

Agiloft Reference Manual

CONTENTS

1. Developer Guide	4
1.1 SOAP and REST API Web Services	5
1.1.1 API Security	7
1.1.2 SOAP API Setup	11
1.1.2.1 SOAP API Call Basics	14
1.1.2.2 Test the SOAP Interface	17
1.1.2.3 EWAttachFromSOAPAttachment	33
1.1.2.4 EWCreate	37
1.1.2.5 EWCreateAndRead	42
1.1.2.6 EWDelete	48
1.1.2.7 EWGetChoiceLineId	53
1.1.2.8 EWLogin	57
1.1.2.9 EWLogout	60
1.1.2.10 EWRead	62
1.1.2.11 EWRemoveAttached	66
1.1.2.12 EWRetrieveAttachedAsSOAPAttachment	70
1.1.2.13 EWSearchTable	74
1.1.2.14 EWSearchTablePaginated	79
1.1.2.15 EWSearchTableWithQuery	85
1.1.2.16 EWSearchTableWithQueryPaginated	91
1.1.2.17 EWSelectAndRead	98
1.1.2.18 EWSelectFromTable	103
1.1.2.19 EWUpdate	106
1.1.2.20 SOAP Examples and Hints	110
1.1.3 REST Interface	112
1.1.3.1 REST - Create	120
1.1.3.2 REST - Read	122
1.1.3.3 REST - Update	124
1.1.3.4 REST - Delete	126
1.1.3.5 REST - Select	128
1.1.3.6 REST - Search	131
1.1.3.7 REST - GetChoiceLineId	136
1.1.3.8 REST - Attach	138
1.1.3.9 REST - Remove Attachment	140
1.1.3.10 REST - Retrieve Attachment	141
1.1.3.11 REST - Login	143
1.1.3.12 REST - Action Button	146
1.1.3.13 REST - Async Status	147
1.1.3.14 REST - Table	148
1.1.3.15 REST - Saved Search	150
1.1.4 Using OAuth2 to Access REST API	152
1.2 ESA Developer Guide	160
1.2.1 ESA XML Use	167
1.2.2 ESA Interface Reference	183
1.2.3 Calls To and From the ESA	210
1.2.4 Building a Custom ESA	226
1.2.5 ESA HelperAPI Interface	233
1.2.6 Using the Sample ESA	238
1.2.7 HTTPS and Command Line ESA Types	250
1.2.8 Java ESA Development	259
1.3 Database Interface	285
1.3.1 Reviewing your Configuration File	286
1.3.2 Create Read Only MySQL Users	288
1.3.3 Identify the Database Listening Port	290
1.3.4 Configure ODBC Connection	291
1.3.5 Query Tables	292
1.3.6 Encrypt and Decrypt Database Passwords	294

1.3.7 Optimize MSSQL for Development or Test Environments	295
1.4 Custom Scripts	297
1.4.1 Script Operations Flow	298
1.4.2 Linked Fields in Scripts	299
1.4.3 Input File Example	300
1.4.4 Stringifiers Description	301
1.4.5 Perl based Scripts	304
1.4.5.1 Perl API	305
1.4.5.2 EWget Module	306
1.4.5.3 EWset Module	312
1.4.5.4 Perl Scripts and External Programs	316
1.4.5.5 Perl Sample Scripts	335
1.4.6 Java Scripts	340
1.4.6.1 Sample Java Scripts	345
1.4.7 Python Scripts	367
1.4.7.1 ALget Common Use	369
1.4.7.2 ALset Common Use	372
1.4.7.3 ALperf Common Use	375
1.4.7.4 Sample Python Scripts	378
1.4.7.5 Script Structure and Basics	383
1.4.8 Reset Record ID Numbers	384

Developer Guide

The topics in this section will assist developers who want to use custom code and scripting within Agiloft, or to use the SOAP and REST APIs.

- SOAP and REST API Web Services
 - SOAP API Setup
 - REST Interface
- ESA Developer Guide
- Database Interface
- Custom Scripts

SOAP and REST API Web Services

Agiloft offers a SOAP and RESTful API which can be used to perform common system tasks on records. SOAP Web Services are enabled per-knowledgebase in the setup menu, while REST services are on by default and are only restricted by group permissions and license types.

API access requires you to have an Enterprise or Web Service license. Although it is possible to start Web Services in the Setup menu, you will receive an error message if you attempt to access the APIs without the correct license. For more information on upgrading your license, see [Licenses](#).

 The SOAP/REST interface is most suited for online integration between systems, rather than processes that involve bulk synchronization. There are more suitable ways of synchronizing many records with Agiloft. These include:

- Pushing Changes to Production
- Importing Record Data
- Microsoft Exchange Synchronization
- External System Adapter
- System integrations such as Salesforce and SCCM

Sample API Code

To generate sample Web Services code in Java or .NET for any table, go to **Setup [Table]**, go to the API tab, and click Download Java Sample or Download .NET Sample. The sample file contains a set of table-specific code samples for each of the API functions.

Create, Read, Update Delete and Select operations on this table

Download Java Sample

Download .NET Sample

This tab also offers a Download HTML option that generates HTML code you can use to embed a form with a Submit button, allowing the selected user to create a record in the table. Simply select the appropriate user, choose the layout to use to display the form, and click Download HTML.

Name to use for the record creator:

☒ guest

☐ User: 

Lay out fields using

End User Layout 

[Download HTML](#)

Types of Information Provided by the APIs

The REST/SOAP API is primarily used to access and modify table records. The API can't be used to modify other system data, like table names, saved searches, table columns, or permissions. For system access to these elements, you can use the Database Interface.

API Security

In **Setup > System > Manage Web Services**, there are some options to configure the security of SOAP and REST Web Services. At the top of the page, you can Enable WS if they aren't activated, or Disable or Re-enable WS if they are currently enabled.

Group Permissions and IP Restrictions

The next section controls group permissions and IP address restrictions for SOAP, followed by the same settings for REST. First, you can select the groups that are permitted to use the interface. If a user does not belong to a group that is selected in these drop-down lists, they are restricted with a 403 access error, even if they have create /edit/delete permissions on the affected records.

Four **global variables** allow you to define a set of IP addresses for SOAP and REST blacklisting and whitelisting:

- Security: REST IP Blacklist
- Security: REST IP Whitelist
- Security: SOAP IP Blacklist
- Security: SOAP IP Whitelist

IP addresses should be comma separated, and address ranges should be indicated with a dash.

Example

10.168.6.102, 10.169.7.132-10.169.7.150

REST Authentication

The REST client application must log in using valid credentials for a knowledgebase. Agiloft verifies these credentials and if valid creates an HTTP session that the client can use in subsequent calls.

Every REST call should contain the user's credentials in the form `login={login}&password={password}`.

Passing Credentials in the Parameter Body with POST

If you are in a production system and do not wish to expose user credentials as plain text, you can avoid passing the login or password in REST calls by using POST instead of GET to pass the parameters in the request body. POST requests support the /ewws/EWRead, /ewws/EWSelect, /ewws/EWCreate, /ewws/EWUpdate, /ewws/EWDelete methods. For more information on constructing a REST request via POST, see [URL Conventions](#).

JSON Web Tokens

You can use JSON Web Tokens (JWT) to retrieve data using an access token instead of using login credentials each time. For more information, see [REST - JSON Web Tokens](#).

Access Permissions

Call permissions are based on the authorized user's permissions in the Agiloft KB, so it will return only the data that the API user has access to. Please note that in addition to the user's group permissions, it is necessary to grant REST access to the group via **Setup > System > Manage Web Services > Groups allowed for REST**. Any operation that the user doesn't have access to will be rejected.

Encryption

SOAP and REST calls can be encrypted by system admins. For example, consider a call to read data from a contract record: `http://[localhost]/ewws/EWRead?$kb=KBNAME&$table=contract`

Omit the variables likely to change, such as the ID value, from the string.

1. Go to **Setup > Access > Automatic Login Hotlinks**.

2. Enter the string constructed above into the input box, select an expiration time, and click Encrypt.

Encrypt Hotlink

[https://myurl.com/ewws/EWRead?\\$KB=Demo&\\$Stable=Contracts&\\$id=450&\\$login=user&\\$password=pass&\\$lang=en&id=44450](https://myurl.com/ewws/EWRead?$KB=Demo&$Stable=Contracts&$id=450&$login=user&$password=pass&$lang=en&id=44450)

Expiration: 1 Hours

Encrypted hotlink:

[https://myurl.com/ewws/EWRead?\\$genhotlink=y7v+JbCXlxWtdp6oniK+NXkULZFbWID9N9IuVj88h5gZ7mxZay+2alw+3Lbq8SKRJ07fCW30HTeROtwzAl2xB8xvGwzBSvEk4FHgT/YWGgs5bmrqdTKvl/xRovfkfepLsrheJEAwRt5OfVGKjQHU+wpwIUOo9AWzW1zEuD81fGCSubtx4G6Cpe5eWgMLOCFIw6TJcDxNbbKN5yJUbwzeGVRjBchiBEfUJG1UINOIuogqzWR4N+Fok9BaCHS2dot5PeVhas0J7jCfvB6HmMQSkOD9](https://myurl.com/ewws/EWRead?$genhotlink=y7v+JbCXlxWtdp6oniK+NXkULZFbWID9N9IuVj88h5gZ7mxZay+2alw+3Lbq8SKRJ07fCW30HTeROtwzAl2xB8xvGwzBSvEk4FHgT/YWGgs5bmrqdTKvl/xRovfkfepLsrheJEAwRt5OfVGKjQHU+wpwIUOo9AWzW1zEuD81fGCSubtx4G6Cpe5eWgMLOCFIw6TJcDxNbbKN5yJUbwzeGVRjBchiBEfUJG1UINOIuogqzWR4N+Fok9BaCHS2dot5PeVhas0J7jCfvB6HmMQSkOD9)

Encrypt

3. The encrypted URL can be used in the same way as the standard URL, and parameters can be added to it in the standard way.



Example

{encrypted URL}&\$lang=en&id=450

The system trusts the encrypted URL, so no password is required.

Statefulness

Though stateless in appearance Agiloft Web Services are stateful by nature. A certain set of information can be shared as a "session" between multiple calls and there is usually no good reason to re-create it on every call.

A typical integration scenario obeys the pattern of "login, do multiple calls, logout".

The statefulness is supported via a widely used pattern of obtaining a session token first via the EWLogin method and explicitly passing it with each subsequent call performed.

Once the session is finished it is advisable to destroy the session with a call to EWLogout to ensure proper freeing of server resources.

For more information, see <https://nordicapis.com/defining-stateful-vs-stateless-web-services/>.

Failed Logins

The SOAP API handles a series of failed logins in the same fashion as the GUI, i.e. it will produce an error if the account becomes blocked due to multiple unsuccessful login attempts.

SSL Support

The Agiloft SOAP API relies on the underlying application server and front-end web server (if present) for SSL support.

SOAP API Setup

The SOAP-based Web Services API in Agiloft is enabled on a per-knowledgebase basis, and uses a token obtained at login to identify the session on subsequent calls.

Enable SOAP Web Services in a Knowledgebase

1. Click the **Setup** gear in the top-right corner and go to **System > Manage Web Services**.
2. By default, the text reads: SOAP Web Services are currently Off
3. Click Enable WS. The screen refreshes, and the text reads: SOAP Web Services are currently On
4. At this point, SOAP Web Services is on, the WSDL is automatically generated for the knowledgebase, and you will be able to access the knowledgebase via the WSDL. See [Obtain the WSDL Document for your Knowledgebase](#).
5. Once enabled, the endpoint for the SOAP Web Service for knowledgebase ABC will become available at:
`https://SERVER/ewws/KBNAME/EWSv2Service?wsdl`
 - a. You can access the WSDL document for the purpose of generation of the client code at the following address: `https://SERVER/ewws/KBNAME/EWSv2Service?wsdl`.
 - b. Note that by default for security reasons the WSDL document uses `localhost` as the endpoint. You can either override this immediately after obtaining the WSDL or override the endpoint location in the client code.
6. The Web Services package will persist when the application server restarts, and will be regenerated by the installer when the server is updated.

If any changes are made to the knowledgebase which should also be reflected in the WSDL, re-enable Web Services then refresh the knowledgebase.

SOAP Web Services WSDL

The WSDL defines a set of generic operations common to all knowledgebases, a set of data types specific to the current knowledgebase, and a set of table-specific variants of the generic operations. Each table in the knowledgebase when the WSDL is generated is represented as a complex type definition.

For example

- The `EWSelectandRead` operation is generic and enables you to retrieve information from the knowledgebase.

- The `EWSelectandRead_WSCase` operation is specific to the Support Cases table and contains all of the data types that are specific to that table.

Table types extend a common `EWWSBaseUserObject` type. The generic methods use this common ancestor for their arguments and result types where relevant. When a generic method is invoked for a specific table, it accepts the table name as a string parameter and returns the corresponding specific descendant type as the result. The code in the client must perform the necessary typecasting, if required. The name of the table is not used, as it can be deduced from the invoked method.

Import the WSDL into your Development Environment

This section provides sample instructions for Apache Axis. For instructions about other development platforms, see your platform's product documentation. *Agiloft* SOAP Web Services follow industry standards and have been tested to work with Java, .NET, PHP, Perl and Python client applications.

Once you have the WSDL file, you need to import it into your development environment to generate the necessary objects for building client Web service applications. For detailed instructions on testing the WSDL, see: [Test the SOAP Interface](#).

Import the WSDL into Apache Axis

Java environments access the API through Java objects that serve as proxies for their server-side counterparts. Before using the API, you must first generate these objects from your organization's WSDL file. Each SOAP client has its own tool for this process. For Apache Axis, use the `WSDL2Java` utility.



Before you run `WSDL2java`, you must have Axis installed on your system and all of its component JAR files must be referenced in your classpath.

The basic syntax for `WSDL2Java` is `java -classpath pathToJAR/fileName org.apache.axis.wsdl.WSDL2Java`

`-a pathToWsdldoc`

The `-a` switch generates code for all elements, referenced or not, which may be necessary depending on your WSDL.

If you have JAR files in more than one location, list them with a semicolon separating the files. For example, if the Axis JAR files are installed in `C:\axis-1_3`, and the WSDL is named `MyKB.wsdl` and is stored in `C:\myKB`:

```
java -classpath c:\axis-1_3\lib\axis.jar;c:\axis-1_3\lib\axis-ant.jar;  
c:\axis-1_3\lib\axis-schema.jar;c:\axis-1_3\lib\commons-discovery-0.2.jar;  
c:\axis-1_3\lib\commons-logging-1.0.4.jar;c:\axis-1_3\lib\jaxrpc.jar;  
c:\axis-1_3\lib\log4j-1.2.8.jar;c:\axis-1_3\lib\saaaj.jar;  
c:\axis-1_3\lib\wsdl4j-1.5.2.jar; org.apache.axis.wsdl.WSDL2Java  
-a C:\myKB\MyKB.wsdl
```

This command will generate a set of folders and Java source code files in the same directory in which it was run. After these files are compiled, they can be included in your Java programs for use in creating client applications.

For most Java development environments, you can use wizard-based tools for this process instead of the command line.

For more information about using WSDL2Java, see <http://ws.apache.org/axis/java/reference.html>.

SOAP API Call Basics

The SOAP API calls implement specific knowledgebase operations that your client applications can invoke at runtime to perform tasks, such as:

- Search and read data in your knowledgebase.
- Add, update, and delete data in your knowledgebase.

Using your development environment, you can construct Web Service (WS) client applications that use the standard WS protocols to:

- Log in to the knowledgebase and receive the session token to be used for subsequent calls.
- Perform SQL-based searches across the knowledgebase data.
- Create, read, update, and delete data.
- Trigger workflow transitions and business rules processing.

Characteristics of API Calls

All SOAP API calls are:

- Service requests and responses - your client application prepares and submits a service request to the server via the API, the agiloft.com Web Service processes the request and returns a response, and the client application handles the response.
- Synchronous - once the API call is invoked, your client application waits until it receives a response from the service. Asynchronous calls are not supported.
- Committed automatically - every operation that writes to an agiloft.com object is committed automatically. This is analogous to the AUTOCOMMIT setting in SQL. Each API call is done in one transaction.
- Followed by a delay inserted after the operation has completed. The delay is set to one second by default and is configurable via the global variable [Web Services Delay \(WSDelay\)](#).
 - An operation on a record may invoke rules and other functions, which need to be allowed enough time and resources to complete.
 - Additionally, the delay is needed because client applications could mistakenly use web services, resulting in a flood of requests.

Factors that Determine Data Access

When using the API, the following factors determine access to your organization's data:

- Your knowledgebase has to have **SOAP API access enabled**.
- The generated WSDL lists all data structures and fields that existed in the knowledgebase at the time of generation. The generated WSDL file contains all of the objects that are available in your knowledgebase. Via the API, a client application can access objects that are defined in your WSDL file.
- The generated WSDL is user-agnostic, but when invoking a method in the SOAP Interface with credentials of a certain user, the runtime permissions of that user come into play. This may result in some fields appearing empty when the user doesn't have read permission, for example.

Run-time access via the SOAP API is governed by the same access permissions as the GUI. Certain operations on the tables and fields can be performed only if the combined permissions in the logged-in user's group list permit such access.

For example, values for the fields that are not visible to a given user are not returned in the results of the query. Another often reported issue is when one is not able to write a structure, retrieved in the very previous call, back to the server simply because the user has the necessary read permissions, but not the write ones.

Please refer to the Groups section in your User Manual or Online Help for full details regarding permissions.

- As such, the API respects table-level permissions configured for the given user group in the knowledgebase.
- Further, the API fully respects the ownership concept and related table-level and field-level permissions configured in the knowledgebase.
- Ownership for a record is determined in runtime at the time of the call and if modified in the call itself, comes into effect immediately upon the completion of the API call.
- Ownership changes to one object instance do not normally automatically cascade to other object instances unless specifically configured so via a Linked Field with changes propagation. For example, if ownership changes for a given Account, ownership does not then automatically change for any Contract associated with that Account - each ownership change must be made separately and explicitly by the client application.
- The configuration of the workflow for a given knowledgebase may influence the ability to create or delete a record in a certain state or to change the state of a record.
- Business rules, especially validation ones, may affect the success of create and modification operations.
- Success also depends on whether a particular change would compromise the referential integrity of your knowledgebase. For example, the data used in Linked Fields sets is validated in create and update calls. In a similar fashion the delete calls are handled in a way to ensure the integrity of the data is maintained and will fail otherwise.
- Certain features that affect the [agiloft.com](https://www.agiloft.com) user interface are not accessible or implicitly enforced via the API. For example:

- Layouts define whether a given field is shown or hidden, but the API does not enforce such layout-specific field restrictions or validations in create and update calls. It is up to the client application to enforce any such constraints, if applicable.
- Record types can control which layouts users with different profiles can see. However, via API all fields available for the table are presented.
- Dependent choices and conditional visibility are not enforced by the API.

If any such constraints are required, it is up to the logic in the client application to enforce them explicitly.

Error Handling

In the event of an error the API calls return SOAP fault messages with additional information. There are currently six types of faults distinguished by the SOAP API:

- `EWIntegrityException`

Signals that the knowledgebase setup has become incompatible with the client application logic/expectations.

- `EWOperationException`

Signals that the operation may not be performed because of some dependencies between agiloft.com functions. This message should be considered in the context of the operation. It may signal a permanent condition, for example workflow forbidding transition from one state to another; or a temporary one, for example the record is locked by another user from another GUI or API session.

- `EWPermissionException`

Signals that the username used to trigger the API call lacks sufficient privileges to perform certain operations.

- `EWSessionException`

Signals that client session has expired or has been removed.

- `EWUnexpectedException`

Signals that an unknown, not handled and unexpected exception has happened on the server side. These exceptions should be reported to the vendor for further investigation. The message contains a token that helps to trace the root cause of the problem.

- `EWWrongDataException`

Signals that data passed by client is wrong in the context of the operation.

Test the SOAP Interface

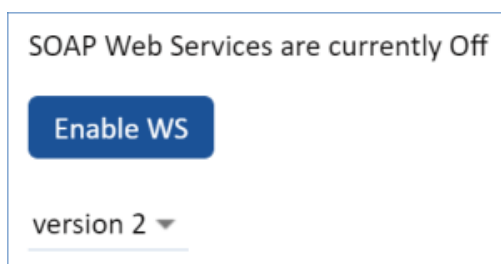
This topic explains how to establish the connection to the SOAP interface via the WSDL, and test several of the available requests. Ideally these tests should be performed on a 'clean' knowledgebase, prior to setting up complicating factors such as WS-Security. The instructions in this topic use SOAPUI as a reference point, but you can use another tool and adapt the instructions if you prefer.

Prerequisites

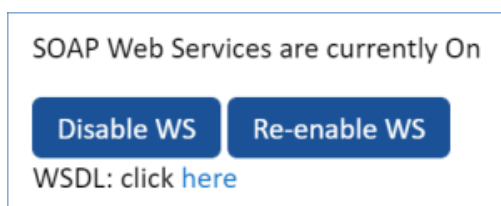
- You must have the admin login details for your knowledgebase.
- The knowledgebase must have an Enterprise license. See [Licensing](#) for more information.
- Download and install SoapUI from here: <https://www.soapui.org/downloads/soapui.html>. The Open Source edition works for the testing described here.

Enable Web Services in the KB

1. In the knowledgebase, click the **Setup** gear in the top-right corner and go to **System > Manage Web Services**.
2. Select version 2 of SOAP and click Enable WS. Wait for web services to be enabled, and the wizard to refresh.



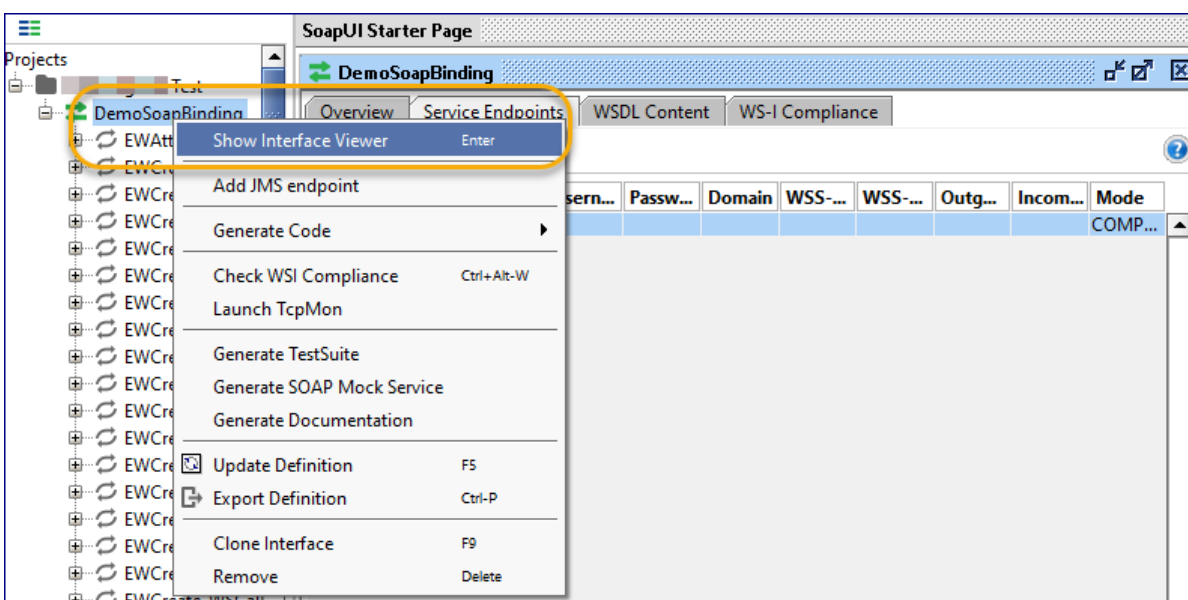
3. The Enable WS button is replaced by two new buttons, and the message now says SOAP Web Services are on.



4. Make sure that only the admin group has access to the SOAP and REST services.
5. Click the link to the WSDL, which opens a new tab in the URL format: `http://<SERVER>/ewws/<KBNAME>/EWSserviceAPIv2?wsdl`

Add the WSDL to SOAPUI

1. In SOAPUI, click the SOAP icon to create a new SOAP project.
2. Enter a name for the project, and in the Initial WSDL field paste the URL to the Agiloft WSDL.
3. Ensure that Create sample requests for all operations? is selected, and click OK. This creates the project, with all of the available SOAP calls listed.
4. Expand the first level to see the full project menu, right click DemoSoapBinding, and select Show Interface Viewer.



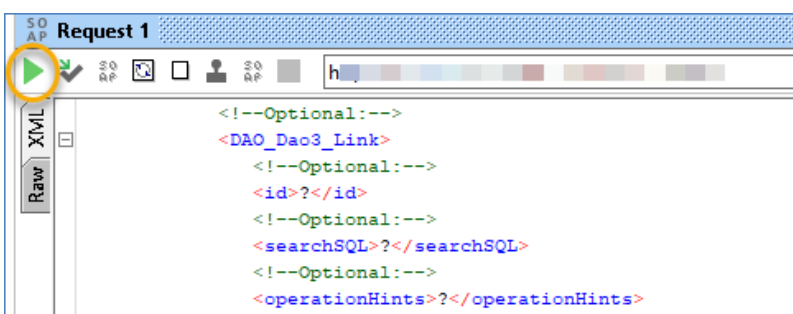
5. Open the Service Endpoints tab of the interface viewer, and check the details of the WSDL URL. It should be formed as above. Select the endpoint and click Assign.
6. In the Assign Endpoints dialog, select All Requests from the drop-down list, and click OK.
7. Exit the interface viewer.

Test the SOAP Calls

At this point, you can test the calls from SOAP to the knowledgebase. To test a call from the list:

1. Expand it to see the request, then double click the request.
2. Replace the existing parameters - which are written in the format `<ParameterName>?</ParameterName>` - with the relevant values from the knowledgebase

- Click the green arrow to run the request.



Follow the example test cases below to test some of the available requests. Many of the examples below use the Support Cases table (WSCase) to test the requests.

EWLogin

- Add the following values to the EWLogin request:

```
<dem:EWLogin>
    <String_1>KBNAME</String_1>
    <String_2>admin</String_2>
    <String_3>admin_password</String_3>
    <String_4>en</String_4>
</dem:EWLogin>
```

- Click Run. This returns something similar to the following response, which includes a session string between the <result> tags:

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Header/>
  <env:Body>
    <demo:EWLoginResponse xmlns:demo="http://Demo.api.ws.enterprisewizard.com">
      <result>78658161687694</result>
    </demo:EWLoginResponse>
  </env:Body>
</env:Envelope>
```

- Change the password in string_3 to an incorrect value and click Run.
 - An error message similar to the following appears: <message>[http-0.0.0.0-80-1] [1618616486d] login/password combination admin/***** for knowledgebase KB1</message>
- Set the username and password to an existing, non-admin user and click Run.

- a. An error message similar to the following appears:

```
<ns1:EWPermissionException xmlns:ns1="http://api.ws.enterprisewizard.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <message>not allowed, please check logs</message>
</ns1:EWPermissionException>
```

5. Enter the correct admin username and password again and click Run.
 - a. Write down the session string, which will be used in later tests.

EWRead

1. Add the following values to the EWRead_WSCase request and click Run:

```
<dem:EWRead_WSCase>
  <sessionId>SESSIONID_FROM_LOGIN_RESPONSE</sessionId>
  <recordID>0</recordID>
</dem:EWRead_WSCase>
```

- a. An error message similar to the following appears: <message>[http-0.0.0.0-80-6]
[1503916816216] no data found for 0</message>
2. Replace the recordID with an existing Support Case record in the KB.
 - a. The response is an XML string with the details of the Support Case record.

EWSelectFromTable

1. In the Support Cases table, create a new record and assign it to the system user 'Ralph Knowles'; or if the user does not exist, assign them to another user and replace String_3 below with their name.
 - a. Add the following values to the EWSelectFromTable request and click Run:

```
<dem:EWSelectFromTable>
  <String_1>SESSIONID_FROM_LOGIN_RESPONSE</String_1>
  <String_2>case</String_2>
  <String_3>assigned_person='Ralph Knowles'</String_3>
</dem:EWSelectFromTable>
```

- b. The response returns the ID of the Support Case, as well as any other Support Case records assigned to that user.

2. Modify the string_3 parameter with a SQL query, which returns the record with the highest ID from the table. Note that the [DB_TABLE_NAME] can be found in the Table Name field of the General tab of the [Table wizard](#). Click Run.
 - a. The response returns the ID of the highest record number for the Support Case table.

EWSelectAndRead

1. Add the following values to the EWSelectAndRead_WSCase request and click Run:

```
<dem:EWSelectAndRead_WSCase>
  <sessionId>SESSIONID_FROM_LOGIN_RESPONSE</sessionId>
  <where>assigned_person='Ralph Knowles'</where>
</dem:EWSelectAndRead_WSCase>
```

- a. The response returns the details of any records assigned to user Ralph Knowles.
2. Modify the <where> condition so that it finds only one record; for example:


```
<where>assigned_person='Ralph Knowles' and wfstate=1</where>
```

, and click Run.
 - a. The response contains the full data of that record.

EWCreate

1. Open the EWCreate_WSCase request.
2. Remove all of the field tags apart from problem_Description and summary, and add the parameter values in this way, then click Run:

```
<dem:EWCreate_WSCase>
  <sessionId>SESSIONID_FROM_LOGIN_RESPONSE</sessionId>
  <WSCase>
    <problem_Description>Test ticket for SOAP<
  /problem_Description>
    <summary>SOAP test create</summary>
  </WSCase>
</dem:EWCreate_WSCase>
```

3. This request creates a very basic new record in the Support Cases table. The response gives the record ID in this way:

```
<result>361</result>
```

.
4. In the KB, locate the new record to verify that it was created successfully.

EWCreateAndRead

1. Open the EWCreateAndRead_WSCase request.
2. Remove all of the field tags inside <WSCase> apart from problem_Description and summary, and add the parameter values in this way, then click Run:

```
<dem:EWCreateAndRead_WSCase>
  <sessionId>SESSIONID_FROM_LOGIN_RESPONSE</sessionId>
    <WSCase>
      <problem_Description>Test ticket 2 for SOAP<
/problem_Description>
      <summary>SOAP test create and read</summary>
    </WSCase>
  </dem:EWCreateAndRead_WSCase>
```

3. This request creates a basic new record in the Support Cases table. The response gives the full data for the new record.
4. In the KB, locate the new record to verify that it was created successfully.

EWUpdate

1. Open the EWUpdate_WSCase request.
2. Remove all of the field tags apart from ID and Priority and add the parameter values in this way, then click Run:

```
<dem:EWUpdate_WSCase>
  <sessionId>SESSIONID_FROM_LOGIN_RESPONSE</sessionId>
  <WSCase>
    <id>RECORD_ID_FOR_UPDATE</id>
    <priority>OPTION_CRITICAL</priority>
  </WSCase>
</dem:EWUpdate_WSCase>
```

3. This request changes the priority of the Support Case record to Critical. The response gives the full data for the updated record.
4. In the KB, locate the record and check that its priority was changed.

EWDelete

1. Add the following values to the EWDelete request and click Run:

```
<dem:EWDelete>
  <String_1>SESSIONID_FROM_LOGIN_RESPONSE</String_1>
  <String_2>case</String_2>
  <arrayOfLong_3>
    <value>RECORD_ID_FOR_DELETE</value>
  </arrayOfLong_3>
  <DeleteRule_4>APPLY_DELETE_WHERE_POSSIBLE</DeleteRule_4>
  <arrayOfLong_5>
  </arrayOfLong_5>
</dem:EWDelete>
```

2. This request deletes the record number in the KB. The response shows no record data.
3. In the KB, search for the record and confirm that it was deleted.
4. Execute the same request with the identical settings and check that the response is similar to the following:


```
<message>[http-0.0.0.0-80-5][1504006477104] Operation cannot be done. Not all
records can be removed.</message>
```

EWGetChoiceLineID

1. Add the following values to the EWGetChoiceLineID request and click Run:

```
<dem:EWGetChoiceLineId>
  <String_1>SESSIONID_FROM_LOGIN_RESPONSE</String_1>
  <String_2>case</String_2>
  <String_3>priority</String_3>
  <String_4>Critical</String_4>
</dem:EWGetChoiceLineId>
```

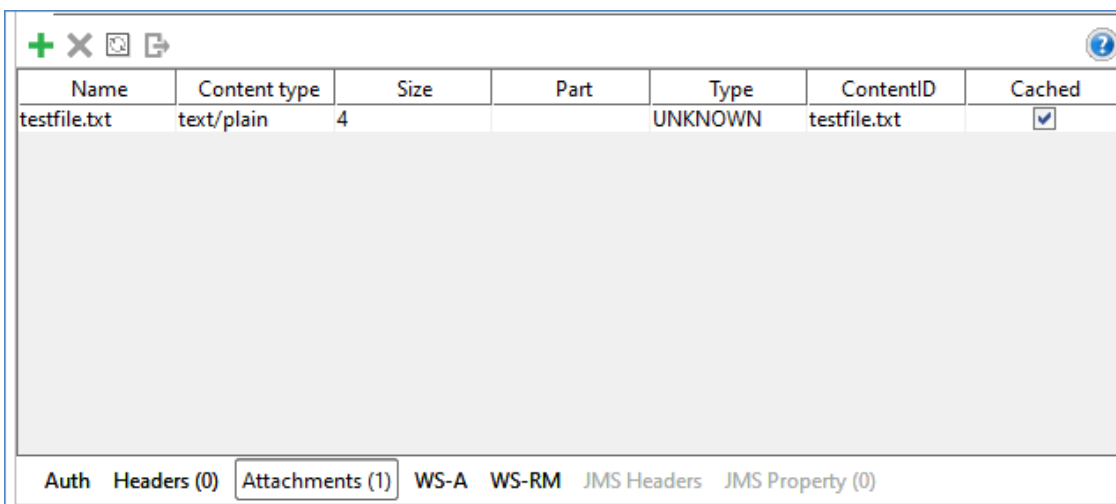
2. This request returns the line number in the Priority choice list for the value Critical. The response is a value like `<result>6</result>`.
3. Change the `<String_4>` value to a non-existent priority and click Run.
 - a. The response is similar to the following message: `<message>[http-0.0.0.0-80-4][1504007797277] No choice line found for value Defcon 6</message>`
4. Change the `<String_3>` value to some non-choice field like 'summary' and click Run.
 - a. The response is similar to the following: `<message>[http-0.0.0.0-80-4][1504007877613] unexpected exception: null</message>`.

EWAttachFromSOAPAttachment

1. Create a txt file with the file name `testfile`, and save it to a location on your system.
2. Add the following values to the `EWAttachFromSOAPAttachment` request, replacing the `<key>` value with an existing Support Case record number:

```
<dem:EWAttachFromSOAPAttachment>
  <sessionId>SESSIONID_FROM_LOGIN_RESPONSE</sessionId>
  <tableName>case</tableName>
  <key>419</key>
  <fieldName>downloadable_files</fieldName>
  <fileName>testfile.txt</fileName>
</dem:EWAttachFromSOAPAttachment>
```

3. Click Attachments below the request window, then click + and select the `filename.txt` file from your system.



Name	Content type	Size	Part	Type	ContentID	Cached
testfile.txt	text/plain	4		UNKNOWN	testfile.txt	☒

Auth Headers (0) Attachments (1) WS-A WS-RM JMS Headers JMS Property (0)

4. Click Run. The request attaches the file to the record in the Support Case table. The response is similar to `<result>1</result>`.
5. In the KB, open the record and check that the file has been attached to the Downloadable Files field.



6. Create a new file and execute the same request again with it. Check that the result has been incremented to `<result>2</result>`.
 - a. In the KB, open the record and check that the new file was added to the Downloadable Files field.

EWRetrieveAttachedAsSOAPAttachment

1. Add the following values to the EWRetrieveAttachedAsSOAPAttachment request, replacing the <key> value with the Support Case record number from the example above, then click Run:

```
<dem:EWRetrieveAttachedAsSOAPAttachment>
  <sessionId>SESSIONID_FROM_LOGIN_RESPONSE</sessionId>
  <tableName>case</tableName>
  <key>419</key>
  <fieldName>downloadable_files</fieldName>
  <filePosition>1</filePosition>
</dem:EWRetrieveAttachedAsSOAPAttachment>
```

2. The request retrieves the attached file from the Downloadable Files field with a file position of 1. The response doesn't show any data, but below the response window the Attachments button indicates that there is 1 attachment.

Name	Content type	Size	Part	Type	ContentID
file=0-1504013130...	application/octet-... 4		file	MIME	<file=0-15040131...

Headers (0)	Attachments (1)	SSL Info	WSS (0)	JMS (0)
-------------	-----------------	----------	---------	---------

3. In the request, change the <filePosition> value to an incorrect attachment number like 80, then click Run.
 - a. The response is similar to the following: <message>[http-0.0.0.0-80-7][1504013330298] Blob field id in table case does not contain 81 files, but less (2) for key 419</message>

EWRemoveAttached

1. Add the following values to the EWRemoveAttached request, replacing the <long> value with the Support Case record number from the example above, then click Run:

```
<dem:EWRemoveAttached>
  <String_1>SESSIONID_FROM_LOGIN_RESPONSE</String_1>
  <String_2>case</String_2>
  <long_3>419</long_3>
  <String_4>downloadable_files</String_4>
  <int_5>1</int_5>
</dem:EWRemoveAttached>
```

2. The request removes the first attached file from the Downloadable Files field. The response is similar to `<result>1</result>`, with the number in the `<result>` tag being the number of remaining attached files.
3. In the KB, open the record and check that the first file was removed from the Downloadable Files field.

EWSearchTable

1. Add the following values to the EWSearchTable request, then click Run:

```
<dem:EWSearchTable>
  <String_1>SESSIONID_FROM_LOGIN_RESPONSE</String_1>
  <String_2>case</String_2>
  <arrayOfString_3>
    <value>id</value>
    <value>summary</value>
  </arrayOfString_3>
  <String_4>Open Cases</String_4>
</dem:EWSearchTable>
```

2. The request searches the Support Cases table for all records with a status of Open. The response shows the field values for each of the open cases.
3. Change the search name in String_4 to a non-existent search and click Run. The response is similar to `<message>No search=Open Cas for table=WSCase</message>`.

EWSearchTablePaginated

1. Add the following values to the EWSearchTablePaginated request, then click Run:

```
<dem:EWSearchTablePaginated>
  <String_1>SESSIONID_FROM_LOGIN_RESPONSE</String_1>
  <String_2>case</String_2>
  <arrayOfString_3>
```



```

        <value>id</value>
        <value>summary</value>
    </arrayOfString_3>
    <String_4>Open Cases</String_4>
    <int_5>0</int_5>
    <int_6>1</int_6>
</dem:EWSearchTablePaginated>

```

2. The request searches the Support Cases table for the first page number (int_6) of the first record (int_5). The response shows the fields on page 0 of the first record in the table.
3. Modify the <int_6> value to 2 (record count), then click Run.
 - a. The response shows the fields from the first two records.
4. Modify the <int_5> value to 1 (page number), then click Run.
 - a. The response shows the first two records on page 1 of the table.
5. Modify the <int_value> to 777, then click Run.
 - a. The response shows no records.
6. Modify <int_5> to 0, and <int_6> to 1000, then click Run.
 - a. The response shows all records in the table.

EWSearchTableWithQuery

1. Add the following values to the EWSearchTableWithQuery request, then click Run:

```

<dem:EWSearchTableWithQuery>
    <String_1>SESSIONID_FROM_LOGIN_RESPONSE</String_1>
    <String_2>case</String_2>
    <arrayOfString_3>
        <value>id</value>
        <value>summary</value>
        <value>priority</value>
    </arrayOfString_3>
    <String_4>Open Cases</String_4>
    <String_5>priority='Low'</String_5>
</dem:EWSearchTableWithQuery>

```

2. The request searches the Support Cases table for all open cases with a priority of Low. The response shows the fields for all records in Open status with a Priority of Low, including the ID Summary and Priority fields.
3. Change the condition for <String_5> to something like <String_5>priority='Low'&&summary~='How'</String_5>, then click Run.
 - a. The response should return no errors and there should be a record data where the Summary field contains the word 'How'.

4. Change the search in <String_4> to a non-existent value, then click Run.
 - a. The response is something like <message>No search=Cases for table=WSCase</message>

EWSearchTableWithQueryPaginated

1. Add the following values to the EWSearchTableWithQueryPaginated request, then click Run:

```
<dem:EWSearchTableWithQueryPaginated>
  <String_1>SESSIONID_FROM_LOGIN_RESPONSE</String_1>
  <String_2>case</String_2>
  <arrayOfString_3>
    <value>id</value>
    <value>summary</value>
  </arrayOfString_3>
  <String_4>Open Cases</String_4>
  <String_5>priority='Low'</String_5>
  <int_6>0</int_6>
  <int_7>1</int_7>
</dem:EWSearchTableWithQueryPaginated>
```

2. The request searches the Support Cases for open cases with a priority of Low, in the first record (int_7) on the first page (int_6) in the table. The response shows all fields from that record.
3. Modify the <int_7> value to 2, to find the first two records, then click Run.
 - a. The response shows all the fields in the first two records on the first page of the table that match the search query.
4. Modify the <int_6> value to 777 to find the first two records on the 777'th page of the table, then click Run.
 - a. The response shows no records.
5. Set the <int_6> value to 0, and <int_7> to 1000, then click Run.
 - a. The response shows the fields for all records in the table.

EWLogout

1. Add the following values to the EWLogout request, then click Run:

```
<dem:EWLogout>
  <String_1>SESSIONID_FROM_LOGIN_RESPONSE</String_1>
</dem:EWLogout>
```

2. The request logs out the session. The response doesn't have any errors.
3. Execute any of the previous requests with the same values. The response is similar to `<message>[http-0.0.0.0-80-7][1504083695004] not logged in or this session has timed out</message>`

Deactivate Web Services

1. As the admin user, log into the testing knowledgebase, then go to **Setup > System > Manage Web Services**.
2. Click Disable WS.
3. The message changes to "SOAP Web Services are currently Off".
4. Try to perform the test case for EWLogin.
5. The response is similar to HTTP Status 404 - /ewws/Demo/EWServiceAPIv2.

Testing SOAP Security

The following examples enables you to test the security of your SOAP application. Leave your instance of SOAPUI configured as above.

Add WS-Security to Web Services

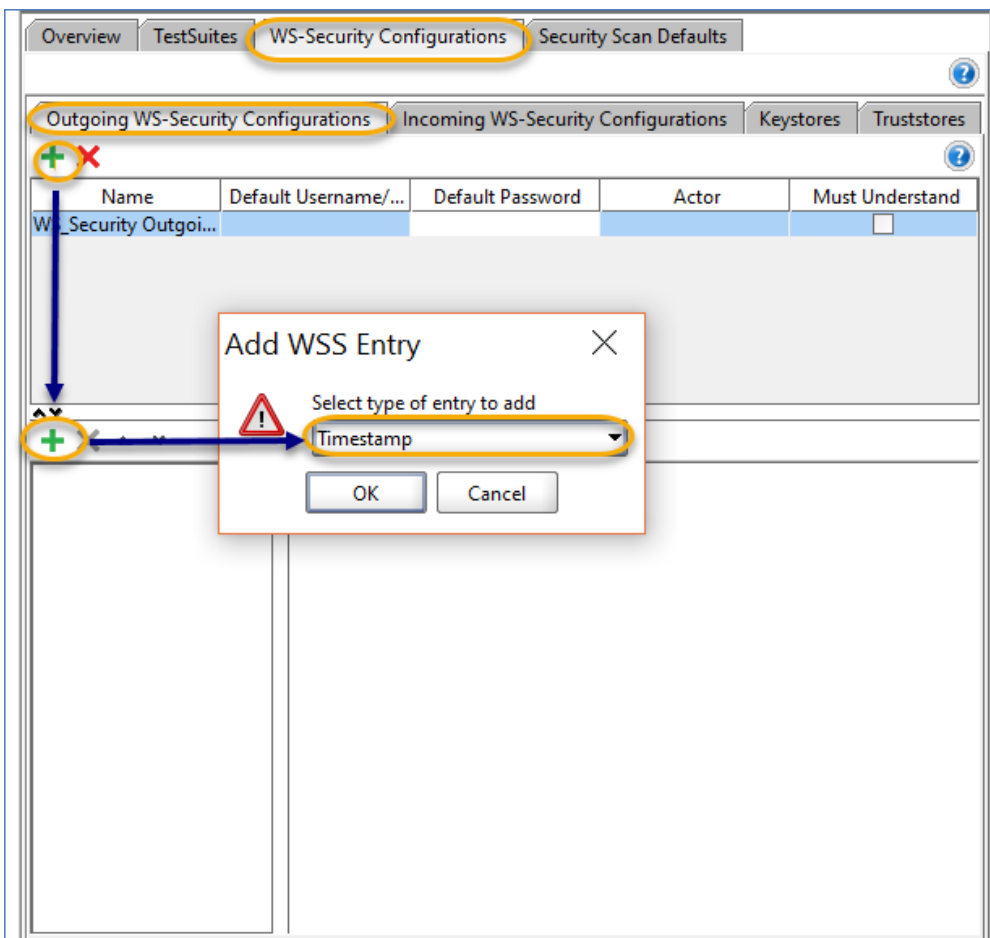
1. In the KB, go to **Setup > System > Manage Web Services**. Check that the wording states "SOAP Web Services are currently Off".
2. Check that 'version 2', and the WS-Security enabled checkbox, are both selected.
3. Write down the current TTL value - 10 by default.
4. Click Enable WS.
5. Select the 'admin' group for SOAP groups, even if it already shows the group as being selected.
6. Click Finish, then reopen the Web Services screen.
7. Check the wording states "SOAP Web Services are currently On".

Login Without Security

1. In SOAPUI, try to perform the test case for EWLogin.
2. The response says something like `<faultstring>This service requires <wsse:Security>, which is missing.</faultstring>`.

Configure SOAPUI for WS_Security

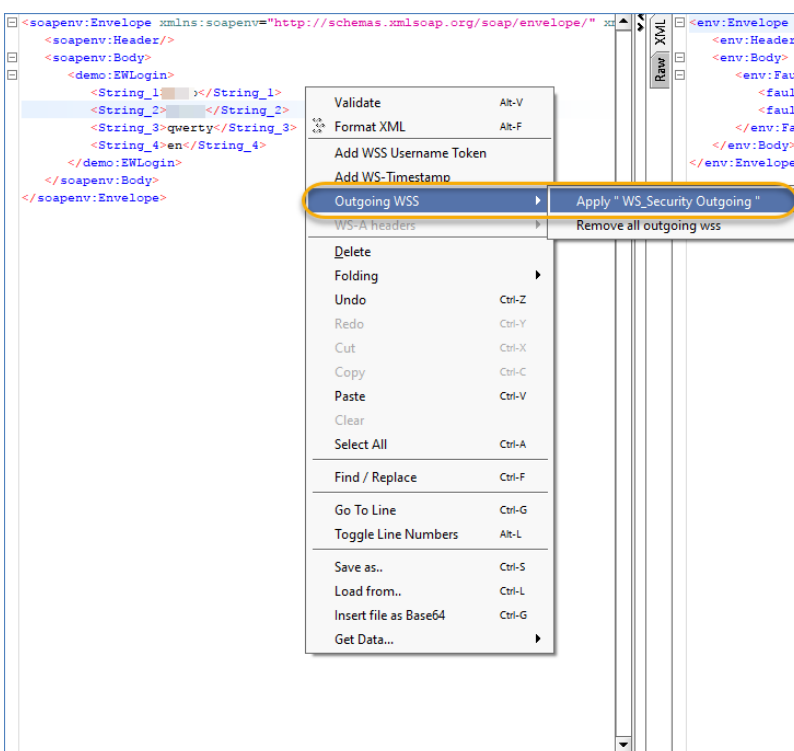
1. In SOAPUI, double click the project root to open the settings wizard.
2. Open the WS_Security Configurations tab to the Outgoing WS-Security Configurations tab.
3. Click + then add a name in the dialog, and save it.
4. In the bottom pane, click + then in the ADD WS Entry dialog, select Timestamp. Click OK.



5. Click OK.
6. In the Time To Live field that appears, enter the TTL value in the Agiloft Web Services wizard.
7. Exit from the settings wizard.

Login with Security

1. In SOAPUI, select the EWLogin request, right click in the window and select **Outgoing WSS > Apply <WS Config Name>**.



2. Click Run.
3. The response returns new timestamp data similar to the following:

```
<wsse:Security ...>
  <wsu:Timestamp wsu:Id="timestamp">
    <wsu:Created>2016-10-12T13:54:35.482Z</wsu:Created>
    <wsu:Expires>2016-10-12T13:54:45.482Z</wsu:Expires>
  </wsu:Timestamp>
</wsse:Security>
```

EWSearchTable with Security

1. Reopen the [EWSearchTable](#) request and add the Session ID from the previous example, then click Run.
2. The response is similar to `<faultstring>This service requires <wsse:Security>, which is missing.</faultstring>`.
3. Right click in the window and select **Outgoing WSS > Apply <WS Config Name>**, then click Run.
4. The response returns the record data, with an additional `<wsse:Security>` section with timestamps.

EWAttachFromSOAPAttachment

Attaches a file into a field of the table record in your knowledgebase.

Syntax

```
int n = ew.EWAttachFromSOAPAttachment(String sessionId, String  
                                     tableName, long id, String fieldName, String fileName);
```

Usage

Use the EWAttachFromSOAPAttachment call to attach a file passed as a SOAP attachment into a file or image field in the table using the specified file name.

Rules and Guidelines

When attaching files, consider the following rules and guidelines:

- The username that was used to obtain the specified session token must have sufficient access rights to modify individual records within the specified table. Please verify specific permissions via **Setup > Access > Manage Groups > (Edit Group) > Table > (Edit Table) > Permissions**.
- Agiloft allows specifying fine-grained access permissions on the field level. The username that was used to obtain the specified session token must have sufficient access rights to be able to update the field content. Please verify specific permissions via **Setup > Access > Manage Groups > (Edit Group) > Table > (Edit Table) > Field Permissions**.
- This call returns the current number of the attached files in the specified field.
- One can use EWRead method to obtain an array of file names for the attached fields in the specified field.
- Only one file per call is attached.
- The file name does NOT have to be unique across multiple calls against the same record and field.
- In general, you use EWAttachFromSOAPAttachment when you know in advance the identifiers of the records to retrieve. The client application may use the likes of EWSelectFromTable call to obtain record identifiers beforehand or to take the identifiers from the id field of the data structures and linking classes.

Basic Steps for Attaching Files

Attaching files to records involves the following basic steps:

1. Determine the id of the record you want to update. You may want to use the EWSelectFromTable or EWSearchTable calls to get the identifiers of the records based on some search condition or get the identifier from a previous EWCreate call, or get the id of a linked record from the linking class after performing EWRead or EWUpdate.
2. Wrap your file on the client-side in the matter specific to your client-side SOAP environment.
3. Call EWAttachFromSOAPAttachment.
4. Process the results.

Example Task

In MyKB knowledgebase as user A, attach from the client-side filesystem file B.txt located in the current directory to the field Additional Files of case #456 and return the total number of files currently attached.

The task is completed by performing the following steps:

1. Login to MyKB with "A" and "password" and English as the local language.
2. Prepare the file for transmission.
3. Invoke attach call.
4. Get total number of files currently attached.
5. Logout.

Sample Code - Java

You can generate sample Web Services code for any table by selecting **Setup > Tables > (Edit Table) > API > Download Sample**.

```
public int attach() throws Exception {
    EWServiceAPI binding = new EWServiceAPIServiceLocator().getDemo();
    try {
        String sessionId = binding.EWLogin("MyKB", "A", "password", "en");
```



```

DataHandler dataHandler =
    new DataHandler(new FileDataSource("B.txt"));
binding.addAttachment(dataHandler);
int n = binding.EWAttachFromSOAPAttachment(sessionId, "case", 456,
"additional_files", "B.txt");
return n;
} finally {
binding.EWLogout(sessionId);
}
}

```

Arguments

Note: The file to be attached is passed as a SOAP attachment.

Name	Type	Description
sessionId	String	Session token
tableName	String	The name of the table where the record is.
id	long	The identifier of the record to attach files to.
fieldName	String	The name of the field to attach files to.
fileName	String	The name of the file to be used in File or Image field.

Response

The current number of files attached in the specified field.

Faults

EWSessionException - client not logged in or session has expired; client should re-login.

EWPermissionException - user used to create the session lacks sufficient privileges to read the record.

EWWrongDataException - client has supplied wrong data, for instance ID cannot be found.

EWOperationException - the operation has been blocked by one of Agiloft functionalities, for example a table-level lock.

EWIntegrityException - specified table cannot be found or its primary key cannot be identified.

EWUnexpectedException - an unexpected exception has happened; user should report this for investigation.

EWCreate

Adds a record to your knowledgebase data.

Syntax

```
long id = ew.EWCreate(String sessionId, String table,  
    EWWSBaseUserObject object)  
or for the table-specific call:  
long id = ew.EWCreate_WSCase(String sessionId, WSCase object)
```

Usage

Use EWCreate to add records to the tables, such as Support Case or People, in your knowledgebase.

The EWCreate call is analogous to the INSERT statement in SQL.

Rules and Guidelines

When creating records, consider the following rules and guidelines:

- The username that was used to obtain the specified session token must have sufficient access rights to create individual records within the specified table. Please verify specific permissions by navigating to **Setup > Access > Manage Groups**, editing the relevant group, clicking the Table tab, editing the relevant table and clicking the Permissions tab.
- Agiloft allows specifying fine-grained access permissions on the table field level. The username that was used to obtain the specified session token must have sufficient access rights to set every field being specified in the data object supplied in the call when creating the record. Consider this if you are using record data obtained from or just used in another call like EWRead or EWUpdate - read and update permissions of a particular user may not match the created ones. Please verify specific permissions by navigating to **Setup > Access > Manage Groups**, editing the relevant group, clicking the Table tab, editing the relevant table and clicking Field Permissions.

- The identifier of the new record is calculated automatically unless the field-level permissions allow the client to supply the pre-calculated one and the client supplied one.
- Certain fields can be defined to have default values. If permissions allow, these may be overwritten by data supplied in the call.
- In the context of overwriting the default values the API and WSDL distinguish between an empty (null) value set explicitly - i.e. the one that should overwrite the default - and a value not set at all in the EWCreate call - i.e. where default should remain as is.
- For required fields that do not have a pre-configured default value, you must supply a value. For all other fields in the record that do not have pre-configured default value, if you do not explicitly specify a value, then its value is empty (null).
- Agiloft allows one to establish relationships between the records in different tables and ensures the data integrity once the links are forged. See [Linked Fields](#).
- When creating the records in the table it is required to explicitly specify the type of the record. This is especially important in the case when the record is created for the true subtype, such as Contacts.Customer. In this case the table specified in the call will be "Contacts" and the type of the record will be "Customer". For the top-level subtypes the type of the record is equal to the table name. Please consult the Table Wizard for the specific names of the tables/subtypes in your KB, available by navigating to **Setup > Tables** and editing your table.
- Fields that are present in the table directly are filled in as simple values.
- Some environments like .NET require a special property set for simple-type fields to properly handle empty values - <name>Specified = true for .NET.
- Fields from the Linked Fields relationships that allow values not present in the donor tables are present both in the table, for the case when the value is non-source, and in the linking classes, for the case when the value is truly imported and the link is forged.

Linked Fields

Fields that are imported into the table from another table as a part of a Linked Fields set have to be filled via special linking classes.

To forge a link the client has to create one or more instances of the linking class and select one of 3 options:

1. Specify the identifier previously obtained through the call to EWSelectFromTable, EWCreate or EWUpdate in the "id" field of the linking class. Please note: some environments like .NET require a special property set for simple-type fields to properly handle empty values - <name>Specified = true for .NET.
2. Supply example data in the fields of the linking class that will be used to run Query-by-Example.

3. Supply the SQL statement similar in form to the one used for EWSelectFromTable in the special field "searchSQL". Such SQL is evaluated prior to the creation of the record to identify the actual value for the linked record identifier(s). This evaluation is done in exactly the same manner as when selecting from the table. Normally this method would be used if the query is indirect, such as "where full_name like 'John%'".

See examples of all of the above in the sample code below.

If any of the query mechanisms are used to identify the related records one has to take into account if the relationship allows multiple records to be linked. If the query returns multiple results and the relationship doesn't allow multiple records to be linked an EWOperationException is thrown.

Choice Fields

The values for choice columns should be supplied as instance(s) of the enumerated types described in the WSDL.

Basic Steps for Creating Records

Creating records involves the following basic steps:

1. Instantiate the concrete subclass of EWWSBaseUserObject such as WSCase or WSContact. Populate its fields with the data that you want to add.
2. Call create, passing in the table name and the data map.
3. Process the results.

Example Task

In MyKB knowledgebase as user A, create a case describing a lost present, with High priority, linked to contact #125, owned by a Support Person and assigned to a Support Team.

The task is completed by performing the following steps:

1. Login to MyKB with "A" and "password", English as the locale language.
2. Create a new data structure, fill in the fields.
3. Specify a customer link to the Contacts record 124.
4. Specify the assignee Support Team from the Team table.
5. Specify an ownership link to the Contact record identified by the full name "Support Person".

6. Create the record.
7. Logout.

Sample Code - Java

```
public long create() {
    EWServiceAPI binding = new EWServiceAPIServiceLocator().getMyKB();
    String sessionId = binding.EWLogin("MyKB", "A", "password", "en");
    WSCase wsCase = new WSCase();
    wsCase.setSummary("A case of a lost present");
    wsCase.setProblem_description("People keep losing things"+
    +" bought for their dearest.");
    WSCaseContacts_Dao3_Link5 contact = new WSCaseContacts_Dao3_Link5();
    contact.setId(125L); // forge link by specifying the identifier
    wsCase.setDAO_Dao3_Link5(contact);
    WSCaseTeams_Dao3_Link3 assignedTo = new WSCaseTeams_Dao3_Link3();
    assignedTo.setAssigned_To("Support Team"); // forge link by QBE
    wsCase.setDAO_Dao3_Link3(assignedTo);
    wsCase.setPriority(WSCasePriority.OPTION_HIGH);
    WSCaseContacts_Dao3_Link7 ownedBy = new WSCaseContacts_Dao3_Link7();
    ownedBy.setSearchSQL("full_name='Support Person'"); // forge link by
    SQL
    wsCase.setDAO_Dao3_Link7(ownedBy);
    wsCase.setType("case");
    final long id = binding.EWCreate(sessionId, "case", wsCase);
    binding.EWLogout(sessionId);
    return id;
}
```

You can generate sample Web Services codes for any table by selecting **Setup > Tables > [Select Table to Edit] > API > Download Sample**.

Arguments

Name	Type	Description
sessionId	String	Session token
tableName	String	The name of the table where the record is to be created - only for generic methods.

obj	EWWSBaseUserObject or a specific descendent like WSCase or WSContact for table-specific calls.	The descendant of EWWSBaseUserObject - one of the complex types described in the WSDL that correspond to the tables on the Agiloft side
-----	--	---

Response

The identifier of the record created.

Faults

EWSessionException - client not logged in or session has expired; client should re-login.

EWPermissionException - username used to create the session lacks sufficient privileges to perform record modification.

EWWrongDataException - client has supplied the wrong data.

EWOperationException - creation operation has been blocked by an Agiloft function.

EWIntegrityException - specified table cannot be found or its primary key cannot be identified.

EWUnexpectedException - an unexpected exception has happened; user should report this for investigation.

EWCreateAndRead

Adds a record to your knowledgebase data and reads the result.

Syntax

