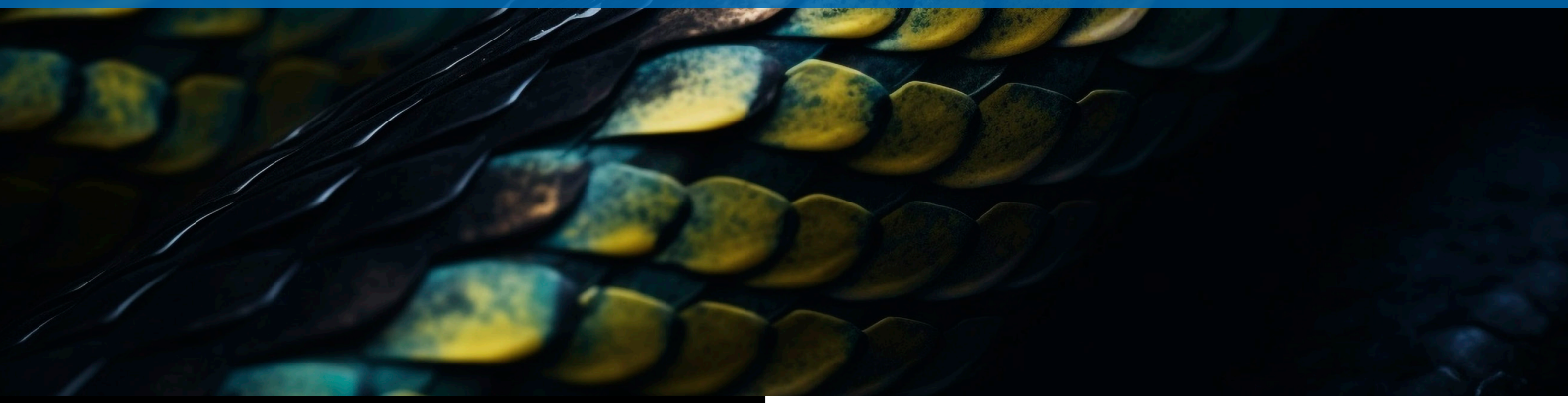
efficient iP®

Solution Note

Object-Oriented DDI Automation with Python

Simplify SOLIDserver automation while retaining full REST API flexibility



Executive Summary

Direct REST API calls give automation teams broad access to SOLIDserver, but they also require developers to manage service-specific methods, parameters, identifiers, responses, and relationships between DDI resources in application code.

`SOLIDserverRest.adv` adds an object-oriented Python model for resources such as IPAM spaces and networks. The same authenticated session also retains generic API access for concise or specialized operations. This hybrid approach lets developers use DDI objects where context and lifecycle matter, while preserving the complete API reach required for DNS, reporting, verification, and other focused service calls.

The Challenge

A typical infrastructure workflow may need to identify the correct IPAM space, select an approved parent network, find available subnet capacity, create a subnet in the right hierarchy, identify a free host-address candidate, publish a DNS record, and verify the result.

Implementing every step through direct API calls is possible, but the script must handle method-specific parameters, object identifiers, hierarchy, response normalization, error handling, and data exchange between stages. This can make automation harder to read, reuse, and maintain.

