USI 2.0 Components & Enhancements

USI 2.0

The Unified Storage Infrastructure (USI) is the common code base between all EMC OEM platforms. USI 2.0 has a single client API (REST) that facilitates the execution of numerous storage operations including creating, expanding, and removing storage, asset introspection, and more. USI 2.0 offers a fusion of several exciting components that have all been fine-tuned for an embedded/OEM install.

USI 2.0 provides scalability because it can operate in either a converged or distributed architecture. All of its components can run standalone or together. In addition, it is possible to leverage and balance multiple instances of each component.

USI 2.0 is also fault tolerant and highly available. It leverages guaranteed delivery and transaction recovery mechanisms to ensure that operations are not only delivered but also executed.

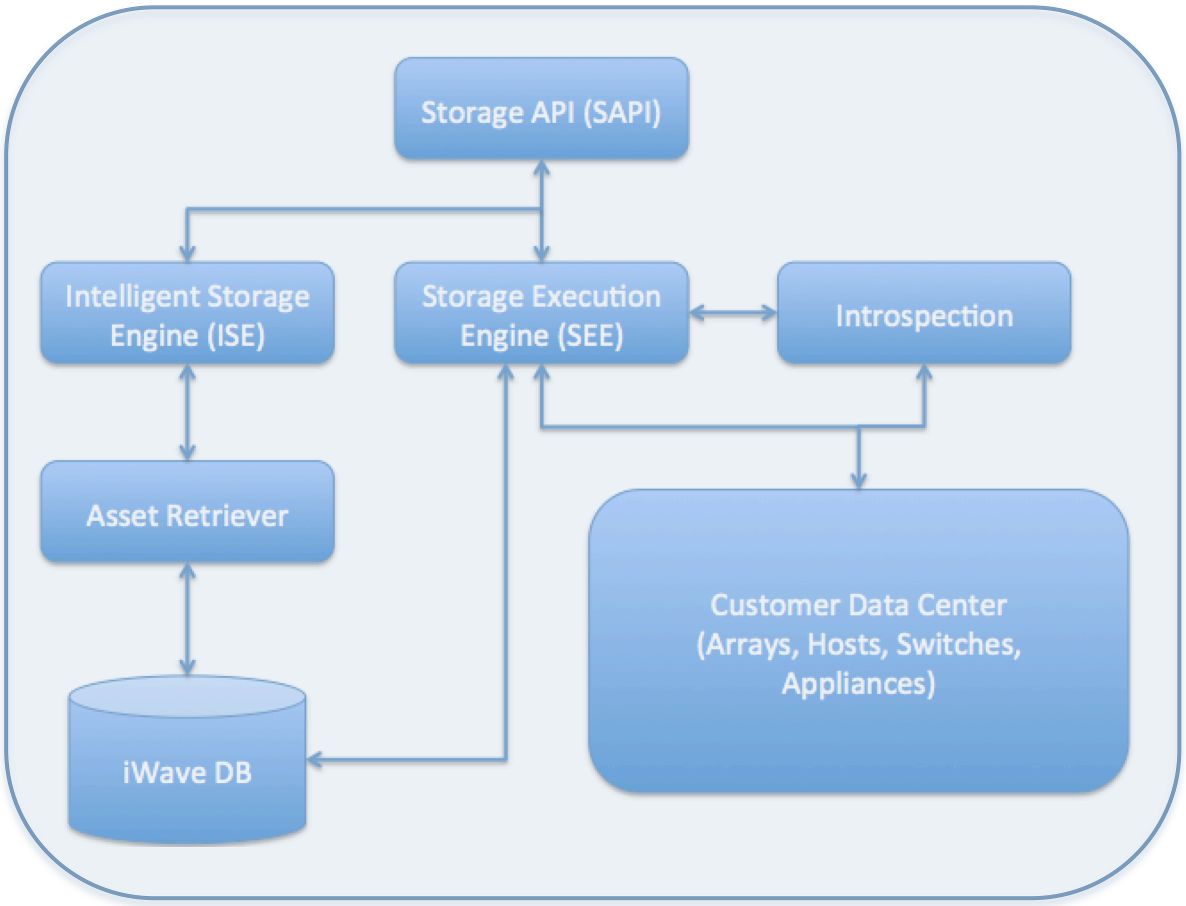
Storage API (SAPI)
Intelligent Storage Engine (ISE)
Storage Execution Engine (SEE)
Introspection
Asset Retriever
Customer Data Center (Arrays, Hosts, Switches, Appliances)
iWave DB

Figure 1 – Logical Component View

USI 2.0 Components

Storage API (SAPI)

The Storage API (SAPI) is USI's public entry point. It offers a RESTful web service interface as well as a legacy WSDL based interface, which is deprecated but still supported for legacy customers.

SAPI delivers several types of operations including:

- Storage Operations – Select assets for generic requests and execute requests.
- System Operations – Query the status of components, etc.
- Scheduling Operations – Allow Storage Operations to be scheduled.

Storage Operations

Selection

The Selections resource is used to direct the Intelligent Selection Engine (ISE) to determine what resources should be provisioned. Once ISE makes a decision, based on its internal rules and policies, the selection is stored and can be used later by the Storage Execution Engine to either execute the selection or test its execution.

