# Application Configuration

Application is configured by setting parameters in application.xml. By default this file is in WEB-INF folder in a web deployment. Should there be a need, it is possible to keep it elsewhere though. We insist on an applicationId as the only required attribute in this, though we do not use it anywhere!!. For a useful application, you will probably set many more attributes. Remember that false is the default value for all our attributes.

|  |  |  |
| --- | --- | --- |
| Attribute name | Required? | Description |
| applicationId | yes | Unique name within a corporate infrastructure. We do not use it directly anywhere by default, but it should be useful if you have to get privileges to access corporate infrastructure like SSO. |
| sendTraceToClient | no | Simplity produces a simple trace of what happened during a service execution. This is logged using the logging infrastructure you choose, or onto the console of the server/log File by default. During development, it is quite handy for the developer to get them on to the client as well. For security reasons, you must set this to false in production. |
| cacheComponents | no | One of the considerations of software development is the time and effort required for test-fix cycles. When you use Simplity, most of the functionality is delivered with components that are not java programs. They are xml files. This provides us with an opportunity to do end-to-end test-fix cycles without a re-build and restart, so long as the changes are to the xml files. However, we can avoid the over-head of repeated loading of these files in production by caching them on a need basis. If you set this attribute to true, xml files are loaded on a need basis, but once loaded, they are cached for repeated use. |
| dbVendor | no | Required if data base is used in the application. oracle, mssql, postgre and mysql and h2 are supported readily. Simplity has a fairly compact set of modular routines that wrap and hide jdbc driver from general use. With this approach, it is fairly easy to add support for other vendors on a need basis. |
| connectionString | no | Not recommended. Better to use dataSource. Could be quite helpful during development. Traditional connection string to connect to the rdbms using jdbc. To be used with dbDriverClassName |
| dbDriverClassName | no | As per jdbc specification for the driver when connection string is used. |
| dataSourceName | no | Preferred method to connect to the data base. This is the JNDI look-up for the class you have written for database crendentials. |
| loginServiceName | no | This designated service is executed as a login-service, or after a login, depending on your infra-structure development. Login service can have two purposes:  1. Authentication : this service may take login credentials from a client and authenticate. If the site uses SSO cookie, this service may validate the SSO cookie and get the login credential from central system.  2. Getting session/global parameters: Most practical applications use some attributes (like roles, privileges and preferences) of the logged in user across services. Login service may fetch these attributes and set them as session parameters. Simplity provides a simple way to set these attributes, and automatically makes them available to all services, as if they are part of the input that is sent from client.  Simplity uses an internal dummy logic as a default that clears the login id without any verification. |
| logoutServiceName | no | If specified, this service is invoked when the client explicitly requests a logout as well as when there is a session expiry. |
| userIdIsNumber | no | Simplity operates in a client-session, and every client session is associated with a userId. This, by default is assumed to be a text field. Use this attribute to change that to a number. Of course we do not provide flexibility to have a Date or Boolean as data type of userId ☺ |
| autoLoginUserId | no | A convenient way to start development quickly before an elaborate security arrangement is made. Ensure that this value is a number ifyou have setuserIdIsNumber=”true”. Once you set this attribute, Simplity automatically creates a session for this userId, and all service requests are executed with this userId as the requester.  Note that the designated loginService is invoked with this userId whiel trying an aut-login. Either you do not designate any service, or ensure that the designated service expects no credentials to clear this login. |
| httpCacheManager | no | Do you want to cache service responses at the http(web) tier? We have provided an example named org.simplity.http.SimpleCacheManager. You may use that, or write your own on similar lines. It has to implement HttpCacheManager interface. Provide fully qualified class name of the manager here. |
| serviceCacheManager | no | Do you want to cache service responses at the service(app) tier? This is similar to httpCacheManager, except that http session is not available for your class, and hence has to implement ServiceCacheManager interface. Provide fully qualified class name of the manager here. |
| exceptionListener | no | Any inconsistency in the application design, for example using a dataType that is not designed, results in an ApplicationException. This is caught at the highest level. Typically such an exception requires technical team’s intervention. You may want to pipe this to any corporate level exception handling with this application. |
| logSqls |  | During development, it would be quite helpful for developers to see the exact sqls that are run by jdbc driver. However, the corporate may have some security restriction and suppress these during production. Set this attribute to true to log the actual sqls. |
| accessController | no | plugin that can be used to implement user-id specific access control to services. By default Simplity allows access to all services to any logged-in users. It is upto the service designer to deny it. However, if access control is to be implemented outside of a service design, then you can write a plugin that implements AccessController interface and specify the class name in this attribute. Access control is consulted by service-agent before invoking a service for a userId. |
| schemaDetails | no | Use this facility if your application needs to work with more than one schema. Main login credentials that is specified using connectionString ordataSource is used by default. Other schemas are to be specified in the section. A service can specify schema name to be used instead of the default one, and simplity takes care of changing the schema for that service. Of course the schema should be from the list specified here. |
| attachmentAssistant | no | Simplity provides plugin for saving and retrieving attachments (files) as an alternative to saving them as part of an rdbms row. Corporates may have central infrastructure, in which case you write your class that implements AttachmentAssistant interface. Leave this null if you can get away with the default plugin available (that requires setting attachmentsFolderPath attribute. |
| attachmentsFolderPath | no | A simple approach to save attachments is to use a folder for the same. We provide a ready plugin in case this approach is suitable for you. Just specify a folder that is accessible to web tier as well as app tier, and Simplity manages the rest. |

## Deploying the Application

Simplity based application can be hosted in a variety of ways, because the application by itself assumes no infrastructure except a JVM. Service request can be routed through Agents. HttpAgent is one such agent. It is designed to be called from a servlet. We provide a sample servlet called Serve that assumes that security is handled by other means. Similarly we provide DefaultLogin and defaultLogout servlets.

These servlets need to be configured in web.xml. If Simplity client is used, Serve should be mapped to \*.\_s, DefaultLogin to \*.\_i and defaultLogout to \*.\_o. In case you use attachments, Stream needs to be mapped to \*.\_s we use these extensions to provide maximum flexibility in deploying the application with different folder structures.

Refer to web.xml file in the tutorial for an example.

# Message

Messages help us send non-data response to client in a structured and manageable way. Messages can have placeholders for run-time values. Messages can be translated to the desired language before responding back to client.

Messages are organized in one or more files under msg/ folder. Attributes of messages are:

|  |  |  |
| --- | --- | --- |
| Attribute name | Required? | Description |
| name | yes | Should be unique across all files. |
| text | yes | Text of the message in English. This may contain place holders for run-time parameters like $1, $2 etc.. Message is formatted at run time by replacing these place holders with the values that are made available. |
| messageType | yes | error, warning, info and success. We have fixed meaning for type=error. Other types are for mean for client to render them with appropriate style. |

# Data Type

A data type component defines the restrictions on the values a field can have. Its primary purpose is to validate values that are received from the client. Data types are organized into one or more files inside /dt/ folder. Data type names are to be unique across all files for a project. Simplity internally uses some data types that are defined in kernel.xml. All the names start with \_ to avoid clash with project specific data types. Though it is alright to use these data types, we recommend that you define all the data types that you use in your project.

