MEISTER 2.0 SUITE

MEISTER V2.0 SUITE AND SDK

Andre Rosenthal
Gateway Architects, LLC
2600 Dallas Parkway suite # 600, Frisco, TX

Table of Contents

Product Document

Document Name:	Meister 2.0 SDK
Document ID:	MV2SDK0001
Document Owner:	Andre Rosenthal
Document Version:	1.0
Document Date:	Monday, 26 August 2019
Document Status:	Draft

Introduction

Meister Suite is a full solution designed to give SAP the true agnosticism needed to make back-end solutions and functions available through the Internet. Its revolutionary approach to a NoSQL vision of SAP transactions where the user is detached from knowing it is running an SAP transaction gives Meister the unique ability to perform a single call which returns all required data points with extreme performance.

Business Benefits

From a business standpoint, all SAP back-end line of businesses, hereinafter SAP LOB, are visible, extensible, streamlined, controlled, secured, and efficiently transferred in a non-intrusive fashion to any choice of UI devices. Not only Meister Suite reuses the existing customers' infrastructure, being UI5 under Fiori, Microsoft SharePoint, or just a web site, but is also maintains the integrity of the SAP system by allowing only what the user profile dictates, viz., no new rights are given beyond the capacity to transmit data through the Meister Suite engine. Authentication is provided either via X.509 or OAuthr 2.0, and authorizations are delegated to the backend giving the enterprise peace of mind since nothing that is not available today via SAP GUI would be executed via Meister Suite.

Agility

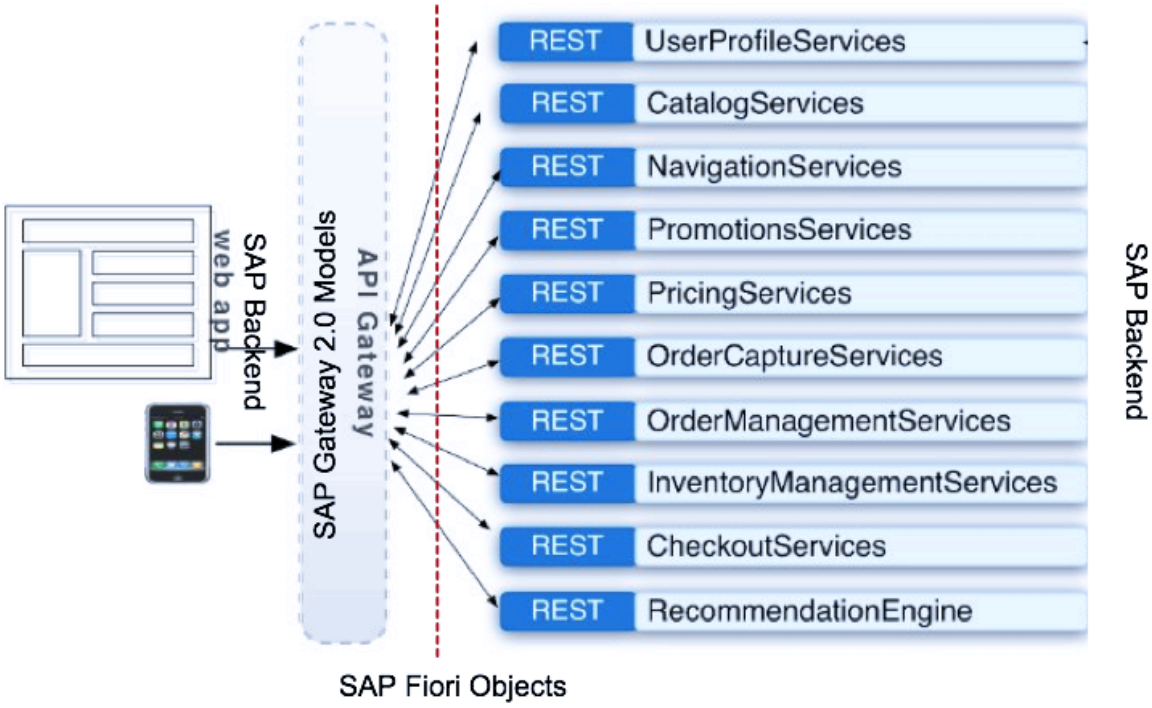
Since Meister Suite does not require a predefined model like Fiori, it promotes agility and efficiency to the enterprise as no coding is necessary to be crafted to support models at the gateway. Rather, Meister Suite proprietary JTA2 compiler generates models automatically during runtime thus allowing the enterprise to realize a much faster to-go-market approach. Additionally, Meister Suite is the only product for SAP that is capable of transmitting a complex payload with table of tables without limits of size and complexity. To achieve superb performance, Meister Suite proprietary byte compression reduces the size of the payload by over 80% on average and is backwards compatible with all compression engines on the market.

