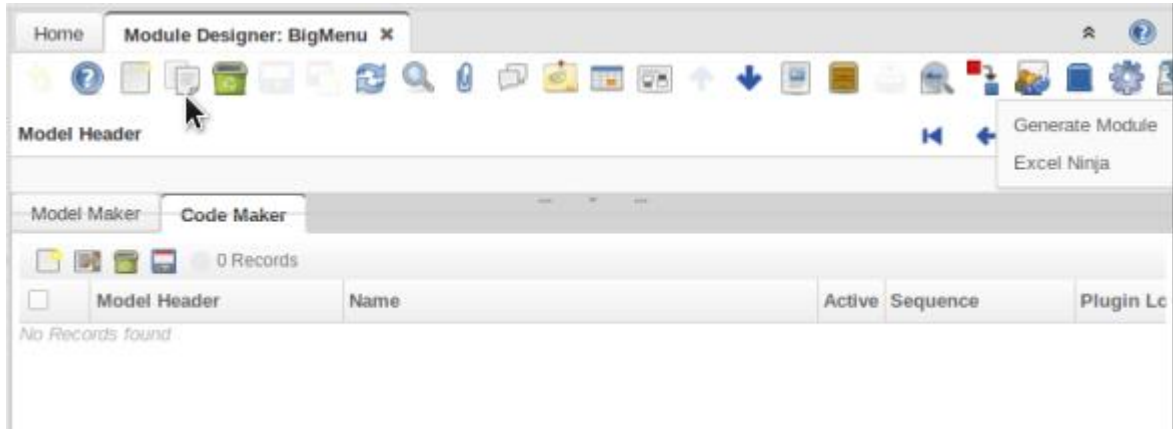
Filename	
	1_RO_ModelHeader.csv
	2_RO_ModelMaker.csv
	3_RO_CodeMaker.csv
	ANinja.zip

Store each sheet as its namesake as shown below. In MacBook, this can be done at one shot. But in Libre Office, you have to do it each sheet at a time. The numbering prefix is vital as it will ensure they are imported later sequentially.

Compress the 3 files into a zip and you can give it any name. Here I called it ANinja.zip. What matters is the individual CSV file names with the numerical sequence prefix.

Direct Excel To Ninja

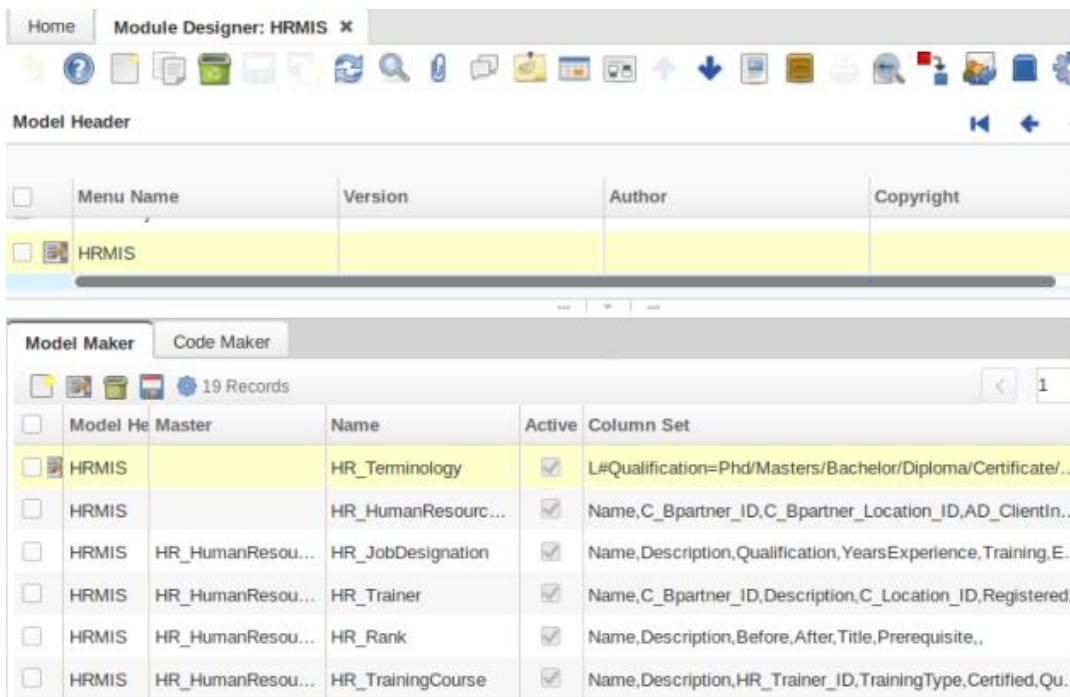
Again, I introduced one really simple but great innovation as we speak. I just made a process that will now free you from even exporting the CSVs, zip them up, attach them to the ModelMaker tab, and run Import Model! All you have to do is at the main tab of Module Designer, run Excel Ninja.



Then at the opened dialog box, give the path to your Excel. You can use Saved Parameters to store your favorite Excel sheets name and location.

Run and that is it! Your whole Excel sheets of header, model and code are in! As in my HRMIS sample, it took hardly 2 seconds.

You can now run Generate Module!.



Chapter 2 - Ninja Generate Module

This assumes you have setup your iDempiere instance. You may run this either in your binary instance or Eclipse development environment. And you should have got the Ninja plugin working. You may fetch them as follows:

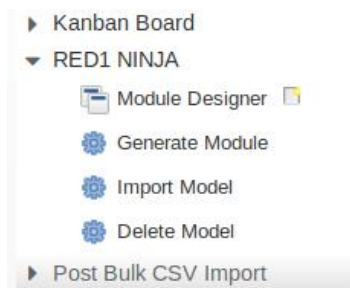
1. Binary - <https://sourceforge.net/projects/red1/files/ModuleCreator/>
2. Source - <https://bitbucket.org/red1/org.red1.ninja>

I am using the source on Eclipse. I also have a blank plugin (as a new plugin to house the generated code and attached a new 2Pack of the data models) fetched from

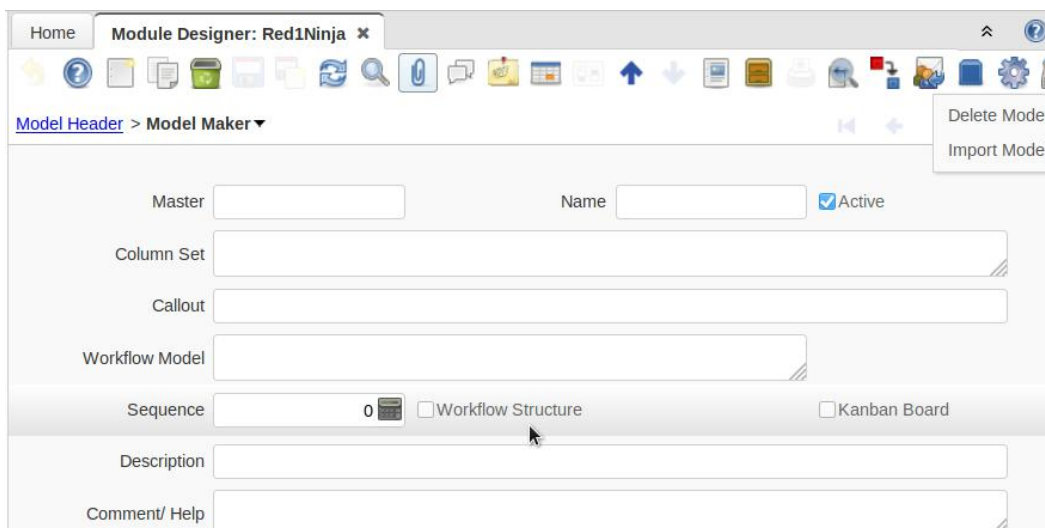
3. <https://bitbucket.org/red1/org.red1.blank>



From the Menu tree, call up the Module Designer



Inside the Module Designer, make a new record, and just attach the zipped CSVs to the ModelMaker window. No need to put in any details. Now, you can either run the attached process *Import Model* or at the Header, *Generate Module*.



After Import Model you will get a positive result at the Description box:

Description	Bulk CSVs in ZIP file successfully imported - New Records: 1
-------------	--

Go to the Model Header main tab and refresh.

Home | Module Designer: NewMenu x

Model Header > Model Maker

Data requested

Master: C_BPartner | Name: MY_Model | ☒ Active

Column Set: Description,IsActive

Callout:

Workflow Model: | ☐ PrintFormat

Sequence: 1 ☐ Workflow Structure ☐ Kanban Board

Description:

Comment/ Help: Workflow Structure will generate Doc Mgmt Structure with Process. Use with KanbanBoard in later Ninia model.

You will see a new record *New Menu* and sub detail record for Model Maker. It corresponds to the imported CSVs. You will also notice that there are more columns that were not included in the Excel. They, particularly **Callout** and **Workflow Model**, can be included whenever needed. Later, during the second round of explanation, I shall address these columns usage.

You can also go to the Code Maker tab and see that it has also been populated accordingly.

Home | Module Designer: NewMenu x

Model Header > Code Maker

Name: MY_Model | ☒ Active

Sequence: 1

Plugin Location: /usr/local/org.red1.blank/src/

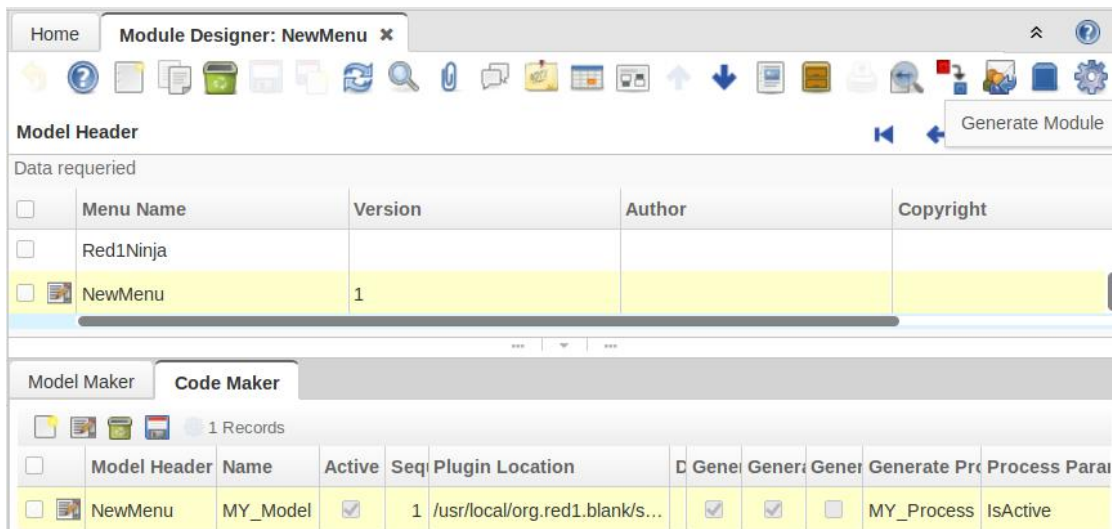
Description:

☒ Generate Model ☒ Generate Model Factory ☐ Generate Translation

Generate Process: MY_Process

Process Parameters: IsActive

Now, we can execute them with the header tab process icon, **Generate Module**.



