

# **SAGA.M31**

## **– Galaxy –**

Whitepaper for a new form of Data  
Integration

© 2004, SAGA D.C. GmbH  
<http://www.sagadc.com>

# Table of Contents

<b>1. EXECUTIVE SUMMARY.....</b>	<b>1</b>
<b>2. THE REASON FOR A NEW TECHNOLOGY .....</b>	<b>2</b>
2.1. The experiences from the past .....	2
2.2. Cost saving aspects.....	3
2.3. Demand of the market.....	4
<b>3. THE FACTORY PROCESS .....</b>	<b>6</b>
3.1. How does that all work? .....	6
3.2. A more detailed view .....	7
3.2.1. Record an action.....	8
3.2.2. Customize the recorded action .....	10
3.2.3. Build an Information Container .....	12
3.2.4. Grant permission for the Container to Users .....	19
3.2.5. LU Status Page.....	20
3.2.6. Future Steps .....	21
<b>4. THE ROADMAP OF SAGA.M31 –GALAXY–.....</b>	<b>22</b>
4.1. Two different Versions.....	22
<b>5. FEATURES OF SAGA.M31 – GALAXY – .....</b>	<b>23</b>
<b>6. SYSTEM REQUIREMENTS .....</b>	<b>24</b>
6.1. Basic Version.....	24
6.2. Enterprise Version.....	24

# 1. Executive Summary

In the past decades, IT developers worldwide spent several million hours to create code that implement data from different sources into their projects. The complexity of a project has risen as the source of information gets more heterogeneous.

How often was a migration process delayed, just because of the data source was not possible to implement in a specified time?

The reason is as simple as making a telephone call. Does anybody today really think about the complexity for the underlying technology that builds a phone system? Making a phone call from a mobile telephone to a telephone system that is part of a Voice-over-IP switch is, from a technical aspect, a complex task. From a user it is simply expected that it works. This approach was the driving factor that has us created an information factory called **SAGA.M31 – Galaxy –**.

Looking into a factory process, the customer is interested in getting a specific product with certain specifications and a unique Part number. From the view of the customer, he does not care what parts this specific product was built with. He requests a specific quality and product. The “Factory” has to take care that these requirements are fulfilled. On the other side, the manufacturer of this product has to deal with several suppliers providing the parts for his product. A good structure normally provides the same parts with identical qualities from different suppliers to make sure that the manufacturer will not be in risk if a supplier increases the price.

So at the end, a product is build from several parts, provided by different and redundant suppliers. That exactly is, what **SAGA.M31 – Galaxy –** does.

It provides a unique Part number (the information container) that contains parts from different suppliers (the information fields) consistently. If an information supplier has to be changed, due to a migration to another platform (Host->SAP) or because of the result from a A&M process, the information container (the product) is not changed. What has been changed is the source of information.

Using this technology, embedded in simple XML structured Requests / Response Processing on a modern platform (WebSphere, TomCat, WebLogic etc), **SAGA.M31 – Galaxy –** provides an interface that can save millions of Euros (Dollars) because nobody is requested to change the application when the source of information has changed.

The complexity of different data-sources like IBM Mainframes, iSeries Platforms, Databases, Directory Servers or ERP-Systems is reduced to a simple XML Request / Response. **SAGA.M31 – Galaxy –** takes care of the consistent data delivery mechanism and structure.

It's a revolution of Data Integration. It was never easier to access mainframe data and database information parallel and without coding hundreds of lines for a simple field. Using **SAGA.M31 – Galaxy –** it is no longer necessary to store Mainframe Data on a intermediate platform because it is too complex to access this data. Use realtime access to this company critical source without knowing how it works. **SAGA.M31 – Galaxy –** does take care on this and knows exactly what and where the information is located.

## 2. The reason for a new technology

### 2.1. The experiences from the past

As a company that has more than 15 years experiences in Data Integration within the Host environment, we learned that it is in most cases pretty complex to get a project into production where the information are stored on different platforms. Also, from a business point of view, we often faced the problem that a project was first delayed and finally stopped because the customer decided to migrate to another system on a different platform.

One of our company slogans “Think strategic, act pragmatic” did help in some circumstances, but most customers delayed their plans until the migration is complete. Well, we all know that these kind of migration projects normally does not follow the plan and in the majority of all projects, the deadline is moved into the future.

Another issue is the way, how applications are written. Even if the applications are object oriented, they have to deal with the source of the information. Programming languages like JAVA™ or .NET™ are structured to run on different operating systems and should therefore be “platform independent”. This is true (in most cases) for the code but never for the source of data. A JAVA-Programmer today is not requested to know the differences in allocating Storage, maintaining the threads or anything like this. What he still needs to know is the answer to a simple question: “Where is my data?”

Standardization processes made it easier to implement SQL Statements for different databases (even if there are different flavors of SQL implemented), LDAP is a simplification of accessing Data stored in a Directory Structure. But if the source of information is changed from a database to a Directory-Service, every program that deals with this data needs to be accessed and at leased checked if there are any dependencies that are related to the underlying technology of the data storage.

Making it a little bit more complex is simply to tell a JAVA Developer, that the data resides on the mainframe or an iSeries platform. If the data storage is a Database and the company provides all necessary access methods to this database, the task is more or less like a SQL implementation. But due to the nature of these information, most companies do not allow direct access to the databases behind CICS and/or IMS Transactions. This is a valid approach because of all validation processes that takes place before a data is stored in the Database. This is the backbone for the company and the main reason why this transaction “still” resides on this platform. Besides the reliability and security, the experienced and highly proved program that sits behind the transaction is the reason for this approach.

If the JAVA programmer now needs to deal with Mainframe transactions and “screen scraping”, he has lost is major goal of business logic, completely out of his focus. Trying to bring a Banking-Transaction programmer into a Host-Transaction **and** Banking-Transaction programmer is, from our experiences, unproductive, not business oriented and finally also not reliable.