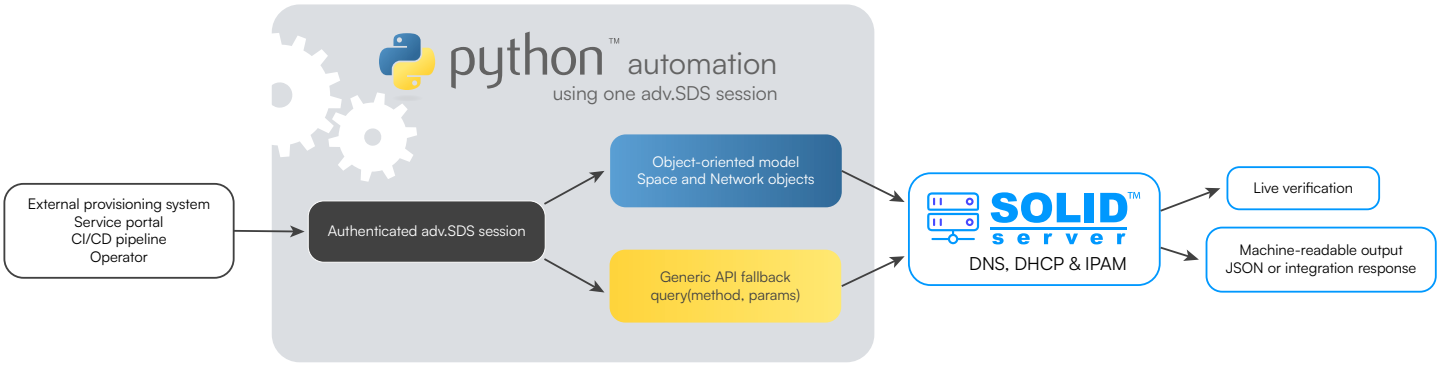
At the same time, an object model should not restrict access to specialized API functions. Automation teams need a practical way to choose the most appropriate abstraction for each operation without changing clients or building parallel integrations.

Solution Overview

`SOLIDserverRest.adv` provides two complementary access patterns through one authenticated `adv.SDS` session:

- **Object-oriented DDI operations:** Classes such as `Space` and `Network` represent resources, refresh their state, express relationships, and perform lifecycle operations.
- **Generic API queries:** `query(method, params)` supports direct list, information, reporting, DNS, and specialized service-method calls.

Both patterns share the same SOLIDserver session, authentication context, timeout settings, and error-handling framework.



Object-Oriented DDI Operations

The advanced model makes the DDI intent visible in Python code. The following example represents a parent network and asks SOLIDserver to find available child subnets:

```
space = sdsadv.Space(sds=sds, name=space_name)
space.refresh()

parent = sdsadv.Network(sds=sds, space=space)
parent.set_address_prefix(block_address, block_prefix)
parent.refresh()

candidates = parent.find_free(
    prefix=requested_prefix,
    max_find=limit,
)
```

A selected subnet can then be linked to the parent, created, refreshed, and queried for free host-address candidates:

```
subnet = sdsadv.Network(sds=sds, space=space, name=subnet_name)
subnet.set_address_prefix(subnet_address, subnet_prefix)
subnet.set_parent(parent)
subnet.set_is_terminal(True)
subnet.create()
subnet.refresh()

free_addresses = subnet.find_free_ip(max_find=limit)
```

The code mirrors the DDI model: a network belongs to a space, has a parent, can be created, and can be queried for free capacity.

Generic API Query Fallback

Some operations are clearer as direct API calls. For example, after selecting a zone, fully qualified domain name, and address, the workflow can create an A record through the same advanced session:

```
client.query(
    "dns_rr_create",
    params=record_parameters,
)
```

The same fallback can be used for lightweight inventory, reporting, and final verification. Developers retain direct access to the broader SOLIDserver API without opening another connection.

USE CASE

Solution Use Case: Policy-Controlled Network and DNS Provisioning

A service portal, cloud provisioning platform, CI/CD pipeline, workflow engine, custom application, or operator can initiate a request for network and DNS resources. Python automation then translates that request into controlled DDI operations through one authenticated `adv.SDS` session.

A typical workflow includes the following stages:

1. **Validate the request and resolve its context:** Confirm the requested prefix, naming information, metadata, and applicable allocation policy. Resolve the appropriate IPAM space and identify the networks in which the request is allowed to operate.
2. **Identify available capacity:** Use `Space` and `Network` objects to inspect the current SOLIDserver state and return suitable free-subnet or host-address candidates.
3. **Select and create the required resources:** Apply organizational selection rules, create the selected subnet in the correct hierarchy, refresh its state, and capture the resulting object identifiers and metadata.