```
EWWSBaseUserObject o = ew.EWCreateAndRead(String sessionId, String table,  
                                           EWWSBaseUserObject object)  
or for the table-specific call:  
WSCase case = ew.EWCreateAndRead_WSCase(String sessionId, WSCase object)
```

Usage

Use EWCreateAndRead to add records to the tables, such as Support Case or People, in your knowledgebase and read the result in one call, potentially retrieving values for the fields set by business rules and other back-end logic.

The EWCreateAndRead call is analogous to an INSERT statement in SQL immediately followed by SELECT.

Rules and Guidelines

When creating records, consider the following rules and guidelines:

- The username that was used to obtain the specified session token must have sufficient access rights to create individual records within the specified table. Please verify specific permissions via **Setup > Access > Manage Groups > [Edit Group] > Table > [Select Table] > Permissions**.
- Agiloft allows specifying fine-grained access permissions on the table field level. The username that was used to obtain the specified session token must have sufficient access rights to set every field being specified in the data object supplied in the call when creating the record. Consider this if you are using a record data obtained from or just used in another call like EWRead or EWUpdate - read and update permissions of a particular user may not match the created ones. Please verify specific permissions via **Setup > Access > Manage Groups [Edit Group] > Table > [Select Table] > Field Permissions**.
- The identifier of the new record is calculated automatically unless the field-level permissions allow the client to supply the pre-calculated one and the client supplied one.

- Certain fields can be defined to have default values. If permissions allow, these may be overwritten by the data supplied in the call.
- In the context of overwriting the default values the API and WSDL distinguish between an empty (null) value set explicitly, i.e. the one that should overwrite the default, and a value not set at all in the EWCreate call, i.e. where default should remain as is.
- For required fields that do not have a pre-configured default value, you must supply a value. For all other fields in the record that do not have a pre-configured default value, if you do not explicitly specify a value, then its value is empty (null).
- Agiloft allows one to establish relationships between the records in different tables and ensures the data integrity once the links are forged. See Linked Fields.
- When creating the records in the table it is required to explicitly specify the type of the record. This is especially important in the case when the record is created for the true subtype, such as Contacts.Customer. In this case the table specified in the call will be "Contacts" and the type of the record will be "Customer". For the top-level subtypes the type of the record is equal to the table name. Please consult the Table wizard for the specific names of the tables/subtypes in your KB, available via **Setup > Tables > [Select Table to Edit]**.
- Fields that are present in the table directly are filled in as simple values.
- Some environments like .NET require a special property set for simple-type fields to properly handle empty values - `<name>Specified = true` for .NET
- Fields from the Linked Fields relationships that allow values not present in the donor tables are present both in the table, for the case when the value is non-source, and in the linking classes for the case when the value is truly imported and the link is forged.

Linked Fields

Fields that are imported into the table from another table as a part of a Linked Fields set have to be filled via special linking classes.

In WSDL a Linked Fields set takes form of `DAO_Dao3_Link<N>` field in the complex data structure that corresponds to the target table, where N is a sequential number assigned automatically at the time of the set creation, e.g. `WSCase.DAO_Dao3_Link3`.

As a value such fields can take one or more `WS<Table1><Table2>_Dao3_Link<N>` data structures - linking classes, where Table1 is the target table and Table2 is the donor table of the Linked Fields relationship, e.g. `WSCaseTeams_Dao3_Link3`.

Unfortunately, at this moment one has to rely on investigating the actual sets of fields inside the classes to trace the fields, visible in the Field wizard in the GUI, back to the main object property.

Non-source values for Linked Field sets that allow them are present directly in the table itself additionally to those in the Linking Classes, if the link was in fact forged.

To forge a link the client has to create one or more instances of the linking class and select one of 3 options:

1. Specify the identifier previously obtained through the call to EWSelectFromTable, EWCreate or EWUpdate in the "id" field of the linking class. Please note: some environments like .NET require a special property set for simple-type fields to properly handle empty values - .<name>Specified = true for .NET.
2. Supply example data in the fields of the linking class that will be used to run a Query-by-Example.
3. Supply the SQL statement similar in form to the one used for EWSelectFromTable in the special field "searchSQL". Such SQL is evaluated prior to the creation of the record to identify the actual value for the linked record identifier(s). This evaluation is done in exactly the same manner as when selecting from the table. Normally this method would be used if the query is indirect, such as "where full_name like 'John%'".

See examples of all of the above in the sample code below.

If any of the query mechanisms are used to identify the related records, one has to take into account if the relationship allows multiple records to be linked. If the query returns multiple results and the relationship doesn't allow multiple records to be linked an EWOperationException is thrown.

Choice Fields

The values for choice columns should be supplied as instance(s) of the enumerated types described in the WSDL.

The original text values have undergone the following transformations:

1. Spaces replaced by "_"
2. Dashes replaced by "MINUS"
3. Pluses replaced by "PLUS"
4. Prefixed with "OPTION_"
5. Converted to upper case

One has to perform the reverse transformation to get to the text value.

Unsupported Types of Fields

Related tables and embedded search results are not supported by the SOAP interface.

Basic Steps for Creating and Reading Records in One Call

Creating records involves the following basic steps:

1. Instantiate the concrete subclass of EWWSBaseUserObject such as WSCase or WSContact. Populate its fields with the data that you want to add.
2. Call EWCreateAndRead, passing in the table name and the data map or the table-specific variant with the data map only.
3. Process the results.

Example Task

In MyKB knowledgebase as user A create a case describing a lost present, with High priority, linked to contact #125, owned by a Support Person and assigned to the Support Team. Return the value from "date_created" field.

The task is completed by performing the following steps:

1. Login to MyKB with "A" and "password" and English as the local language.
2. Create a new data structure, fill in the fields.
3. Specify a customer link to the Contacts record 124.
4. Specify the assignee Support Team from the Team table.
5. Specify an ownership link to the Contact record identified by the full name "Support Person".
6. Create the record.
7. Get date_created value from the returned object.
8. Logout.

Sample Code - Java

```
public Date create() {  
    EWServiceAPI binding = new EWServiceAPIServiceLocator().getMyKB();  
    try {  
        String sessionId = binding.EWLogin("MyKB", "A", "password", "en");
```

```

WSCase wsCase = new WSCase();
wsCase.setSummary("A case of a lost present");
wsCase.setProblem_description("People keep losing things"+
+" bought for their dearest.");
WSCaseContacts_Dao3_Link5 contact = new WSCaseContacts_Dao3_Link5();
contact.setId(125L); // forge link by specifying the identifier
wsCase.setDAO_Dao3_Link5(contact);
WSCaseTeams_Dao3_Link3 assignedTo = new WSCaseTeams_Dao3_Link3();
assignedTo.setAssigned_To("Support Team"); //forge link by QBE
wsCase.setDAO_Dao3_Link3(assignedTo);
wsCase.setPriority(WSCasePriority.OPTION_HIGH);
WSCaseContacts_Dao3_Link7 ownedBy = new WSCaseContacts_Dao3_Link7();
ownedBy.setSearchSQL("full_name='Support Person'"); //link by SQL
wsCase.setDAO_Dao3_Link7(ownedBy);
wsCase.setType("case");
WSCase wsCase = (WSCase) binding.EWCreateAndRead(sessionId, "case",
                                                wsCase);

return wsCase.getDate_Created();
} finally {
binding.EWLogout(sessionId);
}
}

```

You can generate sample a Web Services code for any table by selecting **Setup > Tables > [Edit Table] > API > Download Sample**.

Arguments

Name	Type	Description
sessionId	String	Session token
tableName	String	The name of the table where the record is to be created (only for generic methods).
obj	EWWSBaseUserObject or a specific descendent like WSCase or WSContact for table-specific calls.	The descendant of EWWSBaseUserObject - one of the complex types described in the WSDL that correspond to the tables on the Agiloft side

Response

The record data as a descendant of EWWSBaseUserObject - a complex structure described in WSDL.

Faults

EWSessionException - client not logged in or the session has expired; client should re-login.

EWPermissionException - username used to create the session lacks sufficient privileges to perform the record modification.

EWWrongDataException - client has supplied the wrong data.

EWOperationException - creation operation has been blocked by one of Agiloft functionalities.

EWIntegrityException - specified table cannot be found or its primary key cannot be identified.

EWUnexpectedException - an unexpected exception has happened; user should report this for investigation.

EWDelete

Deletes one or more individual records from your knowledgebase data.

Syntax

```
ew.EWDelete(String sessionId, String tableName, long[] keys,  
            DeleteRule deleteRule, long[] replacementKeys);
```

Usage

Use EWDelete to delete one or more existing records from any given table.

The operation accepts a list of identifiers for the records to be deleted in one transaction. The transaction will either succeed if each individual record has been successfully deleted or a rollback is performed otherwise.

The EWDelete call is analogous to the DELETE statement in SQL.

Rules and Guidelines

When deleting records, consider the following rules and guidelines:

- The username that was used to obtain the specified session token must have sufficient access rights to delete individual records within the specified table.
- Agiloft allows one to establish relationships between the records in different tables and ensures the data integrity once the links are forged.

In the presence of such relationships when the delete operation is performed via GUI, the user who triggers the operation would be prompted via a series of dialogs to resolve the conflicts in each case.

When the same operation is triggered programmatically it is necessary to specify the policy to be applied for conflict resolution beforehand via a deleteRule argument.

This is achieved via a deleteRule parameter that can take one of the four values:

- ERROR_IF_DEPENDANTS - fails when there are any dependents,

- `APPLY_DELETE_WHERE_POSSIBLE` - tries to delete all dependent records, when delete cannot be done an attempt to unlink the record is made,
- `DELETE_WHERE_POSSIBLE_OTHERWISE_UNLINK` - same as above,
- `APPLY_UNLINK` - tries to unlink dependent records,
- `UNLINK_WHERE_POSSIBLE_OTHERWISE_DELETE` - tries to unlink all dependant records, when unlink cannot be done an attempt to delete the record is made,
- `REPLACE_WITH_ANOTHER` - tries to link dependent records to the substitute one specified in the `replacementKeys`.

Normally the decision which policy to use is taken at the "design" time and depends on the structure of the knowledgebase.

Notes

- If the specific strategy fails the error message returned will suggest alternatives to be used. One may use REST interface to find the best strategy.
- If `APPLY_DELETE_WHERE_POSSIBLE` or `DELETE_WHERE_POSSIBLE_OTHERWISE_UNLINK` strategies are used and the configuration of the knowledgebase allows it, a special "Fast Delete" algorithm is used, the same as via the GUI.
- Certain objects cannot be deleted via the API.

Basic Steps for Deleting Records

Deleting records involves the following basic steps:

1. Determine the id of each record that you want to delete. For example, you might call `EWSelectFromTable` to retrieve the identifiers for the set of records that you want to delete based on a specific criteria.
2. Construct the `keys[]` array and populate it with the identifiers of each record that you want to delete.
3. If the `REPLACE_WITH_ANOTHER` delete policy is used construct the substitute `replacementKeys[]` array.
4. Call `EWDelete`, passing in the table name, the `keys[]` array, the policy identifier and the substitute keys array, if necessary.
5. Process the results.

Example Task

In MyKB knowledgebase as user A delete case records #1234, #556, #123456 performing a "cascade" delete for all dependent records.

The task is completed by performing the following steps:

1. Login to MyKB with "A" and "password", English as the locale language.
2. Construct the keys array.
3. Call EWDelete with for case table with these keys and APPLY_DELETE_WHERE_POSSIBLE as delete policy.
4. Logout.

Sample Code - Java

You can generate a sample Web Services code for any table by selecting **Setup > Tables > [Edit Table] > API > Download Sample**.

```
public void delete() {
    long[] keys = new long[] { 1234L, 556L, 12346L };
    try {
        EWServiceAPI binding = new EWServiceAPIServiceLocator().getMyKB();
        String sessionId = binding.EWLogin("MyKB", "A", "password", "en");
        binding.EWDelete(sessionId, "case", keys,
            DeleteRule.APPLY_DELETE_WHERE_POSSIBLE, null);
    } catch ( EWWrongDataException e) {
        System.out.println(" EWWrongDataException encountered:\n\n" +
            e.getMessage());
    } catch ( EWOperationException e) {
        System.out.println(" EWOperationException encountered:\n\n" +
            e.getMessage());
    }
}
```


Arguments

Name	Type	Description
sessionId	String	Session token
tableName	String	The name of the table that contains records to be deleted.
keys	long[]	The array of one or more identifiers of the records to be deleted.
deleteRule	DeleteRule	The policy to be used to resolve the conflicts that may arise if the records being deleted are referenced elsewhere. May be one of the following: 1. ERROR_IF_DEPENDANTS 2. APPLY_DELETE_WHERE_POSSIBLE 3. APPLY_UNLINK 4. REPLACE_WITH_ANOTHER
replacementKeys	long[]	The array of substitute identifiers to be used with the REPLACE_WITH_ANOTHER delete policy has to contain the same exact number of elements, with possibly the same values. An equivalent of a null value can be safely passed instead for any other delete policy.

Response

The call does not return any value. A transaction is either completed as a whole or a rollback is performed.

Faults

EWSessionException - client not logged in or the session has expired; client should re-login.

EWPermissionException - user used to create the session lacks sufficient privileges to perform a record deletion.

EWWrongDataException - supplied data is wrong; caused by either the record to delete cannot be found – cause, array index, and key of the problematic record returned as parameters – or replacement failed – cause. array index, key of the problematic record and key of the replacement one as parameters.

EWOperationException - delete operation cannot be done – cause, array index and key of the problematic record returned as parameters; either the record has dependents – number of dependents as parameter; or the delete and the unlink are both not possible – relationship description as parameter; or the unlink is not possible – relationship description as parameter; or the replacement is not possible – relationship description as parameter; or the delete has failed – cause, array index, key of problematic record as parameters.

EWIntegrityException - specified table cannot be found or its primary key cannot be identified.

EWUnexpectedException - an unexpected exception has happened; user should report this for investigation.

EWGetChoiceLineId

Obtains the internal identifier that corresponds to a choice value for use in SQL-based expressions.

Syntax

```
long id = ew.EWGetChoiceLineId(String sessionId, String  
    tableName,  
    String fieldName, String value);
```

Usage

Use EWGetChoiceLineId to obtain the internal identifier that corresponds to the choice text value for further use in SQL-based expressions such as the where parameter of EWSelectFromTable or the value of searchSQL property when forging links to other tables via Linked Field sets.

SQL-based expressions are evaluated on the database level where choice values are stored as identifiers. These identifiers change if the knowledgebase is copied or re-imported, or the values in the choice list are re-created. It is therefore advisable to use the EWGetChoiceLineId call to obtain the id value at runtime immediately before the SQL call.

Rules and Guidelines

When querying for choice value identifiers, consider the following rules and guidelines:

- The value parameter is intentionally specified as String rather than an enumerated WSDL type. This is to give client applications more flexibility.

If the client needs to lookup the identifier of the choice value enumerated in WSDL, the client should convert the enumerated value into String. Since enumerated WSDL types that correspond to the choice values are based on Strings, this should always be possible although specific details depend on the client Web Services environment used.

When converting one of the WSChoice_ values back to String, please note that the original text values have undergone the following transformations:

1. Spaces replaced by "_"
2. Dashes replaced by "MINUS"
3. Pluses replaced by "PLUS"
4. Prefixed with "OPTION_"
5. Converted to upper case

You need to perform the reverse transformation to get to the text value.

- If the factors that trigger the change of the internal choice identifiers are somehow detected on the client, then the values returned by this call can be safely cached on a per KB basis between multiple calls. As described above, the internal choice field identifiers can be changed when a knowledgebase is copied or re-imported, or if the values in the choice list are re-created.

Example Task

In MyKB knowledgebase as user A, find all cases with High priority.

The task is completed by performing the following steps:

1. Login to MyKB with "A" and "password", English as the locale language.
2. Obtain the identifier for the choice value High for the field Priority in the table Case.
3. Run the SQL query to obtain the identifiers.
4. Logout.

Sample Code - Java

You can generate a sample Web Services code for any table by selecting **Setup > Tables > [Edit Table] > API > Download Sample**.

```
public long[] selectCasesPriorityHigh() throws Exception {
    EWServiceAPI binding = new EWServiceAPIServiceLocator().getMyKB();
    String sessionId = binding.EWLogin("MyKB", "A", "password", "en");
    try {
        long high = binding.EWGetChoiceLineId(sessionId, "case", "priority",
            "High");
        long ids[] = binding.EWSelectFromTable(sessionId, "case",
            "priority="+high);
        return ids;
    } catch (EWSessionException e) {
        // Normally the client should just re-login and re-try the call.
    }
}
```

```
throw e;
} catch (EWWrongDataException e) {
// Since the field name and the choice value being looked up are most
likely
// hard-coded this is really a show-stopper that should be reported to
the
// developer/implementer
throw e;
} catch (EWIntegrityException e) {
// Since the table name is hard-coded this is a show-stopper that
should be
// reported to the developer/implementer
throw e;
} catch (EWUnexpectedException e) {
// This exception should be reported to the vendor.
// The message contains a token that will help to trace the root cause
of
// the problem.
throw e;
} finally {
binding.EWLogout(sessionId);
}
}
```

Arguments

Name	Type	Description
sessionId	String	Session token.
tableName	String	The name of the table where the choice field is located.
fieldName	String	The name of the choice field in the table.
value	String	The choice text value.

Response

The identifier that corresponds to the choice text value.

Faults

EWSessionException - client not logged in or the session has expired; client should re-login.

EWWrongDataException - the specified field does not exist in the table or the choice value passed cannot be found.

EWIntegrityException - the specified table cannot be found or its primary key cannot be identified.

EWUnexpectedException - an unexpected exception has happened; user should report this for investigation.

EWLogin

Syntax

```
String sessionId = ew.EWLogin(String KB, String user, String  
    password, String language);
```

Usage

Use the EWLogin call to log in to Agiloft and start a client session. A client application must log in and obtain the session token (sessionId) before making any other API calls.

When the client application invokes the EWLogin call, it passes the knowledgebase name - not the label - the username and the password as user credentials. The language code is used in localization of error messages. Specify "en" for English if not sure.

After logging in the client has to pass the obtained session token with every subsequent call as a parameter.

Example Task

Log into MyKB knowledgebase as user A with password "password" and English as the local language.

The task is completed by performing the following steps:

1. Obtain the Service SOAP stub.
2. Call EWLogin.
3. Handle results.

Sample Code - Java

```
public String login() {
    EWServiceAPI binding = new EWServiceAPIServiceLocator().getMyKB();
    String sessionId = binding.EWLogin("MyKB", "A", "password", "en");
    return sessionId;
}
```

You can generate a sample Web Services code for any table by selecting **Setup > Tables > [Select Table to Edit] > API > Download Sample**.

Arguments

Name	Type	Description
KBName	String	The name of the KB to log into.
user	String	The username to be used to create the session defines access privileges for all further calls within this session.
password	String	The corresponding password.
language	String	ISO-639 language code ("en", "de", "fr").

Response

The session token to be used in all subsequent calls.

Faults

EWPermissionException - user lacks the sufficient privileges to log in.

EWWrongDataException - the parameters supplied are not valid to log in.

EWUnexpectedException - an unexpected exception has occurred; user should report this for investigation.

EWLogout

Syntax

```
void ew.EWLogout(String sessionId);
```

Usage

Use EWLogout call to explicitly indicate the end of the session to facilitate freeing up of the server resources.

Generally, client applications are not required to logout. Sessions will expire automatically after a predetermined length of inactivity, which can be configured via the global variable "disconnect_timeout". However, the explicit logout ensures that the server resources are freed up faster which aids scalability.

It is advisable to enclose the EWLogout call in the Java finally block, or equivalent in other languages.

Sample Code - Java

```
public String login() {  
    EWServiceAPI binding = new EWServiceAPIServiceLocator().getMyKB();  
    String sessionId = binding.EWLogin("MyKB", "A", "password", "en");  
    try {  
        doSomeOperations(binding, sessionId);  
    } finally {  
        binding.EWLogout(sessionId);  
    }  
}
```

You can generate a sample Web Services code for any table by selecting **Setup > Tables > [Select Table to Edit] > API > Download Sample**.

Arguments

Name	Type	Description
sessionId	String	Session token

Faults

EWUnexpectedException - an unexpected exception has happened; user should report this for investigation.

EWRead

Retrieves the record data from your knowledgebase.

Syntax

```
EWWSBaseUserObject o = ew.EWRead(String sessionId, String  
    tableName, long id);  
or for the table-specific call:  
WSCase wsCase = ew.EWRead_WSCase(String sessionId, long id);
```

Usage

Use the EWRead call to retrieve individual records from the table. The client application passes the tableName and an identifier of the record to be read in the generic call; and just the record identifier in the table-specific call.

Rules and Guidelines

When creating records, consider the following rules and guidelines:

- The username that was used to obtain the specified session token must have sufficient access rights to read individual records within the specified table. Please verify specific permissions by navigating to **Setup > Access > Manage Groups**, editing the relevant group, clicking the Table tab, editing the relevant table and clicking Permissions.
- Agiloft allows specifying fine-grained access permissions on the field level. The username that was used to obtain the specified session token must have sufficient access rights to be able to read field content. Please verify specific permissions via by navigating to **Setup > Access > Manage Groups**, editing the relevant group, clicking the Table tab, editing the relevant table and clicking Field Permissions..
- This call does not return records that have been deleted.
- This call never returns null when the record is not found.

- In general, you use EWRead when you know in advance the identifiers of the records to retrieve. The client application may use the EWSelectFromTable call to obtain record identifiers beforehand or take the identifiers from the id field of the data structures and the linking classes.
- Client applications can use EWRead to perform a client-side join. For example, a client application can run a query to obtain a set of opportunity records, iterate through the returned opportunity records, obtain the accountId for each opportunity, and then call EWRead to obtain account information for those accountIds.

Linked Fields

Values for the fields that are imported into the target table from the donor table as a part of a Linked Fields set are available via special linking classes.

In WSDL a Linked Fields set takes form of a DAO_Dao3_Link<N> field in the complex data structure that corresponds to the target table, where N is a sequential number assigned automatically at the time of the set creation, such as WSCase.DAO_Dao3_Link3.

As a value, such fields can take one or more WS<Table1><Table2>_Dao3_Link<N> data structures; linking classes, where Table1 is the target table and Table2 is the donor table of the Linked Fields relationship, such as WSCaseTeams_Dao3_Link3.

Unfortunately, at this moment one has to rely on investigating the actual sets of fields inside the classes to trace the fields, visible in the Field Wizard in the GUI, back to the main object property.

Non-source values for Linked Field sets that allow them are present directly in the table itself additionally to those in the Linking Classes, if the link was in fact forged.

Choice Fields

The values for choice columns are returned as instance(s) of the enumerated types described in the WSDL.

The original text values have undergone the following transformations:

1. Spaces replaced by "_"
2. Dashes replaced by "MINUS"
3. Pluses replaced by "PLUS"
4. Prefixed with "OPTION_"
5. Converted to upper case

One has to perform the reverse transformation to get to the text value.

Unsupported Types of Fields

Related tables and embedded search results are not supported by the SOAP interface.

Basic Steps for Reading Records

Reading records involves the following basic steps:

1. Determine the id of the record you want to update. You may want to use the EWSelectFromTable call to get the identifiers of the records based on some search condition or get the identifier from a previous EWCreate call, or get the id of a linked record from the linking class after performing another EWRead or EWUpdate.
2. Call EWRead.
3. Process the results.

Example Task

In MyKB knowledgebase as user A, read case #456 and return the value from the "date_created" field.

This task is completed by performing the following steps:

1. Login to MyKB with "A" and "password" and English as the local language.
2. Read record 456 from table Case.
3. Get date_created value.
4. Logout.

Sample Code - Java

```
public Date read() throws Exception {
    EWServiceAPI binding = new EWServiceAPIServiceLocator().getDemo();
    try {
        String sessionId = binding.EWLogin("MyKB", "A", "password", "en");
        WSCase wsCase = (WSCase) binding.EWRead(sessionId, "case", 456);
        return wsCase.getDate_Created();
    } finally {
```

```
binding.EWLogout(sessionId);
}
}
```

You can generate a sample Web Services code for any table by selecting **Setup > Tables > [Select Table to Edit] > API > Download Sample**.

Arguments

Name	Type	Description
sessionId	String	Session token.
tableName	String	The name of the table where the record is to be read (only for generic methods).
id	long	The identifier of the record to be read.

Response

The record data as a descendant of the EWWSBaseUserObject - a complex structure described in WSDL.

Faults

EWSessionException - client not logged in or the session has expired; client should re-login.

EWPermissionException - user used to create the session lacks the sufficient privileges to read the record.

EWWrongDataException - client has supplied the wrong data.

EWOperationException - the operation has been blocked by an Agiloft function such as a table-level lock.

EWIntegrityException - the specified table cannot be found or its primary key cannot be identified.

EWUnexpectedException - an unexpected exception has happened; user should report this for investigation.

EWRemoveAttached

Removes the attached file specified by a position in the named field of the specified table record.

Syntax

```
int n = ew.EWRemoveAttached(String sessionId, String  
                             tableName, long id, String fieldName, int position);
```

Usage

Use the EWRemoveAttached call to remove an attached file from a File or Image field in a record in the table.

Rules and Guidelines

When removing attached files, consider the following rules and guidelines:

- The username that was used to obtain the specified session token must have sufficient access rights to modify individual records within the specified table. Please verify specific permissions via **Setup > Access > Manage Groups > (Edit Group) > Table > (Edit Table) > Permissions**.
- Agiloft allows specifying fine-grained access permissions on the field level. The username that was used to obtain the specified session token must have sufficient access rights to be able to update field content. Please verify specific permissions via **Setup > Access > Manage Groups > (Edit Group) > Table > (Edit Table) > Field Permissions**.
- This call requires the position of the attached file in the specified field. Position numbering starts from 0.
- One can use the EWRead method to obtain an array of file names for the attached field in the specified field. The sequence of file names in the array will correspond to the positions of the files.
- Only one file per call is removed.
- In general, you use EWRemoveAttached when you know in advance the IDs of the records to retrieve. The client application may use the likes of EWSelectFromTable call to obtain record identifiers beforehand or take the identifiers from the id field of the data structures and linking classes.

Basic Steps for Removing Attached Files

Removing files from records involves the following basic steps:

1. Determine the id of the record you want to update. You may want to use the EWSelectFromTable or EWSearchTable calls to get the identifier of the record based on some search condition or get the identifier from a previous EWCreate call, or get the id of a linked record from the linking class after performing EWRead or EWUpdate.
2. Determine the position of the file in the field.
3. Call EWRemoveAttached.
4. Process the results.

Example Task

In MyKB knowledgebase, as user A, remove all files named B.txt currently attached in the field Additional Files of case #456 and return the total number of files currently attached.

The task is completed by performing the following steps:

1. Login to MyKB with "A" and "password" and English as the local language.
2. Read the record to get the list of the files.
3. Find the position of the file.
4. Invoke the remove file call.
5. Get the total number of files currently attached.
6. Logout

Sample Code - Java

You can generate sample Web Services code for any table by selecting **Setup > Tables > (Edit Table) > API > Download Sample**.

```
public int remove() throws Exception {
    EWServiceAPI binding = new EWServiceAPIServiceLocator().getDemo();
    try {
        String sessionId = binding.EWLogin("MyKB", "A", "password", "en");
```

```

WSCase wsCase = (WSCase) binding.EWRead(sessionId, "case", 456);
String[] fileNames = wsCase.getAdditional_Files();
int n = 0;
if (fileNames!=null) {
    for (int i = 0; i < fileNames.length; i++) {
        if ("B.txt".equals(fileNames[i])) {
            n = binding.EWRemoveAttached("case", 456,
                "additional_files", i);
        }
    }
}
return n;
} finally {
binding.EWLogout(sessionId);
}
}

```

Arguments

Name	Type	Description
sessionId	String	Session token
tableName	String	The name of the table where the record is.
id	long	The identifier of the record to attach files to.
fieldName	String	The name of the field to attach files to.
position	int	The position of the file to be removed.

Response

The current number of files attached in the specified field.

Faults

EWSessionException - client not logged in or session has expired; client should re-login.

EWPermissionException - user used to create the session lacks sufficient privileges to read the record.

EWWrongDataException - client has supplied wrong data, for instance ID cannot be found.

EWOperationException - the operation has been blocked by and Agiloft function, for example a table-level lock.

EWIntegrityException - specified table cannot be found or its primary key cannot be identified.

EWUnexpectedException - an unexpected exception has occurred; the admin user should report this for investigation.

EWRetrieveAttachedAsSOAPAttachment

Retrieves an attached file from the specified field of the table record in your knowledgebase.

Syntax

```
ew.EWRetrieveAttachedAsSOAPAttachment(String sessionId, String  
    tableName, long id, String fieldName, int position);
```

Usage

Use the EWRetrieveAttachedAsSOAPAttachment call to retrieve an attached file from a File or Image field in the table record.

Rules and Guidelines

When retrieving attached files, consider the following rules and guidelines:

- The username that was used to obtain the specified session token must have sufficient access rights to read individual records within the specified table. Please verify specific permissions via **Setup > Access > Manage Groups > (Edit Group) > Table > (Select Table) > Permissions**.
- Agiloft allows specifying fine-grained access permissions on the field level. The username that was used to obtain the specified session token must have sufficient access rights to be able to read the field content. Please verify specific permissions via **Setup > Access > Manage Groups > (Edit Group) > Table > (Edit Table) > Field Permissions**.
- This call requires the position of the attached file in the specified field. Position numbering starts from zero.
- One can use the EWRead method to obtain an array of file names for the attached fields in the specified field. The sequence of file names in the array will correspond to the positions of the files.
- Only one file per call is retrieved.

- In general, you use `EWRetrieveAttachedAsSOAPAttachment` when you know in advance the identifiers of the records to retrieve. The client application may use the likes of the `EWSelectFromTable` call to obtain record identifiers beforehand or take the identifiers from the `id` field of the data structures and the linking classes.

Basic Steps for Retrieving Attached Files

Retrieving files from records involves the following basic steps:

1. Determine the id of the record you want to update. You may want to use the `EWSelectFromTable` or `EWSearchTable` calls to get the identifiers of the records based on some search condition; or get the identifier from a previous `EWCreate` call; or get the id of a linked record from the linking class after performing `EWRead` or `EWUpdate`.
2. Determine the position of the file in the field.
3. Call `EWRetrieveAttachedAsSOAPAttachment`.
4. Process the results.

Example Task

In MyKB knowledgebase as user A, retrieve the first file named B.txt currently attached in the field Additional Files of case #456 in the current directory.

The task is completed by performing the following steps:

1. Login to MyKB with "A" and "password" and English as the local language.
2. Read the record to get the list of the files.
3. Find the position of the file.
4. Invoke the retrieve file call.
5. Logout.

Sample Code - Java

You can generate sample Web Services code for any table by selecting **Setup > Tables > (Edit Table) > API > Download Sample**.

