# Hello World

## Creating the Project

Let us build a web-application ground-up. Let us use Tomcat as the servlet container.

* Create a new “Dynamic Web Project” in eclipse. Use jre 1.6 or above.
* Using your favorite build process, ensure that simplity-core.jar is accessible. (Simplest thing to do is to put that in WEB-INF/lib folder, and use that as external jar on eclipse build path)
* Configure web.xml to have the following entry.

<servlet>

<servlet-name>s</servlet-name>

<servlet-class>org.simplity.http.Serve</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>s</servlet-name>

<url-pattern>\*.\_s</url-pattern>

</servlet-mapping>

* Copy comp and schema folders of Simplity to WEB-INF folder.
  + Comp is where we will be adding components of our applications.
  + Schema has the grammar that will guide us in creating the components.
  + Add the two .xsd files as reference using
    - windows -> preference -> xml -> xml-catalog -> user specified entries
* Copy simplity folder under WebContent. This has just one .js file that we are going to use on the client.

We are done with project set-up.

## Creating the service

Create a file named helloWorld.xml under comp/service/tp with the following content.

<?xmlversion=*"1.0"*encoding=*"UTF-8"*standalone=*"no"*?>

<service

xmlns=*"http://www.simplity.org/schema"*

name=*"helloWorld"*

>

<actions>

<setValue actionName=*"sayHello"* fieldName=*"hello"*

fieldValue=*"Hello World"*/>

</actions>

<outputData fieldNames=*"hello"* />

</service>

We are done with the service definition.

## Creating the client

Create helloWorld.html with the following content

<!DOCTYPEhtml>

<html><head>

<meta charset=*"ISO-8859-1"*>

<title>Hello World</title>

</head>

<body onload="Simplity.getResponse(‘helloWorld’).then(Simplity.pushdataToPage, Simplity.showMessage);">

<h1 id=*"hello"*></h1>

<script type=*"text/javascript"*src=*"sayHello.js"*></script>

<script type=*"text/javascript"*src=*"js/simplity.js"*></script>

</body>

</html>

Are we done?

Yes we are done.

Not just with the client application, but with our entire HelloWorldApplication

In summary, we created the following artifacts for the application:

1. helloWorld.xml as a service
2. helloWorld.html as the client

What about other dependent jar files? What about Controller? What about action mapping? What about DAO? Entity Bean? Service configuration? ? ? ?

It is not for nothing that we call this Simplity.

Go ahead and say Hello World by running your app under eclipse/tomcat..

# Say Hello World from Java Class

It is but natural that we love our programming language. After all most of our ‘ahaa’ moments that make our day happen when we successfully beat the hell out of a bug that was so elusive. Of course we never have to blame ourselves for allowing that bug to sneak into our code in the first place, because we are all humans.

Here we go

HellowWorld.java

**Package** org.simplity.tutorial;

**import** org.simplity.kernel.value.Value;

**import** org.simplity.service.ServiceContext;

**import** org.simplity.service.tp.LogicInterface;

/\*\*

\* We say "Hello World" by adding a field named hello to context

\* **@author** James Bond

\*/

**Public class** HelloWorldAction **implements** LogicInterface {

@Override

**public** Value execute(ServiceContext ctx) {

Value value = Value.*newTextValue*("Hellooooo World from a Java class that James Bond has written!!");

ctx.setValue("hello", value);

**return** value;

}

}

And change your service to call this java class rather than setting value directly.

<?xmlversion=*"1.0"*encoding=*"UTF-8"*standalone=*"no"*?>

<service

xmlns=*"http://www.simplity.org/schema"*

name=*"helloWorld"* dbAccessType=*"none"*

>

<actions>

<logic

actionName=*"sayHello"*

className=*"org.simplity.tutorial.HelloWorldAction"*/>

</actions>

<outputData fieldNems=”hello” />

</service>

That works fine.

## Hello World – under the hood

1. When you type the URL <Http://......hellowWorld.html>, browser loads this static resource
2. Browser triggers onload function after loading helloWorld.html
3. Code inside simplity/simplity.js makes an ajax call to “/s.\_s”
4. URL ‘a.\_s/ is mapped to Simplitiy’s web agent who gets called from tomcat. Authentication and other security is cleared here, and any session parameters are added to the data received from client.
5. Web-agent in turn calls Simplity’ service-agent with this data. This call could be a simple method invocation, or a message-based call. It all depends on the deployment architecture.
6. Service-agent locates the service and calls it with the data that is sent from client.
7. Data received from client is unpacked and validated as per input specification for this xml. Continues only if input is in order.
8. Actions of the service are executed.
9. Output data is prepared as per output specification of the service.
10. The path is retraced back to the client-agent.
11. Client agent invokes then() which is the default controller.
12. Default controller pushes data to the page based on simple naming convention.

As you can see, Simplity implements a layered architecture internally, but does not require you to know anything about, let alone provide your own code/class for that.