We have seen, in our projects, several programmers that started enthusiastic by accessing the mainframe and finally, after several months of developing, fail because the complexity of a mainframe transaction is more than a simple SQL Call. Screen-Scraping means that a program that was written to communicate with a Human is now handled programmatically. Questions like: “what should we do when a message

occurs that we do not expect?" are typical for these way of implementation. The complexity and amount of messages that might occur in a mainframe transaction are nearly unlimited, which makes it way too complex to handle all of them. This approach on the other side ends in most cases in an unreliable or instable system that finally is then decided to never get into production.

A company like **SAGA D.C.** is every day confronted with this and other issues of mainframe integration. So, we've learned how to use different approaches to make the application finally more reliable and therefore ready to go into production. Our track record of projects and customers shows that we learned our lesson.

With this experience and our knowledge of the current and near future technology, we started with a project called "H2O" (Host-to-Other).

Pretty soon we realized that we were too focused on the mainframe. And we extended the architecture from Mainframe Implementation into the Data-Integration. If we can provide the information that resides on a mainframe in the same structure and way as other sources, we can provide a new way of independency to the Information platform.

Technologies like Web-Services are focused on a similar approach. However, these services need to get information from somewhere. Using **SAGA.M31 – Galaxy** – this new technologies are able to provide the information **from** any kind of data storage. So we start to provide a new back-end implementing an easy to use front-end.

## 2.2. Cost saving aspects

A company needs to implement their data into several other applications. This might be B2B, B2C or internal processes that makes it necessary to access data. The past 50 years of Data-Processing (today called Information Technology) has created a huge demand on cost saving aspects. This might not always be a good approach for the labors but at least the competition requested to save money. For this reason, more and more processes have been brought to the Computer Systems that makes it more simple (or at least should make it) to retrieve specific information. In the past this was a task of a Batch-Job at night that finally generated a ton or paper that somebody has to read. Today, in the time of real time processing, we expect that our system is providing the information immediately (on Demand) and accurate. This has built the new industry that is called the IT-Industry. It's not only that every processor needs to be faster than it's predecessor by at least the factor of two but also requested more and more developer to get control of the zillions of terabyte of data that resides in the IT-Center.

Let's assume that a process has implemented a Store, a Stock and the production. This is typical for today's information integration. All these application needs to exchange information. Assuming that all application are from one manufacturer, it is supposed that this information flow works seamless. That is one of the reason, from our point of view, that nobody really has looked into a data simplification yet and that the major provider of such applications (SAP as example) are so often installed even if the user are not really satisfied with the functionality.

This approach is far away from the old "Best of Breed" approach that took place in the mid 70's until the end 80's. At this time, the organization was forced to use the best product in the market for a particular task. The complexity of integration and information flow has played into the hands of the "Multi-Application Vendor" even if everybody knows that experts on one specific area are not automatically experts on

another area. You would not drive an SUV on a German Motorway with a speed of 250 KM/h (well there are some of them but they have some weak points too). Also you would not use your Sports car to drive into the Mud. Another way to look at this is the plans in the 80's that somebody has written an Spreadsheet application for the mainframe or people thought that based on the Memory and Disk sizes, every Mainframe application could be easily handled on a PC. The "Best of breed" approach is very valid for Operating Systems, Hardware Platforms and also Programming Languages. But it seems that this is not true for applications. Most CIO's today would not take the risk of making a decision for different vendors for applications that finally have to "play" together.

Another issue in the same context is the complexity of data integration in applications. There are several products on the market that deals with a Knowledge-Base for Application Data. These products cost hundred-thousand of Euros (Dollars) just to provide the information which data is used in what application. Based on this Knowledge, a decision to change a field in one application makes it transparent to the developers what relations exist and how much this finally costs. This information finally stops in many situations a change for the better product because of the complexity and costs that are associated with this. If this information is now also provided to Partners (B2C) than it's nearly impossible to make a decision because every partner now needs to change his application because of the planned change. This is stopping projects even if it were necessary or beneficial for the company to change. So the decision to live with the Beast is made and new technology or more sophisticated applications will never be used.

If it would be possible to implement a single request to obtain information and to get this information consistently and independently from the underlying technology and/or platform, the cost saving would be huge and would open the door to come back to the "Best of Breed" approach that had so many benefits over the current approach.

### **2.3. Demand of the market**

Organizations today are facing the requirements that they need to react quickly on demands that their market generates. For this reason several companies have implemented middleware solutions that "download" information from the mainframe and continuously update this information. Besides the fact that this is a cost factor that would have also be noted in the previous paragraph, this is also not a good approach for actuality of data. Looking back to the days when a Stock-Card has been taken when somebody took a part out of the stock that was than read in during a nightly batch-job, this seems to be the same approach. Every night the actual information are stored or updated between the two platforms. The results are Articles that are not available when ordered (unsatisfied customer) or loosed customers because of a price-attack from the competitor where the update process took too long. This is still reality in today's "Information Technology" and used often if there is a media break between the platforms where the information are stored and processed.

The employee that updates data on the mainframe using the CICS Transaction modifies the data in real time. A purchase order from the Web is "normally" booked

into a Order-Book and then processed separately. If it is a modern implementation this is done via a Message Queue Mechanism. But this approach is still asynchronously. If an order in this queue is taking the last piece from stock, the customer has to be informed, that a specific article could not be delivered because it went out of stock. Even more worse is the opposite scenario where a customer would like to purchase something via the Online-Store. Unfortunately this article is booked as "Out-of-Stock" because some minutes before somebody ordered several hundreds of this article. The customer will not place the order because he wants to get all his articles from one supplier. During the order processing it is now found that the order that took all articles from the stock is not valid. Either the credit card is expired, entered wrong or something else is indicating that this order should not be fulfilled. That means, there are several hundreds article back on stock but the customer who wanted to buy some is gone. For this reason most companies use the "optimistic" approach and leave at least the Minimum-Amount on the stock to keep the customers on their store. Leaving the Virtual-Store and entering a real store would not show this problem. The reason is pretty simple, if somebody wants to buy something he has to pay immediately and not asynchronously. Therefore the stock either exist or not. Having a customer in a store is extremely important and to satisfy his expectations too. If he orders an article he is expecting that this article is on stock and can be delivered. Otherwise he is changing the store. An undelivered article is creating unsatisfied customers and should never occur.