From Script to execution

Using Meister 2.0 yields an incredible flexibility where a solution architect could craft a script (or mini blueprint for a spring in Agile) and at the same time create the necessary Json documents via the SDK to produce a ready for consumption endpoint for the UX layer. The UX at this time could just scaffolds the endpoint into a seed test unit for UX requirements given that the field content should be immaterial for the UX execution, provided of course that the façade pattern is enforced as to detach the content from the semantics of its value, and focusing only on the syntactical nature of the field in question.

Traditional consumption of Rest Services at SAP

Traditional OData protocol-based APIs are tightly coupled with their implementation, and any modification done at either side of the tier cascade into a myriad of changes on all ends. The picture below clearly shows the main issues with this kind of API:



Each one of the services at the right of the picture is a single Fiori application and the navigation between these Rest Services implemented as a model at the API gateway code in SAP Gateway 2.0

Not only the call is chatty and inefficient, but also prone to maintenance issues and performance.

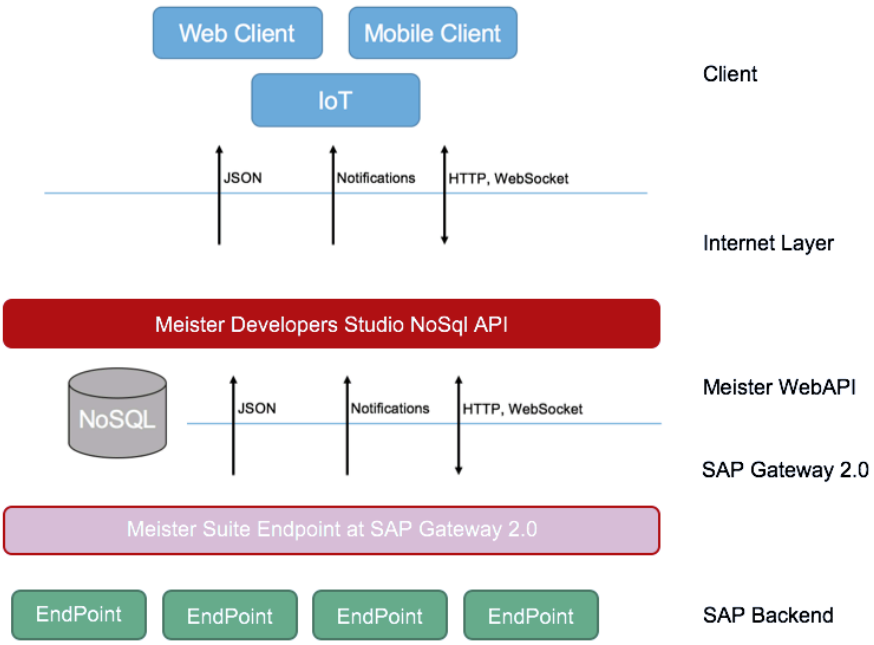
Meister Approach to Rest Services

API Layer of Meister – Microservices Architecture

To completely eliminate the issues found at the traditional approach, Meister introduces a modern view of transactions supported by NoSQL transient datastores and implemented with Microservices Architecture. The following properties are present at Meister Development Studio:

- Object Oriented Programming (OOP) – implemented at the SAP backend R3
- Web service API – the exposed methods of the Meister Suite as NoSQL API calls
- Meister Endpoint Manager – each Meister Endpoint is exposed as a corresponding NoSQL Json document.
- Single Responsibility Principle (SRP) focus – each Meister Endpoint executes a transaction on its own
- Interface Segregation Principle (ISP) focus – each Meister Endpoint segregates the whole process from end to end.

