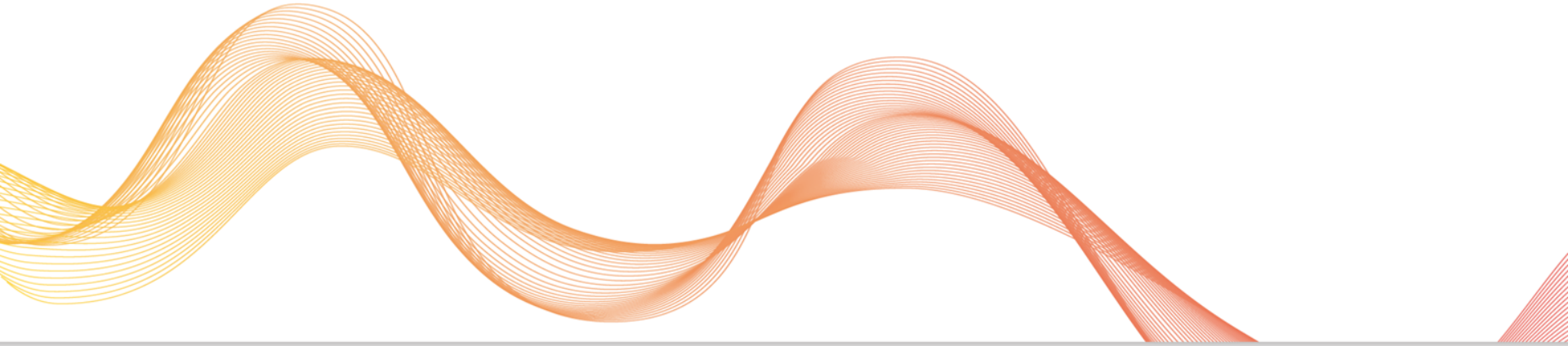
ASAM | Association for standardisation of automation and measuring systems



Release Presentation

ODS Web Services Version 1.1.1

Open Data Services Via Web Services

Release Date: 2014/09/30

Deliverables

- The document represents a companion standard recommendation for interacting with ODS services using web services in 3 levels:
  - Basic entry level
  - Intermediate Level
  - Advanced Level
- Each containing
  - use cases
  - function chapters
  - descriptions
  - input and output objects

Introduction

- ODS Web Services intends to ease basic Create, Read, Update, and Delete (CRUD) tasks for lightweight clients such as browsers and tablets as well as to provide a simple interface for other systems.
- Focus on managing ODS metadata rather than on importing & exporting measurement values

Compatibility with other standards

- ASAM ODS
  - v1.1.1 is fully compatible and tested with ASAM ODS 5.1
  - No known limitations or exceptions
- Responses of the ODS web server to client API calls are compatible to the ASAM ODS XML schema in the sense that the return values are schema-compliant XML-strings