The way to solve these issues is a more real time oriented architecture. Even if everybody is talking about SOA (Service Oriented Architecture) we are talking about RTA (Real Time Architecture) that updates data via the standard transactions on the real systems without any delay. This could only be done if the data integration is simple, reliable, secure and consistent.

**SAGA.M31 – Galaxy** – does exactly what is needed. It provides the real time architecture with simple to use data integration.

## 3. The factory process

### 3.1. How does that all work?

Before going into the details how all this works, it might be useful to shift the view away from the current Information Technology towards a more know Factoring process.

As mentioned in the first paragraph, we look on the information technology as a factory. This item should clarify some words we use in our technology to make it easier to understand.

A production process of an article (let's assume a car – Audi A6) consists of thousands of different parts that are assembled into a new product – the car.

A customer wants to purchase a new car and make a decision to buy a new Audi A6. So he puts together his wish list (and is surprised by the price after putting in all the Extras he likes ...). At the end, he orders a car with some features. The customer knows that he can expect an Audi A6 with several features. He knows the color, the exterior and interior, the power of the engine and some more “key features”. What he don't know and where he normally has no right to decide anything is what manufacturer of the tires he will receive (he knows the size – that's it), who made the Battery (he knows the Amps), who build the seats (he knows Leather seats) and so on. So it is a mixture between detailed information and some Order numbers.

This is the **Request** to the Factory: “Build and deliver an Audi A6 with the following features....”

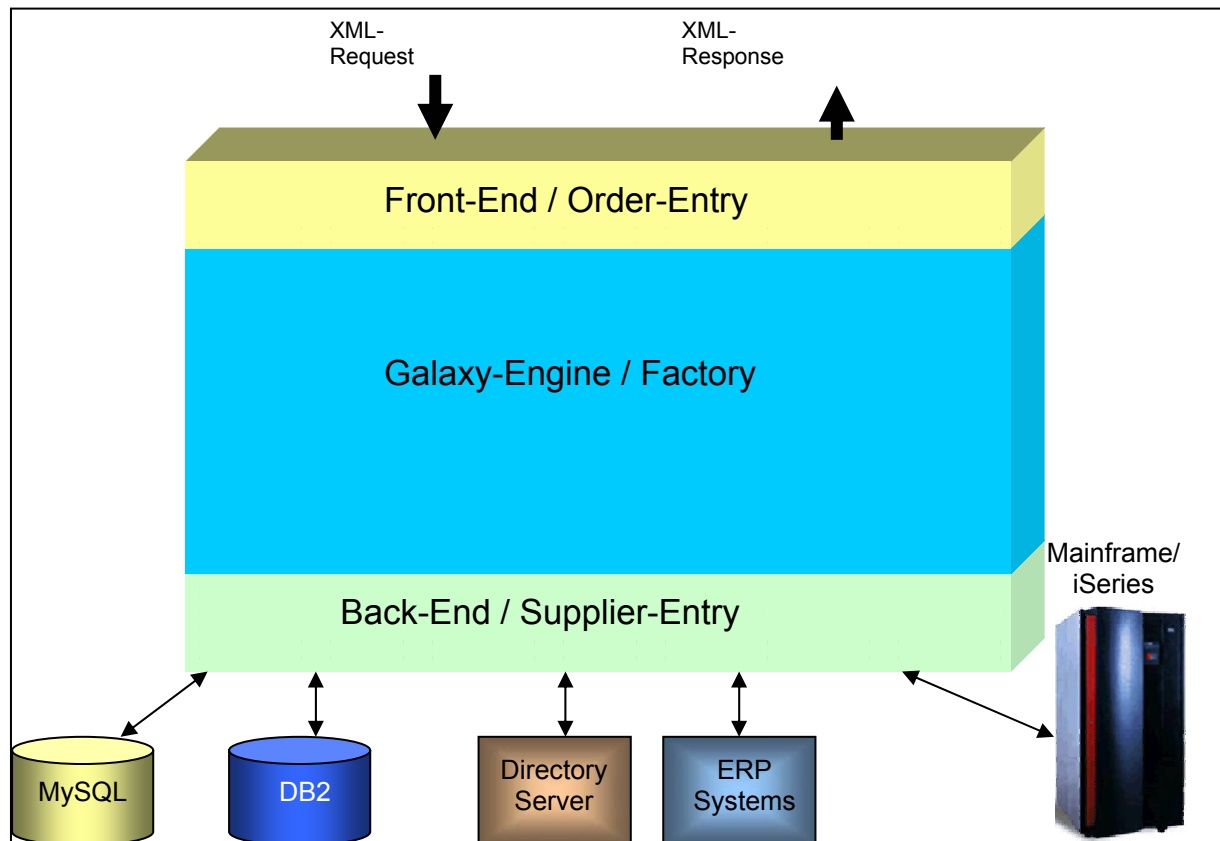
Inside the factory, the manufacturer (Audi) now decides what parts he needs to build that car. Based on contracts with the supplier, a Part-List (Pick-List) is generated. This Pick-List contains internal numbers that represents the different parts used to build that car. These numbers (the production part numbers) are bound to several supplier Part numbers, but this is not part of the Pick-List. This relation is built during the order process to have enough parts available to build the cars ordered.

Depending on the contingent that exists with the supplier of some parts – let's assume the tires – the Manager for the stock is checking if it is necessary to order new tires. If this is true, he has to decide what supplier he has to choose based on all contracts for this specific type. Assuming he decides to order Pirelli Tires, he has to order the specific Pirelli Order number. This is because Pirelli might know that Audi use a different numbering scheme but Pirelli has its own Part Numbers. This Number on the other side is completely different to Michelin's Part Number. Anyway, at the end the specification of the tires is responsible for the order.