Data types may also be used to automate validations on the client side. There are utilities to generate scripts for data types that can be triggered to validate data before making a service request.

### Common attributes for all data types

|  |  |  |
| --- | --- | --- |
| Attribute name | Required? | Description |
| name | yes | Should be unique across all files. |
| messageName | no | Specific message name to be used to raise an error if a field of this data type fails validation |
| sqlType | no | Use this feature if you want to use record definitions to create data base tables |
| formatter | no | Use this feature if you use data types on your client side. This is a java script function name that is triggered while rendering the field value |
| valueList | no | Feature to create enumerated types. For example “red,blue,yellow” or “1:red,2:blue,3:yellow”. In the first case value is text, while in the second case value is number, but client uses it as text |
| description | no | documents this data type. Also, this is used as message in case messageName is not specified |

### Text data type

|  |  |  |
| --- | --- | --- |
| Attribute name | Required? | Description |
| minLength | no | minimum number of characters |
| maxLength | no | maximum number of characters |
| regex | no | pattern as per java syntax |

### Numeric data type

|  |  |  |
| --- | --- | --- |
| Attribute name | Required? | Description |
| minValue | no | minimum integer value, ignoring fractions |
| maxValue | no | maximum integral value ignoring fractions |
| nbrFractionDigits | no | 0 means this is integral. Number of decimal places to maintain. For example if you want to have amount with cents/paisa like 12.34, you would set this to 2. |

### Date data type

|  |  |  |
| --- | --- | --- |
| Attribute name | Required? | Description |
| maxDaysIntoPast | no | What is the lower limit on the date. This is relative to current date (today). For example if the delivery must be at least 10 days into the future, then you set this to -10. If the product has to be at least 10 days old, then you set this to 10. |
| maxDaysIntoFuture | no | What would be upper limit on the date relative to today. |
| hasTime | no | Date fields are confusing at times because of the way computing world treats them. For business purpose, date has no time-zone complication. This is different from an instance of time, say time-of-arrival that requires the time-zone. Computing world uses the same format to store both, and hence confusion at times.  if hasTime is set to true, then this data is considered to be an instance of time, else it is considered to be a date (a day in the calendar) |

### Boolean data type

Client may use 0 for false and 1 for true.

# Record

Table, entity and data-structure are the other possible names for this component. Essentially we define a group of fields into a record and use it for different purposes. A record helps us in implementing one of the golden rules of design : DRY (Do not Repeat Yourself) Once a record is defined, we can use it to represent a database table/view, or a data structure for a parameter for a stored procedure, or set of fields that are expected as input from client etc. We have three types of records:

1. Storage: represents a record that is stored as part of your data base. A storage record maps directly to a table in your RDBMS. Note that only storage records can be used for writing to the data base. (add/modify/delete)
2. View: represents a record as seen by end-users. Typically a view has either a subset of fields from a storage record, or it is formed by picking fields from a set of related storage records. For example when you show details of a customer on a page, you may want to show total order value of this customer, number of pending orders etc.
3. Structure: this is a set of related fields that are utilized by some part of your application, but they are not directly mapped to any field in a storage record. These fields are temporary data elements that are calculated at run time to implement some business rules.   
     
   Structure is also used to model the object/struct definition in your RDBMS that are used by stored procedure.

Records are organized in one or more files under rec/ folder. Attributes of a record are :

|  |  |  |
| --- | --- | --- |
| Attribute name | Required? | Description |
| name | yes | Unique name within a module. |
| moduleName | no | Should match the folder structure under which this component is saved. If a.b.c is the published name of this record then module name is a.b, record name is c, and the component is saved as /a/b/c.xml under the root folder meant for records (/rec/). |
| recordType | yes | storage, view or dataStructure |
| fields | yes | list of fields. |
| tableName | no | This is the name of the table or view in the database. Defaults to name. |
| okToSelectAll | no | A simple check against an unconditional select from the table to avoid too many rows getting extracted. If false, filter action on the record will raise an error if no conditions are specified. |
| keyToBeGenerated | no | If set to true, key is generated using appropriate method based on the underlying rdbms during a add action. |
| childrenToBeRead | no | Comma separated list of fully qualified record names that are the child tables of this table whose rows are also to be selected/read when we read/filter rows using this record. This attribute is used only when we use an auto-service or on-the-fly service. |
| childrenToBeSaved | no | Comma separated list of fully qualified record names that are the child tables of this table whose rows are also to be saved when we save rows using this record. This attribute is used only when we use an auto-service or on-the-fly service. |
| listFieldName | no | If this table is the source of a list of values for a drop-down, you can use this record name for auto-service or on-the-fly-service. Each row in the list has two columns. First column is the primary key of this table. Second column would be this column. For example if this table has all the countries, then listFieldName=”countryName” |
| listGroupKeyName | no | This attribute is also meant for drop-down list. For example if you have states of several countries in this table, and you want to send states for a given country name, then use valueListKeyName=”contryCode” |
| defaultSheetName | no | Defaults to name. This name is used as default for data sheets that are associated with this record. |
| sequenceName | no | name of the sequence to be used to get the next key value from in case oracle is used. Ignored for non-oracle use. Defaults to tableName\_SEQ. |
| suggestionKeyName | no | This is required if this record is to be used for a suggestion service. This is the field name against which typed characters are to be matched. |
| suggestionOutputNames | no | Comma separated list of field names to be extracted, in addition to the key field mentioned above, for each row in the suggestion list. |
| readOnly | no | Must be set to false for any write operation using this record. |
| defaultRefRecord | no | Required if this is a view. This is the fully qualified name of the default reference record for fields. |
| schemaName | no | In case the underlying table/view is not part of the default schema. If this is specified, designer has to ensure that any record-based I/O action, like saveAction, is used in a service that uses this schema. A service cannot use components from different schemas. |
| useTimestampForConcurrency | no | It is common practice to detect ‘dirty-update’ using last modified time stamp. Example,user A reads a row, user B also reads the same row while user A is still looking at it. User A updates the row. Now, user B, who is unaware of the latest values A has updated, makes changes to this row, and submits for update. This, in most case, should not allowed because amongst other things, user B may be unintentionally over-writing user A’s changes.  Ensure that you have designated a field as fieldType=”modifiedAt” and its dataType=”timestamp”. And that you have extracted the timestamp for the client to send it back for the update service request. |
| okToCacheList | no | Applicable if you have set up this application for caching (refer to application.xml settings). The auto service (on-the-fly-service) list\_recordName can be marked for caching by setting this attribute to true. Caching would be across all users. If listGroupKeyName is specified, then caching will happen for each value of that field, else it wouldn’t depend on input. |
| sqlStructName | no | Name with which this structure is defined as a type in your rdbms and it is a complex strucrure. A structure is complex if any of it field is non-primitive i.e. any of its field is a child-record or an array (of primitive or child-records) |
|  |  |  |