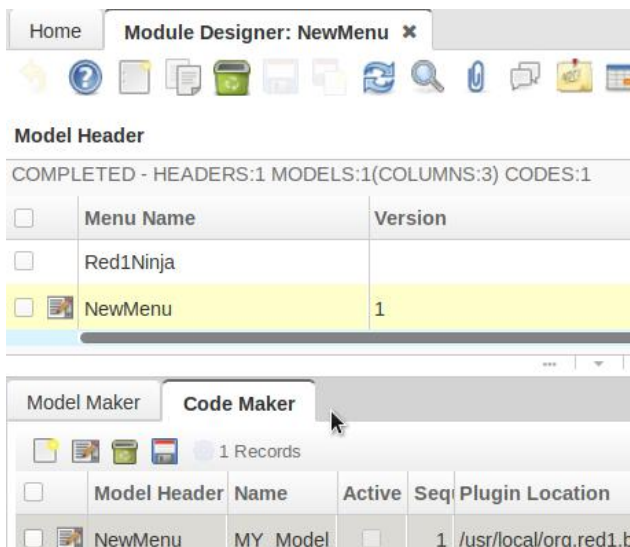
The screenshot shows the SAP Module Designer interface. The 'Model Header' table is displayed with the following data:

<input type="checkbox"/>	Menu Name	Version	Author	Copyright
<input type="checkbox"/>	Red1Ninja			
<input type="checkbox"/>	NewMenu	1		

The 'Code Maker' tab is also visible, showing a table with 1 record:

<input type="checkbox"/>	Model Header	Name	Active	Seq	Plugin Location	D Gener	Gener	Gener	Generate Pro	Process Para
<input type="checkbox"/>	NewMenu	MY_Model	<input checked="" type="checkbox"/>	1	/usr/local/org.red1.blank/s...	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	MY_Process	IsActive

Click OK, without checking anything. You will receive the result within 5 seconds and it is all done. Notice the message on the top of the panel.



The screenshot shows the SAP Module Designer interface after the 'Generate Module' action. The 'Model Header' table is displayed with the following data:

<input type="checkbox"/>	Menu Name	Version
<input type="checkbox"/>	Red1Ninja	
<input type="checkbox"/>	NewMenu	1

The 'Code Maker' tab is also visible, showing a table with 1 record:

<input type="checkbox"/>	Model Header	Name	Active	Seq	Plugin Location
<input type="checkbox"/>	NewMenu	MY_Model	<input type="checkbox"/>	1	/usr/local/org.red1.t

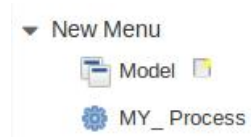
COMPLETED - HEADERS:1 MODELS:1(COLUMNS:3) CODES:1

This means one menu header is created with a data model that has 3 significant columns, and its code stub set.

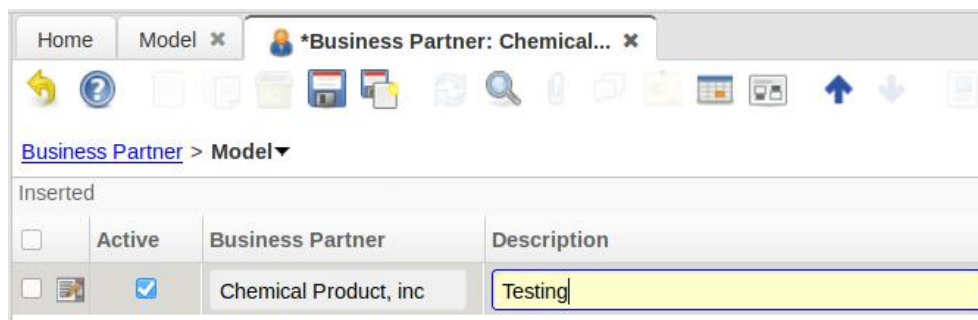
Notice that the Active box in the details are unchecked and they are grayed out. For both Model Maker and Code Maker. This means that they have executed properly. If they are still active, then something went wrong, and you have to check the console logs.

Generated Module

Now we can restart and login into the Garden Admin role to examine the new model.



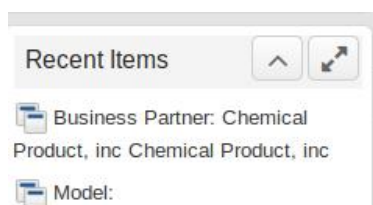
A new menu item appears under the main menu tree with a window item called Model and a Process item called MY_Process. So congratulations, you basically passed Ninja Basics. But wait there is more. Remember our original Model Maker entry was a Master-Detail setting for it to be under the C_Partner model or Business Partner window. So call that up and you can see there is a new tab:



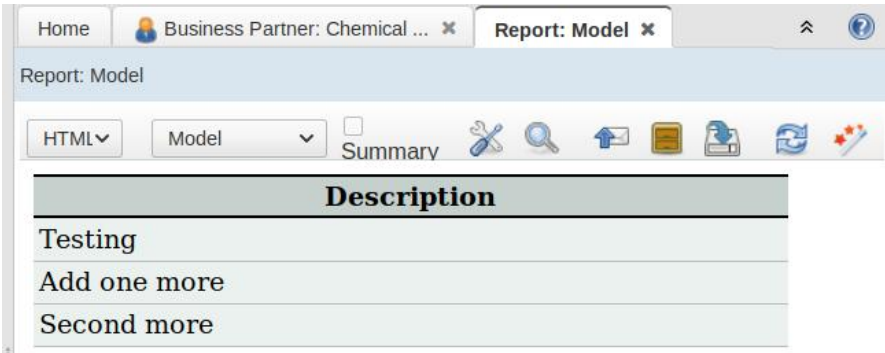
You can see it is correctly linked to individual parent records. Lets try producing some sub detail data and see if it works.



It does work and without needing its code plugin to be active. That means the meta-data framework, thanks to Compiere and improved iDempiere holds very well. If you check the left hand lower panel, you will see under recent items, it is also tracked.

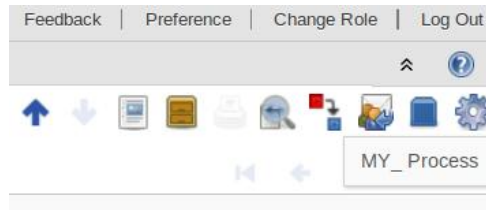


Now, let the AD framework show more of its ready made goodness and glory. Click on the report icon button on the top menu bar. OK it through and see an automated report is generated:



Just imagine, a single Excel detail can translate quite easily and instantly into your ERP system. From now on, your imagination is the only limit. But wait, there is more. Look for the gear icon on the top right and click on it.

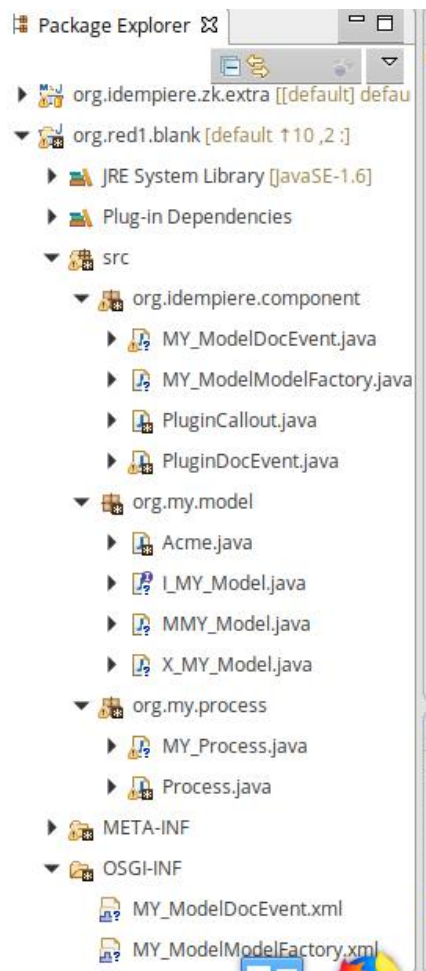
Generated Code Stubs



I also remove the repetitive boiler plate and scaffolding making of a new plugin. At the org.red1.blank plugin ready made for use, you can see the MY_Process which was defined in the CodeMaker tab. Try to test it. You will get an error message.



This is because the code side has not come into play. We shall now leave this data side and go to the code side by going to the Eclipse IDE. Expand the blank plugin.



Under the **src** folder, under **org.idempiere.component** are the DocEvent and ModelFactory classes for the MY_Model item.

These saves the developer from coding or creating them from scratch. Even copy/paste is not needed. You may delete the other java classes that are just place holders.

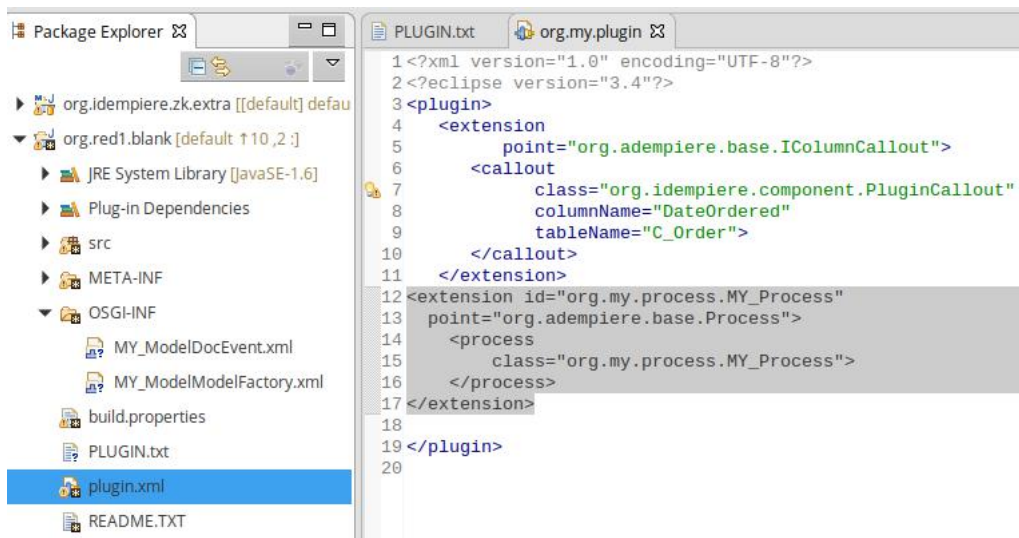
Under the **org.my.model** structure, are the ORM classes. The I_ and X_ and Model classes of MY_Model. Again you may delete the Acme place holder class after this.