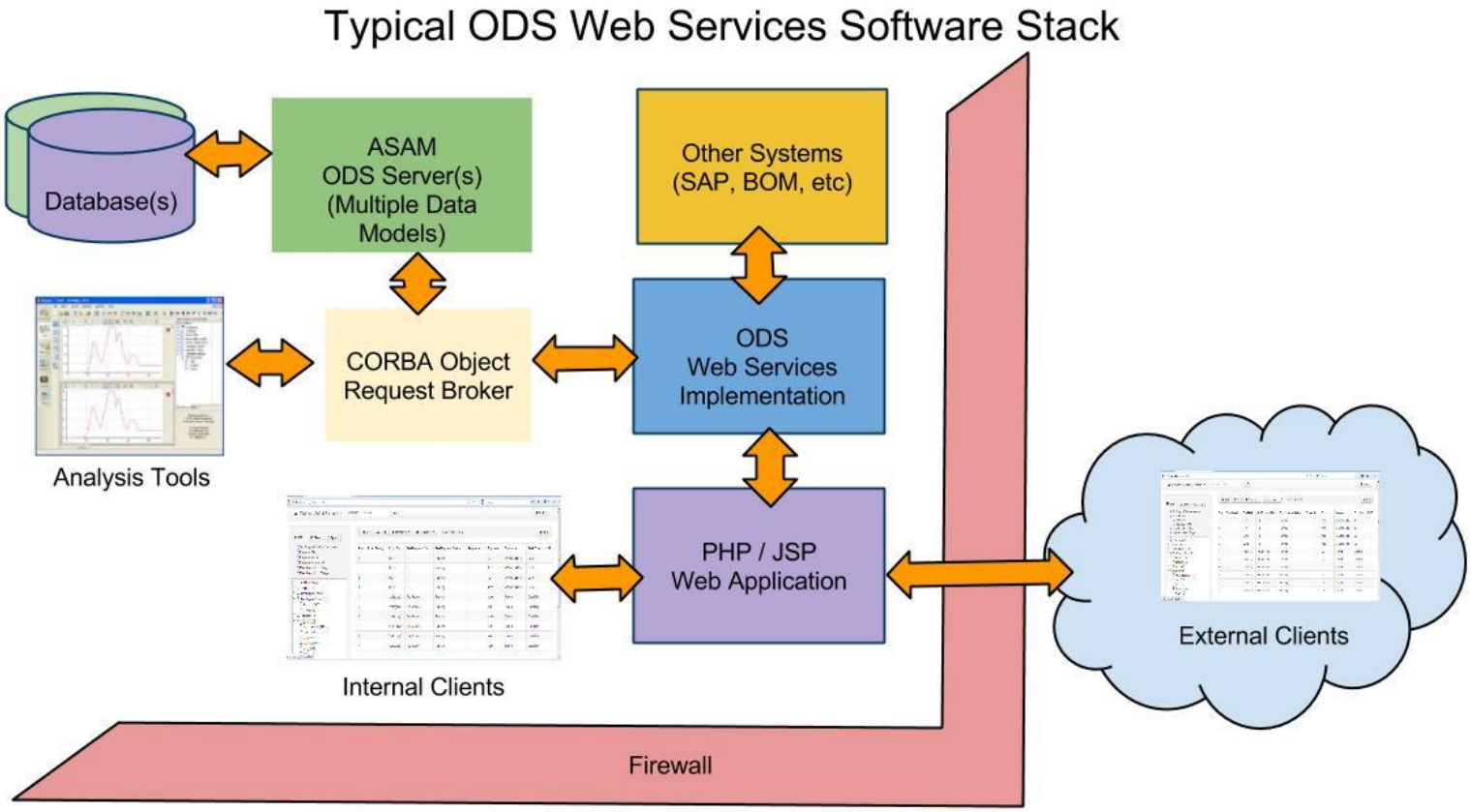
Marketing

- Use-cases:
  - Interface with External Systems
  - Maintaining Users
  - Previewing Data
  - Running Queries
  - Building navigation trees
  - Handling AoParameterSets
  - Cascading Delete
  - Handling Rich Data
  - Multiple Simultaneous Connections

General Concept of ASAM ODS

- ASAM ODS (Open Data Services) is that part within the portfolio of ASAM standards which focuses on persistent storage and retrieval of such data, thereby establishing standards for such application areas.
  - A common data model (base model) for the unambiguous and complete storage of data.
  - Interfaces (API, Application Programmer's Interfaces) to access data of ASAM ODS compatible systems and tools in a standardized way.
  - A standardized, easy to use, text-based exchange format (ATFx, ASAM Transport Format) in order to exchange ASAM ODS data (including its meta-information) between different systems and different platforms.
  - A database model for the (wide-spread) relational databases.
  - A set of application models, reflecting typical scenarios for the use of ASAM ODS and which easily allow a mapping between data originating from different companies.

General Concept



Use Cases

- ODS Web Services (ODSWS) is intended to provide a simple HTTP(S) interface to ASAM ODS servers for many, but not all, of the ODS OO-API calls.
- Overriding goal was to simplify basic Create, Read, Update, and Delete (CRUD) tasks for lightweight clients such as browsers and tablets as well as to provide a simple interface for other systems.
- While ODSWS uses a REST-Like /JSON interface, it is not intended for direct use by public web clients; rather it is meant to be the ODS component of another system that is properly equipped to handle authentication and other security issues.
- Maintaining Channel Names
- Virtually all ASAM ODS systems use AoQuantity instances for providing some consistency and control over Measurement Quantity or channel names.
- ODSWS provides a very simple mechanism for clients to validate channel names against the data in the ODS server.

Use Case: Maintaining Channel Names

- Virtually all ASAM ODS systems use AoQuantity instances for providing some consistency and control over Measurement Quantity or channel names.
- ODSWS provides a very simple mechanism for clients to validate channel names against the data in the ODS server.
- Provides simple API calls to create new channel names when necessary.

Use Case: Interface with External Systems

- Many ODS systems utilize meta-data that originates in other systems.
- ODS was not intended to provide full BOM functionality, yet UnitUnderTest or other build information is often critical to putting measurement data into context.
- ODSWS provides a simple mechanism for external systems to provide basic UUT or TestSequence information.

Use Case: Lookup Lists

- Many applications would be well served if they allowed users to choose values from lists rather than forcing the users to type values into free-form text fields.
- Using lookup lists helps maintain the integrity of the data, improves data consistency, reduces import errors, and promotes data sharing & reuse.
- ODSWS provides very simple methods to obtain catalog data such as units, channel names, test names or other attributes stored in ODS tables.
- Using a simple HTTP GET, a client can obtain values to populate list boxes on web pages or even thick clients.