Field

Fields make up a record. Attributes of a field are:

|  |  |  |
| --- | --- | --- |
| Attribute name | Required? | Description |
| name | yes | Unique within a record. |
| fieldType | no | data – holds data for this field  primaryKey – this is the primary for this record  parentKey – links to primary of parent record  foreignKey – links to primary key of a related record, (not the parent record)  createdTimeStamp – maintains time stamp when row is added. Simplity sets value to this field automatically if save action is executed using this record.  createdByUser – userId that created this row. Automatically populated during a save operation  modifiedByUser – user id that last modified this row. Automatically populated during a save operation  modifiedTimeStamp – maintains time stamp when row is last modified. Simplity sets value this field automatically if save action is executed using this record. Also, value this field is used for checking concurrency during a modify action.  view – all fields in a view have this value.  temp – this field is not part of the table. Default for all fields in a dataStructure  record – this is a child-record or child-data-structure in a complex data model. If any field is of this type, then the record is to be structure, because we do not support such columns in tables. This child record is specified as referRedrecordName. sqlTypeName should be specified for this to be used as a stored procedure parameter.  recordArray – same as record except that it is an array of that data structure.  valueArray – this field has an array of primitive values. |
| dataType | yes | One of the declared data types. |
| columnName | no | Name of the database column. Defaults to name. |
| isNullable | no | Is this a nullable column in the database. |
| referredRecord | no | Required for a field that is referred, or a field in a view, but defaults to the value specified at the record level. (defaultRefRecord) |
| referredField | no | field name in the referred record |
| valueList | no | Similar to valueList in dataType. To be used only if it is not specified as part of data type. |
| isRequired | no | Is this field mandatory when a client is expected to send fields for this record? Typically this is for an add operation. Simplity takes care making it optional for actions like filter and read. |
| defaultValue | no | Applicable if this field is optional. Value to be used if client has not specified value for this field. |
| basedOnField | no | This is for inter-field validation. This is used if this field is required if the other field is specified. Say A and B are optional. But if A is specified, then B should also be specified. |
| otherField | no | If one of the two fields is to be specified, then use this feature. You need to specify otherField for one of them. That is, if one of A or B is required, you may set otherField=”B” for A. DO not specify otherField=”A” for B also. |
| fromField | no | If this field is a to-field of a from-to duo, specify this attribute. For one duo, setting on one field is sufficient. However, we have provided fromFiled as well as toField to allw a field to be part f more than one pairs. |
| toField | no | refer to fromField. |
| messageName | no | Name of message component to be used in case this field fails validation. |
| sqlTypeName | no | required if this field is non-primitive (that is, valueArray, record or recordArray) This is the name of the type definition in your RDBMS. |
|  |  |  |

# SQL

SQL is the language that is used to direct the rdbms to retrieve data, manipulate data, or change database structure itself. This component allows you to design a dynamic sql to be executed at run time, based on run-time values. This component is to be used in a service using ReadWithSql Action or ExecuteSql Action.

|  |  |  |
| --- | --- | --- |
| Attribute name | Required? | Description |
| name | yes | Unique within a module. |
| moduleName | no | Should match the folder structure under which this component is saved. If a.b.c is the published name of this sql then module name is a.b, sql name is c, and the component is saved as /a/b/c.xml under the root folder meant for sqls (/sql/). |
| sqlType | yes | singleSelect : Data extraction sql. At most one row is returned by design. (like primary-key based read)  mutliselect : Data extraction sql. More than one row may be returned. (like filter)  update: data manipulation sql. Number of Rows affected is returned. |
| preparedStatement | yes | sql formatted properly as a prepared statement. Simplity does not check the syntax of the sql. Developer has to ensure that the prepared statement and the sql parameters are designed properly. |
| inputParameters | no | Refer ro SqlParamter attributes. Specifies the parameters that are in the right sequence for each of the ‘?’ in the prepared statement. |
| inputRecordName | no | If you have a record whose fields are in the right order for this prepared statement, use this feature rather than listing inputParameters. |
| outputRecordName | no | This is the preferred way to specify the output. |
| ouputParameters | no | In case you do not have a re-use for a record as output, you may list the output parameters instead of specifying a record name. Provide the output fields in the right order. name and dataType are the attributes of an output parameter. |

# SQL Parameter

Sql parameter is used for both input and output parameters

|  |  |  |
| --- | --- | --- |
| Attribute name | Required? | Description |
| name | yes | Field name or column in a data sheet that has the value for this parameter at run time. |
| dataType | yes | one of the declared data types. |
| defaultValue | no | if value is not found, this value is used |
| isRequired | no | defaults to false. An error is generated if this is true and there is no defaultValue and the parameter has no value at run time. |

# Stored Procedure

This component is a wrapper on a procedure that is available in the rdbms. For example a PL-SQL procedure saved in Oracle. You specify all the details that are required to invoke a procedure. These details can be extracted from db using metadata. We intend to develop a tool for that. At this time, you have to manually specify details and ensure that the types and sequence match.

Stored procedures, once defined like this, can be used in any service with executes action, or in your custom jva class.

|  |  |  |
| --- | --- | --- |
| Attribute name | Required? | Description |
| name | yes | Unique within a module. This need not be the actual procedure name inside the db. In fact, we recommend that you follow a camelCase naming convention that is more common than the ‘\_’ that is common in db. |
| moduleName | no | Should match the folder structure under which this component is saved. If a.b.c is the published name of this procedure then module name is a.b, procedure name is c, and the component is saved as /a/b/c.xml under the root folder meant for stored procedure (/sp/). |
| procedureName | yes | name of the procedure in the data base. |
| parameters | no | list of parameters. Refer to Procedure Parameter attributes. |
| firstParameterIsForReturnedValue | no | It is possible that your procedure is designed to return a value. (Not recommended, but no harm if it is already designed that way) In such a case, you should specify that as the first parameter in parameters, and set this attribute to true. |
| outputRecordNames | no | It is possible to write procedures that return one or more result sets. Provide a comma separated list of fully qualified record corresponding to the result sets. |

# Procedure Parameter

Stored procedure parameter has the following attributes.

|  |  |  |
| --- | --- | --- |
| Attribute name | Required? | Description |
| name | yes | Field name or column in a data sheet that has the value for this parameter at run time. This may or may not be the same as the name with which stored procedure is defined. Remember that the procedure parameters are passed by order, and not by matching names. |
| dataType | yes | one of the declared data types. |
| defaultValue | no | if value is not found, this value is used |
| isRequired | no | defaults to false. An error is generated if this is true and there is no defaultValue and the parameter has no value at run time. |
| inoutType | no | input, output or inOut |
| sqlObjectType | no | relevant if this parameter is a data structure/object as per db definition. This MUST match the declaration in the sp. recordName would be the record that models this data structure. It can be a regular table/view or a complex struct that has its fields with non-primitive type. For example a struct may contain other structs or array of primitives/structs |
| recordName | no | required if sqlObjectType is specified. This record must have fields that match the order and type as in the data-structure/object definition in the rdbms. |
| isArray | no | true if the parameter is an array |
| sqlArrayType | no | required if isArray is set to true. Should match the name as defined in the rdbms. |
|  |  |  |

# Service

In Service Oriented Architecture (SOA) it is all about services. In this approach, server is viewed as a logical entity that can respond to all the published services.