Under **org.my.process** is the new MY_Process class.

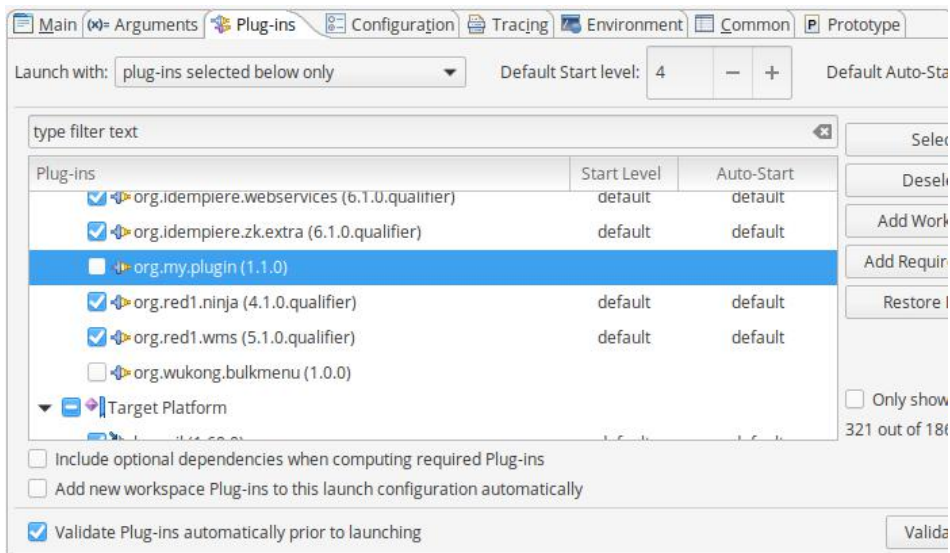
Under the OSGI-INF folder are two new XMLs. They both relate to the DocEvent and ModelFactory java classes respectively. Again, delete the other placeholders.

There is one task to do here before we can activate this new plugin to be used for coding further any logic.

Further down the blank plugin structure (hidden from the screenshot view), you will find **PLUGIN.txt**. Open it up, copy the content, and open the **plugin.xml** and insert it in as an added extension. This is to allow the Process to be linked when called.

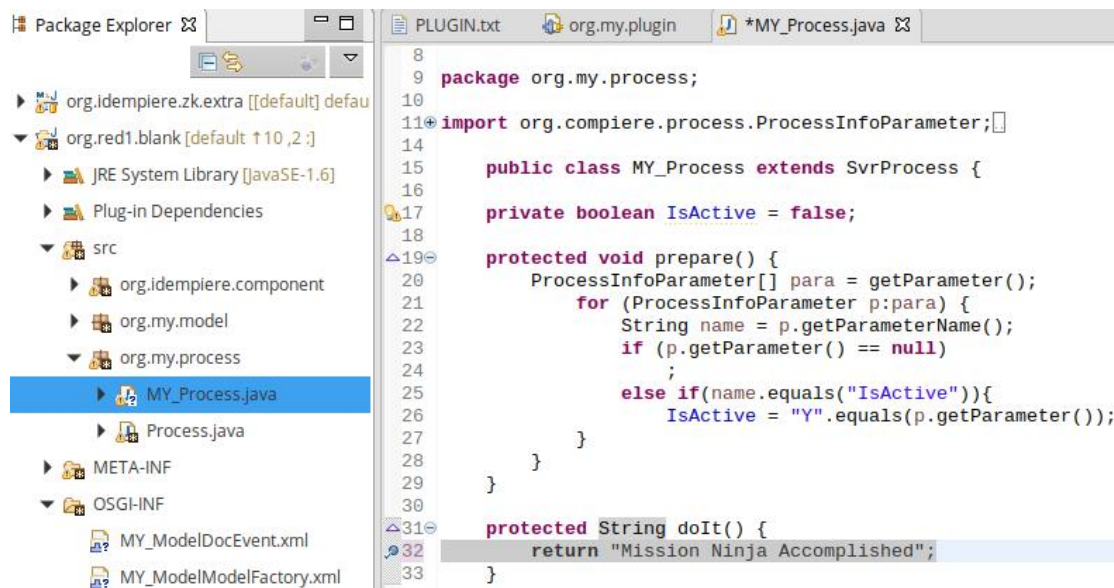


Now, restart your Eclipse Run/Debug As Configuration, and you will notice the blank plugin referred to as org.my.plugin.



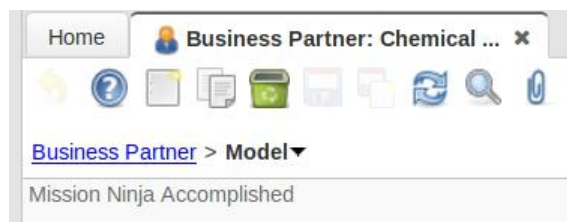
You need to check that to include it. You now have full programmatic control over your model, and soon you can readily use as well as introduce business code logic easily.

You can test again the process and see. But first let's try something to make you jump. Go to the src folder and open the MY_Process.java



Put a break at line 32 and add some comments within the quotes. Now press the MY_Process button and see.

First you will get the break button to pause the execution. This means that the plugin is active and you can now modify that java class. Press continue and you shall receive the success with that comment displayed.



Cheat Sheet

Here I provide a cheat-sheet that summarizes the steps taken. You may call this the Ninja Starter cheat-sheet.

1. Install iDempiere source code in Eclipse IDE

2. Install Ninja Plugin in Eclipse IDE

3. Install Blank Plugin in Eclipse IDE

(Estimated newbie time - 2 days. Estimated expert time - half day)

4. Export Excel Template first 3 sheets as CSVs in same sheet names

5. Zipped up as any name

6. Launch iDempiere in Eclipse with Ninja plugin checked

7. Attach to Model Maker tab of Module Creator

8. Run Import Model or Generate Module

(NOW STEPS 4,5,7 and 8 are deprecated with the Excel Ninja process. Time from 2 mins to 10 secs)

A. Run Excel Ninja from Module Creator main tab process

B. Set to location of your Excel and OK

9. Refresh to new record and run Generate Module

10. In Eclipse, insert PLUGIN.txt into plugin.xml

11. Restart iDempiere in Eclipse with My Plugin checked

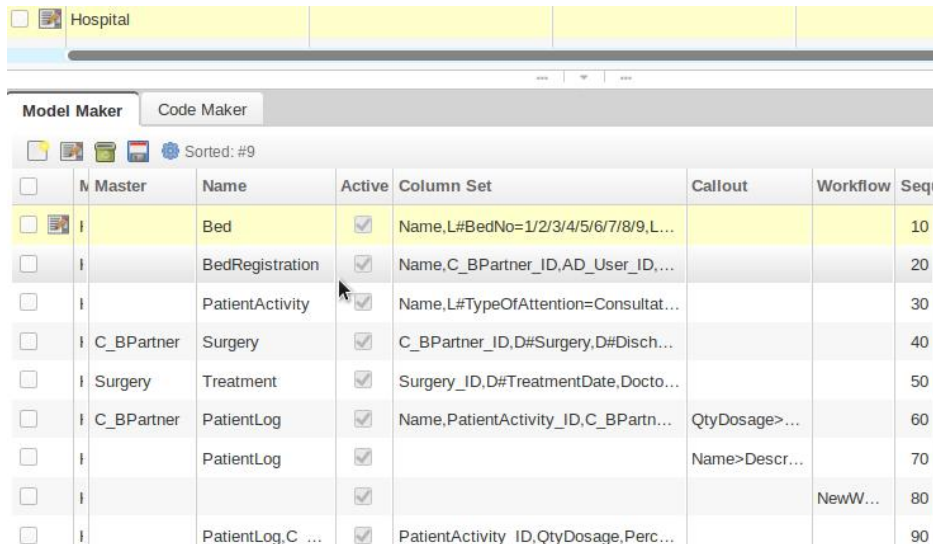
12. Login Client of GardenWorld and start entering data and exporting report

(Estimated newbie time - > 2 hours. Estimated expert time - 5 mins).

Next, we shall go through the the other powerful actions of the Ninja.

Chapter 4 - Data Types and Callout

Now we examine the Model Maker again, and up close with sample imported content already present when you have the Ninja plugin running. Look at the first header record 'Hospital'.



	Master	Name	Active	Column Set	Callout	Workflow	Seq
<input type="checkbox"/>		Bed	<input checked="" type="checkbox"/>	Name,L#BedNo=1/2/3/4/5/6/7/8/9,L...			10
<input type="checkbox"/>		BedRegistration	<input checked="" type="checkbox"/>	Name,C_BPartner_ID,AD_User_ID,...			20
<input type="checkbox"/>		PatientActivity	<input checked="" type="checkbox"/>	Name,L#TypeOfAttention=Consultat...			30
<input type="checkbox"/>	C_BPartner	Surgery	<input checked="" type="checkbox"/>	C_BPartner_ID,D#Surgery,D#Disch...			40
<input type="checkbox"/>	Surgery	Treatment	<input checked="" type="checkbox"/>	Surgery_ID,D#TreatmentDate,Docto...			50
<input type="checkbox"/>	C_BPartner	PatientLog	<input checked="" type="checkbox"/>	Name,PatientActivity_ID,C_BPartn...	QtyDosage>...		60
<input type="checkbox"/>		PatientLog	<input checked="" type="checkbox"/>		Name>Descr...		70
<input type="checkbox"/>			<input checked="" type="checkbox"/>			NewW...	80
<input type="checkbox"/>		PatientLog,C_...	<input checked="" type="checkbox"/>	PatientActivity_ID,QtyDosage,Perc...			90

Look at the lower detail tab, Model Maker and you can see many entries. Some of them are Master-Detail and some are not but standalone models. We shall examine the first one up close. The Bed Model. Click on its Edit Record to bring it up as main panel display.

[Model Header](#) > **Model Maker**▼

Data requested

Master Name ☒ Active

Column Set

The Name means it will be create a Window Tab and Table model named 'Bed'. It shall have properties or Field-Columns as stated in the ColumnSet, separated by commas:

1. **Name** as String data type.
2. **Bed No** as list of values "1,2,3,4,5,6"
3. **Floor** as list of values "1,2,3,4"
4. **Wing** as list of values "North,South,East,West"
5. **In Service** as a Y/N checkbox.

Here we encountered 3 type of data types: String, Reference of fixed Data List, and Checkbox.

We go to the next model, BedRegistration. Take note, that we remove spaces and during generation, the leading caps will space them out. So in this case, it shall appear as Bed Registration.