Note that we do not resort to code-generation either. We use a simple automation techniques.

You will get clarity as we take you through more realistic examples.

# Using a Complex Stored Procedure

Let us take an example of a stored procedure. Let us define every possible data type for procedure parameter.

Primitive types:

1. Integer
2. Decimal
3. Text
4. Date
5. Boolean

Object Structure

1. Object (structure)

Array/table of these types. That makes it 12 types of parameters. Let us also define each of these as output, making it exactly two dozen parameters.

Since our objective is to demonstrate how to interact with a SP, we do not do much in the procedure.

Here is the procedure in Oracle.

<Oracle procedure code pasted here>

## Declaring data types and data structures in Simplity

A client page interacts with user to get input fields that are sent to server.

* Server validates all input for data type and mandatory requirements
* If input is ok, server invokes the stored procedure and returns the output data back to client

<we will add the details here>

<we will add more examples>

# 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. |

# Application Configuration

|  |  |  |
| --- | --- | --- |
| Attribute name | Required? | Description |
| applicationId | yes | Unique name within a corporate infrastructure that can be used to identify and authenticate for this application to interact with other applications and services on the network. |
| sendTraceToClient | no | Should the trace/log text accumulated during the execution of a service be sent to the client? Very helpful during development. Developers can quickly figure out issues/bugs without the need to login to server etc.. |
| cacheComponents | no | False during development and true during production. Setting no helps developer to execute/run the latest component without the need to do builds. This reduces the cycle time for modify-test. However this should be set to true in production to avoid the overhead of parsing and loading xmls into objects repeatedly. |
| dbVendor | no | Required if data base is used in the application. oracle, mssql, postgre and mysql are supported readily. It is fairly simple to extend the support to other rdbms systems. |
| connectionString | no | Not recommended, but 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. |
| loginServiceName | no | Service that is fired under two situations:  explicit login request from client:  This service is invoked with loginId and password. Service should do the authentication. If the service succeeds, it should also have value for userId for this loginId. Also, it should set session fields that are to be made available to all subsequent services.  Autologin:  If login/authentication is outside this application (typically centralized like sso) then the login service is triggered before executing any other service. Login service can make use of this feature to set session parameters. |
| logoutServiceName | no | If specified, this service is invoked when the client explicitly requests a logout. Note that this service is not triggered on session expiry. |
| userIdIsNumber | no | User id field is assumed to be text by default. You may change that to number with this option. |
| sessionHelperClassName | no | Java class that implements org.simplity.http.WebSessionHelper interface. Simplity uses an instance of this class to manage session variables. Session management is also associated with login requirements.Simplity provides three default classes. HelperForNoLogin during development if session variables are not used, DefaultHelper that uses a simple session management, and PassiveHelper when this application uses Simplity only as a module, and session is not managed by Simplity. You may write your session manager based on the application design |
| sessionHelperClassName | no | Way to seamlessly work as a module of another application. SessionHelper is used to extract session parameters into Simplity managed data strucrures. |
| cacheManagerClassName | no | Utilize any centralized caching services, or develop a separate utility for caching responses to service request, and wire it with this application. |
| exceptionListenerClassName | no | Way to wire 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. |
| sessionParams | no | Comma separated list of session parameters to be extracted when session for the application is maintained by non-simplity application. |
| userIdNameInSession | no | name of the session parameter that has userId |
| allowServiceSpecificSchema | no | If this application need to work with more than one schema, then the login credentials should set a default schema. Any service that needs to work with a schema other than the default one, it should use schema attribute. Since setting schema is an over-head, we need to enable this feature at the applicaton |

## Hosting under a Web Server

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 a 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. we use these extensions to provide maximum flexibility in deploying the application with different folder structures.

Apart from this, we need an entry for org.simplity.http.SessionListener as a session-listener. Also, the default folder for comp is WEB-INF/comp. In case, for whatever reason, your component folder resides elsewhere, we need a context parameter “comp-folder” that specifies the folder relative to application root.

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

# SQL

SQL is the language that is used to direct the rdbms to manipulate retrieve data, manipulate data, or change database structure itself. This component allows you to design a dynamic sql to e executed at run time, based on run-time values. This component is to be exceuted 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 rows 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. Develepor 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 output parameters. |

Sql parameter is used for both input and output parameters

# SQL

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

|  |  |  |
| --- | --- | --- |
| 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.

|  |  |  |
| --- | --- | --- |
| 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 ProcedureParameter 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. |

# 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 |

### 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 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 implement some business rules.