Most of the services in Simplity are configured ( or assembled ) using other basic building blocks. A service may also be implemented by a Java code, but that is for special cases.

A service is uniquely identified by clients with its published name. We recommend a qualified name of the form module.serviceName. For large projects you may go in for module.subModule.serviceName etc..

<diagram that shows service as input-specification , service-definition and output-specification.>

Simplity has several features that simplify designing transaction processing systems that use RDBMS as their data base.

### Service Context

Service context has the following attributes.

* userId – This is the value that identifies the logged-in user for whom this service is being executed. In case the data base tables have special columns like createdBy and modifiedBy, this is the value that would be pushed such columns. Also, actions may use this value to check for user privileges. This is a read-only field, and can be copied to any field using CopyUserId Action.
* messages – service accumulates messages during the execution of service. You ay add a message explicitly with AddMessage Action.
* fields – this is a collection of name-value pairs. Values are of type text, integer, decimal, date or boolean. You may explicitly set value of a field using SetValue Action. Several actions end up adding fields to this collection. When execution of actions complete, service uses this collection to get values of fields as specified in output specification.
* data sheets – Collection of tabular data identified by sheet name. Each row of the sheet has the same number of cells, one for each column. Several actions use sheets as inputs and create new sheets. You may manipulate a sheet using actions like AddColulumn, CopyRows, RenameSheet etc. Data sheets can be created and manipulated in your Java code with a rich set of APIs exposed by the DataSheet interface as well as Fields interface.
* custom objects – Collection of any Object, indexed by unique name. Accessible to your java code, and can be used to pass parameters across logic actions. Note that you should always try to use the standard fields and data sheets as far as possible, and resort to custom object only as a last resort. all the data and messages during the course of a service execution. Data includes fields and data sheets. Service context also has provisions to carry any arbitrary object that can be exchanged between logic actions.

### Service Execution

1. A service context is created for this service with the userId for the logged-in user.
2. Input data is parsed as per input specification into the service context. If there are any errors, service returns the error message(s) as response.
3. Session fields are copied into the service context. At this point, all session fields, input fields and input data sheets are available in the service context.
4. If this service specifies data base access, a JDBC connection to the db is established.
5. If the service specifies write access, a transaction is started for this connection.
6. Service execution now starts with the first action.
7. Service execution continues in a sequence, except for JumpTo and Loop action.
8. If any exception is thrown during the execution, service rolls back the transaction and returns to client with the error message.
9. When the service completes, either after executing the last action, or because of a JumpTo directive, context is checked for error messages. In case of any error message, transaction is rolled back, and the service responds with the error message(s)
10. If there are no error messages in the context, then the transaction is committed. A response is prepared based on the output specification.
11. Service now returns with the response, as well as any session fields that may have been requested back.

Service has the following attributes

|  |  |  |
| --- | --- | --- |
| Attribute name | Required? | Description |
| name | yes | Should be same as the file name (without .xml) in which this specification is saved. This would be the last part of the qualified name. |
| moduleName | no | Should match the folder structure under which this component is saved. If a.b.c is the published name of this service then module name is a.b, service name is c, and the component is saved as /a/b/c.xml under the the root folder meant for saving services. |
| dbAccessType | yes | none – no db access  readOnly – only read access. No write.  readWrite – write access, under a transaction.  autoCommit – write access but no transactions. Writes are immediately committed and no roll-back is possible. Should be used with care, as the service may end-up with half of its work done.  subservice – this service consists of other services and no other action. Each such service manages its own dbAccess. |
| inputData | no | specified data that is expected from client |
| outputData | no | specifies what data need to be sent t client on success of this service. In case of error, output data is ignored and error messages are sent instead. |
| actions | yes | actions to be executed |
| schemaName | no | special case where an application uses more than one schema. Default schema is assigned to the login credentials associated with application. Use this feature for a service that uses a schema other than this default one. |
| requestTextFieldName | no | Special case where a custom logic is designed to process the request data that is received as string. For example json. Request string is copied to this field as it is and the service starts executing the actions. |
| responseTextFieldName | no | Special case where the string with which this service need to respond back is already decided and formatted by some action into a field. Output specification is ignored and text from this field is used as response. |
| referredserviceForInput | no | fully qualified name of another service that has the same input requirement. This is a simple feature to avoid duplicating input specification across related services that are expecting the same input by design. |
| referredserviceForOutput | no | fully qualified name of another service that has the same output requirement. This is a simple feature to avoid duplicating output specification across related services that are expected to respond with same set of data by design. |
| executeInBackground | no | true if this service must be executed in background always. Service request would return with an information message that the service has been successfully initiated. |
| canBeCachedByFields | no | applicable if caching is enabled for the application. Possible values  null – default. Do not cache  “” (empty string) – cache this. Example : countries in the world.  “fieldName” – cache this based on values of field1. Example : stateNames cached by country code. You can specify more than one fields as a comma separated list.  “\_userId” – cache this per user. That is, response is same for a given user, but different for different users. Example : user preferences  Though this is very similar to caching for a value of a field, the caching strategy could be different because of user-sessions.  “\_userId,fieldName,field2,..” – combination of above two cases. That is, this is to be cached per user, but also based on values of one or more field values. |
| className | no | if this service is implemented as a java class. This is the qualified class name that implements org.simplity.ServiceInterface.  Note that other attributes are all irrelevant, except testCases. |

## Input Data

Input specification determines the input data that the client need to provide for this service.

|  |  |  |
| --- | --- | --- |
| Attribute name | Required? | Description |
| inputFilds | no | refer to attributes of Input Field |
| inputRecords | no | refer to attributes of Input Record |
| attachmentFields | no | Applicable if attachment management is enabled for this application. Refer to attachmentAssistsnt in application.xml  comma separated list of field names that are associated with attachments. That is, a field that is designed to carry the key to an attachment that is stored outside. For example resumeId could be a field in the employeeMaster that is the key to the document that is stored outside. Simplity takes care of picking the attachment from temporary storage based on the key in this field, saving it into permenant area using attachmentAssistant, and replacing the key in this field to the new key for tthis permenant storage. |
| attachmentColumns | no | similar to attachmentFields, but meant for columns in data sheets. Comma separated list of sheetName.columnName. |
|  |  |  |