```

public int retrieve() throws Exception {
    EWServiceAPI binding = new EWServiceAPIServiceLocator().getDemo();
    try {
        String sessionId = binding.EWLogin("MyKB", "A", "password", "en");
        WSCase wsCase = (WSCase) binding.EWRead(sessionId, "case", 456);
        String[] fileNames = wsCase.getAdditional_Files();
        int n = 0;
        if (fileNames!=null) {
            for (int i = 0; i < fileNames.length; i++) {
                if ("B.txt".equals(fileNames[i])) {
                    binding.EWRetrieveAttachedAsSOAPAttachment("case", 456,
                        "additional_files", i);
                    Object[] attachments = binding.getAttachments();
                    AttachmentPart attachmentPart = (AttachmentPart) attachments[0];
                    final InputStream inputStream =
                        attachmentPart.getDataHandler().getInputStream();
                    BufferedOutputStream bos =
                        new BufferedOutputStream(new FileOutputStream("B.txt"));
                    int b;
                    while ((b = inputStream.read()) != -1) bos.write(b);
                    bos.flush();
                    bos.close();
                    break;
                }
            }
        }
        return;
    } finally {
        binding.EWLogout(sessionId);
    }
}

```

Arguments

Name	Type	Description
sessionId	String	Session token.
tableName	String	The name of the table where the record is.
id	long	The identifier of the record to attach the files to.
fieldName	String	The name of the field to attach the files to.
position	int	The position of the file to be retrieved.

Response

The file as a SOAP attachment.

Faults

EWSessionException - client not logged in or the session has expired; client should re-login.

EWPermissionException - user used to create the session lacks the sufficient privileges to read the record.

EWWrongDataException - client has supplied the wrong data, for instance ID cannot be found.

EWOperationException - the operation has been blocked by an Agiloft function, for example a table-level lock.

EWIntegrityException - the specified table cannot be found or its primary key cannot be identified.

EWUnexpectedException - an unexpected exception has happened; user should report this for investigation.

EWSearchTable

Executes a pre-configured named Saved Search against the specified table and returns an array of record data for the records that match the search.

Syntax

```
EWWSBaseUserObject[] os = ew.EWSearchTable(String sessionId, String  
tableName, String[] fieldNames, String searchName);
```

Usage

Use the EWSearchTable call to search for records in the specified table based on a Saved Search pre-configured in the GUI.

Rules and Guidelines

When querying records, consider the following rules and guidelines:

- The username that was used to obtain the specified session token must have sufficient access rights to read individual records within the specified table. Please verify specific permissions via **Setup > Access > Manage Groups > (Edit Group) > Table > (Select Table) > Permissions**.
- Agiloft allows specifying fine-grained access permissions on the field level. The username that was used to obtain the specified session token must have sufficient access rights to be able to read field content. Please verify specific permissions via **Setup > Access > Manage Groups > (Edit Group) > Table > (Edit Table) > Field Permissions**.
- This call does not return records that have been deleted.
- This method never returns null. In the case when no records are found an empty array is returned.
- *Special note on memory management:* As WS integration implies pass-by-value semantics, the memory allocated for the resulting array will be released once the data is sent to the client and server-side JVM's garbage collector considers it eligible for discarding.

- If the client-side environment is the one with a garbage collector, the memory used by client-side array will be cleared once the client-side garbage collector considers it eligible.
- When the client environment uses explicit memory management, the client is responsible for freeing up the used memory explicitly.
- When using the EWSearchTable method, only the fields explicitly listed in the call are read. Within the values returned for the fields requested explicitly, a null value, where nillable="true", means the actual null value was retrieved. However, the rest of the fields – those not listed in the fieldNames array – will also appear on the wire as nillable="true" elements due to limitations of the underlying Web Services stack.
- To read all fields, use "*" string constant as the only element in the fieldNames array.

Linked Fields

Values for the fields that are imported into the target table from the donor table as a part of a Linked Fields set are available via special linking classes.

In WSDL a Linked Fields set takes form of a `DAO_Dao3_Link<N>` field in the complex data structure that corresponds to the target table, where N is a sequential number assigned automatically at the time of the set creation (e.g. `WSCase.DAO_Dao3_Link3`).

As a value such fields can take one or more `WS<Table1><Table2>_Dao3_Link<N>` data structures - linking classes, where Table1 is the target table and Table2 is the donor table of the Linked Fields relationship, for example `WSCaseTeams_Dao3_Link3`.

Unfortunately, at this moment one has to rely on investigating the actual sets of fields inside the classes to trace the fields, which are visible in the Field Wizard in the GUI, back to the main object property which is not visible in the GUI.

Non-source values for Linked Field sets that allow them are present directly in the table itself and additionally to those in the Linking Classes, if the link was in fact forged.

Choice Fields

The values for choice columns are returned as instances of the enumerated types described in the WSDL.

The original text values have undergone the following transformations:

1. Spaces replaced by "_"
2. Dashes replaced by "MINUS"
3. Pluses replaced by "PLUS"
4. Prefixed with "OPTION_"
5. Converted to Upper Case

One has to perform the reverse transformation to get to the text value.

Unsupported Types of Fields

Related tables and embedded search results are not supported by the SOAP interface.

Basic Steps for Searching Records with a Pre-configured Saved Search

1. Create a Saved Search in the GUI.
2. Perform the call using the name of the search as the last parameter.
3. Handle the results, specifically the situations where there are no elements, one element, or more than one element in the returned array.

Example Task

In MyKB knowledgebase, as user A, find all cases assigned to the user used to login. Return summaries as a String array.

The task is completed by performing the following steps:

1. Login to MyKB with "A" and "password" and English as the local language.
2. Search for cases using My Assigned search.
3. Logout.

Sample Code - Java

You can generate sample Web Services code for any table by selecting **Setup > Tables > (Edit Table) > API > Download Sample**.

```
public String[] search() throws Exception {  
    EWSERVICEAPI binding = new EWSERVICEAPIServiceLocator().getDemo();
```

```

"case",
    try {
        String sessionId = binding.EWLogin("MyKB", "A", "password", "en");
        EWWSBaseUserObject[] records = binding.EWSearchTable(sessionId,

        new String[] {"summary"}, "My Assigned");
        String[] result = new String[records.length];
        for (int i=0; i<records.length; i++) {
            result[i] = records[i].getSummary();
        }
        return result;
    } finally {
        binding.EWLogout(sessionId);
    }
}

```

Arguments

Name	Type	Description
sessionId	String	Session token.
tableName	String	The name of the table where the query has to be performed.
fields	String array	The list of fields to read.
savedsearch	String	The name of the Saved Search to run.

Response

An array of the records as descendants of EWWSBaseUserObject - a complex structure described in WSDL.

Faults

EWSessionException - client not logged in or the session has expired; client should re-login.

EWPermissionException - user used to create the session lacks the sufficient privileges to run the query.

EWWrongDataException - client has supplied the wrong data.

EWOperationException - the operation has been blocked by an Agiloft function, for example a table-level lock.

EWIntegrityException - specified table cannot be found or its primary key cannot be identified.

EWUnexpectedException - an unexpected exception has occurred; the admin user should report this for investigation.

EWSearchTablePaginated

Executes a pre-configured named Saved Search against the specified table and returns an array of record data for the records that match the search. This operation variant allows iteration through results page by page, also called pagination.

Syntax

```
EWWSBaseUserObject[] os = ew.EWSearchTablePaginated(String sessionId, String  
tableName, String[] fieldNames, String searchName, int page, int  
limit);
```

Usage

Use the EWSearchTablePaginated call to search for records in the specified table based on a Saved Search pre-configured in the GUI. Specify page and limit parameters to retrieve data page by page.

Rules and Guidelines

When querying records, consider the following rules and guidelines:

- The username that was used to obtain the specified session token must have sufficient access rights to read individual records within the specified table. Please verify specific permissions via **Setup > Access > Manage Groups > [Edit Group] > Table > [Select Table] > Permissions**.
- Agiloft allows specifying fine-grained access permissions on the field level. The username that was used to obtain the specified session token must have sufficient access rights to be able to read field content. Please verify specific permissions via **Setup > Access > Manage Groups > [Select Group to Edit] > Table > [Select Table] > Field Permissions**.
- This call does not return records that have been deleted.
- This method never returns null. In the case when no records are found an empty array is returned.

- Special note on memory management: As WS integration implies pass-by-value semantics, the memory allocated for the resulting array will be released once the data is sent to the client and server-side JVM's garbage collector considers it eligible for discarding.
- If the client-side environment is the one with a garbage collector, the memory used by client-side array will be cleared once the client-side garbage collector considers it eligible.
- When the client environment uses explicit memory management, the client is responsible for freeing up the used memory explicitly.
- When using the `EWSearchTablePaginated` method, only the fields explicitly listed in the call are read. Within the values returned for the fields requested explicitly, a null value, `nullable="true"`, means the actual null value was retrieved. However, the rest of the fields – those not listed in the `fieldNames` array – will also appear on the wire as `nullable="true"` elements due to limitations of the underlying Web Services stack.
- To read all fields, use `"*"` string constant as the only element in the `fieldNames` array.
- Page numbers start with 0 (zero). A limit – or page size – value of zero indicates "all records" and so all records are returned on page 0 when limit 0 is specified; otherwise, with a non-zero page number, an empty result is returned – no records.
- When a page is not found, empty result is returned.
- A call using pagination always returns a page worth of data. However, to truly take advantage of pagination all other parameters must remain the same. If the table, fields, saved search or limit on a subsequent call is different from the previous one, the underlying query is automatically rebuilt and re-run.
- As such only one "open" query is allowed per client session. If the client requires multiple queries to be iterated in parallel, the client code should create multiple sessions using the same login credentials.
- This method doesn't support multi-threading – the client is responsible for restricting access to a single thread i.e. one client thread = one session = one open query.
- The query will remain "open" until the session is closed – an explicit logout is performed by the client or session timeout occurs – or the application server discards the underlying low-level objects as a result of resource management. In this case the query will be rebuilt and re-run automatically on the next call.
- If the query is rebuilt and re-run, the result of the next call may not be fully consistent with the results of the previous call, as the underlying data may have changed, for example a record was deleted, or the sort order for the search in question was changed. Therefore, the dataset may appear to have "holes" and/or the logical page boundaries may shift when iterating the query page by page.

Linked Fields

Values for the fields that are imported into the target table from the donor table as a part of a Linked Fields set are available via special linking classes.

In WSDL a Linked Fields set takes form of a DAO_Dao3_Link field in the complex data structure that corresponds to the target table, where N is a sequential number assigned automatically at the time of the set creation, for example WSCase.DAO_Dao3_Link3.

As a value such fields can take one or more WS_Dao3_Link data structures - linking classes, where Table1 is the target table and Table2 is the donor table of the Linked Fields relationship, for example WSCaseTeams_Dao3_Link3.

Unfortunately, at this moment one has to rely on investigating the actual sets of fields inside the classes to trace the fields, which are visible in the Field wizard in the GUI, back to the main object property, which is not visible in the GUI.

Non-source values for Linked Field sets that allow them are present directly in the table itself and additionally to those in the Linking Classes, if the link was in fact forged.

Choice Fields

The values for choice columns are returned as instances of the enumerated types described in the WSDL.

The original text values have undergone the following transformations:

1. Spaces replaced by "_"
2. Dashes replaced by "MINUS"
3. Pluses replaced by "PLUS"
4. Prefixed with "OPTION_"
5. Converted to Upper Case

Perform the reverse transformation to get to the text value.

Unsupported Types of Fields

Embedded search results are not supported by the SOAP interface.

Basic Steps for Searching Records with a Pre-configured Saved Search

1. Create a Saved Search in the GUI.

2. Perform the call using the name of the search as the "searchName" parameter.
3. Specify page and limit values.
4. Handle the results, specifically the situations where there are no elements, one element, or more than one element in the returned array.
5. Iterate to the next page as required.

Example Task

In MyKB knowledgebase, as user A, find all cases assigned to the user used to login. Return the first 40 summaries as two String arrays, up to 20 elements each.

The task is completed by performing the following steps:

1. Login to MyKB with "A" and "password" and English as the local language.
2. Search for cases using My Assigned search.
3. Logout.

Sample Code - Java

You can generate sample Web Services code for any table by selecting **Setup > Tables > [Edit Table] > API > Download Sample**.

```
public String[][] search() throws Exception {
    EWSERVICEAPI binding = new EWSERVICEAPIServiceLocator().getDemo()
    try {
        String sessionId = binding.EWLogin("MyKB", "A", "password", "en");
        String[][] result = new String[2][];
        EWSBaseUserObject[] records1 = binding.EWSearchTablePaginated(sessionId,
"case",
        new String[] {"summary"}, "My Assigned", 0, 20);
        result[0] = new String[records1.length];
        for (int i=0; i<records1.length; i++) {
            result[0][i] = records1[i].getSummary();
        }
        EWSBaseUserObject[] records2 = binding.EWSearchTablePaginated(sessionId,
"case",
        new String[] {"summary"}, "My Assigned", 1, 20);
        result[1] = new String[records2.length];
        for (int j=0; j<records2.length; j++) {
            result[1][j] = records2[j].getSummary();
        }
    }
}
```



```

        return result;
    } finally {
        binding.EWLogout(sessionId);
    }
}

```

Arguments

Name	Type	Description
sessionId	String	Session token.
tableName	String	The name of the table where the query has to be performed.
fields	String array	The list of fields to read.
searchName	String	The name of the Saved Search to run.
page	int	The page number.
limit	int	The records per page limit, or page size.

Response

An array of the records as descendants of EWWSBaseUserObject - a complex structure described in WSDL.

Faults

EWSessionException - client not logged in or the session has expired; client should re-login.

EWPermissionException - user used to create the session lacks the sufficient privileges to run the query.

EWWrongDataException - client has supplied the wrong data.

EWOperationException - the operation has been blocked by an Agiloft function, for example a table-level lock.

EWIntegrityException - specified table cannot be found or its primary key cannot be identified.

EWUnexpectedException - an unexpected exception has occurred; the admin user should report this for investigation.

EWSearchTableWithQuery

Executes a preconfigured named Saved Search against the specified table and/or an ad-hoc query and returns an array of record data for the records that match the criteria.

Syntax

```
EWWSBaseUserObject[] os = ew.EWSearchTableWithQuery(String sessionId, String  
    tableName, String[] fieldNames, String searchName, String query);
```

Usage

Use EWSearchTableWithQuery call to search for records in the specified table based on a Saved Search pre-configured in the GUI and/or to filter the records with an ad-hoc query.

Rules and Guidelines

When querying records, consider the following rules and guidelines:

- The username that was used to obtain the specified session token must have sufficient access rights to read individual records within the specified table. Please verify specific permissions via **Setup > Access > Manage Groups > (Edit Group) > Table > (Edit Table) > Permissions**.
- Agiloft allows specifying fine-grained access permissions on the field level. The username that was used to obtain the specified session token must have sufficient access rights to be able to read field content. Please verify specific permissions via **Setup > Access > Manage Groups > (Edit Group) > Table > (Edit Table) > Field Permissions**.
- This call does not return records that have been deleted.
- This method never returns null. In the case when no records are found an empty array is returned.
- **Special note on memory management:** As WS integration implies pass-by-value semantics, the memory allocated for the resulting array will be released once the data is sent to the client and the server-side JVM's garbage collector considers it eligible for discarding.
- If the client-side environment is the one with a garbage collector, the memory used by client-side array will be cleared once the client-side garbage collector considers it eligible.

- When the client environment uses explicit memory management, the client is responsible for freeing up the used memory explicitly.
- When using EWSearchTableWithQuery method, only the fields explicitly listed in the call are read. Within the values returned for the fields requested explicitly, a null value, where nillable="true", means the actual null value was retrieved. However, the rest of the fields – those not listed in the fieldNames array – will also appear on the wire as nillable="true" elements due to limitations of the underlying Web Services stack.
- To read all fields, use "*" string constant as the only element in the fieldNames array.
- The main difference from using the EWSelectAndRead method, which uses an SQL where clause, is that ad-hoc queries operate on a higher level, can use logical field names, are capable of recognizing choice values and high-level relationships between table fields, and can use advanced and time-based criteria.
- If the query doesn't parse according to the grammar, an attempt is made to parse the parameter value as a sequence of identifiers using a different grammar. If both fail, the parameter value is treated as a Full-Text Search query.
- The ad-hoc query grammar is described at the end of this section.

Steps for Searching Records with a Saved Search and/or Ad-hoc Query

1. Optionally create a Saved Search in the GUI.
2. Perform the call using the name of the search and additionally filter the results with an ad-hoc query or use the ad-hoc query without the search.
3. Handle the results, specifically the situations where there are no elements, one element, or more than one element in the returned array.

Example Task

In MyKB knowledgebase, as user A, find all cases assigned to the user used to login with low priority. Return summaries as a String array.

Completion of the task is performed by the following steps:

1. Login to MyKB with "A" and "password" and English as the local language.
2. Search for cases using My Assigned search, additionally filtering by low priority.
3. Logout

Sample Code - Java

You can generate sample Web Services code for any table by selecting **Setup > Table > [Select Table to Edit] > API > Download Sample**.

```
public String[] search() throws Exception {
    EWServiceAPI binding = new EWServiceAPIServiceLocator().getDemo();
    try {
        String sessionId = binding.EWLogin("MyKB", "A", "password", "en");
        EWWSBaseUserObject[] records = binding.EWSearchTableWithQuery(sessionId, "case",
            new String[] {"summary"}, "My Assigned", "Priority=Low");
        String[] result = new String[records.length];
        for (int i=0; i<records.length; i++) {
            result[i] = records[i].getSummary();
        }
        return result;
    } finally {
        binding.EWLogout(sessionId);
    }
}
```

Arguments

Name	Type	Description
sessionId	String	Session token
tableName	String	The name of the table where the query has to be performed.
fieldNames	String array	The list of fields to read
searchName	String	The optional name of the Saved Search to run
query	String	The ad-hoc query

Response

An array of the records as descendants of EWWSBaseUserObject - a complex structure described in WSDL.

Faults

EWSessionException - client not logged in or session has expired; client should re-login.

EWPermissionException - user used to create the session lacks sufficient privileges to run the query.

EWWrongDataException - client has supplied wrong data.

EWOperationException - the operation has been blocked by an Agiloft function, for example a table-level lock.

EWIntegrityException - specified table cannot be found or its primary key cannot be identified.

EWUnexpectedException - an unexpected exception has happened; the admin user should report this for investigation.

Informal Grammar Description for Ad-hoc Queries

Field names are usually column labels as seen in the UI. However, DB and User column names are accepted too. Both field names and values may be surrounded by single quotes ('). If they contain spaces or some weird characters then quoting is mandatory. For example:

Example	Result
Priority=Low	OK
'Priority'='Low'	OK
Bug Priority=Low	Invalid
'Bug Priority'=Low	OK
'Bug Priority'=Very Low	Invalid
'Bug Priority'='Very Low'	OK

Simple criteria

Simple criteria has the form of

```
<field name><operator><value>
```

where operator is one of:

Operator	Definition
=	equals
!=	not equals
~=	contains
!~=	doesn't contain
>=	greater or equals
<=	lesser or equals
>	greater
<	lesser
<<	included by
!<<	not included by

The included/not included by operators (<<, !<<) expect a comma-separated list of values, without spaces, at the right-hand side of the equation and checks if field value is included (or not-included) in this list. In other words, Priority << High,Low is a short-hand for Priority=High || Priority=Low, where || is the OR operator, as described below.

Logical criteria

Allows to combine other criterias using AND and OR operators. '&&' is AND, '||' is OR.

Operator precedence

Expression is evaluated from left to right, braces may be used for grouping. For example 'A && B || C' means '(A && B) || C'.

Time-based criteria

Allows to set relative date constraints. The form is <field name><operator><mode><value>, where operator is one of =,!=,<,>,<=,>=, mode is either '-', which means 'old', '+', which means 'in the future' or '#', which means 'absolute'. 'value' is an integer followed by a single character:

--	--

m	minute
h	hour
w	week
M	month
y	year

Examples:

Date<-1y	'Date' is less than one year old
Date>=+10m	'Date' is greater or equal than 10 minutes in the future
Duration=#2h	'Duration' is exactly two hours

Currently, more complex expressions like 'two years, one month and three hours' are not supported.

Advanced criteria

Advanced criteria has the form of <field name>:<from>-><to> and means 'field <field name> has changed from *from* to *to*'. Either from or to – but not both at the same time – may hold '?' meaning 'any value'.

This criteria searches through record history, and thus the history column must exist and must track changes to the specified field. All simple and time-based criteria have implicit 'now' flag set, which means that they will match the current record state, not the state when advanced criteria has been satisfied. In the other words, if we have a record with the following modification history, with the bottom state being the most recent:

State=Open, Priority=Low State=Closed, Priority=Low State=Closed, Priority=High

Then 'State:Open->Closed && Priority=Low' will not find it, but 'State:Open->Closed && Priority=High' will.

More examples

- Status=Open && ('Assigned To' = john || 'Assigned To' = jane)
- (Priority>High || Summary ~= Urgent) && State:Closed->Reopened
- OS=Windows,Linux && 'Modification Date' < -1y

EWSearchTableWithQueryPaginated

eExecutes a preconfigured named Saved Search against the specified table and/or an *ad hoc* query and returns an array of record data for the records that match the criteria. This operation variant allows pagination - iteration through results page by page.

Syntax

```
EWWSBaseUserObject[] os = ew.EWSearchTableWithQueryPaginated(String sessionId,  
String tableName, String[] fieldNames, String searchName, String query, int page,  
int limit);
```

Usage

Use EWSearchTableWithQueryPaginated call to search for records in the specified table based on a Saved Search pre-configured in the GUI and/or to filter the records with an ad-hoc query. Specify page and limit parameters to retrieve data page by page.

Rules and Guidelines

When querying records, consider the following rules and guidelines:

- The username that was used to obtain the specified session token must have sufficient access rights to read individual records within the specified table. Please verify specific permissions via **Setup > Access > Manage Groups > [Edit Group] > Table > [Select Table] > Permissions**.
- Agiloft allows specifying fine-grained access permissions on the field level. The username that was used to obtain the specified session token must have sufficient access rights to be able to read field content. Please verify specific permissions via **Setup > Access > Manage Groups > [Select Group to Edit] > Table > [Select Table] > Field Permissions**.
- This call does not return records that have been deleted.
- This method never returns null. In the case when no records are found an empty array is returned.

- Special note on memory management: As WS integration implies pass-by-value semantics, the memory allocated for the resulting array will be released once the data is sent to the client and the server-side JVM's garbage collector considers it eligible for discarding.
- If the client-side environment is the one with a garbage collector, the memory used by client-side array will be cleared once the client-side garbage collector considers it eligible.
- When the client environment uses explicit memory management, the client is responsible for freeing up the used memory explicitly.
- When using EWSearchTableWithQueryPaginated method, only the fields explicitly listed in the call are read. Within the values returned for the fields requested explicitly, a null value - nillable="true" - means the actual null value was retrieved. However, the rest of the fields - those not listed in the fieldNames array - will also appear on the wire as nillable="true" elements due to limitations of the underlying Web Services stack.
- To read all fields, use "*" string constant as the only element in the fieldNames array.
- The main difference from using the EWSelectAndRead method, which uses an SQL where clause, is that ad-hoc queries operate on a higher level, can use logical field names, are capable of recognizing choice values and high-level relationships between table fields, and can use advanced and time-based criteria.
- If the query doesn't parse according to the grammar, an attempt is made to parse the parameter value as a sequence of identifiers using a different grammar. If both fail, the parameter value is treated as a Full-Text Search query.
- The ad-hoc query grammar is described at the end of this section.
- Page numbers start with 0 (zero). A limit (page size) value of zero indicates "all records" and so all records are returned on page 0 when limit 0 is specified; otherwise, with a non-zero page number, an empty result is returned (no records).
- When a page is not found, empty result is returned.
- A call using pagination always returns a page worth of data. However, to truly take advantage of pagination all other parameters must remain the same. If the table, fields, saved search, query or limit on a subsequent call is different from the previous one, the underlying query is automatically rebuilt and re-run.
- As such only one "open" query is allowed per client session. If the client requires multiple queries to be iterated in parallel, the client code should create multiple sessions using the same login credentials.
- This method doesn't support multi-threading – the client is responsible for restricting access to a single thread i.e. one client thread = one session = one open query.
- The query will remain "open" until the session is closed - an explicit logout is performed by the client or session timeout occurs - or the application server discards the underlying low-level objects as a result of resource management. In this case the query will be rebuilt and re-run automatically on the next call.
- If the query is rebuilt and re-run, the result of the next call may not be fully consistent with the results of the previous call, as the underlying data may have changed; for example a record was deleted, or the sort order for the search in question was changed. Therefore, the dataset may appear to have "holes" and/or the logical page boundaries may shift when iterating the query page by page.

Unsupported Types of Fields

Embedded search results are not supported by the SOAP interface.

Steps for Searching Records with a Saved Search and/or Ad-hoc Query

1. Optionally create a Saved Search in the GUI.
2. Perform the call using the name of the search and additionally filter the results with an ad-hoc query or use the ad-hoc query without the search.
3. Handle the results, specifically the situations where there are no elements, one element, or more than one element in the returned array.
4. Iterate to the next page as required.

Example Task

In MyKB knowledgebase, as user A, find all cases assigned to the user used to login with low priority. Return the first 40 summaries as two String arrays, up to 20 elements each.

Completion of the task is performed by the following steps:

1. Login to MyKB with "A" and "password" and English as the local language.
2. Search for cases using My Assigned search, additionally filtering by low priority.
3. Logout

Sample Code - Java

```
public String[] search() throws Exception {
    EWSERVICEAPI binding = new EWSERVICEAPIServiceLocator().getDemo();
    try {
        String sessionId = binding.EWLogin("MyKB", "A", "password", "en");
        String[][] result = new String[2][];
        EWSBaseUserObject[] records1 = binding.EWSearchTableWithQueryPaginated
```

```

(sessionId, "case",
    new String[] {"summary"}, "My Assigned", "Priority=Low", 0, 20);
result[0] = new String[records1.length];
for (int i=0; i<records1.length; i++) {
    result[0][i] = records1[i].getSummary();
}
EWSBaseUserObject[] records2 = binding.EWSearchTableWithQueryPaginated
(sessionId, "case",
    new String[] {"summary"}, "My Assigned", "priority=Low", 1, 20);
result[1] = new String[records2.length];
for (int j=0; j<records2.length; j++) {
    result[0][j] = records2[j].getSummary();
}
return result;
} finally {
    binding.EWLogout(sessionId);
}
}

```

You can generate sample Web Services code for any table by selecting **Setup > Table > [Select Table to Edit] > API > Download Sample**.

Arguments

Name	Type	Description
sessionId	String	Session token
tableName	String	The name of the table where the query has to be performed.
fieldNames	String array	The list of fields to read
searchName	String	The optional name of the Saved Search to run
query	String	The ad-hoc query
page	int	The page number
limit	int	The records per page limit (page size)

Response

An array of the records as descendants of EWSBaseUserObject - a complex structure described in WSDL.

Faults

EWSessionException - client not logged in or session has expired; client should re-login.

EWPermissionException - user used to create the session lacks sufficient privileges to run the query.

EWWrongDataException - client has supplied wrong data.

EWOperationException - the operation has been blocked by an Agiloft function, e.g. table-level lock.

EWIntegrityException - specified table cannot be found or its primary key cannot be identified.

EWUnexpectedException - an unexpected exception has happened; the admin user should report this for investigation.

Informal Grammar Description for Ad-hoc Queries

Field names are usually column labels as seen in the UI. However, DB and User column names are accepted too. Both field names and values may be surrounded by single quotes ('). If they contain spaces or some weird characters then quoting is mandatory. For example:

Example	Result
Priority=Low	OK
'Priority'='Low'	OK
Bug Priority=Low	Invalid
'Bug Priority'=Low	OK
'Bug Priority'=Very Low	Invalid
'Bug Priority'='Very Low'	OK

Simple criteria

Simple criteria has the form of

```
<field name><operator><value>
```

where operator is one of:

Operator	Definition
=	equals
!=	not equals
~=	contains
!~=	doesn't contain
>=	greater or equals
<=	lesser or equals
>	greater
<	lesser
<<	included by
!<<	not included by

The (not) included by operator expects a comma-separated list of values without spaces at the right-hand side and checks if field value is included into this list. In other words `Priority << High,Low` is a short-hand for `Priority=High || Priority=Low`, where `||` is OR, as described below.

Logical criteria

Allows to combine other criterias using AND and OR operators. '&&' is AND, '||' is OR.

Operator precedence

Expression is evaluated from left to right, braces may be used for grouping. For example '`A && B || C`' means '`(A && B) || C`'.

Time-based criteria

Allows to set relative date constraints. The form is `<field name><operator><mode><value>`, where operator is one of `=,!=,<,>,<=,>=`, mode is either `'-'`, which means 'old', `'+'`, which means 'in the future' or `'#'`, which means 'absolute'. 'value' is an integer followed by a single character:

--	--

m	minute
h	hour
w	week
M	month
y	year

Examples:

Date<-1y	'Date' is less than one year old
Date>=+10m	'Date' is greater or equal than 10 minutes in the future
Duration=#2h	'Duration' is exactly two hours

Currently, more complex expressions like 'two years, one month and three hours' are not supported.

Advanced criteria

Advanced criteria has the form of <field name>:<from>-><to> and means 'field field name has changed from *from* to *to*'. Either from or to but not both at the same time may hold '?' meaning 'any value'.

This criteria searches through record history, and thus the history column must exist and must track changes to the specified field. All simple and time-based criterias have implicit 'now' flag set, which means that they will match the current record state, not the state when advanced criteria has been satisfied. In the other words, if we have a record with the following modification history, with the bottom state being the most recent:

State=Open, Priority=Low State=Closed, Priority=Low State=Closed, Priority=High

Then 'State:Open->Closed && Priority=Low' will not find it, but 'State:Open->Closed && Priority=High' will.

More examples

- Status=Open && ('Assigned To' = john || 'Assigned To' = jane)
- (Priority>High || Summary ~= Urgent) && State:Closed->Reopened
- OS=Windows,Linux && 'Modification Date' < -1y

EWSelectAndRead

Executes a query against the specified table and returns an array of record data for the records that match the specified criteria.

Syntax

```
EWWSBaseUserObject[] os = ew.EWSelectFromTable(String sessionId, String  
        tableName, String whereClause);  
or for the table-specific call:  
WSCase[] cases = ew.EWCreateAndRead_WSCase(String sessionId, String whereClause)
```

Usage

Use EWSelectFromTable call to search for records in the specified table.

The SQL query supplied should only include the "where" clause from a construct similar to:

```
select primary-key-field-name from table-name where ...
```

Rules and Guidelines

When querying records, consider the following rules and guidelines:

- The username that was used to obtain the specified session token must have sufficient access rights to read individual records within the specified table. Please verify specific permissions via **Setup > Access > Manage Groups > [Edit Group] > Table > [Select Table] > Permissions**.
- Agiloft allows specifying fine-grained access permissions on the field level. The username that was used to obtain the specified session token must have sufficient access rights to be able to read field content. Please verify specific permissions via **Setup > Access > Manage Groups [Edit Group] > Table > [Select Table] > Field Permissions**.
- This call does not return records that have been deleted.

- The queries operate on the database level and should be written in terms of names visible in the Agiloft GUI in the Table wizard in the dbname column. Also see EWGetChoiceLineId for special handling of choice values.
- This method never returns null. In the case when no records are found an empty array is returned.
- Special note on memory management: As WS integration implies pass-by-value semantics, the memory allocated for the resulting array will be released once the data is sent to the client and the server-side JVM's garbage collector considers it eligible for discarding.
- If the client-side environment is one with a garbage collector, the memory used by client-side array will be cleared once the client-side garbage collector considers it eligible,
- When the client environment uses explicit memory management, the client is responsible for freeing up the used memory explicitly.
- It is possible to limit the number of records returned through means available to the underlying database used by your instance of Agiloft e.g. "limit 0,200" for MySQL will limit the number of the returned records to 200.
- It is not possible currently to influence the sequence in which records are returned. They are returned in the effectively arbitrary default order returned by the underlying database.
- When using EWSelectAndRead method all fields of the resulting records are read and passed to the client.

Linked Fields

Values for the fields that are imported into the target table from the donor table as a part of a Linked Fields set are available via special linking classes.

In WSDL a Linked Fields set takes form of DAO_Dao3_Link<N> field in the complex data structure that corresponds to the target table, where N is a sequential number assigned automatically at the time of the set creation, such as WSCase.DAO_Dao3_Link3.

As a value such fields can take one or more WS<Table1><Table2>_Dao3_Link<N> data structures - linking classes, where Table1 is the target table and Table2 is the donor table of the Linked Fields relationship, such as WSCaseTeams_Dao3_Link3.

Unfortunately, at this moment one has to rely on investigating the actual sets of fields inside the classes to trace the fields, visible in the Field wizard in the GUI, back to the main object property - not visible in the GUI.

Non-source values for Linked Field sets that allow them are present directly in the table itself and additionally to those in the Linking Classes, if the link was in fact forged.

Choice Fields

The values for choice columns are returned as instance(s) of the enumerated types described in the WSDL.

The original text values have undergone the following transformations:

1. Spaces replaced by "_"
2. Dashes replaced by "MINUS"
3. Pluses replaced by "PLUS"
4. Prefixed with "OPTION_"
5. Converted to upper case

One has to perform the reverse transformation to get to the text value.

Unsupported types of fields

Related tables and embedded search results are not supported by the SOAP interface.

Basic Steps for Selecting and Reading Records in One Call

1. Compose the whereClause.

Note: "Prepared" statements are not supported, all parameters have to be passed inside of the whereClause string.

2. Perform the call.
3. Handle results, specifically the situations where there are no elements, one element, or more than one element in the returned array.

Example Task

In MyKB knowledgebase, as user A, find all cases assigned to John Doe. Return summaries as a String array.

The task is completed by performing the following steps:

1. Login to MyKB with "A" and "password" and English as the local language.
2. Search for cases assigned to John Doe.
3. Logout

Sample Code - Java

```
public String[] select() throws Exception {
    EWSERVICEAPI binding = new EWSERVICEAPIServiceLocator().getDemo();
    try {
        String sessionId = binding.EWLogin("MyKB", "A", "password", "en");
        EWWSBaseUserObject[] records = binding.EWSelectAndRead(sessionId,
                                                                "case", "assignee='John Doe'");
        String[] result = new String[records.length];
        for (int i=0; i<records.length; i++) {
            result[i] = records[i].getSummary();
        }
        return result;
    } finally {
        binding.EWLogout(sessionId);
    }
}
```

You can generate sample Web Services code for any table by selecting **Setup > Tables > [Edit Table] > API > Download Sample**.

Arguments

Name	Type	Description
sessionId	String	Session token
tableName	String	The name of the table where the query has to be performed - only for generic methods
where	string	The where clause of the SQL select construct

Response

An array of the records as descendants of EWWSBaseUserObject - a complex structure described in WSDL.

Faults

EWSessionException - client not logged in or session has expired; client should re-login.

EWPermissionException - user used to create the session lacks sufficient privileges to run the query.

EWWrongDataException - client has supplied wrong data.

EWOperationException - the operation has been blocked by an Agiloft function, e.g. table-level lock.

EWIntegrityException - specified table cannot be found or its primary key cannot be identified.

EWUnexpectedException - an unexpected exception has happened; the admin user should report this for investigation.

EWSelectFromTable

Executes a query against the specified table and returns record identifiers that match the specified criteria.

Syntax

```
long ids[] = ew.EWSelectFromTable(String sessionId, String  
    tableName, String whereClause);
```

Usage

Use the EWSelectFromTable call to search for records in the specified table.

The SQL query supplied should only include the "where" clause from a construct similar to:

```
select primary-key-field-name from table-name where ...
```

Rules and Guidelines

When querying records, consider the following rules and guidelines:

- The queries operate on the database level and should be written in terms of names visible in the Agiloft GUI in the Table wizard in the dbname column. Also see EWGetChoiceLineId for special handling of choice values.
- This method never returns null. In the case where no records are found an empty array is returned. Special note on memory management: As WS integration implies pass-by-value semantics, the memory allocated for the resulting array will be released once the data is sent to the client and the server-side JVM's garbage collector considers it eligible for discarding.
- If the client-side environment is one with a garbage collector, the memory used by client-side array will be cleared once the client-side garbage collector considers it eligible.
- When the client environment uses explicit memory management, the client is responsible for freeing up the used memory explicitly.

- It is possible to limit the number of records returned through means available to the underlying database used by your instance of Agiloft e.g. "limit 0,200" for MySQL will limit the number of the returned records to 200).
- It is not possible currently to influence the sequence in which records are returned. They are returned in the effectively arbitrary default order returned by the underlying database.

Basic Steps for Selecting Records

1. Compose the whereClause. Note: "prepared" statements are not supported, all parameters have to be passed inside the whereClause string.
2. Perform the call.
3. Handle results, specifically the situations where there are no elements, one element, or more than one element in the returned array.

Example Task

In MyKB knowledgebase, as user A, find all cases assigned to John Doe.

The task is completed by performing the following steps:

1. Log in to MyKB with "A" and "password", English as the local language.
2. Search for cases assigned to John Doe.
3. Log out.

Sample Code - Java

You can generate a sample Web Services code for any table by selecting **Setup > Tables > (Edit Table) > API > Download Sample**.