During the assembly, a person, who is responsible to mount the tires to the wheel, take 5 tires out of a “**Container**”. He doesn't care what manufacturer delivered this tire. This Container is a “Supplier-Container”.

After assembling all parts, a new car is delivered to the customer. In most cases, the car is stored on a truck but under some circumstances, the car is put into a container, the “**Product Container**” and shipped to the customer.

Based on this **view** here the overview of the basic structure of **SAGA.M31 – Galaxy**



Now let's compare the wordings we used before and this picture:

**Order-Entry** – the place where an order enters the production process. **SAGA.M31 – Galaxy** – has a Front-End interface for this. This is nothing new but we need an entry to that system

**Factory** – the place where parts are assembled to a new product. In our terms, we use the term **Engine**. Perhaps we have to re-think about this term because it does a little bit more than simply provide enough power.

**Supplier-Entry** – the place where all suppliers deliver their parts. This is where we receive the different information and register them as our internal production part numbers.

### 3.2. A more detailed view

After all this theoretical words, it's getting time to go into the details – at least a little bit more.

Based on a sample Mainframe Application, here are the steps that are necessary to implement the data into the delivery process.

## SAGA.M31 Galaxy

You are logged in as admin [ [Control Panel](#) | [Logout](#) ]

Menu	:: Main Page
<p><a href="#">Main Page</a> <a href="#">About Galaxy</a></p> <p><b>Connectors:</b> <a href="#">Host Connector</a> <a href="#">SQL Connector</a> <a href="#">LDAP Connector</a></p> <p><b>Container:</b> <a href="#">Create Containers</a> <a href="#">Manage Containers</a></p> <p><b>User Management:</b> <a href="#">Create User</a> <a href="#">Edit/Delete User</a></p>	<p>Welcome to SAGA D.C.'s Galaxy Installation. Galaxy is a virtual Factory that delivers data from configureable sources.</p> <p>This is the main Administrative Console. You can <a href="#">Logout</a> here .</p> <p>To customize, view or build a Container, You can either</p> <ul style="list-style-type: none"> <li><input type="radio"/> view the <a href="#">Container list</a> and edit an existing Container</li> <li><input type="radio"/> or create a new Container</li> </ul> <p>You can choose one of the following connectors and the appropriate option:</p> <p>Using the Host Connector, You are able to:</p> <ul style="list-style-type: none"> <li><input type="radio"/> Record actions using the <a href="#">Actionrecorder</a></li> <li><input type="radio"/> customize recorded actions with the <a href="#">Customizer</a></li> <li><input type="radio"/> control the status of LU on the <a href="#">LU Stauspage</a></li> </ul> <p>For the LDAP Connector, You can:</p> <ul style="list-style-type: none"> <li><input type="radio"/> <a href="#">View all configured Idap requests</a></li> <li><input type="radio"/> or create a new LDAP request</li> </ul> <p>If You want to use the SQL Connector, Your options are to:</p> <ul style="list-style-type: none"> <li><input type="radio"/> <a href="#">View all SQL requests</a></li> <li><input type="radio"/> or create a new SQL request</li> </ul> <p>User Management</p> <ul style="list-style-type: none"> <li><input type="radio"/> You can edit or view existing users by <a href="#">viewing the User list</a></li> <li><input type="radio"/> or you can add new users</li> </ul>

© SAGA D.C. GmbH 2003-2004

### 3.2.1. Record an action

Knowing what a mainframe transaction does is one of the more complex tasks. In most cases, today's developers are sometimes not able to understand the transaction flow process, the shortest path to a certain transaction step and how to handle a Mainframe Screen in the most efficient manner. Therefore it makes sense to have a typical user to walk through the necessary steps to obtain information. To achieve this, we need to record all actions and screens that the user get presented. **SAGA.M31 – Galaxy** – provides a TN3270 Proxy.

To start this proxy, an administrative user is necessary on the Galaxy Application. After selecting the necessary action (start action recording), a panel provides information that are necessary to connect to the proper TN3270 Server. Also a local port (Local on the Galaxy-Server platform) can be defined that will start to listen. After pressing the "Start" Button, the Proxy-Server is ready to receive a Connection.

Now you need to start a 3270 Session with your favorite Emulator but change the Host where you want to connect to point to the Galaxy Server and the port you defined during the Proxy-Configuration. Here is a screen shot for this moment:

## :: Trace Recorder

### Trace rec

Refresh

Status: running

Controls	
Host (3270)	192.168.100.32
Port (3270)	2023
Serverport	2023
Connectionstatus:	connected

Stop Server

Recording status: **deactive**

Recorded action name:

Start Recording

### Usage

As you can see, the target host for the TN3270 Session is the IP-Address "192.168.100.33" and the Port is "2023". (This is an OpenConnect Gateway in our case).

The Galaxy-TN3270 Proxy listens on the Port 2023 (could be any other port as long as it is higher than 1024).

The current status is connected but inactive. That means, the Client is already connected but the Proxy is not recording any traffic yet.

Now a Name for the recorded action should be provided. This name is used later to identify the recorded action.

Here a screenshot for the situation after an action name has been entered and the "Start-Recording" button has been clicked:

## :: Trace Recorder

### Trace re

Refresh

Status: running

Controls	
Host (3270)	192.168.100.32
Port (3270)	2023
Serverport	2023
Connectionstatus:	connected

Stop Server

Recording status: **active**

Recorded action name: Test\_2004\_11\_08\_JG

Stop Recording

### Usag

This is the action recorder that lets you record ac

Now, every screen is captured, including the data that was entered during the session. If you, or the user who has the knowledge how the mainframe transaction works has ended the transaction steps, click on the "Stop Recording" button to end the Proxy-Session.

After the Recording is finished, all necessary steps has been stored in a database.