### inputField

|  |  |  |
| --- | --- | --- |
| Attribute name | Required? | Description |
| name | yes | As per your naming convention, but no spaces please |
| dataType | yes | Ensure that you define the right data type for this field, and not use generic ones like text/date. Data type definition is used for validating the data input. |
| isRequired | no | defaults to optional. Used for validating input. |
| defaultValue | no | in case you want to use a default value for a field that is required. Note that this value has no effect if the field is marked as optional. |

### inputRecord

You use an input record to use a record that you have already defined. You have to use a record if you are expecting an array of data – either an array of primitive data, or a table of data (that is, array of records/data-structure)

|  |  |  |
| --- | --- | --- |
| Attribute name | Required? | Description |
| recordName | yes | fully qualified name of record |
| sheetName | no | Name of sheet (table) to which the array of data is to be extracted into. DO not specify this if data is not expected in an array. |
| fieldnames | no | In case you are expecting a subset of the fields of this record. This is a comma separated list of column names. |
| minRows | no | Minimum number of rows expected. |
| maxRows | no | Maximum number of columns. It is certainly a good practice to put a practical limit. |
| dataPurpose | no | Convenient feature to manage input for the same database table, but for different operations like read, update etc..  read: Only key field of this record is expected, and it is mandatory. To be sued if this service is meant to read a key-based row from the table/view associated with this record  filter: If this service is meant to search/filter rows from the underlying table/view. All fields would be optional, irrespective of the specification in the record. Also, each field can have the operator to be used for the value with the field name as <fieldname>Operator. In case the field is number or date, and the operator is between (><) then an additional value is expected in field name <fieldname>To. Default operator is =. <, <=, >, >=, !=, ~(like), and ><(between) are the valid operators. Refer to filter step.  save : insert of update of records. Primary key is ignored for insert if key is to be generated, while it is mandatory for update. Time stamp field, if specified in record, would be mandatory for update operation.  subset: all fields, other than the primary key, are optional. |
| saveActionExpected | no | If purpose is save we may expect a special field called \_saveAction. You may make this field mandatory with this attribute. Default is to decide the action based on presence of key value. |
| parentSheetName | no | Used for receiving hierarchical data. If specified, rows for this record would be present as arrays in each row of the parent sheet with this sheet name as attribute name. <Refer to an example in tutorial> |
| linkColumnInThisSheet | no | Required if parentSheetName is specified. Name of the column in this sheet that has the key value of the parent sheet. |
| linkColumnInParentSheet | no | Required if parentSheetName is specified. Name of the column in the parent sheet that has the key value. |
|  |  |  |

## Output Data

Output specification determines the data to be sent to client as a response to this service request. You may specify fields, sheet, or hierarchical data as output.

Output specification is used only when the service succeeds. Hence we assume that the service context has all the right data to be sent to the client. We do not carry out any validation.

|  |  |  |
| --- | --- | --- |
| Attribute name | Required? | Description |
| fieldnames | no | comma separated list of fields to be returned |
| dataSheets | no | comma separated list of data sheets to be returned |
| outputRecords | no | refer to Output Record |
| attachmentFields | no | refer to same attribute in Input Data. Simplity manages to retrieve the attachment using the key in this field, saves content into temp area, and changes this field value with the key to this temp area file. |
| attachmentColumns | no | similar to attachmentFields, but meant for columns in data sheets. Comma separated list of sheetName.columnName. |
| sessionFields | no | if you need to save/update session fields, use this feature. For example if the user preferences are saved as session fields, and user changes a preference, then that service will have to reset the session field value. |

### outputRecord

Use a record to specify fields, sheet or hierarchical data.

|  |  |  |
| --- | --- | --- |
| Attribute name | Required? | Description |
| recordName | yes | Fully qualified name of record |
| sheetName | no | If omitted, column field of this record are used for outputting fields. If sheet name is specified, a data sheet with this name is expected in the service context. Rows from this sheet are output as a sheet, or part of hierarchical data based on parentSheetName attribute. |
| parentSheetName | no | Name of the sheet of which this sheet is the child. Rows from this sheet are output as child rows of relevant parent row. |
| linkColumnInThisSheet | no | Required if parentSheetName is specified. Name of the column in this sheet that has the key value of the parent sheet. |
| linkColumnInParentSheet | no | Required if parentSheetName is specified. Name of the column in the parent sheet that has the key value. |

### Actions

When you design a service, obviously you have to do something other than receiving data from client and sending data to client. Theoretically, an empty service just copies output from input. Actions implement the logic/design of your service.

Each action does a part of the over-all work. As far as possible, actions are to be designed independent of other actions in the service, just as a service is designed independent of other services in an application. That means, each action may expect some input data in the service context, then do its part of the job, and write some data back into the service context.

Simplity has been extensively used to deliver services that are part of transaction processing system. We have designed several actions that are well suited to carry out part of such transactions. For example the basic read/write operations from RDBMS.

Service has a set of actions that are executed in sequence by default. However, we provide some features to conditionally execute, or iterate a sub-set over rows of a data sheet.

### Common Action Attributes

|  |  |  |
| --- | --- | --- |
| Attribute name | Required? | Description |
| actionName | no | Required only if you use it as a destination of a jump-to action, or you want to use the returned value from this action. If this action returns a value, that value is added to the service context with field name as <actionName>Result. This can be used by subsequent steps.  We assign a default name to every action for our convenience and hence you may find some fields like “\_action1Result” if you inspect content of context. You should not use these in any of your code, because we may change these in our subsequent versions. |
| executeOnCondition | no | An optional Boolean expression. If present, the expression is evaluated at run time and the action is executed only if it evaluates to true. Else the execution proceeds to next action. |
| executeIfRowsInSheet | no | Another way to set a condition for the action to execute. Not valid if executeOnCondition is specified. This action is executed only if the service context has the named sheet, and the sheet has at least one row |
| executeIfNoRowsInSheet | no | Complement of the above. Action is executed if sheet is not found, or there are no rows in the sheet. |

### RDBMS based Actions

Dealing with data base is the core of any transaction processing system. We have designed a set of actions to help you in that. They have the following common attributes (in addition to all the common attributes of action)

|  |  |  |
| --- | --- | --- |
| Attribute name | Required? | Description |
| successMessageName | no | Name of a message to be sent back to client in case the action succeeds. Note that the message itself could be of type success/failure. For example, if you are looking to validate a new mail id, you may do a key-based read. And if the action succeeds, it means that the mail id already exists, and you would send an error message back to client. |
| successMessageParameters | no | Optional comma separated list of field names, whose values are used to format the message mentioned above. Note that the number of parameters and their sequence should match the parameterization design of the message. Refer to message design for more details. |
| failureMessageName | no | Message to use used if the action fails. |
| failureMessageParameters | no | Optional comma separated list of field names, whose values are used to format the message mentioned above. Note that the number of parameters and their sequence should match the parameterization design of the message. Refer to message design for more details. |
| stopIfMessageTypeIsError | no | This action will result in either a success, or a failure. If a message is associated with the result, and the message happens to be of type error, should the execution stop immediately, or should it continue.  Important to remember that the service will ultimately result in “failure” if any message with error type is added at any time. Updates, if any, will be rolled back. This attribute does not affect this behavior. It only specifies whether the execution should stop immediately, or proceed as of now. |

Now let us understand RDBMS based actions first.