```
public long[] select() throws Exception {
    EWServiceAPI binding = new EWServiceAPIServiceLocator().getDemo();
    try {
        String sessionId = binding.EWLogin("MyKB", "A", "password", "en");
        long[] ids = binding.EWSelectFromTable(sessionId, "case",
            "assignee='John Doe'");
    }
}
```

```

return ids;
} finally {
binding.EWLogout(sessionId);
}
}

```

Arguments

Name	Type	Description
sessionId	String	Session token.
tableName	String	The name of the table where the query has to be performed.
where	string	The whereClause of the SQL select construct.

Response

The identifiers of the records that match the specified criteria.

Faults

EWSessionException - client not logged in or the session has expired; client should re-login.

EWPermissionException - user used to create the session lacks the sufficient privileges to run the query.

EWWrongDataException - client has supplied the wrong data.

EWOperationException - the operation has been blocked by an Agiloft function, for example a table-level lock.

EWIntegrityException - the specified table cannot be found or its primary key cannot be identified.

EWUnexpectedException - an unexpected exception has happened; user should report this for investigation.

EWUpdate

Updates a record in your Agiloft knowledgebase.

Syntax

```
EWWSBaseUserObject res = ew.EWUpdate(String sessionId, String  
    table,  
    EWWSBaseUserObject obj);  
or for the table-specific call:  
WSCase wsCase = ew.EWUpdate_WSCase(String sessionId, WSCase case);
```

Usage

Use EWUpdate to update records in a table, such as Case or Contact, in your knowledgebase. The client application passes the tableName and the map with data to be applied to the record in the generic call and just the map in the table-specific call.

The EWUpdate call is analogous to the UPDATE statement in SQL.

Rules and Guidelines

When updating records, consider the following rules and guidelines:

- The username that was used to obtain the specified session token must have sufficient access rights to update individual records within the specified table. Please verify specific permissions via **Setup > Access > Manage Groups > [Edit Group] > Table > [Select Table] > Permissions**.
- Agiloft allows specifying fine-grained access permissions on the table field level. The username that was used to obtain the specified session token must have sufficient access rights to set every field being specified in the data object supplied in the call on update of the record. Consider this if you are using the record data obtained from or just used in another call like EWRead or EWCreate - read and create permissions of a particular user may not match the updated ones. Please verify specific permissions via **Setup > Access > Manage Groups > [Edit Group] > Table > [Select Table] > Field Permissions**.

- The data object must contain the identifier of the record being modified.
- When updating the records in the table it is required to specify explicitly the type of the record. This is especially important in the case when the record is created for the true subtype, such as Contacts.Customer. In this case the table specified in the call will be "Contacts" and the type of the record will be "Customer". For the top-level subtypes the type of the record is equal to the table name. Please consult the Table Wizard for the specific names of the tables/subtypes in your knowledgebase, available via **Setup > Tables** and editing the relevant table.
- Certain fields can be defined to have default values. If permissions allow these may be overwritten by data supplied in the call.
- A record created by another user, via API or via GUI, may have some required fields not filled as the user didn't have adequate privileges. Any required fields which do not have a value at the time of the call or a preconfigured default value must have a value supplied if they fall within the access privileges of the logged in user that is triggering the EWUpdate call.
- The API and WSDL distinguish between an empty (null) value set explicitly and a value not set at all in the EWUpdate call - i.e. the one that should remain unchanged.
- Fields that are present in the table directly are filled in as simple values.
- Some environments like .NET require a special property set for simple-type fields to properly handle empty values - <name>Specified = true for .NET.
- Agiloft allows one to establish relationships between the records in different tables via Linked Fields and ensures the data integrity once the links are forged.
- Fields from the Linked Fields relationships that allow values not present in the donor tables are present both in the table, for the case when the value is non-source, and the linking classes, for the case when the value is truly imported and the link is forged.

Unsupported Types of Fields

Related tables and embedded search results are not supported by the SOAP interface.

Basic Steps for Updating Records

Updating records involves the following basic steps:

1. Determine the id of the record you want to update. You may want to use the EWSelectFromTable call to get the identifiers of the records based on some search condition or get the identifier from a previous EWCreate call, or get the id of a linked record from the linking class after performing an EWRead.

2. If you do not have it already from one of the previous calls, construct an instance of the complex type that corresponds to the table you are trying to update. Fill in the identifier and those fields that need updating.
3. Call EWUpdate.
4. Process the results.

Example Task

In MyKB knowledgebase, as user A, update a given case, setting the field Escalate to Support Staff to Yes (in this particular KB this triggers a round-robin assignment to the members of the Support Team). Return the name of the employee who is assigned to the case as a result.

This task is completed by performing the following steps:

1. Login to MyKB with "A" and "password" and English as the local language.
2. Set the Escalate to Support Staff field to Yes.
3. Update the record.
4. Read the name of the assignee.
5. Logout.

Sample Code - Java

```
public String update(WSCase wsCase) {
    EWServiceAPI binding = new EWServiceAPIServiceLocator().getMyKB();
    String sessionId = binding.EWLogin("MyKB", "A", "password", "en");
    wsCase.setEscalate_To_Support_Staff(WSCase.WSChoice_Yes_No.OPTION_YES);
    WSCase result = (WSCase) binding.EWUpdate(sessionId, "case", wsCase);
    WSCaseTeams_Dao3_Link3 assigneeLink = result.getDAO_Dao3_Link3();
    String assignee;
    if (assigneeLink!=null) {
        assignee = assigneeLink.getAssigned_To();
    } else {
        assignee = null;
    }
    binding.EWLogout(sessionId);
    return assignee;
}
```

You can generate a sample Web Services code for any table by selecting **Setup > Tables > [Edit Table] > API > Download Sample**.

Arguments

Name	Type	Description
sessionId	String	Session token.
tableName	String	The name of the table where the record is to be updated - only for generic methods.
obj	EWWSBaseUserObject	The descendant of EWWSBaseUserObject - one of the complex types described in the WSDL that correspond to the tables on the Agiloft side.

Response

Updated record data as a descendant of EWWSBaseUserObject - a complex structures of the same type as the object argument.

Faults

EWSessionException - client not logged in or the session has expired; client should re-login.

EWPermissionException - username used to create the session lacks the sufficient privileges to perform record modification.

EWWrongDataException - data causes an index or constraint violation; key, name of problematic column and the value causing the problem as parameters.

EWOperationException - modification operation has been blocked by an Agiloft function.

EWIntegrityException - the specified table cannot be found or its primary key cannot be identified.

EWUnexpectedException - an unexpected exception has happened; user should report this for investigation.

SOAP Examples and Hints

This section contains some general hints and examples for various client-side platforms.

- PHP SOAP Client
- SoapUI
- IIS Integration

PHP SOAP Client

```
$soapSession = new SoapClient("http://host/ewws/KBName/EWSserviceAPIv2?wsdl",
array("exceptions" => false,
"location"=>"http://host/ewws/KBName/EWSserviceAPIv2"));
$sess_id = $soapSession->EWlogin("KBName","user","password","en");
echo "$sess_id\n";
$ids = $soapSession->EWSelectFromTable($sess_id, "case", "1=1");
print_r($ids);
$map = $soapSession->EWRead($sess_id, "case", 521);
print_r($map);
```

SoapUI

When using generic methods a specific descendant of the EWWSBaseUserObject structure needs to be passed. This is achieved by adding the **xsi:type** specialization to the tags defining the structure in the method call parameters.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:tsc="http://KBname.api.ws.agiloft.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<soapenv:Header/>
<soapenv:Body>
<tsc:EWScreate>
<String_1>d57d</String_1>
<String_2>case</String_2>
<EWWSBaseUserObject_3 xsi:type="tsc:WSCase">
<summary>Example 1</summary>
</EWWSBaseUserObject_3>
</tsc:EWScreate>
</soapenv:Body>
</soapenv:Envelope>
```

IIS Integration

If you are integrating Agiloft with Microsoft Internet Information Services (IIS), please modify the web.config file of the IIS to include the `<httpErrors existingResponse="PassThrough" />` tag:

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
<system.webServer>
<httpErrors existingResponse="PassThrough" />
</system.webServer>
</configuration>
```

This will allow proper system exceptions to pass through the IIS API, rather than the default IIS errors. Explanation on the other available values is given at <http://stackoverflow.com/questions/31040671/what-does-existingresponse-passthrough-mean-in-iis>

REST Interface

Agiloft supports REST-style invocations that correspond to CRUD operations: [Create](#), [Read](#), [Update](#), [Delete](#). Additionally limited SQL select functionality is supported via [Select](#), and saved search and ad hoc queries via [Search](#). Attachments can be managed via via the [Attach](#), [RemoveAttached](#), and [RetrieveAttached](#) calls. Login and Logout functions are provided for session security.

This allows you to automate certain actions from external agents such as browser forms, JavaScript/AJAX.

For CRUD operations two invocation styles are supported:

- Pure REST, where CRUD operations map directly to HTTP methods.
- A fallback GET/POST method that can be used instead, as not all user-agents may support all HTTP methods.
- [Select](#) is only available via GET/POST.

Operation	/ewws /REST/...	GET/POST	Returns
Create	POST	/ewws/EWCreate	ID of the newly created record
Read	GET	/ewws/EWRead	encoded record information
Update	PUT	/ewws /EWUpdate	encoded record information after update
Delete	DELETE	/ewws/EWDelete	does not return anything
Select		/ewws/EWSelect	a list of record identifiers and a length of that list
Login		/ewws/EWLogin	a session token, expiration time, and authentication scheme
Logout		/ewws/EWLogout	does not return anything

URL Conventions

The following general conventions apply to how the URL is constructed:

For pure REST, where the `/id` part is omitted for Create (POST), but used for all others:

```
/ewws/REST/{kbName}/{table}[/id]?$login={login}&password={password}&lang={lang}&...
```

For the fallback Get/Post interface:

```
/ewws/{operation}?$KB={kbName}&$table={table}&$login={login}&password={password}&lang={lang}&...
```

The parameters of the POST request can be inserted into the body of the request to conceal the user credentials.

The body (for POST and PUT) or the rest of the URL string should contain the parameter name/value pairs, as per operation specification.

Example

```
GET {server name}/ewws/REST/Demo/Company
/123?$login=user&$password=123&$lang=en&id=123
```

Or

```
POST {server name}/ewws
/EWRead?$KB=Demo&$table=Company&$id=123&$login=user&$password=123&$lang=en&id=1
```

Notes

- Both knowledgebase names and table names are case sensitive when using the REST interface.
- To find the correct case for your table, go to **Setup > Tables**, select your table, click Edit, and look for the Logical Table Name. This is the name and capitalization you should use when accessing that particular table.

Return Values

Return values are returned in an encoded form suitable for the JavaScript `eval()` operation to be applied. Extended characters (like ö) are returned in a UTF-encoded format. After this, the values can be accessed from local variables. Fields with empty values are not returned.

To avoid interfering with variables that may already exist in the client script or document, all table column names in the variables that result from the `eval()` call are prefixed with `EWREST_`. As such, what is returned is escaped using JavaScript rules. The content type of the field is irrelevant for the escaping.

```
EWREST_company_name='Agiloft';
EWREST__1794_full_name=' agiloft.com Admin';
EWREST_website_url=' http://www.agiloft.com'
EWREST_date_updated=' 21 8 06 15:18:43 PM';
EWREST_id=' 21';
```

-  You may want to consider using a JSON decorator to receive a JSON formatted stream instead, since JSON has more readily available parsers. Here is its syntax in a REST call:

```
https://<hostname>/ewws/EWRead/
json?$KB=KB&$table=<table>&$login=admin&$password=<pwd>$lang=en&id=<id>
```

In this case the return result would look like:

```
{"success":true,"message":"","result":{"...","company_name":"Agiloft","_1794_full_name":"Agiloft System","id":21}}
```

HTTP Status Codes

- 200: successful operation
- 400: wrong data specified in the request
- 403: operation not permitted with specified credentials
- 408: request timeout, user may attempt to retry the request
- 409: operation cannot be performed, usually means one of the functionalities has blocked it
- 500: any other problem

Delays

Each call via the REST interface has a delay inserted after the operation has completed. The delay is set to one second by default and is configurable via the global variable [Web Services Delay \(WSDelay\)](#).

- An operation on a record may invoke rules and other functions, which need to be allowed enough time and resources to complete.
- Additionally, the delay is needed because client applications could mistakenly use web services, resulting in a flood of requests.

Security

Use the Login and Logout operations to secure sessions initiated by REST.

- **Login:** Use this operation to authenticate KB credentials and return a token, which can then be used by following requests to avoid including login credentials in request URLs. Input the KB login and password as parameters, and the operation returns a token, expiration time, and authentication scheme to the client. The token can then be used in an Authorization request header, prefixed by the authentication scheme, instead of including the login and password parameters in following requests. By default, the Bearer authentication scheme is used, and the token expiration time is 15 minutes. You can adjust the expiration time by creating a `token_expires_in` global variable and setting the number of minutes, up to 60.
- **Logout:** Use this operation to terminate a session created by the Login operation. This terminates the session associated with the token passed in the Authorization header.

Examples

These examples show the process of creating the token, placing it in the request header, and then terminating the token session. With the token in the header, you can use functions like `Search` without passing in a login and password in the URL.

Usage	Headers	Example Text	Response
Login request		POST https://server/ewws/EWLogin? \$login=user&\$password=passwd& \$KB=Demo&\$lang=en	HTTP/1.1 200 C {"access_token": "eyJhbGciOiJIUzI1NiJ9.eyJkYXRhIjoiZW5kaXQyZm9udCIsImV4c... refresh_token": "...", expiration_time_in_minute": "expires_in": 5," authentication : Bearer "}
Functional request	Authorization: Bearer eyJhbGciOiJIUzI1NiJ9....	GET https://server/ewws/EWSearch? \$KB=Demo&\$table=body&\$lang=en& field=id&field=text&field=body&query=...	
Refresh token	Authorization: Bearer eyJhbGciOiJIUzI1NiJ9....	POST https://server/ewws /EWLogin?\$KB=Demo&\$lang=en&refresh_token=xYZsk ...	HTTP/1.1 200 C {"access_token": "eyJhbGciOiJIUzI1NiJ9.eyJkYXRhIjoiZW5kaXQyZm9udCIsImV4c... refresh_token": "...", expiration_time_in_minute": "expires_in": 5," authentication : Bearer "}
Logout request	Authorization: Bearer eyJhbGciOiJIUzI1NiJ9....	GET https://server/ewws/EWLogout	HTTP/1.1 200 C

Data Encoding

The following conventions are in place to encode aspects of a typical Agiloft knowledgebase:

Simple fields

Simple fields can be filled directly by setting the value for them.

```
... &first_name=John&last_name=Doe&...
```

Choice fields

Choice fields are encoded directly with their text values as seen in the GUI.

```
...&country=USA&...
```



For ad hoc queries, in the Select call choice values should be addressed via the ID values obtained from `GetChoiceLineId`.

Multi-choice fields

Multi-choice fields are encoded as multiple key/value pairs.

```
... &contactMethod=phone&contactMethod=email&...
```

Date, date-time and time fields

Date, date-time and time fields can be encoded with any of 3,275 formats currently supported. The system evaluates the possible formats sequentially and stops when parsing in one of the formats succeeds.

Please refer to the following document to see the list of supported date-time formats: [datetime.txt](#)

Elapsed time fields

can be encoded as "days:hours:minutes:seconds" e.g "0:1:35:15"6

Linked field relationships

If the linked field allows independent values these can be simply assigned to the columns in the main table:

```
...&company_name=Agiloft&...
```

To create a link based on the values of imported columns, you can use Query By Example, expressed with a colon ':' qualifier.

Example values have to be provided for one or more of the imported columns in the following way:

```
...&company_name=:Agiloft&...
```

Or

```
...&company_name=Company:Agiloft&...
```

where donor table name is required if the link type is "single field from multiple tables", but may be omitted in the case of a single donor table.

If the value contains the colon ':' symbol or the question mark '?' symbols, they have to be escaped with backward-slash "\" in the following way:

```
...&company_url=Company:http\://www.example.com&...
```

For more complex queries, possibly including sub-selects, aggregative functions and columns from the donor table that are not imported into the target, you can use a SQL query, expressed with a question mark '?' qualifier. The "where" clause follows the qualifier for the query that will be run against the donor table. The columns in the query have to be referred with their database names rather than logical ones.

Example

To look for Employee records where the Company currency is EUR, and not in the linked set, use the following search:

```
http://localhost:8080/ewws
/EWSearch?$login=admin&$password=qwerty&$lang=en&$table=contacts.
employees&$KB=Demo&query=_1576_company_name0=Company?currency~='EUR'
```

Multiple Values for the Linked Field

These are encoded as multiple key/value pairs:

```
...&company_name=Company:Agiloft& company_name=Company:SaaSWizard&...
```

File and Image Fields (Attached Files)

The REST interface accepts files in POST requests when used with **enctype="multipart/form-data"** encoding.

The name of the form field should match the name of the file or image field. Additionally a field *fieldName\$overwrite* can be specified with any value to instruct the REST interface to override the current data in the file or image field, rather than add.

Application clients can use [REST - Attach](#) operation instead with PUT HTTP method.

Decorators

REST calls allow use of three decorators:

- Asynchronous decorator
 - Can be applied to EWCreate, EWUpdate and EWDelete calls to do "fire-and-forget" type of call.
 - Should be used if the results normally returned by an operation are not important to the caller e.g. a scenario when a lot of records have to be created in the backend.

Use

```
/ewws/async/EWCreate?...  
or  
/ewws/EWCreate/.async?...
```

- Redirect decorator
 - Can be applied to all calls to have a HTTP redirect issued depending on the success of the operation. Useful the scenarios of integration with web sites. Please note the page to which redirect is performed will NOT receive any return data.

- Calls require two parameters to be specified: *\$exiturl* and *\$errorurl* - for redirecting in case of successful operation and in case of error respectively. Both parameters should be absolute URLs and URL encoded if necessary.

Use

```
/ewws/redirect/EWCreate?...&$exiturl=http%3A%2F%2Fwww.google.com&$errorurl=http%3A%2F%2Fwww.google.com%2F404
or
/ewws/EWCreate/.redirect?...&$exiturl=http%3A%2F%2Fwww.google.com&$errorurl=http%3A%2F%2Fwww.google.com%2F404
```

- JSON decorator produces a JSON formatted stream.

Use

```
/ewws/EWRead/.json?...
```

Please note that decorators can be chained. In the case of chaining they are applied from left to right.

Use

```
/ewws/redirect/EWCreate/.async?...&$exiturl=http%3A%2F%2Fwww.google.com&$errorurl=http%3A%2F%2Fwww.google.com%2F404
/ewws/async/EWCreate/.redirect?...&$exiturl=http%3A%2F%2Fwww.google.com&$errorurl=http%3A%2F%2Fwww.google.com%2F404
/ewws/redirect/async/EWCreate?...&$exiturl=http%3A%2F%2Fwww.google.com&$errorurl=http%3A%2F%2Fwww.google.com%2F404
/ewws/async/redirect/EWCreate?...&$exiturl=http%3A%2F%2Fwww.google.com&$errorurl=http%3A%2F%2Fwww.google.com%2F404
```

REST - Create

The EWCreate (POST) operation...

- Creates a record in the specified KB and table with values for the specified fields.
- Accepts the URL with parameters as per general URL conventions and record data encoded as per Data Encoding, both of which can be viewed in the [REST Interface](#).
- Supported Content-Type: application/x-www-form-urlencoded
- Returns the identifier of the newly created record.
- If this operation is used asynchronously, its status can be checked with [EWAsyncStatus](#).

 All parameters must be properly URL-encoded.

Example

Assume an instance of Agiloft is available on localhost, port 8080 and is called "Demo". Create a record for an employee named John Doe, that is assigned to the Service Manager group and Service Management Team, with login 'jdoe' and password 'password'.

Also, assume an instance of Agiloft is available on localhost.

- The Employee table is a sub-table of Contacts (logical table name = contacts.employees).
- To create the user we need to fill in the first_name, last_name, _login and password fields.
- We also need to create links with the groups and teams tables via linked fields that import multi-value fields groups and teams and a single-value field primary team.
- We will use the Query By Example capability and supply the names of the Service Manager group and Service Management Team.

The following request is issued:

```
http://localhost:8080/ewws/EWCreate?$KB=Demo&$stable=contacts.employees&$login=admin&$password=qwerty&$lang=en&first_name=John&last_name=Doe&
```

A result similar to the following will be returned in the case of successful record creation:

```
EWREST_id='353';
```

Here is an example of a JavaScript-based client that invokes the REST interface via AJAX:

```
function xmlhttpGet (strURL) {
    var xmlhttpReq=false;
    var self=this;
    // Mozilla/Safari
```

```

if (window.XMLHttpRequest) {
    try {
        netscape.security.PrivilegeManager.
            enablePrivilege("UniversalBrowserRead");
    } catch (e) {
        alert("Permission UniversalBrowserRead denied.");
    }
    self.xmlHttpRequest=new XMLHttpRequest();
} // IE
else if (window.ActiveXObject) {
    self.xmlHttpRequest=new ActiveXObject("Microsoft.xmlHTTP");
}
self.xmlHttpRequest.open('GET', strURL, true);
self.xmlHttpRequest.onreadystatechange=requestComplete;
self.xmlHttpRequest.send(null);
}
function requestComplete() {
    if (xmlHttpRequest.readyState==4||xmlHttpRequest.readyState=="complete") {
        eval (self.xmlHttpRequest.responseText);
        alert ("Id of new ticket"+EWREST_id);
    }
}
function main() {
    xmlhttpGet('http://localhost:8080/ewws/EWCreate?$KB=Demo&$table=contacts.
employees&$login=admin&$password=qwerty&$lang=en&first_name=John&last_name=Doe&_login

```

REST - Read

The EWRead REST operation:

- Implements the Read operation of the REST interface.
- Accepts the URL with parameters as per general URL conventions which can be viewed in the [REST Interface Overview](#).
- Supported Content-Type: application/x-www-form-urlencoded
- Returns the encoded data of the record.



- The URL must contain the identifier of the record.
- All parameters must be properly URL-encoded.

Example

Assume an instance of Agiloft is available on localhost, port 8080 and is called "Demo". The updated record can be seen in the [Update example](#).

The following request is issued:

```
http://localhost:8080/ewws/EWRead?$KB=Demo&$table=Contacts.
Employees&$login=admin&$password=qwerty&$lang=en&id=358
```

The following result will be returned:

```
EWREST_full_name='John Doe';
EWREST_first_name='John';
EWREST__1576_company_name0='IBM';
EWREST_f_group_0='Service Manager';
EWREST_id='358';
EWREST__106_sw_description='Service Management Team';
EWREST__login='jdoe';
EWREST_date_updated='Dec 27 2017 04:40:24';
EWREST_type='employees';
EWREST_date_created='Dec 27 2017 04:34:38';
EWREST_rep_email='example23@example.com';
EWREST_default_approval_title='Document Approval';
EWREST_last_name='Doe';
```

Here is an example for a JavaScript-based client that invokes the REST interface via AJAX:


```
function xmlhttpGet (strURL) {
    var xmlhttpReq=false;
    var self=this;
    // Mozilla/Safari
    if (window.XMLHttpRequest) {
        try {
            netscape.security.PrivilegeManager.
enablePrivilege("UniversalBrowserRead");
        } catch (e) {
            alert("Permission UniversalBrowserRead denied.");
        }
        self.xmlhttpReq=new XMLHttpRequest();
    }// IE
    else if (window.ActiveXObject) {
        self.xmlhttpReq=new ActiveXObject("Microsoft.xmlHTTP");
    }
    self.xmlhttpReq.open('GET', strURL, true);
    self.xmlhttpReq.onreadystatechange=requestComplete;
    self.xmlhttpReq.send(null);
}
function requestComplete() {
    if (xmlhttpReq.readyState==4||xmlhttpReq.readyState=="complete") {
        eval (self.xmlhttpReq.responseText);
        alert ("Opportunity"+EWREST_opportunity_name_0);    }
    }
function main() {
    xmlhttpGet('http://localhost:8080/ewws/EWRead?$KB=Demo&$table=Contacts.
Employees&$login=admin&$password=qwerty&$lang=en&id=358');
}
}
```

REST - Update

The EWUpdate REST operation:

- Updates the specified record, implementing the Update operation of the REST interface.
- Accepts the URL with parameters as per general URL and record data conventions, both of which can be viewed in the [REST Interface Overview](#).
- Supported Content-Type: application/x-www-form-urlencoded
- Returns the encoded data of the updated record.
- If this operation is used asynchronously, its status can be checked with [EWAsyncStatus](#).
- In place of the usual ID parameter, you may instead use \$searchSQL to identify the record to update, provided that the search returns exactly one record and no more. For example, if you have a unique ext_id field in Agiloft, you can use this to update a single record that has ext_id='a0B2c345' using &\$searchSQL=ext_id='a0B2c345' in the URL.

 All parameters must be properly URL-encoded.

Example

Assume an instance of Agiloft is available on localhost, port 8080 and is called "Demo". Update the record created in the [REST Create example](#) section by linking that employee to an opportunity record for company Agiloft. This operation was not included as part of the create example for illustrative purposes. The same criteria could have been supplied in the create request.

The following needs to be considered:

- The Employee table is a subtable of People.
- The link to Opportunities is created via a Linked Field that imports multi-value fields company_name and opportunity_name.
- We will use the Query By Example capability and supply the name of the company as an example.

The following request is issued:

```
http://localhost:8080/ewws/EWUpdate?$KB=Demo&$table=contacts.employees&$login=admin&$password=qwerty&$lang=en&id=358&_1576_company_name0=:IBM
```

The following result will be returned in the case of successful modification of the record:

```
EWREST_full_name='John Doe';
EWREST__1576_company_name0='IBM';
EWREST_f_group_0='Service Manager';
EWREST_id='358';
EWREST__106_sw_description='Service Management Team';
EWREST__login='jdoe';
```

```
EWREST_date_updated='Dec 27 2017 04:40:24';
EWREST_type='employees';
EWREST_date_created='Dec 27 2017 04:34:38';
EWREST_rep_email='example23@example.com';
EWREST_default_approval_title='Document Approval';
EWREST_last_name='Doe';
```

Here is an example of a JavaScript-based client that invokes the REST interface via AJAX:

```
function xmlhttpGet (strURL) {
    var xmlhttpReq=false;
    var self=this;
    // Mozilla/Safari
    if (window.XMLHttpRequest) {
        try {
            netscape.security.PrivilegeManager.
enablePrivilege("UniversalBrowserRead");
        } catch (e) {
            alert("Permission UniversalBrowserRead denied.");
        }
        self.xmlhttpReq=new XMLHttpRequest();
    } // IE
    else if (window.ActiveXObject) {
        self.xmlhttpReq=new ActiveXObject("Microsoft.xmlHTTP");
    }
    self.xmlhttpReq.open('GET', strURL, true);
    self.xmlhttpReq.onreadystatechange=requestComplete;
    self.xmlhttpReq.send(null);
}
function requestComplete() {
    if (xmlhttpReq.readyState==4||xmlhttpReq.readyState=="complete") {
        alert ("Update completed");
    }
}
function main() {
    xmlhttpGet('http://localhost:8080/ewws/EWUpdate?$KB=Demo&$table=contacts.
employees&$login=admin&$password=qwerty&$lang=en&id=358&_1576_company_name0=:IBM');
}
```

REST - Delete

The EWDelete REST operation:

- Deletes the specified record or records.
- Accepts the URL with parameters as per general URL conventions that can be viewed in the [REST Interface Overview](#). Additionally, the URL must contain one or more record identifiers and the delete rule name, and optionally, the substitute record identifiers.
- Supported Content-Type: application/x-www-form-urlencoded
- This call does not return anything when the operation is successful. The transaction completes only if all records are successfully deleted, otherwise it is rolled back.

 All parameters must be properly URL-encoded.

The **deleteRule** parameter defines one of the following strategies to be applied for dependent records:

- **ERROR_IF_DEPENDANTS** - operation fails when there are any dependent records.
- **APPLY_DELETE_WHERE_POSSIBLE** - tries to delete all dependent records. When delete cannot be done, an attempt to unlink the record is made.
- **DELETE_WHERE_POSSIBLE_OTHERWISE_UNLINK** - same as above.
- **APPLY_UNLINK** - tries to unlink dependent records.
- **UNLINK_WHERE_POSSIBLE_OTHERWISE_DELETE** - tries to unlink all dependent records; when unlink cannot be done an attempt to delete the record is made.
- **REPLACE_WITH_ANOTHER** - tries to link dependent records to the substitute one specified in replacementKeys.

The **subs** parameter is taken in consideration only if the deleteRule **REPLACE_WITH_ANOTHER** is specified and should contain identifiers of records from the same table to be used as substitutes. For each record dependent on the record being deleted, a record with a corresponding replacement key will become the parent one.

- 
- If the specific strategy fails the error message returned will suggest alternatives to be used.
 - If **APPLY_DELETE_WHERE_POSSIBLE** or **DELETE_WHERE_POSSIBLE_OTHERWISE_UNLINK** strategies are used and the configuration of the knowledgebase allows it, a special "Fast Delete" algorithm is used just as via the GUI.

Example

Assume an instance of Agiloft is available on localhost, port 8080 and is called "Demo". Delete the record updated in the [REST Update Example](#).

The following request is issued:

```
http://localhost:8080/ewws/EWDelete?$KB=Demo
&$table=Contacts.Employees&$login=admin&$password=qwerty
&$lang=en&id=358&deleteRule=APPLY_DELETE_WHERE_POSSIBLE
```

Here is an example for a JavaScript-based client that invokes the REST interface via AJAX:

```
function xmlhttpGet (strURL) {
    var xmlhttpReq=false;
    var self=this;
    // Mozilla/Safari
    if (window.XMLHttpRequest) {
        try {
            netscape.security.PrivilegeManager.
enablePrivilege("UniversalBrowserRead");
        } catch (e) {
            alert("Permission UniversalBrowserRead denied.");
        }
        self.xmlhttpReq=new XMLHttpRequest();
    }// IE
    else if (window.ActiveXObject) {
        self.xmlhttpReq=new ActiveXObject("Microsoft.xmlHTTP");
    }
    self.xmlhttpReq.open('GET', strURL, true);
    self.xmlhttpReq.onreadystatechange=requestComplete;
    self.xmlhttpReq.send(null);
}
function requestComplete() {
    if (xmlhttpReq.readyState==4||xmlhttpReq.readyState=="complete") {
        alert ("Delete completed");
    }
}
function main() {
    xmlhttpGet('http://localhost:8080/ewws/EWDelete?$KB=Demo
&$table=Contacts.Employees&$login=admin&$password=qwerty
&$lang=en&id=358
&deleteRule=APPLY_DELETE_WHERE_POSSIBLE');
}
```

REST - Select

The EWSelect REST operation:

- Performs a select on the specified table, implementing the Select operation of the REST interface.
- Accepts the URL with parameters as per general URL conventions which can be viewed in the [REST Interface Overview](#). Additionally, the URL must contain the where clause of the SELECT query to be performed. All parameters must be properly URL-encoded.
- Supported Content-Type : application/x-www-form-urlencoded
- This returns the number of records found, and an encoded data list of record IDs, if any were found.

Only one SQL statement is accepted per call. The API automatically and transparently applies security filters to prevent unauthorized access to information.

 These SQL queries operate on the database level and should be written in terms of dbname column names as visible in the in the Table Wizard in the GUI. Some values for high-level complex field types may have very different representation from what is seen in the GUI. For example, see [REST Interface - GetChoiceLineId](#) for the special handling of choice values.

The SQL query supplied should only include the 'where' clause from a construct similar to:

```
select primary-key-field-name from table-name where ...
```

This returns the number of records found, and an encoded data list of record ID, if any were found.

It is possible to limit the number of records returned through means available to the underlying database used by your instance of Agiloft . For example "limit 0,200" for MySQL will limit the number of the returned records to 200.

It is not possible currently to influence the sequence in which records are returned. They are returned in the effectively arbitrary default order returned by the underlying database. If you wish to impose a sort order on the record sequence, consider using a EWSearch call instead. This will allow you to use a saved search to define a sort order.

Example 1

Assume an instance of Agiloft is available on localhost, port 8080 and is called "Demo". List the Service Request records assigned to 'Ralph Knowles'.

The following request is issued:

```
http://localhost:8080/ewws
/EWSelect?$KB=Demo&$login=admin&$password=qwerty&$table=helpdesk_case&$lang=en&
```

If there are no records found, the following result will be returned:

```
EWREST_id_length = '0';
```

The following result will be returned in the case of three records being found:

```
EWREST_id_length = '3';
EWREST_id_0 = '150';
EWREST_id_1 = '169';
EWREST_id_2 = '325';
```

Example 2

Assume an instance of Agiloft is available on localhost, port 8080 and Demo KB. List the records with summaries containing word 'new' (summary like '%new%').

The following request is issued:

```
http://localhost:8080/ewws
/EWSelect?$KB=Demo&$login=admin&$password=qwerty&$table=helpdesk_case&$lang=en&
```

If there are no records found, the following result will be returned:

```
EWREST_id_length = '0';
```

The following result will be returned in the case of seven records being found:

```
EWREST_id_length = '7';
EWREST_id_0 = '145';
EWREST_id_1 = '147';
EWREST_id_2 = '148';
EWREST_id_3 = '149';
EWREST_id_4 = '151';
EWREST_id_5 = '298';
EWREST_id_6 = '318';
```

Here is an example for a JavaScript-based client that invokes the REST interface via AJAX:

```
function xmlhttpGet (strURL) {
    var xmlhttpReq=false;
    var self=this;
    // Mozilla/Safari
    if (window.XMLHttpRequest) {
        try {
            netscape.security.PrivilegeManager.
enablePrivilege("UniversalBrowserRead");
        } catch (e) {
            alert("Permission UniversalBrowserRead denied.");
        }
        self.xmlhttpReq=new XMLHttpRequest();
```

```

} // IE
else if (window.ActiveXObject) {
    self.xmlHttpRequest=new ActiveXObject("Microsoft.xmlHTTP");
}
self.xmlHttpRequest.open('GET', strURL, true);
self.xmlHttpRequest.onreadystatechange=requestComplete;
self.xmlHttpRequest.send(null);
}
function requestComplete() {
    if (xmlHttpRequest.readyState==4||xmlHttpRequest.readyState=="complete") {
        eval (self.xmlHttpRequest.responseText);
        alert ("Id of new ticket"+EWREST_id);
    }
}
function main() {
    xmlhttpGet('
http://localhost:8080/ewws
/EWSelect?$KB=Demo&$login=admin&$password=qwerty&$table=helpdesk_case&$lang=en&where=

```


REST - Search

The EWSearch REST operation:

- Runs a saved search or an ad hoc query on the specified table, implementing a search-like operation of the REST interface.
- Accepts the URL with parameters as per general URL conventions which can be viewed in the [REST Interface](#). The name of the saved search or an ad hoc query may be included in the call. This operation allows retrieval of results page by page (pagination) using "page" and "limit" parameters.
- Supported Content-Type : application/x-www-form-urlencoded
- Returns the number of records found, and encoded sets of field values - one for each record found. If pagination is being used, the number of records is in the context of the current page and the specified page size.
- Each set will include the ID and the fields used by the table and its subtables to define record ownership, such as requester_login in the examples below. Additional fields can be added to the list by specifying a series of 'field' parameters with logical field names.

 All parameters and names must be properly URL-encoded.

Ad Hoc Searches

In opposition to saved searches, ad hoc searches are limited to run-time queries using the 'query' clause. An example of such a REST query is shown below using the conditions `&query=summary~='test'%26%26priority=3`:

```
http://localhost:8080/ewws
/EWSearch?$KB=Demo&$table=case&$login=admin&$password=qwerty&$lang=en&query=summary~='test'%26%26priority=3
```

-  To search on multiple fields, use a single query parameter that contains all the fields. Connect each field using %26%26, which equates to "&" or and, or using %7C%7C, which equates to "|" or or. Surround each search value in single quotes, and if a field label contains spaces, surround that field label in single quotes as well.

For example: `query=Approver%3D'Denise%20Teller'%26%26'Date Previous Status Change'>'Aug 24 2021 00:00'`

Pagination

- Page numbers start with 0 (zero). A page size limit value of zero indicates "all records" and so all records are returned on page 0 when limit 0 is specified; otherwise with a non-zero page number an empty result is returned, meaning no records.
- When a page is not found, an empty result is returned.
- A call using pagination always returns a page worth of data. However, to truly take advantage of pagination all other parameters must remain the same. If the table, fields, saved search, query or limit on a subsequent call is different from the previous one, the underlying query is automatically rebuilt and re-run.
- As such only one "open" query is allowed per client session. If the client requires multiple queries to be iterated in parallel, the client code should create multiple sessions using the same login credentials.
- This method doesn't support multi-threading – the client is responsible for restricting access to a single thread; i.e. one client thread = one session = one open query.
- The query will remain "open" until the session is closed, that is an explicit logout is performed by the client or session timeout occurs, or the application server discards the underlying low-level objects as a result of resource management. In this case the query will be rebuilt and re-run automatically on the next call.
- If the query is rebuilt and re-run, the result of the next call may not be fully consistent with the results of the previous call, as the underlying data may have changed – for example, a record was deleted, or the sort order for the search in question was changed. Therefore, the dataset may appear to have "holes" and/or the logical page boundaries may shift when iterating the query page by page.
- At this time the REST interface creates a new session and performs an explicit logout for each call. As such, though pagination is available, the query will always be rebuilt and rerun. The ability to issue multiple REST calls within a single session, similar to the SOAP interface, is in development.

Examples of REST Searches

Example 1

Assume an instance of Agiloft is available on localhost, port 8080 and the knowledgebase is called 'Demo'. List the Service Request records that correspond to the saved search 'C: Status is Closed'. Additionally, only return those that have High priority.

The following request is issued:

```
http://localhost:8080/ewws
/EWSearch?$KB=Demo&$login=admin&$password=qwerty&$table=helpdesk_case&$lang=en&
```

If there are no records found, the following result will be returned:

```
EWREST_id_length = '0';
```

The following result will be returned in the case of four records being found.

 If no return fields are specified in the request, it will return the ID, type fields and requester login field, based on the record ownership defined for the Service Request table, based on the permissions definition for the Service Request table.

```
EWREST_length = '4';
EWREST_login_0='ewsystem';
EWREST_type_0='helpdesk_case';
EWREST_id_0='318';
EWREST_login_1='ewsystem';
EWREST_type_1='helpdesk_case';
EWREST_id_1='151';
EWREST_login_2='ewsystem';
EWREST_type_2='helpdesk_case';
EWREST_id_2='146';
EWREST_login_3='internal';
EWREST_type_3='helpdesk_case';
EWREST_id_3='145';
```

Example 2

Assume an instance of Agiloft is available on localhost, port 8080 and Demo KB. Now we want to retrieve particular fields.

The following request is issued:

```
http://localhost:8080/ewws
/EWSearch?$KB=Demo&$login=admin&$password=qwerty&$table=helpdesk_case&$lang=en&
```

If there are no records found, the following result will be returned:

```
EWREST_id_length = '0';
```

The following result will be returned in the case of four records being found:

```
EWREST_length = '4';
EWREST_summary_0='Here is a new service request with some tasks';
EWREST_priority_0='High';
EWREST_summary_1='New Employee Setup for Patricia Smith';
EWREST_priority_1='High';
EWREST_summary_2='Upgrading Our Software';
EWREST_priority_2='High';
EWREST_summary_3='Need New Wireless Card for Laptop';
EWREST_priority_3='High';
```

Example 3

Assume an instance of Agiloft is available on localhost, port 8080 and Demo KB. Now we want to retrieve particular fields in addition to the default ones as per Example 2.

We are retrieving data in pages 2 records at a time and are interested in the 2nd page only. Page numbers counts from 0.

The following request is issued:

```
http://localhost:8080/ewws
/EWSearch?$KB=Demo&$login=admin&$password=qwerty&$table=helpdesk_case&$lang=en&
```

If there are no records found, the following result will be returned:

```
EWREST_id_length = '0';
```

The following result is returned in the case of four records found in total, showing last two ones:

```
EWREST_length = '2';
EWREST_summary_0='Upgrading Our Software';
EWREST_priority_0='High';
EWREST_summary_1='Need New Wireless Card for Laptop';
EWREST_priority_1='High';
```

Here is an example for a JavaScript-based client that invokes the REST interface via AJAX:

```
function xmlhttpGet (strURL) {
    var xmlhttpReq=false;
    var self=this;
    // Mozilla/Safari
    if (window.XMLHttpRequest) {
        try {
            netscape.security.PrivilegeManager.
enablePrivilege("UniversalBrowserRead");
        } catch (e) {
            alert("Permission UniversalBrowserRead denied.");
        }
        self.xmlhttpReq=new XMLHttpRequest();
    } // IE
    else if (window.ActiveXObject) {
        self.xmlhttpReq=new ActiveXObject("Microsoft.xmlHTTP");
    }
    self.xmlhttpReq.open('GET', strURL, true);
    self.xmlhttpReq.onreadystatechange=requestComplete;
    self.xmlhttpReq.send(null);
}
function requestComplete() {
    if (xmlhttpReq.readyState==4 || xmlhttpReq.readyState=="complete") {
```

```

        eval (self.xmlHttpRequest.responseText);
        alert ("Id of new ticket"+EWREST_id);
    }
}
function main() {
    xmlhttpGet('
http://localhost:8080/ewws
/EWSearch?$KB=Demo&$login=admin&$password=qwerty&$table=helpdesk_case&$lang=en&search

```

REST - GetChoiceLineId

The EWGetChoiceLineId REST operation:

- Obtains the internal identifier that corresponds to a choice value for use in SQL-based expressions.
- Accepts the URL with parameters as per general URL conventions which can be viewed in the [REST Interface Overview](#). Additionally, the URL must contain the name of the choice field and the choice list element value.
- Supported Content-Type: application/x-www-form-urlencoded
- This returns the identifier of the choice list element suitable to use in EWSelect.

 All parameters including the choice text value must be properly URL-encoded.

Use EWGetChoiceLineId to obtain the internal identifier that corresponds to the choice text value for further use in SQL-based expressions. This is needed for the "where" parameter of EWSelectFromTable or the query text when forging links to other tables via Linked Field sets.

SQL-based expressions are evaluated on the database level where choice values are stored as identifiers. These identifiers change if the knowledgebase is copied or re-imported, or the values in the choice list are re-created. It is therefore advisable to use the

Use the EWGetChoiceLineId call to obtain the ID value at runtime immediately before the SQL call.

 If the factors that trigger the change of the internal choice identifiers, for instance knowledgebase copy or re-import, or choice list values are re-created, are detected on the client, then the values returned by this call can be safely cached on a per KB basis between multiple calls.

Example

Assume an instance of Agiloft is available on localhost, port 8080 and is called "Demo". Obtain the identifier for choice value 'High' for the field 'Priority' in the table 'Case'.

The following request is issued:

```
http://localhost:8080/ewws
/EWGetChoiceLineId?$KB=Demo&$login=admin&$password=qwerty&$table=case&$lang=en&
```

If there were no matching choice list values found, the HTTP code 400 will be returned with a brief explanation of the problem.

The following result will be returned in the case of the text value being successfully translated into a numeric identifier:

```
EWREST_choiceLineId = '1';
```

Here is an example for a Javascript-based client that invokes the REST interface via AJAX:

```
function xmlhttpGet (strURL) {
    var xmlhttpReq=false;
    var self=this;
    // Mozilla/Safari
    if (window.XMLHttpRequest) {
        try {
            netscape.security.PrivilegeManager.
enablePrivilege("UniversalBrowserRead");
        } catch (e) {
            alert("Permission UniversalBrowserRead denied.");
        }
        self.xmlhttpReq=new XMLHttpRequest();
    } // IE
    else if (window.ActiveXObject) {
        self.xmlhttpReq=new ActiveXObject("Microsoft.xmlHTTP");
    }
    self.xmlhttpReq.open('GET', strURL, true);
    self.xmlhttpReq.onreadystatechange=requestComplete;
    self.xmlhttpReq.send(null);
}
function requestComplete() {
    if (xmlhttpReq.readyState==4 || xmlhttpReq.readyState=="complete") {
        eval (self.xmlhttpReq.responseText);
        alert ("Id of new ticket"+EWREST_id);
    }
}
function main() {
    xmlhttpGet('http://localhost:8080/ewws/EWGetChoiceLineId?$KB=Demo
&$table=case&$login=admin&$password=qwerty
&$lang=en&field=priority&value=High');
}
```

REST - Attach

The EWAttach REST operation:

- Attaches the file passed in the body of the request to the specified field of the record identified by the supplied primary key.
- Accepts the URL with parameters as per general URL conventions which can be viewed in the [REST Interface Overview](#). Additionally, the URL must contain the identifier of the record, the name of the field to attach the file to and the file name to use. The file content is passed in the body of the request.
- Supported Content-Type: multipart/form-data
- Returns the current number of files attached to the specified field.



- All parameters must be properly URL-encoded.
- At present browsers do not support sending PUT requests. This REST operation is intended for the use of application clients. If you need to attach a file from a browser, use the regular POST requests with EWCreate or EWUpdate operations, and specify **enctype="multipart/form-data"** in the browser HTML form.

Example

Assume an instance of Agiloft is available on localhost, port 8080 and is called "Demo". Attach a file to the field "someField" in record 1234 of table "someTable" with a file name as testfile.zip.

The request has to be issued via the PUT HTTP method where the body contains the content of the file to be attached.

```
http://localhost:8080/ewws
/EWAttach?$KB=Demo$table=someTable&$login=admin&$password=qwerty&id=1234&field=
```

The following result will be returned when the file is attached:

```
EWREST_someField.length='1';
```

Example code

```
<form method="post" action="https://demo.agiloft.com/ewws/EWCreate" enctype="
multipart/form-data">
<input name="$KB" value="Demo" />
<input name="$table" value="case" />
<input name="$lang" value="en" />
<input name="$login" value="admin" />
```



```
<input name="$password" value="qwerty" />  
<input type="file" name="inbound_attachments"/>  
<input type="submit" value="submit" />  
</form>
```

REST - Remove Attachment

The EWRemoveAttachment REST operation:

- Removes the attached file specified by the position in the named field of the record identified by the supplied key.
- Accepts the URL with parameters as per general URL conventions which can be viewed in the [REST Interface Overview](#). The URL must contain the identifier of the record, the name of the field to remove the file from and the position of the file in the field.
- Supported Content-Type: application/x-www-form-urlencoded
- Returns the number of files remaining attached in the field.

 All parameters must be properly URL-encoded.

Here is an example for a JavaScript-based client that opens the REST interface via AJAX:

```
function xmlhttpGet (strURL) {
    var xmlhttpReq=false;
    var self=this;
    // Mozilla/Safari
    if (window.XMLHttpRequest) {
        try {
            netscape.security.PrivilegeManager.
                enablePrivilege("UniversalBrowserRead");
        } catch (e) {
            alert("Permission UniversalBrowserRead denied.");
        }
        self.xmlhttpReq=new XMLHttpRequest();
    } // IE
    else if (window.ActiveXObject) {
        self.xmlhttpReq=new ActiveXObject("Microsoft.xmlHTTP");
    }
    self.xmlhttpReq.open('GET', strURL, true);
    self.xmlhttpReq.onreadystatechange=requestComplete;
    self.xmlhttpReq.send(null);
}
function requestComplete() {
    if (xmlhttpReq.readyState==4||xmlhttpReq.readyState=="complete") {
        eval (self.xmlhttpReq.responseText);
        alert ("Id of new ticket"+EWREST_id);
    }
}
function main() {
    xmlhttpGet('http://localhost:8080/ewws
/EWRemoveAttachment?$KB=Demo&$table=someTable&$login=admin&$password=qwerty&id=1234&f
```

REST - Retrieve Attachment

The EWRetrieve REST operation:

- Returns the attached file specified by position from the named field in the record identified by the supplied key.
- Accepts the URL with parameters as per general URL conventions which can be viewed in the [REST Interface Overview](#).
- Returns the Content-Type used when attaching the original file.
- Returns the file content in the body of the response via a GET or POST HTTP method.



- The URL must contain the identifier of the record, the name of the field to retrieve the file from and the position of the file in the field.
- All parameters must be properly URL-encoded.

Here is an example of a JavaScript-based client that invokes the REST interface via AJAX:

```
function xmlhttpGet (strURL) {
    var xmlhttpReq=false;
    var self=this;
    // Mozilla/Safari
    if (window.XMLHttpRequest) {
        try {
            netscape.security.PrivilegeManager.
                enablePrivilege("UniversalBrowserRead");
        } catch (e) {
            alert("Permission UniversalBrowserRead denied.");
        }
        self.xmlhttpReq=new XMLHttpRequest();
    } // IE
    else if (window.ActiveXObject) {
        self.xmlhttpReq=new ActiveXObject("Microsoft.xmlHTTP");
    }
    self.xmlhttpReq.open('GET', strURL, true);
    self.xmlhttpReq.onreadystatechange=requestComplete;
    self.xmlhttpReq.send(null);
}
function requestComplete() {
    if (xmlhttpReq.readyState==4 || xmlhttpReq.readyState=="complete") {
        alert ("A file has been returned");
    }
}
function main() {
```

```
xmlhttpGet('http://localhost:8080/ewws  
/EWRetrieve?$KB=Demo&$table=someTable&$login=admin&$password=qwerty&id=1234&field=som
```

REST - Login

The EWLogin function is used to create JSON Web Tokens (JWT). These are access tokens used to retrieve data with REST. With JWT, the user credentials are used only once, when the user requests the access token. After a token is generated, the same token is used to authenticate the session until the token expires or is closed out.

Tokens must be refreshed before they expire. The default expiration time for the tokens is 15 minutes, but this value can be changed by creating a Text or Long Integer global variable in your KB named token_expires_in.

Generating JSON Web Tokens

In order to get an access token, use the /ewws/EWLogin method. The response is provided in JSON format.

Request Description	Request Response
Type of REST request - POST HTTP-headers - Content-Type: plain/text URL - /ewws/EWLogin Request parameters (they can be filled to request body): \$KB - URL-encoded KB name \$login - URL-encoded login \$password - URL-encoded password \$lang - URL-encoded language; by default - en	Type - JSON Fields: access_token - access token refresh_token - token for refreshing non expired access token expiration_time_unit - time unit; by default - minute expires_in - expiration time in units; by default is 15 minutes authentication_scheme - authentication scheme; by default - Bearer

Request Example

```
$ curl -is -X POST --header "Content-Type: plain/text" "https://your.server.com/ewws/EWLogin?%24KB=Demo&%24login=admin&%24password=*****&%24lang=en"
```

```
HTTP/2 200
server: nginx
Date: Fri, 24 Jan 2020 16:39:54 GMT
content-length: 482
strict-transport-security: max-age=31536000; includeSubDomains
x-frame-options: SAMEORIGIN
strict-transport-security: max-age=31536000; includeSubDomains
x-frame-options: SAMEORIGIN
strict-transport-security: max-age=31536000; includeSubDomains
{"access_token":"XXeyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiIxNTJfYWRtaW4iLCJyb2x1IjoiUkVTVCIsc2lnY5jZSI6IjQ0MTM4NzAiLCJleHAiOiJlE10TY3M
```

Using JSON Web Tokens

To use the generated access token, add the Authorization header to your request by combining the `authentication_scheme` and `access_token` values on one line. These values come from the response received when generating the token.

If you received the values in the example above, the header would be:

Authorization: Bearer XXeyJhbGciOiJIUzI1NiJ9.

eyJzdWIiOiIxNTJfYWRtaW4iLCJyb2xlIjoiaUkVTVCIzInNlYW5jZSI6IjQ0MTM4NzAiLCJleHAiOjE1OTY3M

Example Header

```
$curl -is -X GET --header "Authorization: Bearer XXeyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiIxNTJfYWRtaW4iLCJyb2xlIjoiaUkVTVCIzInNlYW5jZSI6IjQ0MTM4NzAiLCJleHAiOjE1OTY3M"
```

Refreshing JSON Web Tokens

To refresh an access token, use the `/ewws/EWLogin` method.

Request Description	Request Response
Type of REST request - POST HTTP-headers - Authorization URL - /ewws/EWLogin Request parameters (they can be filled to request body): \$KB - URL-encoded KB name \$lang - URL-encoded language; by default - en Request body: refresh_token - your refresh token	Type - JSON Fields: access_token - access token refresh_token - token for refreshing non expired access token expiration_time_unit - time unit; by default - minute expires_in - expiration time in units; by default is 15 minutes authentication_scheme - authentication scheme; by default - Bearer

Refresh Example

```
$ curl -is -X POST --header "Authorization: Bearer XXeyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiIxNTJfYWRtaW4iLCJyb2xlIjoiaUkVTVCIzInNlYW5jZSI6IjQ0MTM4NzAiLCJleHAiOjE1OTY3M"
```

JWT Logout Operations

To close out the session associated with a token, use the `/ewws/EWLogout` method.

Request Description	Request Response
Type of REST request - POST or GET HTTP-headers - Authorization URL - /ewws/EWLogout Request parameters (they can be filled in request body): \$KB - URL-encoded KB name \$lang - URL-encoded language; by default - en	Response code 200 (OK)

Logout Example

```
$ curl -is -X GET --header "Authorization: Bearer XXeyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiIyX2FkbWluIiwicm9sZSI6IlJFU1QiLCJzZWFuY2UiOiI0MiIsImV4cCI6MTU3OTg4NjMzNywia
```

REST - Action Button

The EWActionButton REST operation:

- Executes an Action as an asynchronous call. The status of asynchronous calls can be checked with [EWAsyncStatus](#).
- Must use a POST request.
- Accepts the URL with parameters as per general URL conventions which can be viewed in the [REST Interface Overview](#). The URL must contain the table name, the ID of the record, and the name of the field that contains the action button.
- Supported Content-Type: application/x-www-form-urlencoded
- Returns the record ID and callback ID.

 All parameters must be properly URL-encoded.

Here is an example using the cUrl utility:

```
curl -is -X POST "http://your.server.com/ewws/async/EWActionButton?%24KB=Demo&%24login=admin&%24password=****&%24lang=en&%24table=case&name=ab_field&id=82"
HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: charset=UTF-8; charset=UTF-8
Content-Length: 48
Date: Fri, 14 Aug 2020 10:12:53 GMT

EWREST_id='82';
EWREST_EWCALLBACK_ID='10100_1';
```


REST - Async Status

The EWAsyncStatus REST operation:

- Checks the execution status of an asynchronous call, such as REST - Action Button, REST - Update, or REST - Create.
- Must use a POST or GET request.
- Accepts the URL with parameters as per general URL and record data conventions, both of which can be viewed in the [REST Interface Overview](#). The URL must include the table name and the callback identifier received by the asynchronous operation being checked.
- Supported Content-Type: application/x-www-form-urlencoded
- Returns the response code for the call's execution status:
 - 200: Operation completed successfully.
 - 201: The request is queued for execution.
 - 202: Operation in progress.
 - 501: Operation failed.
 - 523: No information about the given callback ID.



All parameters must be properly URL-encoded.

Here is an example of getting the execution status:

```
curl -is -X GET "https://example.agiloft.com/ewws/EWAsyncStatus?%24KB=Demo&%24login=admin&%24password=****&%24lang=en&%24table=case&callback_id=76406_9"
HTTP/2 200
server: nginx
date: Sat, 15 Aug 2020 10:58:48 GMT
content-type: charset=UTF-8
content-length: 0
strict-transport-security: max-age=31536000; includeSubDomains
```

REST - Table

The EWTable operation...

- Returns a list of all the tables and fields in the system.
- Supports GET and POST requests.
- Accepts the URL with parameters as per general URL conventions and record data encoded as per Data Encoding, both of which can be viewed in the [REST Interface](#).
- Must be used with [REST - Login](#) or OAuth 2.0 authorization.
- Supported Content-Type:
 - For requests: application/x-www-form-urlencoded
 - For returns: application/json;
- For tables, the label and logical table name are returned. For fields, the field name, label, type, and required flag are returned. Linked fields return only the source table information. Choice fields return the ID and name of the selected value. Action buttons, related tables, embedded search results, and embedded communications are not supported.
- This operation is never used asynchronously.

 All parameters must be properly URL-encoded.

Example

Assume an instance of Agiloft is available on localhost, port 8080 and is called "Demo". Retrieve a list of the tables and fields, and include information about any linked fields.

After obtaining an authorization token with [JWT](#), the following request is issued:

```
http://localhost:8080/ewws/EWTable/.json?$KB=Demo&includeLinkedInfo=true
```

A result similar to the following excerpt will be returned:

```
{
  "success": true,
  "message": "",
  "result": {
    "tables": [
      {
        "label": "WMI Sample",
        "logicalName": "wmi_sample",
        "fields": [
          {
            "columnLabel": "ID",
            "columnName": "id",
            "columnType": "BIGINT",
            "columnTypeDomain": "swautoincrementfield"
          }
        ]
      }
    ]
  }
}
```

```

    },
    {
      "columnLabel": "Type",
      "columnName": "type",
      "columnType": "BIGINT",
      "columnTypeDomain": "swobjecttype"
    },
    {
      "columnLabel": "Date Updated",
      "columnName": "date_updated",
      "columnType": "DATETIME",
      "columnTypeDomain": "swdatetimefield"
    },
    {

```

...

REST - Saved Search

The EWSavedSearch operation...

- Returns details about saved searches defined for a table. It returns the name and label of the search; the saved search identifier in the Agiloft database; and the description of the search.
- Supports GET and POST requests. For POST requests, the table parameter can be specified in the POST Request body instead of the URL parameter (?\$table=<logical_name_of_the_table>).
- Accepts the URL with parameters as per general URL conventions and record data encoded as per Data Encoding, both of which can be viewed in the [REST Interface](#).
- Must be used with [REST - Login](#) or OAuth 2.0 authorization.
- Supported Content-Type:
 - For requests: application/x-www-form-urlencoded
 - For returns: application/json
- JSON is the only output format supported, and you must use /.json after EWSavedSearch in the URL.
- The parameter \$table is mandatory and must use the logical name of a table in the KB.
- This operation is never used asynchronously.

 All parameters must be properly URL-encoded.

Example

Assume an instance of Agiloft is available on localhost, port 8080 and is called "Demo". Retrieve a list of saved searches for the Contract table, using its logical name of contract.

After obtaining authorization with EWLogin, the following request is issued:

```
http://localhost:8080/ewws/EWSavedSearch/.json?$table=contract
```

A result similar to the following excerpt will be returned:

```
{
  "success": true,
  "message": "",
  "result": [
    {
      "label": "R:Renewal date is tomorrow and Renewal Type is not one-time",
      "name": "contract|r: renewal date is tomorrow",
      "id": 265185,
      "description": ""
    },
    {
      "label": "R:Contract is expiring tomorrow",
```

```
"name": "contract|r:contract is expiring tomorrow",  
"id": 265186,  
"description":""  
}  
}
```

Using OAuth2 to Access REST API

Agiloft supports an OAuth2 process for authenticating users via token exchange so that they can make REST API requests. This requires you to configure an OAuth2 client in your knowledgebase (KB) by creating an API application, which requests permission and authentication from the user before performing any API requests on their behalf. When an API request is made, the user is redirected to the API application, asked to authenticate themselves and grant the requested permissions, and then redirected to the web application.

Agiloft's implementation of OAuth2 complies with the OAuth 2.0 specification and provides a variety of security benefits, as described in RFC 6749.

Prerequisites

An Enterprise license is required to use REST APIs in Agiloft.

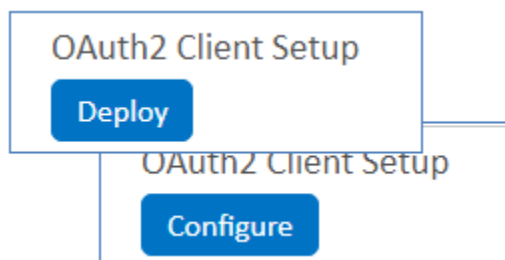
This feature works only with native Agiloft users; this is not compatible with LDAP.

Creating an API Application

The OAuth2 client setup begins with creating and configuring a new API application. When you create the application, you associate it with a specific user in the KB to determine which permissions are used when the server handles a REST API call. All REST API requests will be executed on behalf of this user.

To create an API application:

1. Click the **Setup** gear in the top-right corner and go to **Integration > OAuth2 Client Setup**.
2. Click Deploy or Configure:
 - **Deploy:** The OAuth2 client hasn't been deployed yet. Clicking Deploy configures the OAuth2 client for use in your KB, which can take a few minutes to complete.
 - **Configure:** The OAuth2 client has already been deployed. Clicking Configure takes you to the API Application screen.



3. On the API Application screen, click New to open the API Application Settings wizard.
4. Complete the required fields:
 - a. **Name:** The name that the application uses with Agiloft's REST API.
 - b. **Display Name:** The name of the application that appears in the KB.

- c. **Associate this Application with Contact ID:** The ID of the user associated with the application. All permissions for this user are used when the REST API is called. This field is automatically populated when you select a user in the Full Name field.
 - d. **Full Name:** The name of the user associated with the application.
 - e. **Redirect URI:** The webpage where you want to redirect users at the end of the authorization process. The redirect URI specified in your OAuth request must belong to this URI.
 - f. **Token Expiry in Minutes:** The amount of time in minutes before the access token expires. The default value is 15, and the acceptable range is from 1 to 60.
5. Click Apply Changes.
 6. Click Enable. This activates the application and generates the Client ID and Client Secret, which are used during the token exchange.

Application << >>

OAuth2 Client Identifier

Client ID
xEjxFBwdJrl8Qcm/qJkaSOZI Vghmwyw=

Client Secret
u3MLeHOjtcv9DhXOOLDDh OqQAvJF8NB/M4IBvNvA==

Note: You must keep your Client Secret confidential

Apply Changes

Enable

 If you need to block your application for any reason, click Disable, which replaces Enable after it's been clicked.

Changing Application Settings

Verify that your application settings are correct so that you don't need to change them later. If you need to change your application's settings but have already completed the authentication procedure described below, you need to re-authenticate for your changes to take effect. If you need to make changes but have already received an access token, you need to revoke the access token, make the changes, and then obtain a new access token for your changes to take effect. See [Revoking Tokens](#) for more information.

Changes to the values in the Associate this Application with Contact ID field or the Redirect URI field can sometimes block your application. If this happens, recreate the application with the correct settings.



You can change the Token Expiry in Minutes value without performing the re-authentication procedure. Changes to this field take effect when the next access token is requested.

OAuth Token Exchange

After the API application is created, the OAuth authorization process occurs through a series of HTTP calls made to various endpoints on the KB server. The following section describes the process and serves as a reference for the parameters in each client request and server response.

Authorization Request

The OAuth process starts with the client directing the user's browser to make a request to the `/ewws/oauth` endpoint, where the Content-Type must be in the form `application/x-www-form-urlencoded`. The request includes the following query string parameters:

Parameter	Value	Required	Description
response_type	code	Yes	Defines the response type and must always have a value of code.
client_id	Obtained from API Application Settings wizard	Yes	Identifies an application.
redirect_uri	Absolute URI defined on the API Application Settings wizard, such as <code>https://example.agiloft.com</code>	Yes	Defines where the user is redirected at the end of the authorization process. The value must belong to the set of values specified on API Application Settings wizard.
scope	permissions_for: {CONTACT_ID}	Yes	Defines the permissions that the user is asked to approve, which are determined by the Contact ID defined in the API Application Settings wizard. For example, if the Associate this Application with Contact ID field is set to 222, then this parameter should be set to <code>permissions_for:222</code> .
state	Any string	Recommended	Denotes a value that is returned to the client as a parameter at the end of the authorization process, which verifies the validity of the request. Although this parameter is not required, we highly recommend using it to protect against cross-site request forgery (CSRF), as described in RFC 6749.

Example Authorization Request


```
https://example.agiloft.com/ewws/oauth?response_type=code&redirect_uri=https%3A%2F%
2Ftest.com%2Freceiver&
  client_id=Bvn7k4fIdMEZQrJJ7ZCIQgErLTDbX9L73LThA5YA4W0%3D&scope=permissions_for%
3A213&
  state=LQKFNL023478_3259423
```

The authorization request asks the user to authenticate themselves and then authorize the request, which redirects the user to the URI value defined for the `redirect_uri` parameter.

Successful Request

If the request succeeds, the following query string parameters are sent to the client in response:

Parameter	Value
state	Returns the value of the state parameter from the authorization request.
code	Indicates the authorization code for use in the upcoming access token request. This code is valid for five minutes.
api_access_point	Defines the base URI for all REST API calls that use an access token, if any. If this value is empty, use the base URI for your KB's server.
client	Provides system information.

Example Response to a Successful Request

```
HTTP/1.1 302 Found
Location: https://example.agiloft.com/receiver?client=&state=LQKFNL023478_3259423&
code=EFLJHELFH23487402387LKFEHJFEHF=
```

Failed Request

If the request fails, the following query string parameters are sent to the client in response:

Parameter	Value
error	Defines the error code. See below for a description of each error code.
error_description	Describes why the request failed.

The following error codes are possible:

Code	Description
invalid_request	The request is missing a required parameter, includes an invalid parameter

	value, includes a parameter more than once, or is otherwise malformed.
unauthorized_client	The client is not authorized to request an authorization code using this method.
access_denied	The resource owner or authorization server denied the request.
unsupported_response_type	The authorization server does not support obtaining an authorization code using this method.
invalid_scope	The requested scope is invalid, unknown, or malformed.
server_error	The authorization server encountered an unexpected condition that prevented it from fulfilling the request.

Access Token Request

If the authorization request is successful, the client makes an HTTP POST to the /ewws/otoken endpoint by using the api_access_point value retrieved in the previous step. The following parameters are used:

Parameter	Value	Required	Description
grant_type	authorization_code	Yes	Defines the grant type, which must always be authorization_code.
code	The authorization code obtained in the previous step	Yes	Confirms that the previous step was successful.
client_id	Obtained from the API Application Settings wizard	Yes	Identifies the application.
redirect_uri	Must match the value used in the previous step	Yes	This value must belong to the set of values specified in API Application Settings wizard.

Example Access Token Request

```
POST /ewws/otoken HTTP/1.1
Host: example.agiloft.com
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code&
code=EFLJHELHFH23487402387LKFEHJFEHF&
client_id=Bvn7k4fIdMEZQrJJ7ZCIQgErLTDbX9L73LThA5YA4W0%3D&
redirect_uri=https%3A%2F%2Ftest.com%2Freceiver
```

Token Response

The authorization code from the request is exchanged for OAuth2 tokens, and the server returns a JSON array with the following keys:

Parameter	Value
access_token	The access token, which can be used as an authorization header in the Agiloft REST API.
refresh_token	The refresh token, which can be used to get a new access token. Refresh tokens expire after 28 days of inactivity.
token_type	The value will always be Bearer.
expires_in	The expiration time in minutes for the access token.

Example Token Response

```
{
  "access_token": "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiIyX2FkbWluI",
  "refresh_token": "LKHEFOP932875KJGKJG32423542LHLKHFDFDKLJ.OJ==",
  "token_type": "Bearer",
  "expires_in": 15
}
```

Refresh Request

At any time the client can use the refresh token to receive a new access token. To start a request, the client makes an HTTP POST to the /ewws/otoken endpoint by using the api_access_point value retrieved during the authorization request. The following parameters are used:

Parameter	Value	Required	Description
grant_type	refresh_token	Yes	The value must always be refresh_token.
md5_secret	The first 20 characters of the Client Secret, obtained from API Application Settings wizard	Yes	Authenticates the application.
refresh_token	Refresh token received during the previous step	Yes	Defines the refresh token.

Example Refresh Token Request

```
POST /ewws/otoken HTTP/1.1
Host: example.agiloft.com
Content-Type: application/x-www-form-urlencoded

grant_type=refresh_token&
md5_secret=F64EC9CE9852EE154696&
refresh_token=LKHEFOP932875KJGKJG32423542LHLKHFD_FDKLJ.OJ%3D%3D
```

The refresh token can then be used to generate a new access token. The server returns a JSON array with the following keys:

Parameter	Value
access_token	The access token, which can be used as an authorization header in the Agiloft REST API.
token_type	The value will always be Bearer.
expires_in	The expiration time in minutes of the access token.

Example Response

```
{
  "access_token": "wehRT74i0iJIUzI1NiJ0.weJzdWIiOiIyX2FkbWluI",
  "token_type": "Bearer",
  "expires_in": 15
}
```

Revoking Tokens

Access tokens and refresh tokens can be revoked at any time by the authorizing user. REST operations cannot be performed with a token that has been revoked. If an access token is revoked and it has a corresponding refresh token, the refresh token is also revoked. If a refresh token is revoked, all the access tokens issued from that refresh token are also revoked.

Refresh tokens can be revoked by making a POST call to the `/ewws/orevoke` endpoint by using the `api_access_point` value retrieved during the authorization request. The following parameters are used:

Parameter	Value	Required	Description
revoke_for	OAuth2 refresh token or Client Secret key from API Application Settings wizard	Yes	Defines the refresh token or Client Secret Key.

Example Request to Revoke a Refresh Token

```
POST /ewws/orevoke HTTP/1.1
Host: example.agiloft.com
```

```
Content-Type: application/x-www-form-urlencoded

revoke_for=LKHEFOP932875KJGKJG32423542LHLKHFD_FDKLJ.OJ%3D%3D
```

Response to Revoking a Token

If the request to revoke the token succeeds, an HTTP success code 200 is returned with an empty body or without any body at all. If the request fails, an HTTP code 400 is returned with the error code `INVALID_REQUEST`.

Example Response to a Failed Request

```
{
  "error" : "INVALID_REQUEST",
  "error_description" : "Invalid token."
}
```

Bypassing Content-Type Header Check

 This section is visible only to authenticated users, since this method is not recommended.

In some cases, when designing integrations in accordance with RFC, you might need to bypass the Content-Type header check requirement for REST OAuth2 requests. This method is not recommended, because the Content-Type header check is a security measure, so use this only as a last resort.

To bypass this requirement:

1. In the admin console, create a new Text global variable named `ignore_oauth_content_type` and set it to the IDs of the KBs where you want to bypass the check. For example, you might enter `66,77,13` to bypass three KBs with those IDs.
2. In the KB, go to **Setup > System > Manage Global Variables** and create a Choice global variable named `disable_content_type_validation_at_api_oauth_login`. Assign it the Yes/No choice list and set it to Yes.

Now, with both variables set, `/ewws/oauth`, `/ewws/otoken`, and `/ewws/orevoke` operations won't require a specific Content-Type header.

ESA Developer Guide

An External System Adapter (ESA) is a plugin for Agiloft that enables connectivity with other systems. The ESA provides a means to synchronize data between two systems based on record timestamps. The ESA...

- Receives and parses XML messages from Agiloft, either directly or from the remote proxy
- Processes the messages, usually by accessing external system (XS) data
- Frames an XML response and passes it back to Agiloft.

The Agiloft synchronization subsystem provides facilities for automatic synchronization between Agiloft tables and the corresponding records in some external system. Synchronization can be bidirectional or unidirectional in either direction. Agiloft comes with pre-built ESAs for common systems such as Exchange and Excel. Additional ESAs can be created by customers in Java, or in the language of their choice.

Note: one major advantage to developing the ESA in Java is that it can easily be linked with the bundled Agiloft Remote Proxy that handles the communication with the sync subsystem. If the ESA is developed in another language, you must use the standard I/O streams to communicate with the Remote Proxy.

An addition is necessary for HTTPs ESA's, which can initiate calls beginning a sync to the Agiloft Helper API. A command-line ESA can't do that, because it is only run by sync when Agiloft decides that it is time to synchronize. Such an ESA can't pass a message to Helper API outside of the sync cycle.

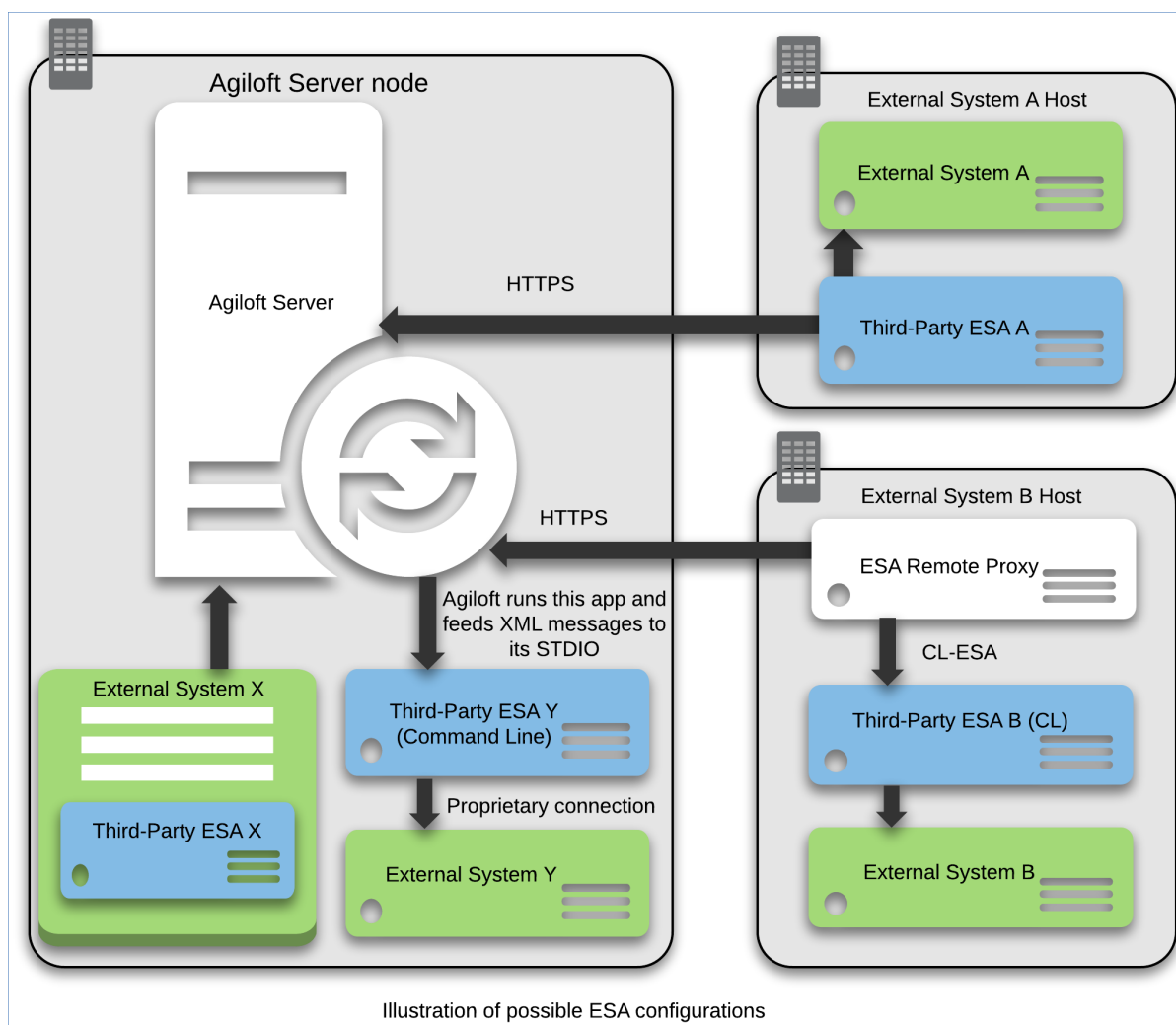
ESA Properties

To be synchronized with Agiloft, an external system must have these properties:

- It must be possible to obtain the modification timestamps for each record. This timestamp is typically contained in the record itself, but this is not absolutely necessary.
- Record IDs should not be changed, or at least there must be a way to determine any record ID as it was in the last sync. Record IDs can be remapped during sync, but the old ID must be made known to the ESA in order to track record matches.
- Records from the external system, as presented during sync by the ESA, must be grouped by structures, which are mapped on a 1-to-1 basis to Agiloft types or tables, such as People/Employee. An ESA structure can be a table, a folder, a file or anything else that logically groups records in the external system. Within a given structure, external records must use the same set of field mappings.

Possible Configurations

This diagram shows a few possible configurations for the external system to work with Agiloft.



The two white components represent the Agiloft server application itself, and the ESA remote proxy application. The four green components represent external systems that might be remote, or co-located on the Agiloft machine. The blue components show possible ESA deployments:

- ESA A is located on the external host, communicating with the external system over some possibly proprietary protocol and communicating with Agiloft over HTTPs, as if over the internet.
- ESA B is a command-line (CL) application, also on the external system host, but is managed (launched) by an ESA remote proxy which, in turn, communicates with Agiloft over HTTPs. The ESA Remote Proxy uses standard I/O streams for message exchanges with the ESA.
- ESA X and ESA Y are similar to ESA A and ESA B, respectively, but are located on the Agiloft host. ESA X is run within its external system and communicates with Agiloft over HTTPs, as if over the internet. ESA Y is run and communicates directly with Agiloft through STDIO.

Choosing an ESA Deployment Type

The following guidelines can help when designing your ESA deployment:

- If your external system is a server or a continually running daemon-like process, implementing the ESA as a module of the system might be a better option, since you can control the life cycle of the ESA. In this case, the ESA should connect with Agiloft over HTTPS.
- For an application that is not always running, use the command-line form.
- If you need to trigger synchronization based upon external system events, use an HTTPS and possibly daemon-like ESA. A command-line ESA cannot connect to Agiloft on its own, and will only be triggered with Agiloft begins a synchronization.
- If you don't need the external system to ever trigger a synchronization, develop your ESA as a command-line application, using standard I/O communication with the XS. If the ESA is not on the Agiloft host, an ESA Remote Proxy will be needed for it to interface with Agiloft, but the ESA will be simpler to develop.
- The Agiloft ESA for QuickBooks must run on the Agiloft host, but otherwise ESAs can fit into any of the above contexts. Firewall configurations will often determine the most convenient deployments, which will in turn dictate certain sync configuration options.
- Once an ESA has been designed, it can communicate with systems of the same type, not just with a single server. For instance, once an Exchange ESA is written, you may set up synchronizations with many Exchange servers, not just with that particular Exchange installation.
- Java classes for message handling are available for any ESA that is to be developed in Java. The [ESA Interface Reference](#) topic describes the extensive reusable Java classes available, and provides a complete ESA in Java.
- Communications use the XML-RPC model, where XML messages are sent to and from the Agiloft server. However, when an ESA is remote, the communication flow is actually reversed in order to allow communication through firewalls. The remote ESA actively polls for the next XML-RPC message on the Agiloft server. Logically though, the communication is the same because the ESA only performs commands issued by the sync subsystem and is therefore passive. This polling is handled by the Java support libraries provided by Agiloft and need only be considered if you are developing an ESA using a programming language other than Java.

Calls To and From an ESA

All communications between Agiloft and the ESA are in the form of XML message exchange, of two kinds:

- Calls - Method Calls
- Results - including exceptions

Most calls come from Sync to the ESA, but the ESA can respond by a call to the Agiloft Helper API.

A remote Command-line ESA can never initiate a syncn cycle, but an HTTPs one can do so, by a call on the Agiloft Helper API. In this case the initial call is `startSync(HelperAPI)`. Agiloft then responds with a `startSync(ESA)` and other calls to the ESA, finishing communication by a void result of `startSync(HelperAPI)`.

Call ID

Every call has a mandatory call-id attribute. This attribute should be set by the caller to a positive integer value. It is recommended but not obligatory that this value would be incremented on each call. A result of a call has a "response-to" attribute, filled by the call-id value of the call, to which this result belongs. This simple mechanism allows monitoring of message flow and detection of communication or application failures.

ESA Data Model

Agiloft stores external system user data in a knowledgebase. Knowledgebases are set up similarly to like separate databases, with many customizable tables containing configurable records, and which may be linked to other tables. One knowledgebase in a server - among many possible instances - is one side of the synchronization process. The other side is your external system, which can be virtually any legacy system, as long as its data can be logically matched against knowledgebase table records and these two requirements are met:

- Every record must have a unique numeric or string ID
- Every record must have a modification timestamp

The ESA must provide the Agiloft sync subsystem with some meta-information about your external system, such as the list of structures and the fields in each table that are to be mapped. When setting up the synchronization configuration, the Agiloft administrator - who does not have to be the ESA developer - maps Agiloft table fields to your external system fields, thus establishing the relationship between them.

Responsibilities of ESA Components

There is a clear separation of responsibilities between the Agiloft synchronization subsystem and the ESA:

- The Agiloft sync subsystem manages the generic synchronization logic:
 - initiating synchronizations
 - comparing records in both systems
 - deciding which records on which systems are to be updated
 - resolving conflicts after an update, and so on.
- The ESA manages the Create/Retrieve/Update/Delete operations on the external system.

An ESA always communicates with the external system and never accesses Agiloft data directly. Instead, it communicates with the Agiloft sync subsystem core that determines what actions are needed and takes care of all the necessary data conversions and record mappings. The ESA manages the IDs, data and other items in the external system. From a synchronization point of view, the ESA is the easy part - it just has to implement the CRUD operations on the external system. Of course, the ESA complexity depends on how your system is accessed and how easily its data can be exposed as tables that can be matched to the corresponding Agiloft tables.

ESA Parameters

An ESA usually requires some initialization parameters in order to do the synchronization, such as external system name, port, schema name, and so on. For the most part, synchronizations are quite simple and it would place an unnecessary burden on the Agiloft administrator to manage the process through configuration files or special tools. The Agiloft subsystem core can manage the following processes:

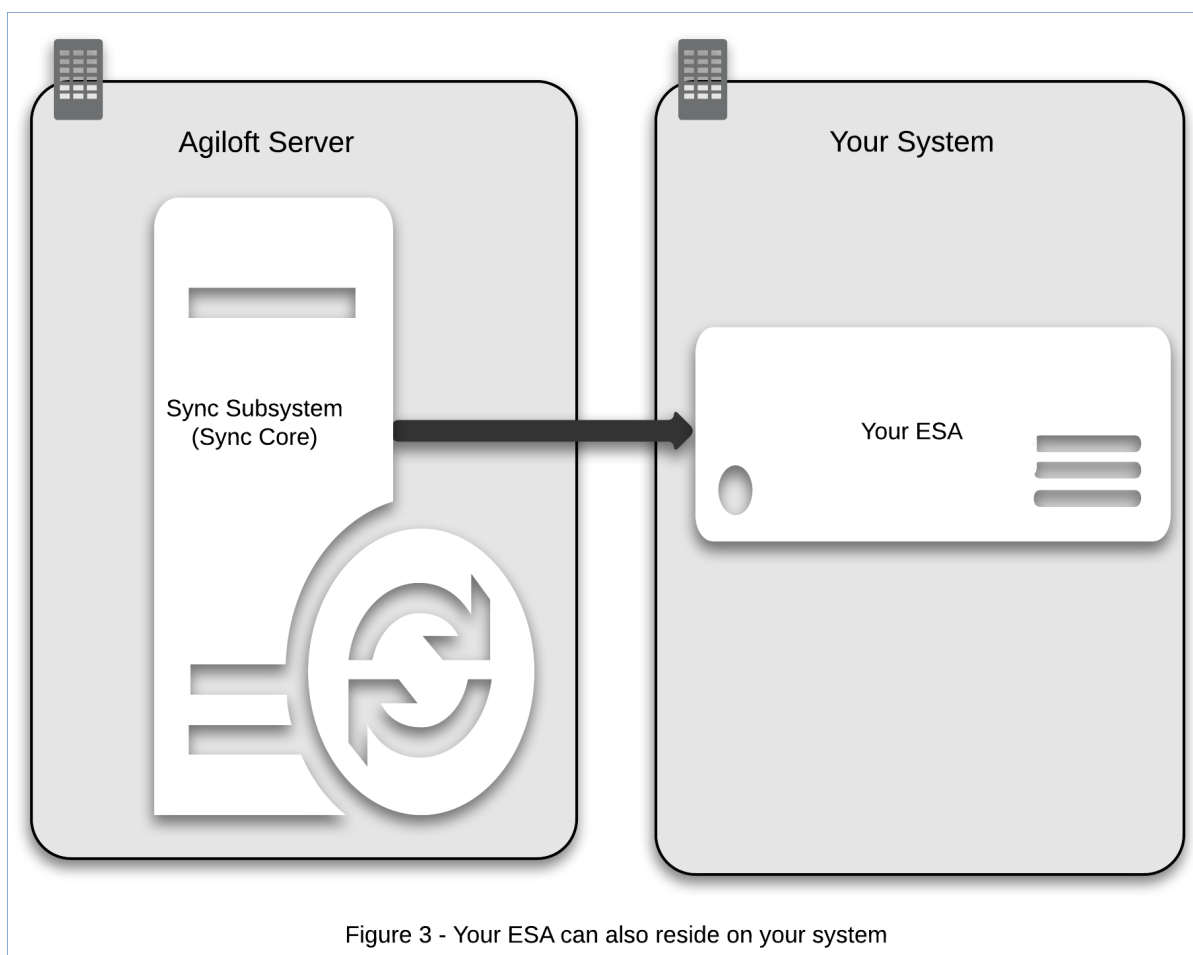
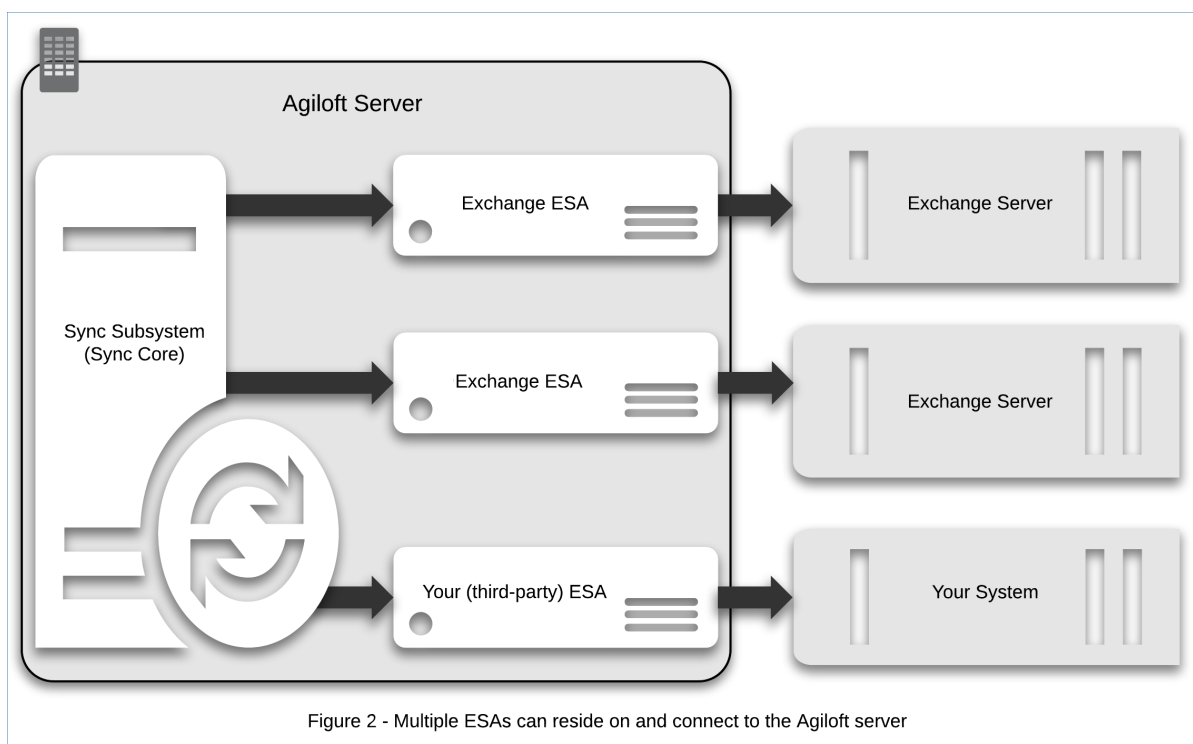
- The ESA provides the Agiloft sync subsystem with a list of parameters it needs, including parameter names and types. Several simple types are supported, such as string inputs, choice fields and so on.
- The Agiloft administrator sets up the sync configuration through an Agiloft GUI that allows the admin to map these parameters to Agiloft tables so that synchronization can take place.
- At run time, the ESA obtains these stored parameter values, using the parameter name as the key.

This mechanism provides a simple way to manage configuration and synchronization data. Your ESA may or may not use this facility. It is not mandatory, but it is recommended to make the configuration easier for the Agiloft administrator.

Sync Configuration

Before running a synchronization, the Agiloft administrator must first set up the configuration. This is done inside the Agiloft knowledgebase, and includes the following aspects:

- The name of the ESA to be synchronized with
- The type of external system and ESA parameters. Usually this includes the server to synchronize with, but some ESAs such as Facebook do not require this information.
- Table and field mappings between the systems.
- A number of other parameters, such as one- or two-way synchronization, conflict resolution settings, polling and so on.



When the configuration is complete, the synchronization is ready to run. Every sync configuration has a unique ID, called the external system ID, which is a handle used to identify the configuration when communicating with the ESA. the external system ID is automatically generated by Agiloft and can be viewed on the Sync Configuration screen.

Pseudocode

Pseudocode for an ESA is:

```
Boolean quit = false;
Do
{
    String xmlMsg = readXmlMessageFromEw();
    Message parsedMsg = parseXmlMessage(xmlMsg);
    // Dispatch message
    Message result;
    Switch (parsedMsg.type)
    {
        Case getModified:
        {
            // Find all modified records ...
            result = new RecordListResultMessage( modified records);
            break;
        }...
        Case update:
        {
            // Update a record
            result = new RecordResultMessage( updated record);
            break;
        }
        ...Case release:
        {
            // Release all resources...
            quit = true;
        }
        Default:
        {
            result = new ExceptionResult("Unkno wn operation");
        }
    }
    String xmlResponse = xmlizeResponse(result);
    sendResponseToEw(xmlResponse);
} While (Not quit)
```

ESA XML Use

XML messages are defined using the W3C XML Schema language. You can find this scheme within the `sync.xsd` example file in `syncHome`. The XML messages schema can be downloaded [here](#). All messages from or to Agiloft must be well-formed XML documents, starting with an XML declaration and a root `<sync>` tag. A complete, formal XML This ensures that Agiloft and the ESA will understand each other.

Schematic Examples

This section contains examples of XML messages in the ESA:

Calls to ESA

All messages are framed by `<sync>` and `<esa-call>` elements and are prepended by an XML declaration. Actual messages look like this:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<sync xsi:noNamespaceSchemaLocation="sync.xsd"
:xsi="http://www.w3.org/2001/XMLSchema-instance">
<esa-call call-id="12345">
<startSync>
<external-system-id>My-External-ID</external-system-id>
</startSync>
</esa-call>
</sync>
```

Results from ESA

All result messages should also include the XML declaration and be wrapped into `<result>` tags, like this:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<sync xsi:noNamespaceSchemaLocation="sync.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<result response-to="12345">
<value>1.0</value>
...</result>
</sync>
```

Calls to HelperAPI

All these messages must be framed by <sync> and <api-call> elements and are prepended by the XML declaration. Actual messages look like this:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<sync xsi:noNamespaceSchemaLocation="sync.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <api-call call-id="12345">
    <startSync asynchronous="true">
      <external-system-id>My-External-ID</external-system-id>
    </startSync>
  </api-call>
</sync>
```

Results from API

These messages are identical in form to the ESA call results.