Master	<input type="text" value=""/>	Name	BedRegistration	<input checked="" type="checkbox"/> Active
Column Set	Name,C_BPartner_ID,AD_User_ID,Bed_ID,L#ClassBed=FirstClass/SecondClass/ThirdClass,D#DateIn,D#DateOut			

There are _IDs and now I explain what they means. An _ID means it is a lookup to a foreign table. The looked up table is the same name as the ID without the _ID suffix. Thus C_BPartner_ID is looking up the C_BPartner table. Unlike the fixed list stipulated by L#, this will be a dynamic list.

The next data type is D# which is obviously the Date data type. Ninja also automatically recognise a Date value as a D#. Thus D#DateOut can be written as DateOut and it shall reproduce the same result. As long the Date is the first part of the value.

Now we go down the list of Model Maker records to examine new data types and script. Take a look at record 6, PatientLog. It is a master-detail under the C_BPartner model.

> Model Maker

Master	<input type="text" value="C_BPartner"/>	Name	PatientLog	<input checked="" type="checkbox"/> Active
Column Set	Name, PatientActivity_ID, C_BPartner_ID, AD_User_ID, Description, C_Location_ID, M_Product_ID, Q#QtyDosage, BedRegistration_ID, Q#MaxDosage, Q#Percentage			
Callout	QtyDosage>Percentage=QtyDosage / MaxDosage * 100			

It has Q# which is a Quantity data type. Note the Callout field has a value:

QtyDosage>Percentage=QtyDosage / MaxDosage * 100

This will be a JSR223 meta-data callout rule. It means the Qty Dosage field when changed shall update the Percentage field with the value of field, QtyDosage divided by the value of field, MaxDosage multiply by 100. Basically this is a percentage calculation.

The JSR script generated will be more complex but that is the work of Ninja. You just use such simpler syntax. After generation, you can see under the Column Callout pointing to a Rule that has the following script.

Event Type	Callout
Rule Type	JSR 223 Scripting APIs
Script	<pre>import java.math.BigDecimal; import java.math.RoundingMode; import org.compiere.util.Env; if (A_Tab.getValue("QtyDosage") != null) { A_Tab.setValue("Percentage", ((BigDecimal)A_Tab.getValue("QtyDosage")).divide((BigDecimal)A_Tab.getValue("MaxDosage"),4, RoundingMode.HALF_UP).multiply(Env.ONEHUNDRED)); } result = "";</pre>

In the next record, we can see something interesting. It is the same PatientLog model, but it has another Callout. This means it shall create just as describe - Name>Description=C_BPartner.C_BP_Group.Name.

Master	<input type="text"/>	Name	PatientLog
Column Set	<input type="text"/>		
Callout	Name>Description=C_BPpartner.C_BP_Group.Name		

The field Name when changed, shall have the Description field updated with the C_BPpartner contained property C_BP_Group which will be looked up and return its value for Name. In other words, it will just pass the BPartner Group's Name and store in the Description field.

Again, this simple script will be converted to a more complex JSR223 rule set, which is as follows:

Home
Rule: beanshell:calloutName... x

Rule
6/6

Search Key beanshell:calloutName

Name beanshell:calloutName

Description beanshell:calloutName from Aladdin Magic Plugin. Thanks to red1 @ red1.org

Comment/Help

Event Type Callout

Rule Type JSR 223 Scripting APIs

Script

```
import org.compiere.util.Env;
import org.compiere.model.MBPartner;
import org.compiere.model.MBPGroup;
MBPartner C_BPpartner = new MBPartner(Env.getCtx(),
(Integer)A_Tab.getValue("C_BPpartner_ID"),null);
MBPGroup C_BP_Group = new MBPGroup(Env.getCtx(),C_BPpartner.getC_BP_Group_ID(),null);
A_Tab.setValue("Description",C_BP_Group.getName());
result = "";
```

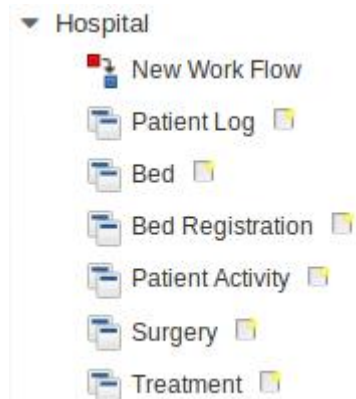
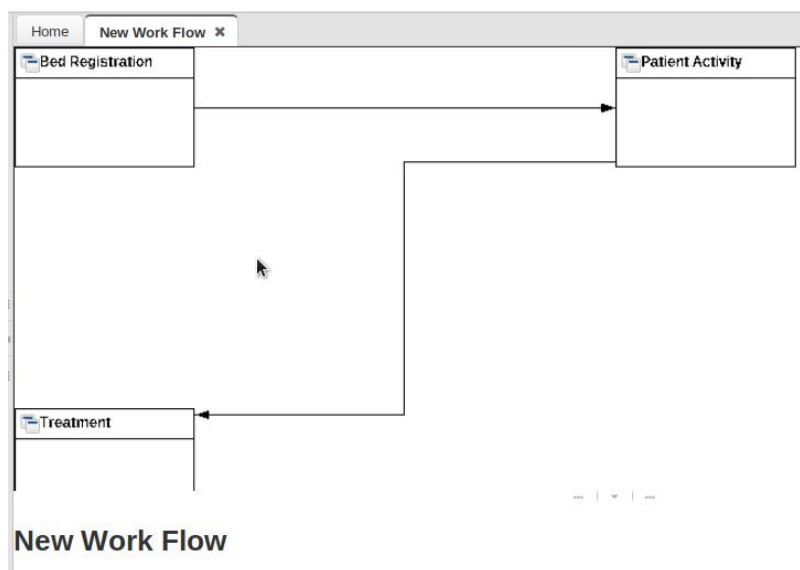
Chapter 5 - Workflow Model and Info-Window

Now we look at the next record that has a Workflow Model value:

Master	<input type="text"/>	Name	<input type="text"/>
Column Set	<input type="text"/>		
Callout	<input type="text"/>		
Workflow Model	NewWorkflow=Bed Registration, Patient Activity, Treatment		

NewWorkflow=Bed Registration, Patient Activity, Treatment.
Note that the equals sign has no spaces before and after, and the commas are tight too. The names of the Windows are exact names as appear in the windows with spaces included.

A new **Workflow Model** will be named **New Work Flow** which shall appear in the Menu Tree. Double clicking it and it shall show open to show a linked icon graph of 3 windows as below.



Next record has an interesting method.

Master	<input type="text"/>	Name	<input type="text" value="Partner, BedRegistration, Bed"/>
Column Set	PatientActivity_ID, QtyDosage, Percentage, DateIn, DateOut		

Its Name field has multiple table models:

PatientLog, C_BPartner, BedRegistration, Bed

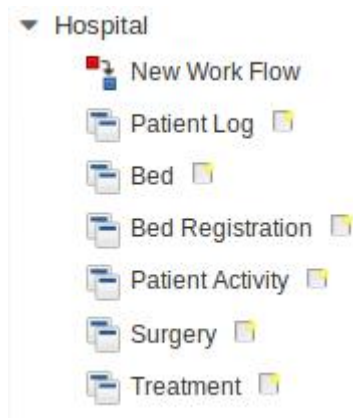
This shall create an **Info-Window** that is based on an SQL View of these tables. Make sure these tables are already created prior to this. Their sequence must be earlier. And make sure their spelling is exact.

The Column Set field contains columns that shall be added Info-Columns. The result shall be as below.

When you run Generate Module for this Hospital set, you should get an all grayed success. You can unchecked the Active columns of the Code Maker side if you just wish to see the model generation success as well the Callout Rules and Info-Window as explained earlier.

Model Header		
COMPLETED - HEADERS:1 MODELS:7(COLUMNS:34) CODES:0		
<input type="checkbox"/>	Menu Name	Version

Then you can relogin as Garden Admin and view the new Hospital module.



Chapter 6 - Spreadsheet Power

Now I show my most recent improvement which was done as I am writing this guide. Take a look at this Ninja Excel sample:

=TEXTJOIN("","",\$DataTyperMaker.B2:R2)			
E	F	G	H
Master	Name	Help	ColumnSet
	HR_Terminology	Preset DataTypes that shall be used later on.	L#Qualification=Phd/Masters/Bachelor/Diploma/Certificate,L#ContractType=Permanent/Contract/Temporary/OneTime,L#PaidBy=Hourly/Daily/Monthly/Project/Adhoc,Y#Certified,Y#YearsExperience,Y#Registered,Q#NoOfDays,Q#NoOfHours,,,,,,,,,

Notice how unwieldy the ColumnSet is. I am making two improvements. First, I am batching all long form hash datatype such as L#, Q#, Y# into a preliminary ModelMaker record so that they need not be repeatedly defined with the hash, as it will have been created and Ninja shall reuse what is done.

Second I create another source for this ColumnSet. Notice the formula behind it (above left of the screenshot. It is:

=TEXTJOIN("","",\$DataTyperMaker.B2:R2)

It is referring to another sheet and joining the contents of the columns in the range selected. So we go to that sheet and it is below:

A	B	C	D	E	F	G	H	I
DATATYPE	List of Static Values	List..	List..	Y/N Box	Quantity			
For Model Maker	L#Qualification=Phd/Masters/Bachelor/Diploma/Certificate	L#ContractType=Permanent/Contract/Temporary/OneTime	L#PaidBy=Hourly/Daily/Monthly/Project/Adhoc	Y#Certified	Y#YearsExperience	Y#Registered	Q#NoOfDays	Q#NoOfHours
DataTypeCode	L#	L#	L#	Y#	Y#	Y#	Q#	Q#
Name	Qualification	ContractType	PaidBy	Certified	YearsExperience	Registered	NoOfDays	NoOfHours
List Values	= Phd/Masters/Bachelor/Diploma/Certificate	= Permanent/Contract/Temporary/OneTime	= Hourly/Daily/Monthly/Project/Adhoc					
1 st	Phd	Permanent	Hourly					
2 nd	Masters	Contract	Daily					
3 rd	Bachelor	Temporary	Monthly					
4 th	Diploma	OneTime	Project					
5 th	Certificate		Adhoc					

Here, it is very obvious. I am setting out the string making even further separated into logical sets so that it is now easy to enter just the values without wielding other signage and symbols.

For instance, the first L# list you only need to enter the values in yellow highlighted columns. Qualification and then the list values below - Phd, Masters, Bachelor, Diploma, and Certificate. It shall automatically result in the top row as

L#Qualification=Phd/Masters/Bachelor/Diploma/Certificate

There is now less human redundant typo activity. And the layout is more elegant and cleaner for the data model designer to work with. Everything is in its own place and consultants and users won't be hunting high and low for any documentation or definition. Changes or corrections will be much faster and even lay users won't be confused at all.