### Read Action

Read a row from the underlying table/view based on the primary key value. Columns from this row are set as field values in the service context. Rows from one r more child-tables can also be read as part of this action. This is a handy-feature when you have header and details design. Like when you read on order, you may want to read all its lines.

|  |  |  |
| --- | --- | --- |
| Attribute name | Required? | Description |
| recordName | yes | Fully qualified record name. This record should have field(s) marked as primary key for this action to be used. |
| inputSheetName | no | If this is not specified, primary key is assumed to be a field in service context. If sheet name is specified, then the keys for read operation are taken from sheet and output rows are put into output sheet.  If sheet name is specified, and if sheet is not received, we try and get the key from fields, as if sheet is not specified. |
| outputSheetName | no | If specified, output row or rows are put into this sheet. If not specified, its behavior depends on inputSheetName. If inputSheetName is specified, then outputSheet defaults to default sheet name for the record. Else the columns from the row are copied as fields. |
| childRecords | no | This feature is available with almost all record based operations. Read rows from one or more child tables for each row in this parent row.  A child record has recordName and sheetName as its attributes. sheetName would default to the value determined based on record.  This feature is feasible only if the child record definition takes care of specifying the right parent record. |

If you have a service with just one read action, then you do not have to actually write such a service. Auto service (on-the-fly-service) feature allows a special service name get\_recordName to be used as if you have written a service with a read action using that record. Child records are read based on settings in record specification.

### ReadWithSql Action

Use a sql to read data rows. Refer to sql definition as a building block for details. Rows from one or more child-tables can also be read as part of this action.

|  |  |  |
| --- | --- | --- |
| Attribute name | Required? | Description |
| sqlName | yes | Fully qualified sql name. |
| inputSheetName | no | If this is not specified, primary input fields for sql are assumed to be fields in the service context. If sheet name is specified, then the input fields for sql are columns in the sheet. |
| outputSheetName | no | If specified, output data is copied to a sheet with this name. Else first (or only) row of the output is copied as fields in the service context. |
| childRecords | no | refer to read action. |

### Filter Action

Read rows from the underlying table/view based on the filtering criteria. Rows from one or more child-tables can also be read as part of this action.

Auto-service is available with a service name pattern of filter\_recordName.

|  |  |  |
| --- | --- | --- |
| Attribute name | Required? | Description |
| filteringRecordName | yes | Fully qualified record name. Filtering criteria is sequenced based on the input received for fields in this record. |
| outputRecordName | no | Defaults to filteringRecordName. Record that is used for selecting output columns. Note that if this is different from filteringRecordName, then care should be taken to ensure that the underlying view has columns to satisfy both the records. |
| inputSheetName | no | If this is not specified, input is expected in fields. If specified, we use first row of this sheet as input. We do no filter for each row of this sheet. |
| outputSheetName | no | Defaults to the sheet name associated with outputRecordName. |
| childRecords | no | A child record has recordName and sheetName as its attributes. sheetName would default to the value determined based on record. A child record should have declared one of its columns as a parentField. |

### Save Action

Add, update or delete rows of data based on specification in a record. While adding we handle primary key generation, as well as values for fields like createdBy, created at etc..

Concurrency is an issue while updating a row. We handle it if you declare a time-stamp field in your record (fieldType=”modifiedTimeStamp” and set use useTimestampForConcurrency=”true”)

Auto service feature is available as save\_recordName.

|  |  |  |
| --- | --- | --- |
| Attribute name | Required? | Description |
| recordName | yes | Fully qualified record name. Obviously, this record should be associated with a table, or in some cases, an updatable view. |
| saveAction | no | add, modify, delete or save. Save is the default. Save is valid only if primary key is to be generated. If key is supplied by client we modify that row, else we add a row with a new key. If key is not generated, then you must specify add/modify explicitly. Special field named \_saveAction is reserved for the value of save action. This is also the name of column in case input is in a data sheet. |
| inputSheetName | no | If null, we assume one row is available as fields in service context. If the sheet has multiple rows, they are saved as a batch using jdbc. This method may have performance issues if thousands of rows are to be saved. You should consider alternate designs like ETL for large volume updates. |
| treatSqlErrorAsNoResult | no | At times, you may design an insert operation that will try to insert failing which you may want to update. In such cases, you may get a sql error on key-violation. By default we would raise an exception. You may alter this behavior with this keyword. |
| childRecords | no | You may save rows for one or more child tables as well using child records. If a parent row is added, then obviously all child rows are to be added. If the parent row is modified, you may either add/modify/delete child rows, or replace existing rows with new set of rows. This is possible if the child table is not referred by any other table. That is, primary key of the child table is not a foreign key in any other table. You indicate this design using replaceRows=”true” |

### ExecuteSql Action

If a save action is not convenient for you to design your data operation, you may design your own sql. Refer to sql component specification. Such a sql is executed as a prepared statement using this action.

|  |  |  |
| --- | --- | --- |
| Attribute name | Required? | Description |
| sqlName | yes | Fully qualified name of a sql component. It should be marked as toBeExecuted=”true”. |
| inputSheetName | no | If null, we assume parameters are as fields in service context. If the sheet has multiple rows, they are executed as a batch using jdbc. This method may have performance issues if thousands of rows are to be saved. You should consider alternate designs like ETL for large volume updates. |
| treatSqlErrorAsNoResult | no | At times, you may design an insert operation that will try to insert failing which you may want to update. In such cases, you may get a sql error on key-violation. By default we would raise an exception. You may alter this behavior with this keyword. |

### ExecuteSp Action

If your db operation is more complex, you may write a stored procedure and execute it using this action. Many applications are developed almost entirely in stored procedure on the server side. We recommend that you take a more practical approach and shift as much logic to application layer as possible. We encourage refactoring complex stored procedure into a set of service actions.

|  |  |  |
| --- | --- | --- |
| Attribute name | Required? | Description |
| procedureName | yes | Fully qualified name of a stored procedure component. |
| sheetNameForInputParameters | no | If null, we assume primitive input parameters are to be taken from fields in service context. Note that arrays/data-structures are always expected in data sheets. If the sheet is specified, we take only one row from it, and ignore other rows. We do not execute the stored procedure repeatedly. In case you need such a design, you may use loop action. |
| sheetNameForOutputParameters | no | Primitive output parameters are copied as fields into the service context by default. You may copy them to a row in a sheet by specifying this attribute. |
| outputSheetNames | no | It is possible for a stored procedure to return one or more result sets. If your stored procedure has such a design, then the sheet names can be over-ridden using a comma separated list. Sheet names generally default to the one derived from the record name. |

### KeyValueList Action

This action is designed specifically for server to send list of possible values for a drop-down for a control (input field) on the client. Auto-service feature is available with naming convention list\_recordName. However this action is provided in case you want to send the list as part of another service.