Use Case: Maintaining Users

- Most ODS systems utilize instances of the AoUser element to control access to the data. ODSWS provides very simple methods to create new users, modify their membership in user groups, change passwords, and to delete existing users.

Use Case: SECURITY

- The ASAM ODS security specification provides multiple levels of granularity for protecting instances from unauthorized access.
- ODSWS allows new Access Control List (ACL) security information to be assigned during the initial creation of an instance element as well as for maintaining the ACL for instances over their lifetime.

Use Case: Previewing Data

- While accessing large amounts of measured data would be impractical via ASCII and HTTP, ODSWS does provide methods to obtain value matrix values.
- Clients can request subsets of channels, ranges of rows, and even unit conversion. Clients can also request a sampling of rows in order to reduce the number of rows returned.
- For example, by requesting every 100th row for a subset of the channels, a client can get a quick overview of the measurement without waiting for all data to be downloaded.

Use Case: RUNNING QUERIES

- The ODS extended query functionality can require a significant amount of low-level API calls for all but the simplest queries.
- ODSWS provides a convenient way to take advantage of the power of ODS query. With one high-level API call, users can submit sophisticated queries and obtain many columns of data.

Use Case: BUILDING TREES

- Many client applications use trees for providing navigation.
- ODSWS provides several methods to populate the information that is typically found in navigation trees.

Use Case: MULTI-VERSIONED DATA

- The ODS base model provides a version attribute to allow clients to track modifications to instances.
- As a result, many instance elements have duplicate names and use the version attribute to distinguish them.
- Unfortunately, the ODS OO-API does not provide a version parameter to the getInstanceByName() API call.
- ODSWS has addressed this limitation and it is now a very simple process to obtain instance data using the instance name plus the version information.

Use Case: PARAMETER SETS

- Parameter sets provide a simple, yet powerful method to store flexible groups of Key:Value pairs of information.
- This gives ODS the flexibility to adapt to changing needs over time as well as to handle a wide variety of data with little pre-configuration.
- Using the idea that parameter Key:Value pairs are essentially dynamic attributes, ODSWS provides convenience methods for both storing and retrieving parameter sets.
- Key:Value pairs used for parameter sets are a native structure for the JSON used by ODSWS.

Use Case: Rich Data

- ODSWS allows clients to request in a single API call a bounty of information about any given instance including its parent(s), children, and siblings.
- Clients can specify which relations are to be used and which ones can be ignored. In a single call, a client could obtain all attributes for an instance of AoMeasurement in addition to the names, ids, and versions of its channels, submatrices, unitsUnderTest, AoSubtest, and test sequences.
- If the measurement has relations to 1 or more parameter sets, ODSWS will automatically obtain the Key:Value:Unit items related to the parameter sets.
- This same API call will also return a simple structure of the instance's ancestors, which, unlike the more rudimentary ASAMPath, includes the instance ids of the ancestors.
- ODSWS will also return the ACL of the instance for the convenience of the client.

Use Case: Cascading Deletes

- When deleting a parent instance, it is often desirable to delete the children.
- ODSWS provides the ability to issue a controlled cascading delete.
- This allows clients to specify which relationships are to be used in determining which items should be deleted.
- While ODSWS makes intelligent assumptions about the order in which deletions should occur, clients may provide their own instructions in order to reduce foreign key & integrity constraint violations.
- ODSWS will return the instance ids of all the records it deleted.
- It will also return the names, descriptions, and URLs of any files stored in the external reference attributes of any deleted records that could otherwise be potentially orphaned.
- The files themselves are not deleted by ODSWS, but the client may utilize the list of files for this purpose.

Use Case: Super User

- Surprisingly, the task of determining whether the logged-in user is a super user is not possible with a single ODS OO-API call.
- ODSWS automatically provides this information when a user logs in.
- Clients may use this information to allow/disallow program functionality.

Use Case: Multiple Simultaneous Connections

- ODSWS allows clients to connect to multiple ODS data models simultaneously.
- This powerful capability could be used for synchronizing meta-data between multiple systems.
- Comparing channel names in one location vs. those in another location becomes trivial.

Use Case: High Level API Calls

- Because of the higher latency associated with HTTP calls, ODSWS strove to combine many low-level API calls into a single call.
- For example, when creating an instance element with ODSWS, it is possible to create the instance, populate it with attribute values, link it to existing elements using pre-defined relations, and assign it security ACL – all in a single API call.
- Updating instances is just as simple.