Now, I didn't stop there. I converted quite all mundane ColumnSet sets to be hands-off and refer to another sheet where all model fields or properties are nicely laid out.

See the following screenshot:

=TEXTJOIN("","",\$ModelMakerSource.A2:A4)			
E	F	G	H
Master	Name	Help	ColumnSet
	HR_Terminology	Preset DataTypes that shall be used later on. Ninja Excel : DataTypemaker	L#Qualification=Phd/Masters/Bachelor/Diploma/Certificate,L#ContractType=Permanent/Contract/Temporary/OneTime,L#PaidBy=Hourly/Daily/Monthly/Project/Adhoc,Y#Certified,Y#YearsExperience,Y#Registered,Q#NoOfDays,Q#NoOfHours,,,,,,,,,
	HR_HumanResourceManagement	Main Tab,C_Bpartner_ID>Co	Name,C_Bpartner_ID,C_Bpartner_Location_ID
HR_HumanResourceManagement	HR_JobDesignation	HR Job Descriptions, AD_User_ID>Substitute	Description,Qualification,YearsExperience,Training,Experience,AD_User_ID,
HR_HumanResourceManagement	HR_Trainer	Trainer Details. C_Bpartner_ID>Trainer	Description,C_Location_ID,Registered,,
HR_HumanResourceManagement	HR_Rank	Rank Details	Name,RankBefore,RankAfter,Title,Prerequisite,Description

Now the ColumnSet sets below the Terminology are also been factored out. Notice the cell function on the top left of the screen. It is another TEXTJOIN referring to a new sheet, ModelMakerSource as shown on the right.

Now each ColumnSet refers to this separate sheet from rows 2 downwards column-wise. Row 1 is for Master-Detail relationship.

The heading names are also referenced into the ModelMaker. Thus, all changes or creation of data-models can be done here without worrying of the actual ColumnSet typo such as the commas. Extra commas are taken care by Ninja and truncated during actual processing.

	A	B	C
1		HR_HumanResourceManagement	HR_HumanResourceManagement
2	HR_HumanResourceManagement	HR_JobDesignation	HR_Trainer
3	Name	Name	Name
4	C_Bpartner_ID	Description	C_Bpartner_ID
5	C_Bpartner_Location_ID	Qualification	Description
6	AD_ClientInfo	YearsExperience	C_Location_ID
7		Training	Registered
8		Experience	
9		AD_User_ID	

Again, the data types are also not an issue as seen in some of these field names. They also reference the previous DataTypeMaker sheet to avoid typos. Thus we always make changes at the same source, at one place rather than all over.

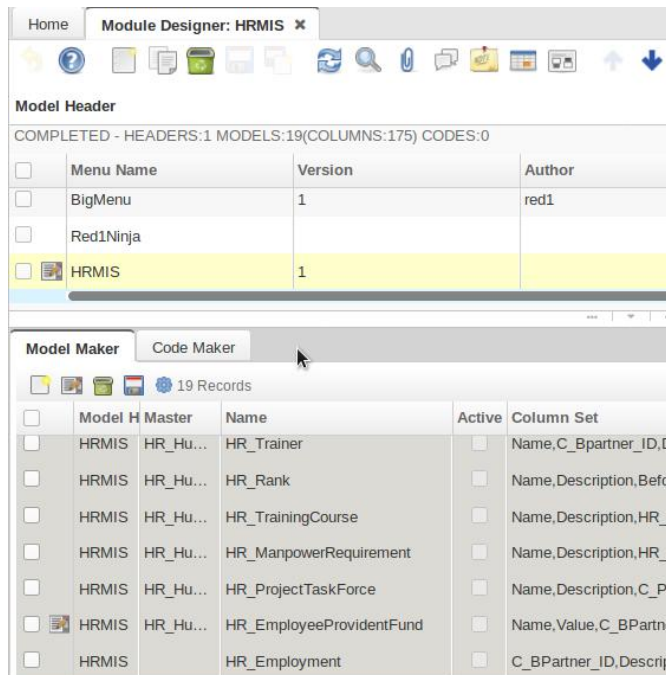
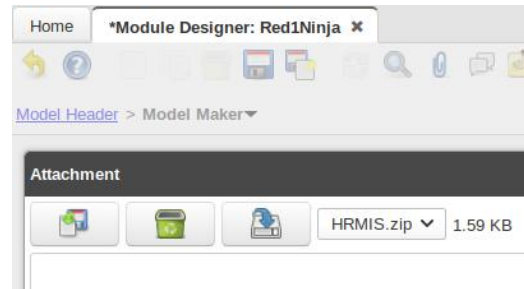
The magic is that, whatever is done here shall translate to exact and actual appearance in the software for your MIS or DBMS or ERP later.

So now i easily complete my data model design within a single day interspersed with long walks and leisure, and when i am done and unleash my Ninja, it is voila! Look at the following.

I upload the sample HRMIS (Human Resource Management Information System) that i am building from scratch at the moment as a brand new module (free) for iDempiere!

Fetch the Ninja_HRMIS.xlsx here - <https://sourceforge.net/projects/red1/files/HRMIS/>

So all I have to do is just zipped up the Excel exported 2_RO_ModelMaker.csv with its 1_RO_ModelHeader.csv and attach to an empty line in ModelMaker of the Module Designer and then run the Generate Module process. I get the new Model Maker called HRMIS (see below), and run Generate Module again on this new set.



Relogin as the Client and you can see at the Main Menu Tree pull down at the top mid of the panel, a brand new menu called HRMIS. Upon opening it up, you see all the models created in its entirety. Relationship, master-details, lookups, lists, and data types.

Simply magic.

On the following screenshot is the important Employment window with its sub-tabs. I quickly arrange its main fields by using the Tab Editor from the Window/Tab Process.

Now, who won't be impressed? You can quickly then go through with your prospect or client or user on the look and feel but more importantly the data model design whether it has taken into account all required fields and data relationship.

And the beauty is that you can do all this changes right at the original source Excel spreadsheet, and then export the CSVs and generate again, all within less than a minute, once you get a hang of it.



Home *Employment x

Employment

Inserted

Client* GardenWorld

Organization* *

Business Partner

Contact

Date Start

Employee Provident Fund

Position

Basic Pay 0

Contract Type

Description

Job Designation

Paid By

Employment History Medical History Payroll Payroll Details Family Members Appraisal Training Leave Application Assistance Request

0 Records

Active	Name	Date End	Date Start	Description	Employment	Exit Interview	Job Designation	Paid By	Position	Rank	User/ Contact
No Records found											

All the data models as designed in the Excel are now transferred into the iDempiere System and ready to use. It can now accept data, stored automatically within the database. The data relationship is all taken care of. Master-detail association correctly managed. There shall be no errors of any sort. All data type referencing be they check-boxes, pull down list, look up dynamic values done.

The most powerful part is as shown before this, is the reporting function. Row based reports are easily made on the fly. Later we shall see Master-Detail PrintFormat creation also done with a Ninja tick box. I also have nice patches to modify the underlying code to make JasperReport add-on (which was integrated into iDempiere to beautify print format) redundant 90% of the time.

Chapter 7 - The Module Design Approach

I have my preferred way of designing modules. Let's look at the HRMIS spreadsheet again:

	HR_HumanResourceManagement	HR_HumanResourceManagement	HR_HumanResourceManagement	HR_HumanResourceManagement	HR_HumanResourceManagement	HR_HumanResourceManagement
HR_HumanResourceManagement	HR_JobDesignation	HR_Trainer	HR_Rank	HR_TrainingCourse	HR_ManpowerRequirement	HR_ProjectTaskForce
Name	Name	Name	Name	Name	Name	Name
C_Bpartner_ID	Description	C_Bpartner_ID	Description	Description	Description	Description
C_Bpartner_Location_ID	Qualification	Description	Before	HR_Trainer_ID	HR_JobDesignation	C_Project_ID
AD_ClientInfo	YearsExperience	C_Location_ID	After	TrainingType	QtyOnHand	C_Campaign_ID
	Training	Registered	Title	Certified	QtyNeeded	DateStart
	Experience		Prerequisite	Qualification	DateStart	DateEnd
	AD_User_ID			Registered	DateEnd	QtyPersonnel

Remember that in any new module, we want to integrate some of the above to core tables such AD_User and C_BPartner. (In appendix we explain such tables in detail.)

There are two ways to put in new tables that needs reference to core tables. Either as a sub-table to the intended main table, or a separate table altogether with just a field within it looking up the Business Partner. In the above screenshot, you see both. The left most is the standalone table. The rest are Master-Detail.

However in this HRMIS example, none of them, both standalone and master-details are children of any core table. By right, HR is part of the C_BPartner (Employee) model. They can be, just that in my HRMIS, I do not wish them to be bundled to the core making them bulky. I wish the HRMIS to be a more loosely coupled module and only use the 2nd approach which is to look up instead. My HRMIS sample module makes all the tables children of either HR_HumanResourceManagement and HR_Employment (to be the stand-in for C_BPartner).

So in the first leftmost table of HR_HumanResourceManagement, I look up C_BPartner by pointing to its main primary key, C_BPartner_ID. In iDempiere (again this is an inherited feature from the original project, Compiere), it automatically refers to data records in the table referenced without the _ID suffix, which is C_BPartner. And the Name column values shall appear as a pull down dynamic list in the HR_HumanResourceManagement table.

Let me give another reason why in this case, I look up instead of make a child of. By looking up, you have the choice of leaving that field blank. Meaning it is not mandatory to have a core record in the Financials ERP core. This is both good or bad depending what you wish to achieve. I wish to have an HRMIS that can exist without the ERP. It can then be minimalist and standalone small MIS catering to a small client who wishes only for a HR Payroll system.

It is not so bad, for being separated from the Business Partner (Employee) main window, because with the Info-Window (another magic step from Ninja) you can easily join HR_Employment to C_BPartner or other models for information joining and reporting.

Now what do we do after defining the above in our spreadsheet? How do we setup the meta-data or configuration or any coding? It is all meta-data within the Application Dictionary and normally from Compiere days you have to do quite a number of steps to do this. But now with my Ninja, you won't notice it. Ninja code solves both for you. All you need is just that Ninja_HRMIS template.

You will notice in the HR_Trainer table, there is another reference to C_BPartner_ID. That is meant for the trainer record ID. The label on the window tab for this table will show 'Trainer'. How is the label changed? It is done via a smart use of a language translation file. This will be in later chapters.

In the HRMIS spreadsheet, we scroll further to the right and see HR_Employment as a new standalone and next to it is a sub-table to it. This is to put in past job experience or employment history of employees. Such data will be created anew. What we shall do is to prepare the data as Excel format also.

This new table is HR_EmploymentHistory. The prefix HR_ is to help identify the table easily as there are about 500 tables already in the core iDempiere. All are prefixed and we shall explain them in the appendix for basics.