Record must specify listFieldName to the column that is to be displayed by the client, while the primary key would be the internal value. In case the record has list of values grouped by another key, like states for a given country, then the record must also specify listGroupKeyName. And the client has to send the value for this group field with that field name, or in a special field name called \_key.

|  |  |  |
| --- | --- | --- |
| Attribute name | Required? | Description |
| recordName | yes | Qualified name of record that represents the table/view. This record must have specified component. |
| outputSheetName | no | Defaults to sheet name for the record. List of values is returned in this sheet. |

### ReadChildren Action

Action that reads rows from one or more child records based on value of a parent key.

|  |  |  |
| --- | --- | --- |
| Attribute name | Required? | Description |
| recordName | yes | Qualified name of record that represents the table/view. This record must have specified component. |
| outputSheetName | no | Defaults to sheet name for the record. Rows are returned in this sheet. |
| parentKeyFieldName | yes | Name of field in the service context that has the value for parent key. |

### Business Logic

We recommend that the business logic be implemented using java code. A typical business logic should not be dependent on the actual rdbms operations, but should be based on data structure (that may well mimic tables in the rdbms). Such an approach separates database i/o operations from logic. A good business logic expects certain inputs in the service context. It carries out its logic and may

* Raise error/warning/info messages
* Calculate new data elements/rows that have to be saved back to the data base. This data is once again saved in the service context. (Actual rdbs operation would be carried out by subsequent actions)

A java class that implements business logic should implement org.simplity.tp.LogicInterface.

While we recommend that the java class dealing with business logic should not do db i/o, there may be complex situations where the logic may have to be mixed with db i/o. For such cases, Simplity provides an interface named org.simplty.ComplexLogicInterface in which the code can access the current db connection using DbDriver.

### LogicAction

|  |  |  |
| --- | --- | --- |
| Attribute name | Required? | Description |
| className | yes | Fully qualified class name that implements org.simplity.tp.LogicInterface. |

### ComplexLogicAction

|  |  |  |
| --- | --- | --- |
| Attribute name | Required? | Description |
| className | yes | Fully qualified class name that implements org.simplity.tp.ComplexLogicInterface. |

And then there are the other actions that help you in managing errands…

### AddMessage Action

You may add a message to service context using this action.

|  |  |  |
| --- | --- | --- |
| Attribute name | Required? | Description |
| messageName | yes | Name of a message component. |
| parameters | no | If the message is parameterized, you may specify name of fields from which values are to be copied for these. , we assume primitive input parameters are to be taken from fields in service context. |
| stopIfMessageTypeIsError | no | By default, we continue to execute the next action, even if the added message has a severity of error. Of course, the service is considered to have failed at the end if the context has at least one message with severity of error. If the execution has to stop immediately after adding this error message, here is the attribute for you. |

### AddColumn Action

Add a column to a data sheet.

|  |  |  |
| --- | --- | --- |
| Attribute name | Required? | Description |
| sheetName | yes | Name of the data sheet in the service context to which message component. |
| columnName | yes | If this column exists, it is replaced, else it is added. If columnValue or columnValueExpression is specified, then the column is added with values for each row. Otherwise the values would be “unknownValue”. |
| columnValueType | yes | text, integer, decimal, date or boolean. |
| columnValue | no | Value that would be same for all rows. For example columnValue=”25” or columnValue=”y”. If the value is not a constant, but the value of a field at run time, use $. For example columnValue=”$parentId”. |
| columnValueExpression | no | Not valid if columnValue is specified. Should e valid expression that may contain columnNames of this sheet as variables. Expression must evaluate to a value type as mentioned in columnValueType. Expression is avaluated for each row while adding the column. |

### CreateSheet Action

You can create a data sheet and add it to the service context. Data sheet has header row and zero or more data rows. Also, the data rows of a data sheet consistently have values for each column. That is each data row has exactly the same number of cells, and the data type of values of these cells are consistent for that column.

|  |  |  |
| --- | --- | --- |
| Attribute name | Required? | Description |
| sheetName | yes | sheet name to be created/replaced |
| data | yes | semicolon separated list of rows, first row being the column names. Each row is a comma separated list of values. For example data=”col1,col2col3;v1,12,b1;v2,23,b2;v3,35673,bbb” |

### CopyRows Action

Copy all rows from a data sheet to another compatible data sheet. Two data sheets are compatible if they have the same number of columns, and the value types are same. Note that the column names could be different but the value types must be same. Rows are appended.

|  |  |  |
| --- | --- | --- |
| Attribute name | Required? | Description |
| fromSheetName | yes | sheet name to copy from |
| toSheetName | yes | sheet name to copy to |

### CopyUserId Action

UserId is a sensitive field that has implication on privileges etc. You should avoid getting it from client. userId id saved in session on login, and is copied into the service context. You may want to copy this value into any other field you want to deal with. Use fieldname=””yourField” .

|  |  |  |
| --- | --- | --- |
| Attribute name | Required? | Description |
| fieldname | yes | field name to which userId is to be copied to |

### JumpTo Action

Actions are executed in sequence by default. However, you may want to either stop or jump to a different action instead of proceeding to the next action.

|  |  |  |
| --- | --- | --- |
| Attribute | Required? | Description |
| toAction | yes | Name of the action to jump to. Following special names can be used  \_stop : stop execution, and simulate as if the service has completed execution  \_error: stop execution signaling an error, so that any transaction is rolled back  \_break: valid inside a loop block. Execution continues after the loop  \_continue: this loop is stopped and the next loop starts |

### Log Action

Log values of fields and sheets into trace at this point.

|  |  |  |
| --- | --- | --- |
| Attribute | Required? | Description |
| names | yes | Comma separated list of names of fields or data sheets to be logged. Values of these fields and sheets are written to the trace. |

### Loop Action

|  |  |  |
| --- | --- | --- |
| Attribute | Required? | Description |
| sheetName | yes | data sheet to loop on. Actions specified for this loop are executed once for each row in this data sheet. Columns from a row are copied as fields in the service context for each loop. This is broken if there is any JumpTo action that may have any special value, or is jumping to an action outside the block. |
| actions | yes | actions that make up the body of this loop |

### RenameSheet Action

Rename a data sheet in the context

|  |  |  |
| --- | --- | --- |
| Attribute | Required? | Description |
| sheetName | yes | existing sheet name |
| newSheetName | yes | new name for the sheet |

### RowExists Action

Check whether a table/view has a row for a given key. You should provide a name for this action, so that the result of this action would be useful. For example if ationName=”checkCustomer”, then you will have a field named checkCustomerResult with a Boolean value in that.