The picture below shows Meister's approach to Microservices Architecture:



Under Meister Development Studio, a client call is executed as a query on the NoSQL database with the respective json request document and json response document. Meister Development Studio traps the call and passes through the SAP Gateway Meister Service running Meister Suite, and the call gets executed via endpoints at the SAP Backend. The json response is then returned to Meister Development Studio and stored in the NoSQL transient database for execution of the client's query and remains at the NoSQL for the duration of the transaction. Not only the Dynamic Single Call was able to execute many endpoints at the same time aggregating the results as it goes, but there is one and only one call to the backend until the transaction is completed. All reads executed on the json document are done at the NoSQL database at the WebApi layer.

Meister Foundation and Core components are the methods of communication between SAP Gateway 2.0 and any SAP backend above ECC 6.0 SP5.The maintenance and repairs are strictly set on the /MEISTER/ namespace and falls outside the registration keyset for usability. That is, all code for Meister Framework is under the /MEISTER/ namespace and injected into the respective SAP stack via TOC (transport of copies), regular external transport, DVD media, or download equivalent. Basis then will import the TR just like any other regular TR which completes the installation of Meister. The SAP reserved namespace /MEISTER/ needs to add to the system via SE09 with modifiable flag. Once basis is ready to upload the TR Meister will add the repair license under Consumer mode at the given SAP stack and the import should continue.

There are two distinct methods to create LOBs using Meister, being **Outside-In** and **Inside-Out**, the former is driven by existing LOB code available at the SAP backend stack, such as BAPI and customer function modules, either RFC enabled or not. The latter is driven my Meister Development Studio, and extension to Visual Studio 2017, which allows the ad hoc definition of json payloads which are then created automatically at the SAP ABAP stack as object-oriented classes.

Regardless of the underlying call on the SAP backend stack, Meister will create a model automatically based on the syntax of the backend call and expose that model as a json package for the client consuming the Meister Service.

Json Documents

Meister Developers Studio operates on the notion of json documents, which are common structures leveraged by any UX platform on all devices. Given that the native languages (as well as traditional frameworks like .NET and Java) are capable of being extended via package imports to operate in NoSQL mode (such as Cassandra

libraries and MongoDB, for instance), the UX staff now needs only to connect to the NoSQL engine of choice and prepare json documents for request and response. Once that is done, and published at the database layer, Meister Development Studios' API is executed and, depending on the mode of operation, development versus execution, creates the necessary support structures on the SAP backend, or execute the call via Meister, respectively.

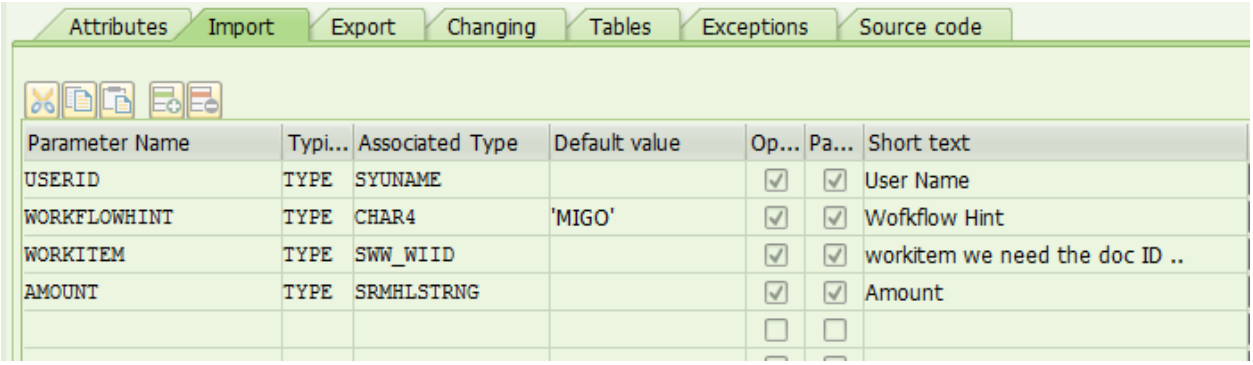
Meister Wrappers – the inside-out model

Regardless of the underlying call on the SAP backend stack, Meister will create a model automatically based on the syntax of the backend call and expose that model as a json package for the client consuming the Meister Service.