The first line belongs to HR_Employment. This means It is a sub tab of HR_Employment table as can be seen below.

See how the properties in the spreadsheet appear on the actual window panel. If any changes wish to be made, the implementer just go back to this sheet and make the changes there. S/he can also add and delete fields and RUN_ImportIempiere to clean back the database and ImportModel again. This allows fast prototyping without running around correcting the AD manually and losing sight of what goes into what.

I	J
	HR_Employment
HR_Employment	HR_EmploymentHistory
Name	Name
C_BPartner_ID	Description
Description	ContractType
ContractType	PaidBy
PaidBy	AD_User_ID
AD_User_ID	DateStart
DateStart	DateEnd
C_BPartner_ID	HR_JobDesignation_ID
HR_JobDesignation_ID	Position
Position	Contact
Contact	T#ExitInterview
Q#BasicPay	BasicPay
HR_Rank_ID	HR_Rank_ID
HR_EmployeeProvidentFund_ID	

The screenshot displays the SAP HRMIS 'Employment History' window. The top bar includes 'Home' and 'Employment x' tabs. The main area shows a form with the following fields: 'Client' (GardenWorld), 'Organization' (*), 'Active' (checked), 'Contact' (James), and 'Contract Type' (Temporary). Below the form is a tabbed interface with 'Employment History' selected. The table below shows 1 record for Steven, with columns for 'Active', 'Name', 'Date End', 'Date Start', and 'Description'.

Active	Name	Date End	Date Start	Description
<input checked="" type="checkbox"/>	Steven			

Hopefully by now, you can begin to grasp the whole idea behind Ninja and the smart integration it has with the spreadsheet to create new models. Knowing such tool doesn't mean we can be all powerful. The GIGO principle still applies. You can use this tool to design a fragmented or elegant module. A tight or loosely coupled design.

Chapter 8 - The Migration Approach

Now, I describe how you can migrate whole legacy data the easiest way into iDempiere. I have simplified and revolutionise the approach to be dead simple which is crucial as migration is one of the scariest and deadliest part of a large MIS or ERP effort. Many projects failed even before it began due to the migration part.

Why does that happen? Firstly, data needs to be scrubbed or verified. There maybe redundant repetitive data, some or lots of it outdated. Mostly will be in multiple sheets of soft copy or worse still have to be input again from scratch and human errors abound.

In iDempiere as inherited from Compiere, there is an Import Loader regime to migrate data into the system. But it has to be properly formatted first. And the way it looks up and populate is hard coded. Thus it can break as time goes by due to changes to data models or you need to code each time you have a new model.

My approach dispense with all that. The principle is to match exactly the data model property headers to the underlying database design. Column for column. It is WYSIWYG - What you see is what you get.

But how do I solve the raw jumbled up formats? I again use the Excel spreadsheet as the killer app. As you will see further in the same HRMIS spreadsheet, there are another three parts of the spreadsheet, to the right of it.

The left or first part is where raw data is taken from the legacy system as is basis. Then the 2nd or middle part is the master sheet where the raw data is combined. Then the right side is where the data is transposed into core iDempiere database models.

I shall explain more clearly by taking the actual data in it.

Now before that, let me explain abit about the two types of actual user data that may exist in the database. One is the core tables. Such as C_BPartner, which is the Business Partner master data that is used for Vendors, Customers and Employees alike. So what we are to use shall be the exact format of this table to inject new data or migrated data from the outside. The point to remember is that to import data into such, requires all field labels or headers to be exact and its prerequisite or mandatory fields to be available.

Here is a list of core master tables that any data to get into it, must fit exactly:

1. C_BPartner - Business Partners
2. M_Product - Products
3. C_Order - Orders
4. C_OrderLine - Order details

Let us try a real life case which I already prepared. Fetch ProductMigration.xlsx from <https://sourceforge.net/projects/red1/files/UnicentaPOS/>. Open it up and go through each sheet.

»	+	Master	1_M_Product_Category	2_M_Product	3_M_ProductPrice	4_M_Product_BOM	5_M_Product_PO	6_M_Replenish
---	---	--------	----------------------	-------------	------------------	-----------------	----------------	---------------

Notice there are a number of sheets beginning from Master, 1_M_Product_Category, 2_M_Product and so on. Right away you can deduce that those with prefixes of 1_ until 6_ are main core tables in iDempiere. You can look through each of them and see that they are matching the actual database tables. Before we open up the Master sheet, let me explain what this spreadsheet is all about. It is to prepare data for BOM or Build Of Materials for a chain of cafe restaurants. Each product BOM has children or formula of ingredients at the precise or estimated quantities so that when a parent product such as Spaghetti or Coffee is ordered, its exact amounts of ingredients used are calculated. This is so that the central store will be notified of each branch's usage automatically. This is one brilliant idea I myself did.

	A	B	C	D	E	F	G	H	I	J
1		Table 1								
2	Category	Name	UOM	Selling Price	Purchase Price	Stock Qty	Vendor	Reorder Qty	Time to Deliver	Location
3	Food	Spaghetti	Each	13			Seed Farm Inc.	10		1 HQ Warehouse
4		Burger	Each	11			Seed Farm Inc.	10		1 HQ Warehouse
5		Ground Nuts	Each	4			Seed Farm Inc.	10		1 HQ Warehouse
6		Pizza	Each	12			Seed Farm Inc.	10		1 HQ Warehouse
7		Lunch Set	Each	16			Seed Farm Inc.	10		1 HQ Warehouse
8	Drinks	Coffee Latte	Each	4			Seed Farm Inc.	10		1 HQ Warehouse
9		Teh Tarik	Each	2			Seed Farm Inc.	10		1 HQ Warehouse
10	Ingredients	Flour	kilogram	3	2		Seed Farm Inc.	10		1 HQ Warehouse
11		Rice	kilogram	6	5		Seed Farm Inc.	10		1 HQ Warehouse

Like mentioned in passing earlier, there are three sides to such a spreadsheet to migrate actual data. The left side is the raw data from the legacy. In this case we do not need it as this is fresh data. The middle is the Master data and it is as shown above. On the right of it, are all core tables data.

When you look through each sheet of the core table data, in each column data, you will see that they are referring to this master data. If you wish to change anything, you just change it here in the Master. It is that simple. The idea is to let you focus on your own legacy data and not worried where the exact fitting table layout is.

Let us look through one by one to see this concept in practice. First we take the first sheet tab to the right labelled 1 M Product Category.

Look at cell C3. It refers to =\$Master.A3. So this way we just edit data in the Master and the rest of the sheets shall change accordingly.

C3		fx Σ =	= \$Master.A3
	B	C	D
2	Value	Name	PlannedMargin
3	Food	Food	0
4	Drinks	Drinks	0
5	Ingredients	Ingredients	0

fx Σ = =\$Master.A10						
Value	Name	C_UOM_ID	M_Product_Category_ID	C_TaxCategory_ID	Description	IsBOM
Spaghetti	Spaghetti	Each	Food	Standard		Y
Burger	Burger	Each	Food	Standard		Y
GroundNuts	Ground Nuts	Each	Food	Standard		Y
Pizza	Pizza	Each	Food	Standard		Y
LunchSet	Lunch Set	Each	Food	Standard		Y
CoffeeLatte	Coffee Latte	Each	Drinks	Standard		Y
TehTarik	Teh Tarik	Each	Drinks	Standard		Y
Flour	Flour	kilogram	Ingredients	Standard		N
Rice	Rice	kilogram	Ingredients	Standard		N
Sugar	Sugar	kilogram	Ingredients	Standard		N
Salt	Salt	kilogram	Ingredients	Standard		N
Meat	Meat	kilogram	Ingredients	Standard		N
Chillis	Chillis	kilogram	Ingredients	Standard		N
Veg	Veg	kilogram	Ingredients	Standard		N
Bun	Bun	kilogram	Ingredients	Standard		N

This table data is a pre-requisite to the M_Product data to be grouped under these new product categories. The row 2 header is the exact match of the table column-names. During Ninja's ImportModel, the rows of data below them will be inserted accordingly.

We now look at the next sheet tab, M_Product. We see again how the cells are referring to the Master sheet for non-redundancy convenience and accuracy.

Here can proceed and examine each tab and see how the new data is been laid out. It is intuitive, and amazingly easy for the data migration process. There can be no error because these sheets are tested in my real user case. I am now sharing for reuse to the public and community.

As I said before, this to me is more superior and handy than the current Import Loader as data is more visible and easily managed in a single master sheet. Also, the new iDempiere window tab direct Import isn't that easy when you have many different table sets to handle. Each table requires you go to that Window Tab and attach the data set individually. That is cumbersome and doesn't allow you single spot visibility and reference. So, for a multi-table data-set such as for BOM sets, this is no longer a hindrance.

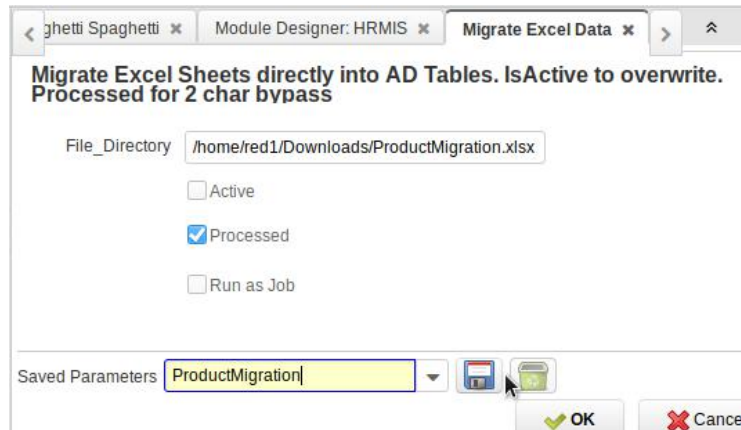
This way is more quicker and zipping them (only the exact core right-hand side tables) into a single zip and using Import Model process at Ninja's ModelMaker lets you do the importing with a single act.

So, this is the way of migrating to existing core tables. But what about non-core? The concept is still the same. But I added one more magic. You may skip the Table-Column creation and jump. No extra steps if your data type references are already created such as L#, D# and Y#. Just put your data in defined headers, and Ninja goes further by creating new tables at the same time while migrating! You can return back to the HRMIS spreadsheet tabs sheets' design to see the same principle. Take those data sheets and just zip them and import them. You only need to import first the HR_Terminology under the Model Maker.

However, if you want the option to code logic, where you have to create the process and model code stubs, you have to include the rest under the ModelMaker column.

So now, for example, to import data directly into one such model, HR_EmploymentHistory, we just need to prepare a sheet containing data that fits the properties in the table. Just remember that since it is a sub table, it needs to have a value at the HR_Employment_ID that already is present in the Name property of the parent HR_Employment table. Just go through those sample data in the HRMIS Excel to get the idea.