USE CASE

4. **Coordinate address and DNS handling:** Depending on the use case, create or reserve a selected address in IPAM, pass it to another provisioning system responsible for the address lifecycle, or use it as input for an associated DNS operation. Focused DNS actions can be performed through the generic API-query interface on the same authenticated session.

5. **Verify the resulting state:** Read the created or updated resources back from SOLIDserver rather than relying only on the response from the write operation. Verification can include the network hierarchy, address state, DNS records, audit information, and other relevant metadata.

6. **Return a structured result:** Provide the initiating system with machine-readable output containing the selected context, created resources, resulting identifiers, verification status, and any information required by subsequent workflow stages.

Inputs and outputs can be exchanged as JSON documents, API payloads, workflow variables, event messages, or native objects from the surrounding automation platform. This allows SOLIDserver automation to participate in wider provisioning and deprovisioning processes without requiring manual interaction.

`find_free()` and `find_free_ip()` can return multiple candidate subnets or addresses. A workflow may select a candidate according to capacity, location, naming, utilization, or other organizational policies. Random selection from the returned list can reduce the probability that concurrent workflows choose the same candidate, but it does not eliminate the race between discovery and creation. Where strict concurrency control is required, the workflow should create or reserve the selected resource immediately where applicable, detect conflicts, and retry with another candidate.

Production implementations can also separate planning from execution through preview or dry-run behavior, approval gates, explicit commit actions, and post-change validation. These controls are implementation choices rather than requirements of the Python library itself.

The same pattern is not limited to subnet and DNS provisioning. It can be extended to address allocation, pools, DHCP scopes, resource metadata, DNS record lifecycle management, reporting, reconciliation, and deprovisioning.

Solution Benefits

Simplify DDI automation development
with Python objects for resources, relationships, and lifecycle operations.

Improve readability and maintainability
by expressing the intended DDI operation directly in code.

Retain complete API flexibility
for focused or specialized SOLIDserver service calls.

Accelerate ecosystem integration
through simple JSON hand-offs with portals, pipelines, workflow engines, and cloud platforms.

Reduce change risk
with dry-run behavior, explicit write execution, and live post-change verification.

Strengthen auditability
by capturing selected context, creating object identifiers, trace information, and verification results.

Key Components

- **SOLIDserver DDI:** Unified DNS, DHCP, and IP address management with REST API access for external automation.
- **SOLIDserverRest:** Python mapping for direct SOLIDserver REST API operations.
- **SOLIDserverRest.adv:** Object-oriented classes for key DDI resources, with generic API access through the same session.
- **Python automation or orchestration platform:** A standalone script, service portal, CI/CD pipeline, provisioning system, workflow engine, or custom NetOps application.

Production Considerations

Production implementations should use a dedicated least-privilege API identity, validate the SOLIDserver TLS certificate, protect credentials outside source code, and retain controlled planning or approval before write execution. They should also handle concurrency, repeated requests, partial failures, and reconciliation. `find_free()` and `find_free_ip()` can return multiple candidate subnets or addresses. Selecting one randomly from the returned list reduces the probability that parallel workflows choose the same candidate. Where strict concurrency control is required, the workflow should create or reserve the selected resource immediately and retry if another process has claimed it in the meantime.

Conclusion

`SOLIDserverRest.adv` makes SOLIDserver automation more expressive without sacrificing REST API coverage. The object model is well suited to stateful operations where DDI context, relationships, and lifecycle matter, while the generic query fallback remains available for concise or specialized calls.

By combining both patterns, organizations can build readable and reusable Python workflows for subnet provisioning, DNS registration, verification, and wider ecosystem integration while keeping SOLIDserver at the center of controlled DDI operations.

efficient iP®

REV: C-260817

Americas
EfficientIP Inc.
1 South Church Street
West Chester, PA 19382-USA
+1 888-228-4655

Europe
EfficientIP SAS
90 Boulevard National
92250 La Garenne Colombes-FRANCE
+33 1 75 84 88 98

Asia
EfficientIP PTE Ltd
60 Paya Lebar Road #11-47
Paya Lebar Square SINGAPORE 409051
+65 6678 7752