Since Meister normalizes the call using DDIC objects, discrete elements present in either Imports or Exports of function modules would be encapsulated in a customer structure and stored on the project as Dictionary Structures. From a consumption standpoint, Meister introduces a request and response Json envelopes to construct the Inside-Out model.

Meister uses the concept of DDIC-Normalization which replaces discrete Imports and Exports variables on Function Modules with DDIC Structures that encompass these importing and exporting variables. That is, Meister automatically normalizes the backend calls either by creating a denormalized version of the SAP backend call (Import, Export, Tables, Change, and Exception) which means Meister can run any of the existing backend calls regardless of these being flagged as remote enabled, or Meister can run specific normalized packages encapsulated as LOBs extensions. This normalization is executed by Request and Response Json documents, knowing that due to the dual role of tables and changes, Meister will duplicate these structures at the Request and Response envelopes. It is up to the implementer of the call to send the necessary data content for each of these DDIC structures.

As an example, an existing function module on the ECC backend may look like:



Which ends up being replicated as is as a json Request with 4 parameters. Should this be a normalized call, only one structure would be found at the import section as an inner structure of the Request envelope.

Meister Endpoints – the outside-in model

Meister promotes the creation of outside-in endpoints via the SDK website and allow the implementer the agility to design the Request and Response Json documents leaving the details of the implementation as a minimal task for the ABAP resources. Meister uses the endpoint technology to define how the payloads will be processed. Each discrete request and response payload have a unique endpoint, which itself could be split into several federated calls within the control of Meister, such as Google Maps, etc. The management of endpoints is driven by the Meister SDK website.

With the Inside-Out process, the Solutions Architect defines the Endpoint at the SAP ABAP stack, and publishes the endpoint to be used by the UX development team via the SDK website without any code being created at SAP. This is useful to take advantage of pre-existing SAP code and make it now visible to the UX layer using Meister as a Rest service.

With the Outside-In process, the Solutions Architect defines the Endpoint and payloads using the Meister Developers Kit (the SDK) and publishes the content as a new set of OOABAP classes. The whole process is driven by Meister itself, leaving the implementation uniqueness to the ABAP development team.

The SDK website provides the support for creation of endpoints with all the support for the OOABAP classes and Meister manages the life cycle of the call as part of its own process.

Error Handling

All Meister errors are permeated from the SAP backend to the UX. In the Inside-out process, the message table could be provided by the SAP OOABAP code as a node in the json response, and in the Outside-In process, the message table and exceptions are predefined by Meister SDK.

Traditional audit trail processes and extensive log is managed by Meister at the framework layer and retrieved by an endpoint driven by the SDK website.

Meister Roles and Profiles

Since Meister is installed on the SAP Backend and on the SAP Gateway 2.0, the authorizations needed to perform these activities are expected to be done by the SAP Basis team.

The roles needed for installing Meister at the security layer are provided at installation time, and consists of:

1) Basis_adm on client XXX of the respective SAP Stacks for the installer user (say AROSENTHAL.)
2) Existing roles and profiles set AROSENTHAL to be maintained or re-added.

Specific customer roles should be applied if possible, and only adding the required objects from the set of roles provided herein is advisable. Both Basis and Security teams should collaborate for this effort.

Meister works either on a service account, or a named user. Regardless of the user preference the required roles are presented herein. Assuming a service user called MEISTER_SRV, the user should exist on both backend and gateway systems and configured with the expected roles.

User MEISTER_SRV needs a few roles which are predefined by Meister Suite and provided by Gateway Architects.

Additionally, user MEISTER_SRV needs rights to session debugging on both Gateway and backend systems as all traffic flows though this user. Please check the latest roles with the corresponding ones on the gateway as they may have changed since the roles were designed.

Two documents were produced by Gateway Architects to instruct on how to correctly install Meister on a Hub environment as well as on an embedded mode.