BREAKING NEWS! Latest Direct To Ninja method also for these sheets to come into the ERP database. Just remember any new data models must already exist. If you want to create new models during migration on the fly, attach them as zip CSVs and use ImportModel instead.



Using **Migrate Excel Data**, run it from your Client that needs the data. Within 3 seconds, the Excel below is turned into migrated data!

Search Key	Name	Description	Product Category	D Tax Category
Burger	Burger		Food	Standard
GroundNuts	Ground Nuts		Food	Standard
LunchSet	Lunch Set		Food	Standard
Pizza	Pizza		Food	Standard
Spaghetti	Spaghetti		Food	Standard

Organi	P Line No	Active	Des	BOM Type	BOM Product	BOM Quantity	Part Type
*	10	✓		Standard ...	Flour-Flour	0.3	
*	20	✓		Standard ...	Salt-Salt	0.04	
*	30	✓		Standard ...	Chilis-Chilis	0.05	
*	40	✓		Standard ...	Veg-Veg	0.1	

Notice the ProductMigration.xlsx Excel sheet names correspond to the AD table names. They are preset to map exactly and in correct queue order. Once you login again to your client, you will notice the migrated records in working order.

PART II

Chapter 1 - My Best Practice

What is the best way to learn programming? Why, it is bugs, of course. Make lots of them. Actually you do not have to. Just strike the keyboard for the first time. 110% you hit a bug. It is natural part of life. First time you try to feed yourself. First time you go on a bicycle. That learning process cannot be stopped. Or else you never learn. Up till adulthood that is our pattern and best practice of learning and living. Trial and error.

What is the best way to learn iDempiere and now with Ninja? Read above or listen carefully to me, someone who has been around the IT world or IoT world if you prefer the latest rehash of old terminology, since 1980. Right after I left high-school. At a time when there were only mini mainframes. No Internet, no mobile phones and certainly no Google to fact-check anything we say.

But experience is something that Google often miss. It is something best not said in words as it can be misleading or becoming a ten commandments cast in stone which gets broken every time it is tested.

My best practice is something I went through uniquely and it cannot be cast in stone. But I be the happiest person if any of them gets broken. I love learning something new. And I will always shout it back out to the world. That is why this project is so made for me. Openness, transparency, no information hiding. So this is my first best practice rule. If you do not publish, you perish. 'Publish or Perish'. Wait, it was already said. So here i am going to rehash that to,

'If Linus Torvalds did not release Linux, someone else will.'

And guess what? You be blamed later anyway. In fact, the original creator of Linux is a university professor teaching software, Dr Andrew Tanenbaum. He called it Minix as a sandbox of Unix, the prevailing server side operating system that time. Linus called it Freax, an acronym of Free and X. Someone else called it Linux and it stuck. Why I put Linus Torvalds name in there? To prove a point more clearly. Prof Dr. Andrew Tanenbaum could have unleash it first, but instead he was too high up in his cathedral. He even criticized Linus for releasing it.

But Linus real mantra isn't that. He is attributed to the phrase,

'Release Early, Update Often'

When he released early, mistakes were abound, and other techies rushed in to correct them. Thus his time spent is actually a very small fraction of the total or actual cost of releasing Linux. This i found to be true in the iDempiere project and in my own. Others gave me bits and parts that eventually helps me more in the long run. I have no worry of anyone stealing my trade secrets because A: I am not the smartest guy in the room, and B: It is still too difficult to make what you steal work the way it supposed to be, no matter how much you read up or open it up. It is much better and cleaner to let your mechanic change your engine oil for you.

In a way, choose your mistakes well. Do not try to learn everything at one time. It takes a long time to arrive at where I am. And i am moving my goal posts every day. Sometimes every hour as i sit to study and improve my work.

In essence, what i write is the culmination of many mistakes and Aha moments.

Do Not Touch The Engine

This is often heard among my project members or students. It is wide spread in quite all my writings. You probably hear the same among car mechanics not to touch the engine to ensure nothing is broken further. Instead check the fusebox. It could be just a matter of a fuse, instead of pulling a long wire to rewire the headlamps. It maybe a quick fix. But it can be a fatal one. Something else may be broken. What is worse, when you change mechanics, the new one has no idea what that wire is for.

OSGi is the technology that allows the core engine to be decoupled from your module or vertical functionality. It is used by Eclipse when IBM acquired Equinox to be contributed together with the IDE (Integrated Development Environment).

Do Not Touch The Core Tables

What this means is not to over-write or even append to any master table. Instead append as a sub-table, or as I recommended in my HRMIS sample, a separate table that has a field that looks up the core table reference.

I give a good example. A classic one as explained before in my BlackPaper here:

<http://www.red1.org/BlackPaper.pdf>

In fact my Black Paper tells everything you need to know what Not to do. So read it together with another PDF below of more best practice that you must do.

Don't Test Alone

If you read my FitNesseTesting pdf here, it says to do it from outside your organisation!

<https://sourceforge.net/projects/red1/files/Testing/iDempiereFitNesse.pdf/download>


Most people test their software from the inside, by the very programmer who programs it. That is an anti-pattern of Software Engineering. Yes, you may test it, but that test is not official nor acceptable that you have passed the testing stage. This is not to be confused with UAT, nor to be passed on to the users during UAT, as some will inadvertently do.

UAT or User Acceptance Trial is as the name suggests. It is for the user to do their own final assessment. But for them to find bugs that are deep is not fair. There are bugs that crop up at year end, or Xmas holidays, when no one is working. Including the software.

And continuously test, test, test, as the mantra goes. This can be achieved by a best practice tool called Jenkins, a fork of Hudson. Here is the iDempiere testing site -

<http://ci.idempiere.org/>.



S	W	Name ↓	Last Success	Last Failure	Last Duration
		delete_maven_cache	2 mo 15 days - #1	N/A	4,7 sec
		globalqss-iDempiere-LCO	5 days 4 hr - #79	5 days 5 hr - #78	1 min 23 sec
		iDempiere	7 days 14 hr - #89	N/A	8 min 20 sec

[Use The Ninja](#)

It is not to self-promote, but I created the Ninja wizard is to implement the best practice forcefully. It facilitates that all modules are built as they should be - as modules, separate, decoupled and talking to the core via clearly defined interfaces.

With nothing changed in the core, you enjoy:

1. No breakage during upward and forward migration.
2. Less maintenance as the ones that break are only your own modules. And you already know where to look. Debugging can be tricky as you work across an enterprise. So narrowing your sight allows you to focus and be up and running to the maximum.
3. Non-repetitive redundant work in setting up new AD models in your module or creating new Java code that is always boilerplate.
4. The Kanban Board document event handling also helps in making your own module less clumsy and spaghetti strewn as event changes to the Document status are handled in a single spot, namely the DocEvent java already generated automatically for you to code just the logic.
5. It avoids you from making human mistakes in the AD changes to setup menus, windows, tables and columns, as well as processes parameters, info-windows, data types, and print formats and thus saves you a lot time even days on end.
6. It gives a simple point of reference as Ninja picks off from a single spreadsheet.

Just Enough Java

Java is perhaps one of the hardest thing to learn, even for me as i was a Cobol programmer in 1982. Object Oriented concepts are really alien, and it took me years , from 1996 when it was at version 1.0 till 2003, perhaps Java 1.6. Today at time of my writing it is at Java 11 and change owners from Sun Microsystems to Oracle Corporation. But with the Compierre project where I could intercept the code during execution and see what is actually happening, I began to master it for the first time.

Today, I still considered myself in the learning process, but I have learned enough to at least craft the basic data model's business logic and leave the IoT style framework handicraft to the top gunners of our project, Carlos Ruiz and Low Heng Sin and Hiep Le Quy. They make up the Circle of Trust that reviews the final code versions in the iDempiere project.

To learn to their level is redundant or too difficult for most people, me included. However to use iDempiere to make specific functionality not covered by the horizontal ERP or MIS, we have to sooner or later touch code. Or at least read the code to understand how things work or go wrong. It is also the hallmark of been open source to have the ability to access code and modify it and eventually improve it and share the improvements on a global scale.

I have instituted the Ninja template to allow such data model coding to create processes, and data model events. The logic will be more focus to real life business functionality and not be burdened by any boilerplate code. Of course the most perfect world is the use of a Business Rules Engine, but that is another big learning curve and so far no one in the top circle has made any mention of it. Other than me. And I cannot get my head around the best way to do it yet. Whether i be using Drools or a Ninja style approach. But I am still thinking about it now and then.

Meanwhile, I feel the Java that is needed is minimal and sufficient to fulfil quite all the needs of any vertical modules that may exist. I have worked on numerous module creations such as Budgeting, Manufacturing, and Warehousing and know the scope of Java skills needed. The Ninja itself as my creation was done using the same Java knowledge. Nothing further.

In my other project work, such as Android Mobile integration to WebServices, and Unicenta POS integration via Active Message Queue, I encountered more Java frameworks and those are particular and specific but not needed here where most business logic resides.

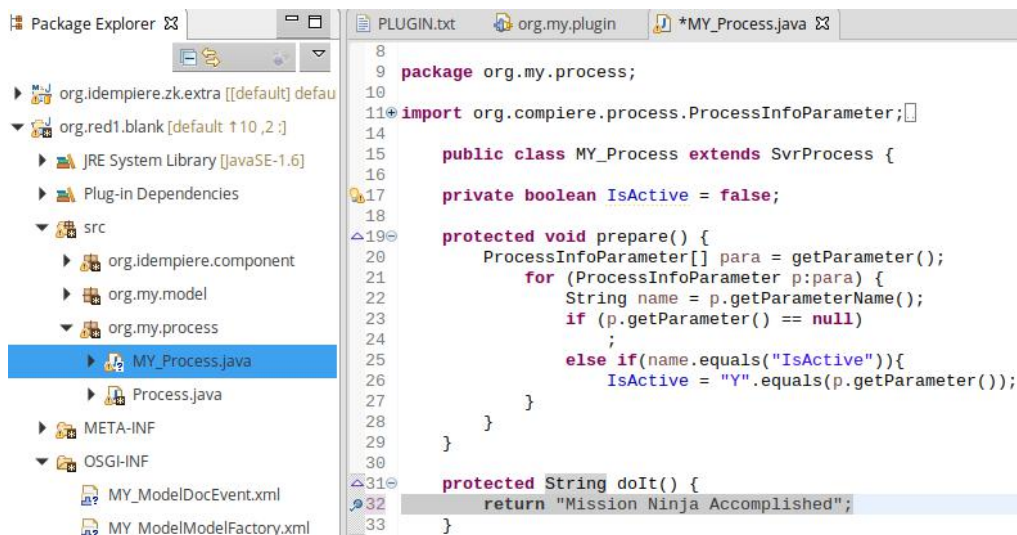
With this sufficient set of Java knowledge, coupled with the Ninja wizard, one should be able to build any module needed in the world.