Messages are organized in one or more files under msg/ folder. Attributes of messages 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. |
| valueListFieldName | 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 valueListFieldName=”countryName” |
| valueListKeyName | no | This attribute is also meant for the case 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. |
| keyGeneratorName | no | For oracle db when key is to be generated. |
| 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. |
| logicalPrimarykeyFields | no | Comma separated list of fields that would be candidate primary key for this table. This is a design feature, but it is not yet used by Simplity for any validation. |

Field

Fields make up a record.

|  |  |  |
| --- | --- | --- |
| 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 this field automatcically if save action is executed using this record.  modifiedTimeStamp – maintains time stamp when row is last modified. Simplity sets value this field automatcically 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 |
| 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 |
| vaueList | 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. |
| fromField | no | refer to fromField. |
| messageName | no | Name of message component to be used in case this field fails validation. |

# 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. However, most features are equally usepecifically for transaction processing systems that use RDBMS as their data.

### 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. |

## Input Specification

Input specification determines the input data that the client need to provide for this service. You may specify one or more inputField as well as one or more inputRecord as your input speificaiton.

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 Specification

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.

Output fields may me specified as a comma separated list of field names. Field values are extracted from service context, and copied to the output with the same name. If you are dealing with output from RDBMS, you may use the record that you might have any ways defined to interact with RDBMS.

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. |

sessionFields

You may have some special situation where your service wants to set some session fields. This is independent of output fields. Use sessionFields attribute for a comma separated list of session field names. These fields are extracted from service context, and passed on to client-agent, in additional any output fields. Note that you may have the same field repeated as output field as well as session field. You have to repeat that in fieldnames=”” as well as sessionFields=””.

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 if you use it as a destination of a jump-to action, or you want to use the returned value from this action. It is a good practice to set short name for all actions anyways. 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. An internal unique name is assigned if name is omitted. |
| 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 |
| executeIfRowsInSheet | 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 al 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. |
| 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. |

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.

|  |  |  |
| --- | --- | --- |
| 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. |
| 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 | 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. |

ReadWithSql Action

Use a sql to read data rows. Refer to sql definition as a building block for details. Rows from one r 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 | 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. |

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.

|  |  |  |
| --- | --- | --- |
| 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 bothe 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=”modifiedAt”)

|  |  |  |
| --- | --- | --- |
| 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. |

ExecuteXp 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. |

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. |

Logic Action

A Java class that implements some logic/algorithm. This class should implement LogicInterface. Service context is the source of data. It is also the place where the output from the algorithm is to be copied to. No db operation is possible in such an algorithm.

|  |  |  |
| --- | --- | --- |
| Attribute name | Required? | Description |
| className | yes | fully qualified name of the class |

ComplexLogic Action

This is similar to Logic Action, except that it can have db operations as well. ComplexLogicInterface is to be implemented by this class. We strongly recommend that you try to use a set of service actions to implement your service design rather than writing a complex logic that involves db operations as well as some business logic. Separation of db operations from logic into separate components improve re-usability as well as maintainability of the service.

|  |  |  |
| --- | --- | --- |
| Attribute name | Required? | Description |
| className | yes | fully qualified name of the class |

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 are 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 |

Suggest Action

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

|  |  |  |
| --- | --- | --- |
| 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 |
| outputSheetName | no | Defaults to sheet name associated with the record. |
| matchType | yes | contains : column value should contain supplied string anywhere  starting: column value should start with the supplied string  userSpecified: decided at run time. Reserved word \_matchStarting can be used by the client as a Boolean to indicate whether to use starting or contains. |

RenameSheet Action

You may just change the name of a data sheet using this action.

|  |  |  |
| --- | --- | --- |
| Attribute name | Required? | Description |
| sheetName | yes | current name of the sheet |
| newSheetName | yes | new name of the sheet |

SetValue Action

Set a field value using an expression that is evaluated at run time. This would replace the value in amount to the discounted amount.

|  |  |  |
| --- | --- | --- |
| 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”. |
| 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. |

Client APIs

login(userId, password);

Use login function to login to Simplity application. This API makes a call to the url “a.\_i” that would be linked to the login-servlet by server set-up. (Default is org.simplity.http.DefaultLogin) On success, a special token is returned by the server. This token is sent as a header attribute with every subsequent request to the server.

It is entirely up to your application set-up as to what happens with this login. For example, during development, you may not even need to login before making any request. If this application is a stand-alone application, it may have to manage its own login, and hence userId and password may be require fields that are used for authenticating the user.

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. The URL invoked is “a.\_o”. This would be mapped to the logout servlet. (Default is org.simplity.http.DefaultLogout)

getResponse(serviceName, jsonData, successFuntion, errorFunction);

This is the main api that is used for invoking services on the server. This uses the URL “a.\_s”. This is 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.

jsonData 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 that is invoked with an object as the only parameter. This object has the same structure as in input. It has an additional array named “\_messages” that may have non-error messages like info, success or warning. Simplity.setDataToPage is a valid function for this. It is a very simple utility to push data to your html page based on some naming convention. Refer to setDataToPage API for details.

errorFunction is your cal 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 till you come up with a fancy way of doing the same.