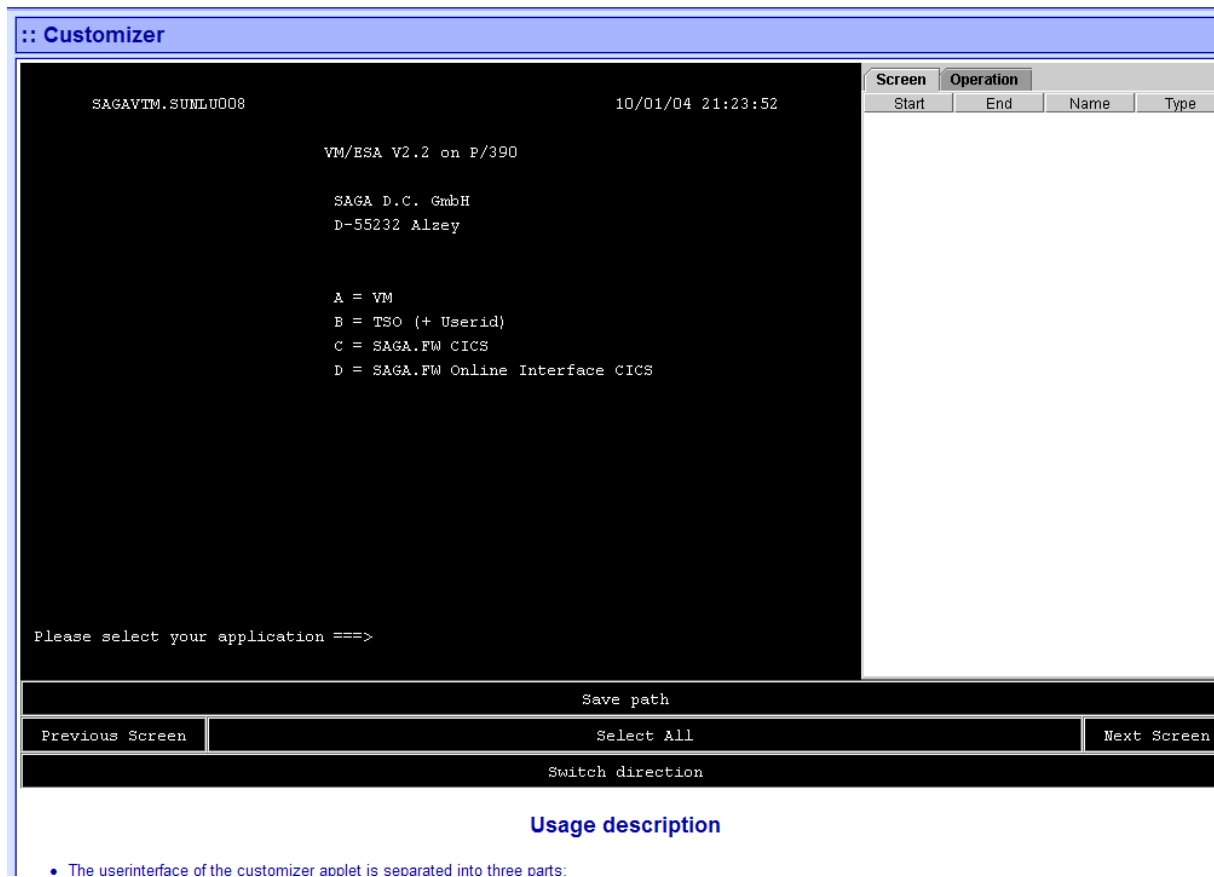
#### 3.2.2. Customize the recorded action

Now the most complex work has to start. To enter the customization process, select the link called "Customize Actions" on the Main-Panel of Galaxy.

On the next page, you will see an overview of all recorded actions where you should select the one that you desired to customize.

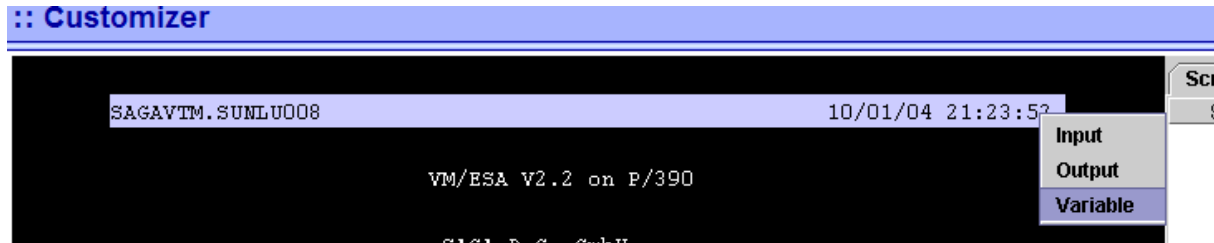
An Java Applet which will load the Recorded Action and present you with the Mainframe screens that were recorded.

A Screenshot:



Now the important steps are to identify all Variable fields on a screen. That means all fields that are not modified frequently. This is a little different approach than other Implementations are using today. The experiences from the past showed us, that Screen Identification is critical and mostly depended on fix locations. The problem starts when these identifiers change or are in a relation to other information on the screen. Therefore we turned the logic to get a better Screen Identification logic. We found that in nearly all cases, our logic fits better than the old one. You can identify the variable portions on the screen by selecting the desired area in the Applet window.

This is, how the selection looks like and what will be shown if you click on the **right** button of your mouse:



Now simply click on **Variable** and the information is stored in the Panel on the right-hand side of your Customizing Applet.

If you click on the **Operation** Tab on the right panel, you can see the data that were entered during the recording. If you need to put variables in fields on the screen, delete the appropriate information and create an **Input** Field for it. This can be

achieved by selecting the field where the data was entered, Right-Click and select **Input**. Provide a unique Name for this field in the right hand Panel by “Double-Clicking” into the Field-row of the table.

The same is valid if you want to create an **Output** Field. Select the Area that you want to provide into your Delivery process and Right-Click “Output”. Now provide a Name for this Action.

Walk through all Screens until the Data Capturing comes to the point that the user who recorded the session started the **Return Process**. This is typically the way out of the Transaction. At the screen where the user started the “Return Process” click on the **Switch Direction** Button on the lower part of the Customization Applet Window. From this point on, the user (in most cases) simply pressed an AID-Key like PF3, PF12, PF4 or something similar. Every Screen can now be configured as “Full Variable” by selecting the whole Screen and save this as a variable portion. The reason is just simply because the **Way-out** is pretty easy to achieve.