POST /sapi/rest/selections		
Submits request to selection storage for provisioning. The request is submitted to ISE, which uses internal rules to determine the ideal storage option. The response is an identifier that can be used to execute the selection using the Execution API.		
Request Details		
Supported Types:	application/json, application/xml	
Fields:	Request	Name of request being submitted for selection
	Requester	Name of requester
	requestParameters	Collection of name value pairs used in performing the selection
Example:	see below	

```
POST /sapi/rest/selections HTTP/1.1
User-Agent: curl/7.21.4 (universal-apple-darwin11.0)
libcurl/7.21.4 OpenSSL/0.9.8r zlib/1.2.5
Host: localhost:8444
Accept:application/json
Content-Type:application/json
Content-Length: 706

{
  "request" : "VMAXeAddStorageNewServer",
  "requester" : "snicklin",
  "requestParameters" : [
    {
      "name" : "username",
      "value" : "user001"
    },
    {
      "name" : "deviceSize",
```

Intelligent Selection Engine (ISE)

As the name suggests, the Intelligent Selection Engine (ISE) adds intelligence to requests. It enriches generic requests by leveraging proprietary, patent-pending intelligence and industry best practice rules/policy enforcement. All rules and policies are modifiable and support multi-tenancy.

ISE is responsible for determining how to best fulfill storage requests. When ISE receives generic requests from SAPI, it uses internal rules and administrator-defined guidelines to make intelligent and informed decisions about what asset to use, where to place storage, where to create zones, etc. After making provisioning decisions, ISE enriches the generic request, turning it into the actual operation that gets passed to the Storage Execution Engine (SEE) for processing. Because ISE makes storage decisions on its own, it frees users from concerns about their system's underlying storage infrastructure.

Introspection Engine

The Introspection Engine collects and stores the information about assets that ISE uses to make storage provisioning decisions. The Introspection Engine permits administrators and/or end users to register assets (e.g. arrays, switches, etc.). Once an asset is registered, the Introspection Engine interrogates it to determine all of its attributes and internal objects and records that information in the Asset Database to be accessed by ISE (via the Asset Retriever). In addition to collecting and storing asset attribute information, ISE can also assemble performance metrics for each asset. The Introspection Engine can run according to a schedule or on an ad hoc basis.

Supported Assets	
Arrays:	VMAX, VNX, Compellent, NetApp
File Systems:	VNX/Celerra, NetApp, Compellent File
Switches:	Cisco MDS, Brocade
Hosts:	Linux, Virtual Hosts (VMware)

Table 4 – Supported Assets

Asset Retriever

The Asset Retriever is the interface that allows ISE to query the Asset Database. It retrieves data from an underlying data store and returns that data into the iWave storage object model. The Asset Retriever can be decoupled from iWave's database, adding flexibility and enabling use with third-party databases.

Storage Execution Engine (SEE)

The Storage Execution Engine (SEE) executes operations passed to it by ISE. Upon receiving an operation request, SEE loads the appropriate execution plan—a list of precondition checks, execution steps, and rollback semantics—then implements those steps to complete the operation. In the event of a failed or interrupted execution, SEE has rollback and resumption capabilities.

- In the OEM environment, SEE also contains the Introspection Engine code and powers that functionality.
- In environments where operations cannot be passed to SEE (event driven), an optional Polling Service can be used to query for operations to be executed.

USI 2.0 Enhancements

USI 2.0 boasts numerous enhancements that significantly advance its capabilities and its efficiency.

- A new filesystem (/usi) replaces all iWave branding (file system, OS user, etc.).
- Numerous bug fixes.
- Simplified development model—new functionality can now be built within hours rather than days.
- All management (e.g. starting, stopping, restart) is completed using a single control script instead of many scripts around the code base.

Storage API (SAPI)

iWave now provides and actively maintains a SAPI REST Client. Customers that want to integrate with us can now leverage our Client rather than having to write their own. While iWave highly recommends using our Client, customers can still write their own if they choose.

Other SAPI enhancements include:

- WSDL and REST interfaces (WSDL deprecated).
- Better logging and error handling.
- Request scheduling.
- System level operations.

Intelligent Storage Engine (ISE)

- Easier, more efficient Setup Helpers for interacting with ISE.
- Rules added for common items (naming conventions, formatting of variables, etc.), eliminating the need for hardcoding in JAVA classes.
- Better logging and error handling.

Storage Execution Engine (SEE)

- Better fault tolerance and failover support. SEE now resumes interrupted transactions upon restart.
- Health check, DB check, and uptime services are now exposed via REST.
- Introspection now available (VMAX, VNX, Compellent, NetApp, Cisco, and Brocade).
- New notification service allows for configurable calls to an endpoint (such as WS Coordinator in SP) for request status notification.
- In the event of a failed execution, results now contain an “Error Message” field.

- When SEE sends an Execution Result to the WS Coordinator, it now sends a REST request to the Poller to do another poll immediately.
- Improved Administrator console—better log viewing, configuration screen, etc.
- Enhanced Database utility scripts—initialize schema, backup schema, backup data, clean schema, etc.
- Internal enhancements.
- Better control of error logging—no stack traces.
- Simplified programming model, which can be exposed as an SDK to deliver new content if desired.
- Single configuration directory for SEE configuration files.
- Provision result now contains a list of updated, deleted, and created items (not just created items, as before), which allows for better asset tracking.
- Support for scheduling at the C level, not just at the Storage API level.