However for the front end or the world of IoT, Ninja or iDempiere is not needed. MIS or ERP is essentially the back-end of a system. The front end it has is sufficient but not as modern as some front-ends that are emerging as we speak. Thus, let them be, as they usually talk to the back end via WebServices and Message Queue technology. I might cover them in this guide at a later version. For now, i shall go to the basic set of Java code that is intended.

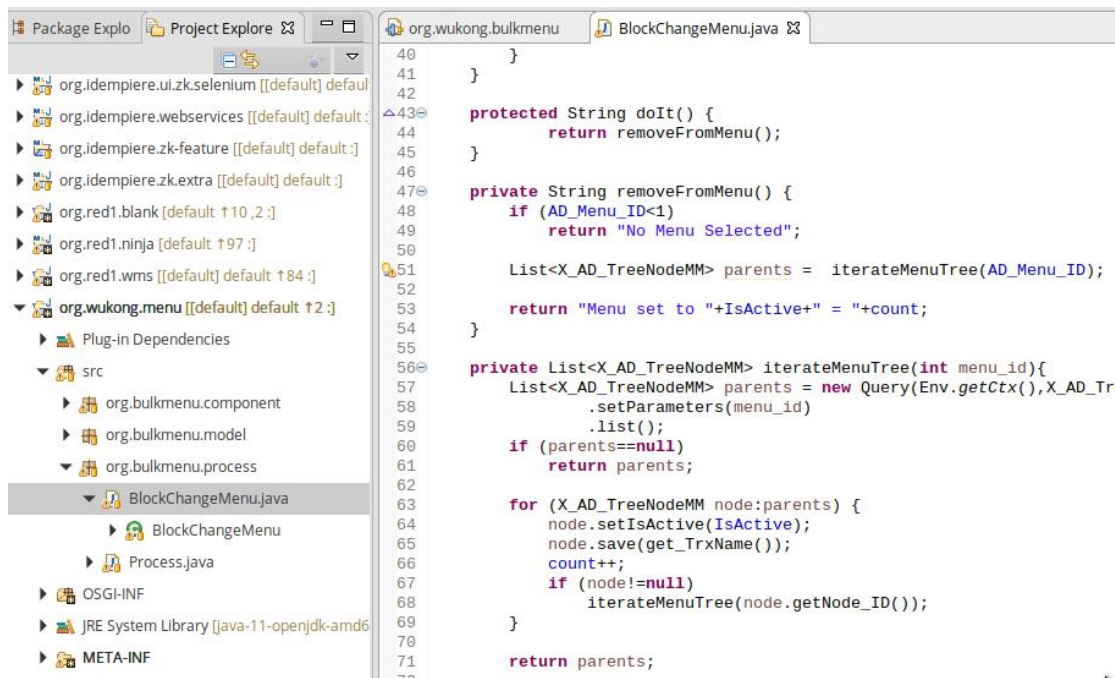
Below is a sample already shown in the early chapter of what is left for the coder to do in the case of a process. First, let us understand what is a process and what does it do. A process is to do something further within a standard data model. The generated data model already has just simple CRUD stuff. CRUD stands for Create, Read, Update and Delete. A process is a more advanced CRUD realm, where data from the table is now used to access further data outside and subject to business rules before updating back to itself or other tables.

An iDempiere process is first defined in the AD, and thus is a meta-data configuration, not needing any hard code but to point to a record or instance, to retrieve the intended full details to be iterated and acted upon. The Ninja setups all the steps in the AD for a process and its parameters, so that the developer only needs to code the pure business logic.

THE SAP KILLER



What is to be coded is at the grayed line 32 of the `doIt()` method.. More logic can be inserted before that line returns any results intended. I will show a typical code sample and then explain what it really does.



This is from the `BlockChangeMenu.java` under my other magic plugin, The Wukong Menu. It basically make any part of the main menu tree appear or disappear. Like magic. The trick is just to mark the checkbox active or not. So let's see the `doIt()` method above.

After it takes the `AD_Menu_ID` parameter from the selection by the user, it checks if it is not selected with, if (`AD_Menu_ID<1`). It will then return to the panel with the message 'No Menu Selected' reminding the user to choose something.

It shall then pass the process to a sub method called `iterateMenuTree(AD_Menu_ID)`. At that method, the passed value is put into a Query method. Below it is copied in full.

```
List<X_AD_TreeNodeMM> parents = new
Query(Env.getCtx(),X_AD_TreeNodeMM.Table_Name,X_AD_TreeNodeMM.COLUMNNAME_Parent_ID"=?",get_TrxName())
    .setParameters(menu_id)
    .list();
```

The Query Program

First let us understand what is this Query method. It is what in the Java world called an ORM or Object Relationship Model where data from a database is queried. Previously even in Compiere, the way it was done is using the SQL or Structured Query Language method. As the name implies, that is a structured and not an object oriented method. What do we mean by that?

Structured is monolithic, linear as how English sentences are written. Below is an example of the Query method rewritten in the old SQL:

```
SELECT * FROM AD_TreeNodeMM WHERE Parent_ID = AD_Menu_ID;
```

Now that looks not too bad and simpler right? Wrong. It is actually a string and it can be hacked during execution by what is known as SQL Injection. It is also bulky and unwieldy. The full code to execute the string is as follows:

```
public <T extends PO> List<T> list() throws DBException
{
    List<T> list = new ArrayList<T>();
    String sql = buildSQL(null, true);

    PreparedStatement pstmt = null;
    ResultSet rs = null;
    try
    {
        pstmt = DB.prepareStatement(sql, trxName);
        rs = createResultSet(pstmt);
        while (rs.next ())
        {
            T po = (T)table.getPO(rs, trxName);
            list.add(po);
        }
    }
    catch (SQLException e)
    {
        log.log(Level.SEVERE, sql, e);
        throw new DBException(e, sql);
    } finally {
        DB.close(rs, pstmt);
        rs = null; pstmt = null;
    }
    return list;
}
```

Now you understand? It is not only bulky but error prone because as a string typed script, the developer can inadvertently make typo errors and string content mistakes are not noticed during compile time but during run-time. Containing this bulky method in a Java pattern inside the ORM framework controls the SQL injection not to be possible by the calling method and strictly housed inside the Query method which has measures to prevent SQL Injection. For example the buildSQL sub method it to build the SQL string is doing it this way:

```
private final String buildSQL(StringBuilder selectClause
```

Notice it is declaring a final String object which does not let the object be mutated during its lifetime after it has been created here. Then the DB method is finally terminated at the end of the Query method with

```
finally {
    DB.close(rs, pstmt);
```



```
rs = null; pstmt = null;
```

The Query methods build the SQL string by taking the arguments from the calling Query method which was sent as objects but parsed to its final SQL formatted string.

This is brought together in object oriented fashion, and only assembled automatically once here and thus guarantees no lazy programmer forgot to do those final declarations. There is no need to repeat the whole DB method each time you wish to handle an SQL. Using a Query method ensures no typo error as you be using object arguments in the script such as TableName, COLUMNNAME, and .list. Any mistake while typing those will not compile but highlighted for you to correct time immediately rather than unknown and during run-time which is fatal.

The shorter Query method immediately returns this bulky work as an object called 'list' that is cast back into the calling method object, in this case, AD_TreeNodeMM. The developer can focus more easily with better visibility. I will now explain about the TableName and COLUMNNAME objects first before going through the rest of the iterateMenu logic.

Those two, TableName and COLUMNNAME types are objects marshaled by the ORM setter and getter framework that has already existed since Compiere days. It is not to be confused with the ORM Query framework I talked about.

The Setter/Getter framework is a basic mechanism to obtain the values of any property and write to it, rather than the the SQL string method. For example,

"SELECT Description FROM AD_User" is now done by MUser.getDescription();

The MUser class extends from the X_class which has the get and set methods. Using the M or model class is to house further routines that we wish to add such as getLines or getParent in a Master-Detail relationship. You can find most of the core ORMs in the org.compiere.model package under the main org.adempiere.base plugin.

However this basic set/get ORM is still insufficient for more versatile operations of obtaining a set of values from any Model class as seen in our example. That is why the ORM Query framework was introduced quite early after the ADempiere fork by Teo Sarca and Victor Perez. It immediately got adopted as our best practice.

(To be cont'd. Comments are welcome. Tell me what you do not understand, or can be written more clearly, and I will edit the content)

Ninja 2.2 idea:

Process to read Excel sheets Header, Model, Code to respective RO_ tables.

Chapter 2 - Support & Warranty

Using Open Source, is like jumping from the frying pan into the fire. If you debunk SAP, you have a good problem. Now what? If you hit a bug, or if something for no reason happens, just as in Murphy's Law - 'Anything that can go wrong, will go wrong'. That often happens in the technical world. Who do you call then?

That is why one of the biggest argument against large organisations using Open Source alternatives is that there is 'No Neck To Choke'. Strange isn't it? The first evil of closed proprietary software is vendor lock-in which locks the users for life to the maker of the software without a chance to escape but until death do they part. Either the death of the user or the death of the vendor, of which can easily happen, the last time I check with the computing world.

But without the vendor lock-in, or protection from the Mafia, you are now all on your own in the open. Software is most likely not the users' business. They have their own businesses that use the software as a tool. It is best that tool is left to the experts to manage.

That is why I been advocating best practice with the setting up of a Centre of Excellence in each country by the community to give professional and commercial support that has to match the debunked SAP consulting houses. Here in this section, I shall give some ideas what a CoE is.

Centre of Excellence

A CoE plays many roles.

1. Act as a reference and resource centre of materials, training and help centre
2. To promote the project to the public and hold conferences or boot camps
3. Conduct certification of professionals and ensure a minimal standard of competency among practitioners of the project's products
4. Open to membership of either paying members or sponsors that have the right to vote, as well as associate members that has a voice too but may not have a vote on the annual agenda.

Below is illustrated the many benefits to each category of the players or stakeholders.

University	Students	Market-Place	Project
More students intake with foreign interest	Start Career Early On The Job Training	Benchmark Statistics to Validate Assumptions	Reference Centre for Successful User Cases
More original research papers with global citations	Overseas Assignments	Source of Resource Outsource	Branding of Project Contributors
More grants and visibility in social media	Earn Income from Project Success	Certification, Compliance, Confidence	Corporate Market Access
Future High Ranking Worldwide	Subject Matter Experience	Corporate Confidence	Market Validation
Joint ownership of Software Apps	Reverse Brain Drain and Unemployment	Freedom from Foreign Software	Increase in Project Funding

The marketplace and the iDempiere project stands to gain a lot in the long run with more comprehensive market validation and adoption by serious big boys. (to be cont'd)