When you’re ready with the Customization process click on the “Save Path” - should be named “Save Action”- Button. After this, you are presented with the Dialog that relates these Information-Fields to the specific Product-Fields. That is the Process where the “Supplier” uses a different Part number than the internal Production Process.

Now you’re mostly ready with the Information providing.

### 3.2.3. Build an Information Container

After you put together all the necessary information from the different Resources, you are ready to build Information Containers. These Containers are the Products that your Service provides. In the Terminology of the Factory Sample above, this would be the car that is being shipped to your customer.

Based upon User Rights a user can request Data. What Information Container you provide depends on your specific needs and implementations. This could be an RSS/RDF Feed or a Portal Implementation with Data from a Timesheet up to different Application that needs Data for further Processing. There is no real limitation in minds. The only logical requirement that exist is that this Container should not change frequently. The Data should be consistent. **SAGA.M31 – Galaxy –** does provide the Functionality. It depends on the Administrator to make it a reality.

To create a Container click on the “**Create Container**”-link on the Main-Page. Here is the output of that request:

**:: Create Container**

Step 1Step 2Step 3

Container Name and Description

Summary	CPU Usage of one VSE Test System
Handle	<input type="text"/>
Description	<input type="text" value="A demonstration for Tables"/>

First you need to define a Summary (shown inside the Container-Browser), a Handle – that is a unique name under that this container can be accessed, else it gets a unique Container-Id, and a short description what this container does.

The next step is to select the Source-Container and add the fields to the newly created container:

## :: Create Container

Step 1 Step 2 Step 3

### Specify Fields for Container

**Container** CPU Usage of one VSE Test System

Fields in connector: SAGA.M31 Hostconnector for SAGA-System

id	name	Connector Name	type	add
50888716	CPU_Elapsed	SAGA.M31 Hostconnector for SAGA-System	string	Add
50888715	CPU_IOS	SAGA.M31 Hostconnector for SAGA-System	string	Add
50888717	CPU_Job	SAGA.M31 Hostconnector for SAGA-System	string	Add
50888718	CPU_Part	SAGA.M31 Hostconnector for SAGA-System	string	Add
50888714	CPU_Percent	SAGA.M31 Hostconnector for SAGA-System	string	Add
50888719	CPU_Prog	SAGA.M31 Hostconnector for SAGA-System	string	Add
10	CPU_Table	CPU Usage of our Mainframe (VSE Test)	table	Add
50888713	CPU_Time_Sec	SAGA.M31 Hostconnector for SAGA-System	string	Add
50888710	Lieferanten-Kurz	SAGA.M31 Hostconnector for SAGA-System	string	Add
50888706	Lieferanten-Name1	SAGA.M31 Hostconnector for SAGA-System	string	Add

### Container Output Fields

Field Name	Name in Connector	Connector Name	Type	Edit	Delete
CPU_Table	CPU_Table	SAGA.M31 Hostconnector for SAGA-System	table	edit	delete

and Resolve Dependencies

### Container Input Fields

Field Name	Name in Connector	Connector Name	Type
------------	-------------------	----------------	------

The Name in the Container-Output-Field Area is the unique name that will stay for the lifetime of this container. The name is independent to the source-container / Connector field.

The Input fields are automatically generated if you click the "Save Names and Resolve Dependencies" Button.

The next step is to finalize the information for the Container:

## :: Create Container

Step 1 Step 2 Step 3

### Save Container

Container	CPU Usage of one VSE Test System
Handle	<i>No handle Specified</i>
Description	A demonstration for Tables

**Container Output Fields**

Field Name	Type	Name in Connector	Connector Name
CPU_Table	table	CPU_Table	SAGA.M31 Hostconnector for SAGA-System

**Input Fields for container**  
*No Input Fields*

[Create Container](#)

If everything is as expected, the “Create Container” Link generate or update an existing container.

The next step is a Sample-Run. This means a test with the created container to see if all information is correct and if the structure of data is as expected.

The Sample Run can be started from the “Manage Container” Link on the Main Page.

## :: Container List

Container Name	Run Container	Edit Container	Delete Container
CPU Usage of one VSE Test System	Sample Run	Edit	Delete
SQL-Lieferanten	Sample Run	Edit	Delete
Sample Supplier Revenue and Address - Please use 1 to 10 as Supplier Number	Sample Run	Edit	Delete

Here the output after Clicking on the “**Sample Run**” Button:

## :: Galaxy Sample Request

### About to run a sample against: Sample Supplier Revenue and Address - Please use 1 to 10 as Supplier Number

Required Input to do a sample run:

Lieferanten-Nr

Show XML Request/Response

Show the Trace

[Back to Conatiner List](#)

You can (as we did) select the "Show XML Request/Response" and "Show Trace" Checkboxes. This will provide you with all information to analyze the results. After the container run successful, the result page is displayed:

The result shows the Information you requested, the XML Files (if requested) and the Trace (in our case from a Mainframe Session). Here the results:

### Run results:

#### Field data

Data Name	Data Value
Lieferanten-Name1	SPEDITION MUELLER_____
Lieferanten-Kurz	MUELLER_____
Lieferanten-PLZ	55232
Lieferanten-Ort	ALZEY_____
Lieferanten-Name2	_____
Lieferanten-Umsatz-lfd-Jahr	506587,14
Lieferanten-Strasse	MAINZER STRASSE 5__

Next are the XML Request / Response information:

## XML Request:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ContainerRunRequest>
  <container id="50954241">Sample Supplier Revenue and Address - Please use 1 to 10 as Supplier Number</container>
  <username>admin</username>
  <password>MD5 Hash of you password</password>
  <data type="field" varType="string" id="51150850" dataItemName="Lieferanten-Nr">
    <value>2</value>
  </data>
</ContainerRunRequest>
```

## XML Response:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ContainerRunResponse>
  <returnCode>0</returnCode>
  <containerRunMessage>Container was run. See results for more details</containerRunMessage>
  <data type="field" varType="string" id="51150854" dataItemName="Lieferanten-Name1">
    <value>SPEDITION MUELLER_____</value>
  </data>
  <data type="field" varType="string" id="51150857" dataItemName="Lieferanten-Kurz">
    <value>MUELLER_____</value>
  </data>
  <data type="field" varType="string" id="51150851" dataItemName="Lieferanten-PLZ">
    <value>55232</value>
  </data>
  <data type="field" varType="string" id="51150855" dataItemName="Lieferanten-Ort">
    <value>ALZEY_____</value>
  </data>
  <data type="field" varType="string" id="51150856" dataItemName="Lieferanten-Name2">
    <value>_____</value>
  </data>
  <data type="field" varType="string" id="51150853" dataItemName="Lieferanten-Umsatz-lfd-Jahr">
    <value>506587,14 </value>
  </data>
  <data type="field" varType="string" id="51150852" dataItemName="Lieferanten-Strasse">
    <value>MAINZER STRASSE 5__</value>
  </data>
</ContainerRunResponse>
```

The XML Request is the only structure that needs to be send to the **SAGA.M31 – Galaxy – Server**. The Response is returned to the requesting Application. The different information is specifically documented but easy to understand.

The XML-Request should contain the userid and the MD5 encoded Password that the user uses on the Galaxy Platform.

Finally the result of the Trace-Output from the Host:

## Trace:

Write to position 1878:c  
Send Aid-Key: Enter  
Screen:

```
-----  
| IESADMS01                      SAGA-IBIS EURO  
| 5686-066 and Other Materials (C) Copyright IBM Corp. 1995 and other dates  
|  
| VV  VV  SSSSS  EEEEEEE  ++  
| VV  VV  SSSSSSS EEEEEEE  ++  
| VV  VV  SS     EE        ++  EEEEEEE  SSSSS  AA  
| VV  VV  SSSSSS EEEEEEE  ++  EEEEEEE  SSSSSSS AAAA  
| VV  VV  SSSSSS EEEEEEE  ++  EE        SS     AA  AA  
|  VV  VV      SS  EE        ++  EEEEEEE  SSSSSS AA  AA  
|  VVVV  SSSSSSS EEEEEEE  ++  EEEEEEE  SSSSSS AAAAAAAA  
|  VV    SSSSS  EEEEEEE  ++  EE        SS     AAAAAAAA  
|                                     ++  EEEEEEE  SSSSSSS AA  AA  
|                                     ++  EEEEEEE  SSSSS  AA  AA  
|  
| Your terminal is A006 and its name in the network is SUNLU028  
| Today is 08/11/2004 To sign on to SDC3CIC2 -- enter your:  
|  
| USER-ID..... _____ The name by which the system knows you.  
| PASSWORD..... Your personal access code.  
| LOGON HERE..... 2 Enter 1 for YES, 2 for NO  
| PF1=HELP 2=TUTORIAL 3=TO VM 4=REMOTE APPLICATIONS 6=ESCAPE (U)  
| 9=Escape (m) 10=NEW PASSWORD 12=LANG SEL  
|  
-----
```

Write to position 1464:\*\*\*\*\*  
Write to position 1384:sagal  
Send Aid-Key: Enter  
Screen:

```
-----  
| IBIS          EPOS Release: 2.01          A006          08.11.04 18:56:45|  
|  
|           W i l l k o m m e n  
|                               i n  D O S L I B.  
|  
|           EEEEEEEEEE      PPPPPPPPP      OOOOOOOO      SSSSSSSSS  
|           ~~~~~          ~~~~~          ~~~~~          ~~~~~  
|  
-----
```

As you can see, all actions are logged (Password or Hidden Fields are not shown).

If the result is what you expected, continue to “deploy” the Container. If not, take the chance and visit us at <http://www.sagadc.org> and look in the Forums to see if somebody can help you or already fixed the problem.

### 3.2.4. Grant permission for the Container to Users

After finishing the Container development (the production is finished) you should provide the users with Access to this container. This can be done via the “View User” Link on the Main Page.

Here is a sample output of the result:

:: User List						
Username	Last Name	First Name	Role	Permissions	Edit User	Delete User
admin	admin	galaxy	Administrator	Permissions	Edit User	Delete
guest	Guestuser	Galaxy	User	Permissions	Edit User	Delete

Clicking on the “Permission” Link will lead you into the Container-Granting process. You will receive a similar page:

:: Change Permissions	
<b>Change Permissions for user: guest</b>	
Container Name	allow to run?
CPU Usage of one VSE Test System	Allow? <input checked="" type="checkbox"/>
SQL-Lieferanten	Allow? <input type="checkbox"/>
Sample Supplier Revenue and Address - Please use 1 to 10 as Supplier Number	Allow? <input checked="" type="checkbox"/>
<input type="button" value="Save Changes"/> or <a href="#">go back to list of all users</a>	

Provide the User Access to the Container by selecting the appropriate Container.

Now the Container can be accessed by the user.

That's all that is necessary to simplify the Information process.

### 3.2.5. LU Status Page

Because it is always helpful in a quiet Mainframe or Midrange Environment to know what status an LU has, here the information that is provided by the LU-Status Page:

:: LU Status			
LulIndex	Status	View	Control
0	Waiting	Current screen	Stop
1	Disconnected	Current screen	Start
2	Disconnected	Current screen	Start
3	Disconnected	Current screen	Start
4	Disconnected	Current screen	Start

Clicking on the "View Current screen" will show the current information on the LU:

**:: Show Current Screen**

[Refresh](#) [Back to LU Staus](#)

Screen:

```

/-----
|
|          SAGAVTM.SUNLU028                               11/08/04 18:32:45
|
|                                VM/ESA V2.2 on P/390
|
|                                SAGA D.C. GmbH
|                                D-55232 Alzey
|
|                                A = VM
|                                B = TSO (+ Userid)
|                                C = SAGA.FW CICS
|                                D = SAGA.FW Online Interface CICS
|
|-----
|
| Please select your application ==>
|-----

```

### **3.2.6. Future Steps**

As the releases of more and more features of Galaxy will roll out, it is fundamental that the basic coverage of our Data Factory exist in a simple structure already. It's not necessary to wait longer.