Use Case: Generating Crumbs in Web Pages

- Many web pages use crumbs as a navigational aid.
- The crumbs essentially show the ancestors of the given object and allow the user a chance to jump to any parent or other object in the hierarchy.
- The ASAM Path object provides most of the information necessary to produce a working set of crumbs, but it lacks a critical piece of information – the instance ids of the ancestors.
- ODSWS solved this problem by producing a list of the ancestors for any given instance.
- The ancestor information includes element name, instance name, instance version, and instance id.
- That should allow any client to easily generate the appropriate values for crumbs.

Use Case: 64 Bit Integers

- ODSWS uses 64-bit integers throughout.
- For older clients that need to convert older T_LONGLONGs into true 64-bit integers – and vice-versa, ODSWS provides 2 simple utilities.
- This simplifies things for clients that do not have register shifting capabilities and for clients that just want to use a standard library instead of writing their own functions.

Use Case: Extending ODS to Other Languages

- While C++ and Java are still very important languages, many (if not most) modern web applications are written in PHP, Rails, Python, node.js and other more recent languages.
- Few of these languages support CORBA libraries.
- To test the design of ODSWS, a number of PHP and node.js tests were performed.
- Adding ODSWS to a typical PHP application requires only a few simple lines of code.

Use Case: Security Information

- Secure environments use many of the following strategies to reduce risk:
  - Network filters such as firewalls and ACL rules on routers should be used to block all connections from unknown or untrusted systems from accessing the ODSWS server.
  - ODSWS or their hosting servers should limit access only to & from certain IP addresses and ports.
  - ODSWS should be limited to non-public local networks and secured virtual private networks.
  - ODSWS should be integrated with strong, complex authentication credentials such as LDAP.
  - ODSWS users should deploy a model of least privileges whereby users and processes only possess the minimum amount of privileges to accomplish required tasks and that more powerful, administrative tasks be reserved for certain, infrequently used accounts.
  - ODSWS or its hosting server should follow best practices for monitoring against, and preventing, common attacks.

General Information: Logical vs. Physical Specification

- Unlike the ASAM ODS Specification with its heavy dependence upon the physical storage specification, ODSWS is purely a logical API with no dependence upon any type of web server or other internal processing on the server or capabilities on the client.
- Examples in the Documentation use an Apache Tomcat J2EE Web application server, JQuery JavaScript clients, and command-line cURL clients.
- These are purely for purposes of illustrating usage of ODSWS and do not imply that these are the only tools allowed.
- ODSWS should work with any computer language that can issue HTTP GET & POST requests.

General Information: ODS Data Models and Server Limitations

- ODS Web Services can be implemented as a proxy to one or more ODS servers.
- It is designed to be agnostic regarding ODS servers, versions, and data models.
- It includes an API call to allow potential clients to determine what ODS versions are supported.

General Information: REST-Like JSON Interface

- The overall style of the API was based upon several popular REST-Like / JSON interfaces currently in use.
- It does not follow the pure REST design patterns.
  - REST guidelines suggest that GET, POST, DELETE, and PUT http calls be used for varying purposes.
  - This suggestion often breaks down in real-world usage.
  - In ODSWS, the type of call and the complexity of the structure are what determine whether GET or POST should be used.
  - Any API calls that could affect the state of the ODS server are required to use POST requests rather than simpler GETs.
  - Many methods allow either GET or POST requests, but several require the sophisticated structures of data that are simply not possible using only a GET.

General Information: Headers and Cookies

- Nothing in this specification requires nor disallows the use of cookies or other header information to be sent along with requests.
- ODSWS makes extensive use of AoSession Id tokens.
  - They may be passed as part of the URL, as part of the POSTed content, or as cookie values.
  - The transport mechanism is not relevant to ODSWS – just the content of the transported information.
  - JSON is the content type for both requests and responses.
  - All clients should specify the formats and character sets that they are sending and prepared to receive.
  - All examples in the Documentation use ‘application/json’ and UTF-8.

Connections

- Virtually all ODSWS API calls require an active connection to an ODS server.
- These connections are referred to in this document as AoSessions.
- Clients can use the openAoSession() and closeAoSession() API calls to connect and disconnect to any applicable ODS server(s).
- Clients can use the listServices() API call to discover which ODS servers and data models are available.
- There are no restrictions imposed by ODSWS on the number of concurrent AoSessions any single client can create
  - listServices – to obtain a list of available ODS services (data models),
  - openAoSession – to connect to an available ODS service,
  - checkAoSession – to determine if a connection to an ODS service is valid,
  - closeAoSession – to disconnect from an ODS service,
  - getModel – to obtain detailed information about an ODS service (data model).
  - getServerInfo – to obtain version information about the ODSWS server.