|  |  |  |
| --- | --- | --- |
| Attribute | Required? | Description |
| recordName | yes | qualified record name |
| fieldName | no | defaults to key field name defined in the record. Field name in the context that has the value for the key field |

### SetValue Action

Set value for a field in service context. Creates a new field or replaces value of an existing field. This is useful if you:

* Know the value of the field at design time. Say you want to set the value to zero. Use fieldValue=”0”. Likewise you may set it to “true”, “false” for Boolean values, “2016-04-23” for date and “abcd” for a text field.
* Want to set it to value of another field. Say you want to set value of a to value of b. Use fieldValue=”$b”
* Value is calculated as an expression. Do not use fieldValue but use expression=””. One of the issue with xml is escaping special characters. With “  
  <setValue fieldName=”a”>  
   <expression>  
   <![CDATA[ b \* c – d + k ]]>

|  |  |  |
| --- | --- | --- |
| Attribute name | Required? | Description |
| fieldName | yes | If this name is present in the service context, it is replaced, else added. |
| fieldValue | no | if the value to be assigned is a constant, or the value of a single field, use this attribute. Example amount=”20” or amount=”$otherAmount”. |
| expression | no | required if fieldValue is not specified. For example expression=”amoaunt – dscountPercent \* amount / 100”. |

### SubService Action

Invoke a service as an action. However, we do not use all aspects of service. For example, when we define a service, we specify input and output. These are not used when you invoke a service as a subservice action. Only the actions within the service are executed. Enclosing service should take care of input and output.

|  |  |  |
| --- | --- | --- |
| Attribute name | Required? | Description |
| serviceName | yes | fully qualified name of service. |

### Suggest Action

Speciffically designed to provide suggested values for a google-suggest-like control on the client. Columns from matching rows are populated into the output sheet, based on a record definition.

In a typical client design, it is quite likely that you have a service specifically to do just this one action. Keeping in our theme of avoiding boiler-plate ode, we provide a pseudo-service/auto-serice/on-the-fly-service for this. A service named suggest\_recordName can be used for this.

|  |  |  |
| --- | --- | --- |
| Attribute | Required? | Description |
| recordName | yes | Fully qualified name of a record component. |
| fieldToMatch | yes | Field name in the record that is to be used for matching with what user has typed so far. For example, it is customerName if user is typing an existing customer name.  By default we use “contains anywhere” for matching. However, you may change that to “starts-with” by sending a Boolean value in a field named \_matchStarting. For example in your json “\_matchStarting”:true. |
| outputSheetName | no | Defaults to sheet name associated with the record. |

## Test Case

We believe that test cases are part of service design and specification. Hence we have made test case part of service.

|  |  |  |
| --- | --- | --- |
| Attribute | Required? | Description |
| testCaseName | no | unique within a service. Readability improves if a name is provided. |
| description | no | for documentation |
| inputJson | no | Data as expected from a client. This is a valid json object that has all the data elements, (fields and possibly data sheets).  Since quotes have to be escaped as “&quot;” specifying a json as an element content or attribute value would be quite error-prone, and unreadable. Hence this attribute is provided as an element and not an attribute. In the element, use CDATA section where you can copy-paste a formatted JSON as it is. For example:  <inputDataAsString>  <![[CDATA]  {  “att1”: “value1”,  “att2”: “value2”,  “dataSheet1”:[  {…….},  {……..},  ]}  ]-> |
| inputFields | no | An input field has fieldSelector and fieldValue. |
| inputSheets | no | An input sheet has   * sheetName - required * columnName – required – comma separated list of column names * data – required - for ease of entry, data is a ‘;’ separated list of rows. Each row is a ‘,’ separated list of values. For example a data sheet with three columns and 2 rows can be specified as  “v1,v2,v3;k1,k2,k3”. Of course, in this case, columnNames would have the names of these three columns.   Note that comma and semicolon cannot be part of your text data. We believe this should not put real restriction on your ability to test the service. If there is a real special case where the service must be tested with values containing comma and semicolon, please use inputDataAsText attribute. |
| outputDataAsString | no | output expected from service in raw form as returned to a client, like JSON. Note that the text is not matched, but the json object is compared for testing. You need not specify the attributes in any specific order.  Once you specify this attribute, other output related attributes are not relevant. |
| nbrErrorsExpected | no | If you are testing for failure, this is the only output related attributes you would specify.  A Service treats validation errors and business errors differently. Typically, a service would like to point out all mistakes in its input. If client sends ten fields, five of which are in error, it is better that the server points out all of them in one response.  However, this ca not be said of business rules. For example the order amount could be above the limit. And there are order lines that may fail other business rules. But the service execution may stop with the first error and return to client with that error. This aspect must be kept in mind while designing test cases expecting number of errors. |
| outputFields | no | if a field name is specified with no value, we assume that the field is expected as an output, but any value would do. If a value is specified, we match that value as well.  we do not have a facility to specify that a field be not output. We believe such a facility is not required in practice. This facility can be added, if required with an attribute named fieldsNotexpected. |
| outputSheets | no | Refer to inputSheets for data related attributes. For output, you may simply provide minRows and maxRows without columnNames and data, if your intent is to just validated number of errors. If your intention is to test the actual data, please use columnNames as well as data. |

# Client APIs

### login(userId, password, successCallbackFn, errorCallbackfn);

Use login function to login to Simplity application. Its use depends on the way you have set up your server deployment.

Simplity recommends that the authentication process be outside of the application. However, this API is provided if your application has to manage its own login. This API makes a call to your server application’s designated login service.

### Logout();

It is a good practice to issue logout() either when the user chooses to do so, or when the browser window is closed. This helps the server optimize its resource better.

Designated logout service on your server app is invoked with this call.

### getResponse(serviceName, jsonText, successFuntion, errorFunction);

This is the main API that is used for invoking services on the server. This uses the URL “a.\_s”. This url is mapped to the common servlet (default is org.simplity.http.Serve).

serviceName is the fully qualified service name like inv.getStokcDetails, or a valid record-based auto-service like filter\_cust.customer.

jsonText is the data you want to send as input for this service. Probably the easiest way is to actually have the javascript object and use JSON.stringify(object) method. Standard object has fields as primitive attributes, and sheets as array attributes.

{“a1”:”v1”, “a2”:20, “s1”:[{“aa1”:”vv1”…},{….}…], “s2”:[…..]…..}

You may use null or empty string if the service expects no data.

successFunction is your call back function. This function is invoked when server responds back a success. An Object instance representing the josn returned by server is passed to this function as a parameter. Response typically contains fields (as attribute-value) and data sheets (as sheetName-array). It also has a special sheet named \_messages that may have non-error messages like info, success or warning. setDataToPage(data, doc) is used as default success function. This function pushes data to your HTML Dom based on simple naming convention. (like a grossly simplified version of angular) This function is described later.

errorFunction is your call back function when there is an error, either while calling the server, or when the server returns. It is called back with an array of error objects. You may use Simplity.showMessages as a default call-back.