See the next pages why....

## 4. The roadmap of SAGA.M31 –Galaxy–

### 4.1. Two different Versions

**SAGA.M31 – Galaxy** – was planned from the very first day with two different versions and license Models.

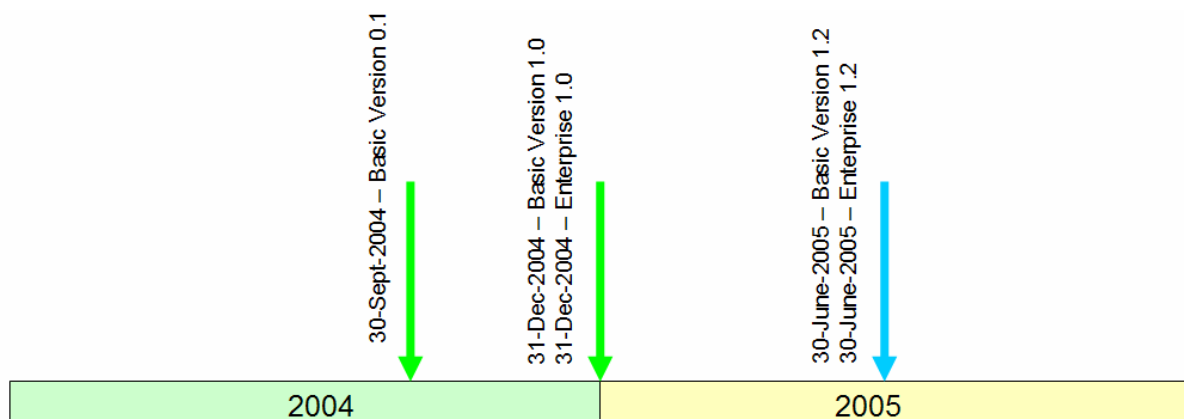
- Basic Version
- Enterprise Version

The main difference exists in the License Models. The **Basic Version** is provided as Freeware via our website <http://www.sagadc.org>. Here is the main place to start an implementation because every new Feature will be implemented in the Basic Version at first.

We decided to not provide **SAGA.M31 – Galaxy** – under an OpenSource Model but as Freeware because we feel that we are too small to compete against the Major Companies in this market. It is of course not any kind of high sophisticated Development; it is simply a change of minds and putting everything in question before finding a new solution. That's one of the reasons why we call it a Galaxy. Only 500 years ago, everybody thought that the Earth is a flat slice and that the sun turns around the Earth. Today we have a much wider view than this but still think in the old, always used paths.

What features are implemented in what version is shown on the next pages. The main difference is the implementation of more performance, auditing and Supplier in the Enterprise Version than in the Basic Version.

Looking on a time horizon, here are our plans with the upcoming release dates:



Like all companies, SAGA depends on Revenue and Profit. However we are committed to provide the releases by the above mentioned dates. In the Feature List, you can see what Features are planned for the different releases. It is mandatory for us that the technology is used and not built for technology only.

## 5. Features of SAGA.M31 – Galaxy –

Here a list of Features that are either already part of the Product or will be provided by the planned release dates as shown above.

	Feature	B	E	Planned Release
Front-End	User Interface HTML	✓	✓	0.1
	User Interface XML	✓	✓	0.1
	Programmatic JAVA Api	✓	✓	1.2 – June 2005 -
	Basic Authentication for Galaxy	✓	✓	0.1
Engine	Skywalker LU-Driver	✓	✓	0.1
	Elevator LU-Driver		✓	1.0 – Dec 2004 -
	Navigation Basic	✓	✓	0.1
	Navigation – with Auto-Route Navi -		✓	1.0 – Dec 2004 -
	Average Response time Providing		✓	1.0 – Dec 2004 -
	Cached Containers		✓	1.0 – Dec 2004 -
	RDF / RSS Automation		✓	1.0 – Dec 2004 -
	XML2Any API	✓	✓	1.0 – Dec 2004 -
	Reporting / Logging /Auditit		✓	1.0 – Dec 2004 -
	Identity Management API	✓	✓	1.0 – Dec 2004 -
	Input Data Management	✓	✓	1.0 – Dec 2004 -
Back-End	TN3270 Host Connector	✓	✓	0.1
	WebConnect Host Connector (JHLLAPI)		✓	1.2 – June 2005 -
	HACL Host Connector		✓	1.0 – Dec 2004 -
	TN5250 Midrange Connector	✓	✓	1.0 – Dec 2004 -
	MySQL Database Connector	✓	✓	0.1
	DB2 Database Connector	✓	✓	1.0 – Dec 2004 -
	LDAP Connector	✓	✓	0.1
	Netegrity Siteminder Connector		✓	1.2 – June 2005
	Tivoli Access Manager Connector		✓	1.2 – June 2005
	RSA CleaTrust Connector		✓	1.2 – June 2005
	SOAP / WebService Connector	✓	✓	1.2 – June 2005
	SAP Connector		✓	1.2 – June 2005
	Other ERP Connectors		✓	- after June 2005 or on Demand

The Colors define the part of the Factory where yellow is the Front-End, blue is the Engine and Green is the Back-End.

## 6. System Requirements

### 6.1. Basic Version

- MySQL Database
- IBM WebSphere Application Server Version 5+ **or**
- Jakarta Tomcat Application Server
- TN3270 Server
- Operating System for above mentioned Application Server

### 6.2. Enterprise Version

- MySQL Database **or** DB2 Database
- IBM WebSphere Application Server Version 5+
- TN3270 Server (Package covers a TN3270 Client)
- IBM Host Access Client Library (Host-on-Demand / PersCom 4.3+) **or**
- OC://WebConnect Pro Version 6.3+