ESA XML Message Schema

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <!--Root element
  is sync, having exactly one "call" or "result"-->

  <xs:element
    name="sync">
    <xs:complexType>
      <xs:choice maxOccurs="1">
        <xs:sequence>
          <xs:element name="esa-call" type="esaCallType"/>
        </xs:sequence>
          <xs:element name="api-call" type="apiCallType"/>
        </xs:sequence>
          <xs:sequence>
            <xs:element name="result" type="resultType"/>
          </xs:sequence>
        </xs:choice>
```

```
</xs:complexType>
</xs:element>
```

```
<!--External System Adapter Calls-->
<xs:group name="esa-calls">
<xs:choice>
<xs:element name="startSync" type="startSyncEsa"/>
<xs:element name="leaseSession" type="noArgCall"/>
<xs:element name="endSync" type="noArgCall"/>
<xs:element name="release" type="noArgCall"/>
<xs:element name="getAllowedModes" type="noArgCall"/>
<xs:element name="needSyncAgain" type="noArgCall"/>
<xs:element name="getCurrentTime" type="noArgCall"/>
<xs:element name="configure" type="configure"/>
<xs:element name="getStructureList" type="localeOnly"/>
<xs:element name="getFieldList" type="getRecordMeta"/>
<xs:element name="getRelations" type="getRecordMeta"/>
<xs:element name="getCollections" type="getRecordMeta"/>
<xs:element name="getParametersMeta" type="localeOnly"/>
<xs:element name="getModified" type="getChanges"/>
<xs:element name="getModifiedPaged" type="getChanges"/>
<xs:element name="getDeleted" type="getChanges"/>
<xs:element name="getDeletedPaged" type="getChanges"/>
<xs:element name="leaseCursor" type="updateCursor"/>
<xs:element name="closeCursor" type="updateCursor"/>
<xs:element name="readDataPage" type="readDataPage"/>
<xs:element name="read" type="read"/>
<xs:element name="create" type="create"/>
<xs:element name="update" type="update"/>
<xs:element name="delete" type="delete"/>
<xs:element name="countRange" type="countRange"/>
<xs:element name="checkDelayedCreate" type="checkDelayedCreate"/>
<xs:element name="checkDelayedUpdate" type="checkDelayedUpdate"/>
<xs:element name="checkDelayedDelete" type="checkDelayedDelete"/>
<xs:element name="getProgressReport" type="noArgCall"/>
<xs:element name="getDetailedReport" type="noArgCall"/>
<xs:element name="syncErrorNotify" type="syncErrorNotify"/>
</xs:choice>
</xs:group>
```

```
<!--Helper API calls-->
<xs:group name="helper-api-calls">
<xs:choice>
<xs:element name="startSync" type="startSyncApi"/>
<xs:element name="getParameter" type="getParameter"/>
<xs:element name="getPollPeriod" type="getPollPeriod"/>
<xs:element name="trackRecordDeletion" type="trackRecordDeletion"/>
<xs:element name="detectDeleted" type="detectDeleted"/>
<xs:element name="isKnownID" type="isKnownID"/>
</xs:choice>
</xs:group>
```

```

<xs:element name="enumerateKnownIDs" type="enumerateKnownIDs"/>
</xs:choice>
</xs:group>

```

```

<!-- Calls -->
<xs:complexType name="callType" abstract="true">
<xs:attribute name="call-id" use="required" type="xs:nonNegativeInteger"/>
</xs:complexType>

<xs:complexType name="esaCallType">
<xs:complexContent>
<xs:extension base="callType">
<xs:group ref="esa-calls"/>
</xs:extension>
</xs:complexContent>
</xs:complexType>

<xs:complexType name="apiCallType">
<xs:complexContent>
<xs:extension base="callType">
<xs:group ref="helper-api-calls"/>
</xs:extension>
</xs:complexContent>
</xs:complexType>

<!-- Esa.leaseSession, Esa.getStructureList, Esa.endSync, Esa.release, Esa.
getAllowedRunModes, Esa.blockMappedFieldsChangesEsa.needSyncAgain, Esa.
getCurrentTime has empty parameters -->
<xs:complexType name="noArgCall">
</xs:complexType>

<!-- External System Adapter calls -->

<!-- Start Sync -->
<xs:complexType name="startSyncEsa">
<xs:sequence>
<xs:element name="external-system-id" type="xs:string"/>
</xs:sequence>
</xs:complexType>

<!-- Esa.getFieldList, Esa.getRelations, Esa.getCollections -->
<xs:complexType name="getRecordMeta">
<xs:sequence>
<xs:element name="structure" type="xs:string"/>
<xs:element name="locale" type="xs:language"/>
</xs:sequence>
</xs:complexType>

<!-- Esa.getParametersMeta, Esa.getStructures -->

```



```

<xs:complexType name="localeOnly">
  <xs:sequence>
    <xs:element name="locale" type="xs:language"/>
  </xs:sequence>
</xs:complexType>

<!--Esa.configure-->
<xs:complexType name="configure">
  <xs:sequence>
    <xs:element name="external-system-id" type="xs:string"/>
    <xs:element name="force" type="xs:boolean"/>
  </xs:sequence>
</xs:complexType>

<!--Esa.getModified, Esa.getModifiedPaged, Esa.getDeleted, Esa.getDeletedPaged -->
<xs:complexType name="getChanges">
  <xs:sequence>
    <xs:element name="structure" type="xs:string"/>
    <xs:element name="after" type="xs:dateTime" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>

<!--Esa.leaseCursor, Esa.closeCursor -->
<xs:complexType name="updateCursor">
  <xs:sequence>
    <xs:element name="cursor-id" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

<!--Esa.readDataPage -->
<xs:complexType name="readDataPage">
  <xs:sequence>
    <xs:element name="cursor-id" type="xs:string"/>
    <xs:element name="page-index" type="xs:nonNegativeInteger" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>

<!--Esa.read-->
<xs:complexType name="read">
  <xs:sequence>
    <xs:element name="structure" type="xs:string"/>
    <xs:element name="id" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

<!--Esa.create-->
<xs:complexType name="create">
  <xs:sequence>
    <xs:element name="structure" type="xs:string"/>
    <xs:element name="record" type="recordType"/>
  </xs:sequence>
</xs:complexType>

```

```

</xs:sequence>
</xs:complexType>

<!--Esa.update-->
<xs:complexType name="update">
<xs:sequence>
<xs:element name="structure" type="xs:string"/>
<xs:element name="last-seen" type="xs:dateTime" id="last-seen-update"/>
<xs:element name="record" type="recordType"/>
</xs:sequence>
</xs:complexType>

<!--Esa.delete-->
<xs:complexType name="delete">
<xs:sequence>
<xs:element name="structure" type="xs:string"/>
<xs:element name="last-seen" id="last-seen-delete" type="xs:dateTime"/>
<xs:element name="id" type="xs:string"/>
</xs:sequence>
</xs:complexType>

<!--Esa.countRange -->
<xs:complexType name="countRange">
<xs:sequence>
<xs:element name="structure" type="xs:string"/>
<xs:element name="min-id" type="xs:string"/>
<xs:element name="max-id" type="xs:string"/>
</xs:sequence>
</xs:complexType>

<!--Esa.checkDelayedCreate -->
<xs:complexType name="checkDelayedCreate">
<xs:sequence>
<xs:element name="structure" type="xs:string"/>
<xs:element name="token" type="xs:string"/>
</xs:sequence>
</xs:complexType>

<!--Esa.checkDelayedUpdate -->
<xs:complexType name="checkDelayedUpdate">
<xs:sequence>
<xs:element name="structure" type="xs:string"/>
<xs:element name="token" type="xs:string"/>
</xs:sequence>
</xs:complexType>

<!--Esa.syncErrorNotify -->
<xs:complexType name="syncErrorNotify">
<xs:sequence>
<xs:element name="error" type="xs:string"/>
</xs:sequence>

```

```
</xs:complexType>
```

```
<!--Helper API-->
```

```
<!--Start Sync-->
```

```
<xs:complexType name="startSyncApi">
<xs:sequence>
<xs:element name="external-system-id" type="xs:string"/>
</xs:sequence>
<xs:attribute name="asynchronous" use="required" type="xs:boolean"/>
</xs:complexType>
```

```
<!--HelperApi.getParameter -->
```

```
<xs:complexType name="getParameter">
<xs:sequence>
<xs:element name="external-system-id" type="xs:string"/>
<xs:element name="name" type="xs:string"/>
</xs:sequence>
</xs:complexType>
```

```
<!--HelperApi.getPollPeriod -->
```

```
<xs:complexType name="getPollPeriod">
<xs:sequence>
<xs:element name="external-system-id" type="xs:string"/>
</xs:sequence>
</xs:complexType>
```

```
<!--HelperApi.trackRecordDeletion -->
```

```
<xs:complexType name="trackRecordDeletion">
<xs:sequence>
<xs:element name="external-system-id" type="xs:string"/>
<xs:element name="structure" type="xs:string"/>
<xs:element name="id" type="xs:string"/>
<xs:element name="time" type="xs:dateTime" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
```

```
<!--HelperApi.detectDeleted -->
```

```
<xs:complexType name="detectDeleted">
<xs:sequence>
<xs:element name="external-system-id" type="xs:string"/>
<xs:element name="after" type="xs:dateTime"/>
<xs:element name="structure" type="xs:string"/>
</xs:sequence>
</xs:complexType>
```

```
<!--HelperApi.isKnownID -->
```

```
<xs:complexType name="isKnownID">
<xs:sequence>
```

```

<xs:element name="external-system-id" type="xs:string"/>
<xs:element name="structure" type="xs:string"/>
<xs:element name="id" type="xs:string"/>
</xs:sequence>
</xs:complexType>

```

```

<!--HelperApi.enumerateKnownIDs -->
<xs:complexType name="enumerateKnownIDs">
<xs:sequence>
<xs:element name="external-system-id" type="xs:string"/>
<xs:element name="known-before" type="xs:dateTime"/>
<xs:element name="structure" type="xs:string"/>
</xs:sequence>
</xs:complexType>

```

```

<!--Data types-->
<!--External Record-->
<xs:complexType name="recordType">
<xs:sequence>

<!--Field values-->
<xs:element name="field" minOccurs="0" maxOccurs="unbounded">
<xs:complexType>
<xs:simpleContent>
<xs:extension base="xs:string">
<xs:attribute name="name" type="xs:string" />
</xs:extension>
</xs:simpleContent>
</xs:complexType>
</xs:element>

<!--Related record IDs-->
<xs:element name="relation" minOccurs="0" maxOccurs="unbounded">
<xs:complexType>
<xs:sequence>
<xs:element name="related" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
<xs:attribute name="id" type="xs:string" use="required"/>
</xs:complexType>
</xs:element>

<!--Collections-->
<xs:element name="collection" minOccurs="0" maxOccurs="unbounded">
<xs:complexType>
<xs:sequence>
<xs:element name="record" type="recordType" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
<xs:attribute name="id" type="xs:string" use="required"/>
</xs:complexType>
</xs:element>

```

```

</xs:sequence>

<!--Record Attributes-->
<xs:attribute name="id" type="xs:string" use="required"/>
<xs:attribute name="old-id" type="xs:string"/>
<xs:attribute name="modified" type="xs:dateTime"/>
<xs:attribute name="new" type="xs:boolean"/>
</xs:complexType>

<!--External Structure Descriptor-->
<xs:complexType name="structureType">
<xs:attribute name="name" type="xs:string"/>
<xs:attribute name="screen-name" type="xs:string"/>
</xs:complexType>

<!--Exception Descriptor-->
<xs:simpleType name="exceptionTypeEnum">
<xs:restriction base="xs:string">
<xs:enumeration value="general"/>
<xs:enumeration value="record"/>
<xs:enumeration value="recorddelayed"/>
<xs:enumeration value="configuration"/>
<xs:enumeration value="alreadyconfigured"/>
<xs:enumeration value="optlockfailed"/>
<xs:enumeration value="concurrentdelete"/>
<xs:enumeration value="skipstructure"/>
</xs:restriction>
</xs:simpleType>
<xs:complexType name="exceptionType">
<xs:sequence>
<xs:element name="message" type="xs:string"/>
<xs:element name="trace" type="xs:string"/>
<xs:element name="configured-to" type="xs:string" minOccurs="0"/>
<xs:element name="external-id" type="xs:string" minOccurs="0"/>
<xs:element name="modified-at" type="xs:dateTime" minOccurs="0"/>
<xs:element name="delay-token" type="xs:string" minOccurs="0"/>
<xs:element name="structure-name" type="xs:string" minOccurs="0"/>
</xs:sequence>
<xs:attribute name="type" type="exceptionTypeEnum" use="required"/>
</xs:complexType>

<!--Enumerated (field or parameter) value-->
<xs:complexType name="enumValueType">
<xs:sequence>
<xs:element name="value" type="xs:string"/>
<xs:element name="screen-name" type="xs:string"/>
</xs:sequence>
</xs:complexType>

<!--External Field Descriptor-->
<xs:simpleType name="externalFieldTypeEnum">

```

```

<xs:restriction base="xs:string">
  <!--Supported Schema types-->
  <xs:enumeration value="ew:enum"/>
  <xs:enumeration value="ew:attachedFiles"/>
  <xs:enumeration value="xsd:string"/>
  <xs:enumeration value="xsd:normalizedString"/>
  <xs:enumeration value="xsd:token"/>
  <xs:enumeration value="xsd:language"/>
  <xs:enumeration value="xsd:duration"/>
  <xs:enumeration value="xsd:date"/>
  <xs:enumeration value="xsd:dateTime"/>
  <xs:enumeration value="xsd:time"/>
  <xs:enumeration value="xsd:gYearMonth"/>
  <xs:enumeration value="xsd:gYear"/>
  <xs:enumeration value="xsd:gMonthDay"/>
  <xs:enumeration value="xsd:gDay"/>
  <xs:enumeration value="xsd:gMonth"/>
  <xs:enumeration value="xsd:boolean"/>
  <xs:enumeration value="xsd:base64Binary"/>
  <xs:enumeration value="xsd:hexBinary"/>
  <xs:enumeration value="xsd:float"/>
  <xs:enumeration value="xsd:decimal"/>
  <xs:enumeration value="xsd:double"/>
  <xs:enumeration value="xsd:byte"/>
  <xs:enumeration value="xsd:unsignedByte"/>
  <xs:enumeration value="xsd:int"/>
  <xs:enumeration value="xsd:unsignedInt"/>
  <xs:enumeration value="xsd:integer"/>
  <xs:enumeration value="xsd:negativeInteger"/>
  <xs:enumeration value="xsd:nonNegativeInteger"/>
  <xs:enumeration value="xsd:positiveInteger"/>
  <xs:enumeration value="xsd:nonPositiveInteger"/>
  <xs:enumeration value="xsd:long"/>
  <xs:enumeration value="xsd:unsignedLong"/>
  <xs:enumeration value="xsd:short"/>
  <xs:enumeration value="xsd:unsignedShort"/>
  <xs:enumeration value="xsd:anyURI"/>
</xs:restriction>
</xs:simpleType>
<xs:complexType name="externalFieldType">
  <xs:sequence>
    <xs:element name="enum-value" minOccurs="0" maxOccurs="unbounded" type="
enumValueType"/>
  </xs:sequence>
  <xs:attribute name="name" type="xs:string" use="required"/>
  <xs:attribute name="screen-name" type="xs:string" use="required"/>
  <xs:attribute name="required" type="xs:boolean" use="required"/>
  <xs:attribute name="identifying" type="xs:boolean" use="required"/>
  <xs:attribute name="updatable" type="xs:boolean" use="required"/>
  <xs:attribute name="updatableOnCreate" type="xs:boolean" use="required"/>
  <xs:attribute name="max-length" type="xs:integer" use="optional" default="-1"/>

```

```

<xs:attribute name="type" type="externalFieldTypeEnum" use="required"/>
</xs:complexType>

<!--External Relation Descriptor-->
<xs:complexType name="externalRelationType">
<xs:attribute name="id" type="xs:string" use="required"/>
<xs:attribute name="related" type="xs:string" use="required"/>
<xs:attribute name="screen-name" type="xs:string" use="required"/>
<xs:attribute name="multiple" type="xs:boolean" use="required"/>
<xs:attribute name="required" type="xs:boolean" use="required"/>
<xs:attribute name="postponable" type="xs:boolean" use="required"/>
</xs:complexType>

<!--External Collection Descriptor-->
<xs:complexType name="externalCollectionType">
<xs:attribute name="id" type="xs:string" use="required"/>
<xs:attribute name="screen-name" type="xs:string" use="required"/>
<xs:attribute name="required" type="xs:boolean" use="required"/>
<xs:attribute name="updatable" type="xs:boolean" use="required"/>
<xs:attribute name="updatableOnCreate" type="xs:boolean" use="required"/>
</xs:complexType>

<!--Cursor Descriptor-->
<xs:complexType name="cursorType">
<xs:attribute name="id" type="xs:string" use="required"/>
<xs:attribute name="page-size" type="xs:positiveInteger" use="required"/>
<xs:attribute name="number-of-pages" type="xs:nonNegativeInteger" use="required"
/>
</xs:complexType>

<!--ESA Parameter Meta-->
<xs:simpleType name="esaParameterTypeEnum">
<xs:restriction base="xs:string">
<xs:enumeration value="single"/>
<xs:enumeration value="radio"/>
<!--<xs:enumeration value="multi"/>-->
<!--<xs:enumeration value="expandable"/>-->
<xs:enumeration value="xml"/>
<xs:enumeration value="custom"/>
<xs:enumeration value="password"/>
<xs:enumeration value="timezone"/>
</xs:restriction>
</xs:simpleType>
<xs:simpleType name="esaParameterItemTypeEnum">
<xs:restriction base="xs:string">
<xs:enumeration value="text"/>
<xs:enumeration value="integer"/>
<xs:enumeration value="float"/>
<xs:enumeration value="boolean"/>
<xs:enumeration value="date"/>
</xs:restriction>

```

```

</xs:simpleType>
<xs:complexType name="parametersMetaType">
<xs:sequence>
<xs:element name="label" type="xs:string"/>
<xs:element name="hint" type="xs:string"/>
<xs:choice>
<xs:element name="render-code" type="xs:string" minOccurs="0"/>
<xs:sequence>
<xs:element name="item-type" type="esaParameterItemTypeEnum"/>
<xs:element name="default-value" minOccurs="0" type="xs:string"/>
<xs:element name="enum-value" minOccurs="0" maxOccurs="unbounded" type="
enumValueType"/>
</xs:sequence>
</xs:choice>
</xs:sequence>
<xs:attribute name="name" type="xs:string" use="required"/>
<xs:attribute name="type" type="esaParameterTypeEnum" use="required"/>
<xs:attribute name="required" type="xs:boolean" use="required"/>
</xs:complexType>
<xs:complexType name="parameterValueType">
<xs:simpleContent>
<xs:extension base="xs:string">
<xs:attribute name="type" type="esaParameterItemTypeEnum"/>
<!--<xs:attribute name="valueScreenName" type="xs:string" use="optional"/>-->
</xs:extension>
</xs:simpleContent>
</xs:complexType>

```

```

<!--Result-->

<xs:complexType name="resultType">
<xs:choice minOccurs="0"> <!-- Result can be empty as well -->

<!--Simple values -->
<xs:sequence>
<xs:element name="value" type="xs:string"/>
</xs:sequence>

<!--Exception Result-->
<xs:sequence>
<xs:element name="exception" type="exceptionType"/>
</xs:sequence>

<!--Record list-->
<xs:sequence>
<xs:element name="record" type="recordType" maxOccurs="unbounded"/>
</xs:sequence>

<!--Structure list-->

```



```

<xs:sequence>
<xs:element name="structure" type="structureType" maxOccurs="unbounded"/>
</xs:sequence>

<!--Fields list-->
<xs:sequence>
<xs:element name="field" type="externalFieldType" maxOccurs="unbounded"/>
</xs:sequence>
  <!--Relations list-->
  <xs:sequence>
  <xs:element name="relation" type="externalRelationType" maxOccurs="unbounded"/>
  </xs:sequence>

  <!--Collections list-->
  <xs:sequence>
  <xs:element name="collection" type="externalCollectionType" maxOccurs="unbounded"
  />
  </xs:sequence>

  <!--Cursor result-->
  <xs:sequence>
  <xs:element name="cursor" type="cursorType"/>
  </xs:sequence>

  <!--ID list-->
  <xs:sequence>
  <xs:element name="id" maxOccurs="unbounded" type="xs:string">
  </xs:element>
  </xs:sequence>

  <!--ESA Parameter Metas list -->
  <xs:sequence>
  <xs:element name="esa-parameter" type="parametersMetaType" maxOccurs="unbounded"
  />
  </xs:sequence>

  <!--Value list -->
  <xs:sequence>
  <xs:element name="parameter-value" type="parameterValueType" maxOccurs="
  unbounded"/>
  </xs:sequence>

  <!--Simple values -->
  <xs:sequence>
  <xs:element name="nullValue" type="xs:boolean"/>
  </xs:sequence>
</xs:choice>
<xs:attribute name="response-to" use="required" type="xs:nonNegativeInteger"/>
</xs:complexType>
</xs:schema>

```

Data Mapped to Snippets

To assist in the coding of messages, many kinds of data are mapped to XML snippets associated with the ESA XML messages schema as follows:

External Records

External records are mapped to XML snippets with the form:

```
<record id="id-value"
old-id="old-id-value"
modified="modificationtimestamp">
<field name="field-name"> field value </field>
...
<field name="field-name">...</field>
<relation id="relation-id"
<related>related-object-id</related>
...
<related>related-object-id</related>
</relation>
...
<relation id="relation-id"
...
</relation>
<collection>
<record> tags, describing collection members.
</collection>
</record>
```

External Structures

External Structures are mapped to XML snippets with the form:

```
<structure name="name-value" screen-name="screen-name"/>
```

External Collections

External Collections are mapped to XML snippets with the form:

```
<collection id="collection-id"
screen-name="screen-name"
required="boolean-value"
updatable="boolean-value"
updatableOnCreate="boolean-value"/>
```

External Fields

External Fields are mapped to XML snippets with the form:

```
<field name="name-value"
screen-name="screen-name"
type="xsd-type-name"
required="true/false"
identifying="true/false"
updatable="true/false"
updatableOnCreate=" true/false"
maxLength="maximum length of string representation">
<!-- If a field is of enumeration type, it should have type "ew:enum"
and contain one or more nested <enum-value> elements: -->
<enum-value>
<value>value</value>
<screen-name>
name to be used in mapping editor
</screen-name>
</enum-value>
</field>
```

External Relations

External Relations are mapped to XML snippets with the form:

```
<relation id="relation-id"
related="related-structure-name"
screen-name="screen-name"
```

```
multiple="Boolean-value"
required="Boolean-value"/>
```

ESA Parameters Meta

The ESA parameters metadata are mapped to XML snippets with the form:

```
<esa-parameter
name="string-name"
type="single | radio | multi | expandable | xml"
required="true | false">
<label>Label to show above input</label>
<hint>Hint to show on the left</hint>
<item-type>
xsd:string | xsd:integer | xsd:double | xsd:Boolean
</item-type>
<default-value>
String-value | Integer-value | Double-value | Boolean-value
</default-value>
<!-- For parameters of type "radio", "multi" and
"expandable", a number of nested <enum-value>
elements should follow: -->
<enum-value>
<value>value</value>
<screen-name>
name to be used in mapping editor
</screen-name>
</enum-value>
</esa-parameter>
```

Cursors

Cursors are mapped to XML snippets with the form:

```
<cursor id="cursor-id"
page-size="records-per-page"
number-of-pages="number-of-data-pages"/>
```

ESA Interface Reference

This topic describes the methods in the ESA Java interface in the `com.supportwizard.sync.interfaces.esa.ExternalSystemAdapter` class. This can also be found in the `com/supportwizard/sync/interfaces/esa/ExternalSystemAdapter.java` file.

The ESA interface is the same for other interfaces, so the class and method comments can still be used.

checkDelayedCreate

Signature	<code>ExternalRecord checkDelayedCreate(String structure, String token)</code>
Description	Checks for a delayed record creation result. If a record creation has been delayed by the ESA, the sync core will call this method to get the actual operation result
Rationale	For some systems, it might be very inefficient to create records one by one, in the order Sync Core calls ESA. In this case, record creations can be delayed to form a single, or a few bulk updates to the external system. If such a delay is required, the <code>create()</code> method should schedule the creation and throw <code>EsaRecordDelayedException</code> exception. After Sync Core performs all <code>create()</code> calls, it will call the <code>checkDelayedCreate()</code> method to get the actual creation result. The moment ESA does the actual External System update is not important for the Sync Core. Common strategies are to either wait until <code>checkDelayedCreate()</code> is called, accumulating all creates or to do some more intermediate updates, one for every N <code>create()</code> calls.
Parameters	■ <code>structure</code> - structure to check operation for ■ <code>token</code> - operation token, as supplied within <code>EsaRecordDelayedException</code>
Returns	Created record
Exceptions	■ <code>EsaRecordException</code> if the delayed operation fails for the record ■ <code>EsaRecordDelayedException</code> if the operation is to be delayed again, or not yet done. Sync will then re-call this method later.

Example - see `ExternalSystemAdapterBase` base class

```
public ExternalRecord
checkDelayedCreate(String structure, String token) throws RemoteException,
EsaException, EsaRecordException {
```

```
throw new EsaException("Delaying is not supported by that
ESA");
}
```

checkDelayedDelete

Signature	Date checkDelayedUpdate (String structure, String token)
Description	Checks for a delayed record deletion result. If a record delete has been delayed by the ESA, the sync core will call this method to get the actual operation result
Rationale	Same as checkDelayedCreate() , but applied to delete() .
Parameters	■ structure - structure to check operation for ■ token - operation token, as supplied within EsaRecordDelayedException
Returns	Nothing
Exceptions	■ EsaRecordException if delayed operation fails for the record ■ OptimisticLockFailureException if the record is modified after the lastSeen timestamp. This means that a record was modified while sync was in progress. ■ EsaRecordDelayedException if the operation is to be delayed again, or not yet done. Sync will then re-call this method later.

Example - see ExternalSystemAdapter base class

```
public void checkDelayedDelete(String structure, String token) throws
RemoteException, EsaException,
EsaRecordException {
throw new EsaException("Delaying is not supported by that
ESA");
}
```

checkDelayedUpdate

Signature	Date checkDelayedUpdate (String structure, String token)
Description	Checks for a delayed record update result. If a record update has been delayed by the ESA, the sync core will call this method to get the actual operation result
Rationale	Same as <code>checkDelayedCreate()</code> , but applied to <code>update()</code> .
Parameters	■ structure - structure to check operation for ■ token - operation token, as supplied within <code>EsaRecordDelayedException</code>
Returns	New/updated record timestamp
Exceptions	■ <code>EsaRecordException</code> if delayed operation fails for the record ■ <code>OptimisticLockFailureException</code> if record is modified after <code>lastSeen</code> timestamp - this means that a record was modified while sync was in progress. ■ <code>EsaRecordDelayedException</code> if the operation is to be delayed again, or not yet done. Sync will then re-call this method later.

Example - see `ExternalSystemAdapterBase` base class

```
public Date checkDelayedUpdate(String
structure, String token) throws EsaException,
EsaRecordException {
throw new EsaException("Delaying is not supported by that
ESA");
}
```

closeCursor

Signature	void closeCursor (String cursorID)
Description	Closes the cursor. This call indicates that Agiloft doesn't need the cursor anymore and guarantees that <code>readDataPage</code> will never be called for this cursor.

Rationale	Indicates that the sync subsystem is not going to use the cursor anymore and ESA may free all cursor-related resources.
Parameters	cursorID - cursor ID
Returns	Nothing
Exceptions	None

Example - see `ExternalSystemAdapterBase` base class

```
public void closeCursor(String cursorID) throws EsaException,
RemoteException {
throw new EsaException("Not implemented");
}
```

Configure

Signature	String configure (String externalSystemID, boolean force)
Description	Indicates that the ESA is used by a Sync configuration. The method is called on the ESA when a new Sync configuration is created. Simple ESAs may just ignore this call and return the passed <code>externalSystemID</code>. If, for any reason, the ESA should not be used by the configuration, the ESA should return null unless the parameter 'force' is true. If force is true, ESA should adjust itself to be used by the configuration.
Rationale	A complex ESA may store additional per-configuration data or may require that only a single sync configuration exist per ESA installation. The <code>configure()</code> method provides a means to control this by notifying the ESA that it is about to be used by a configuration with the given External System ID. If the ESA returns null, the Agiloftadmin is asked to actively confirm that he wishes to use the ESA with the configuration. If he confirms, <code>configure()</code> is called again, with the force parameter set to true. ESA should treat this as a direct order from admin to be used with this External System ID from now on. The ESA may throw a <code>ConfigurationInvalidException</code> from the <code>startSync()</code> method if called with an old inactive External System ID.
Parameters	- externalSystemID - External System ID as set in the sync configuration. - force - indicates that admin insists, confirms a warning, configuring the ESA for this configuration.
Returns	externalSystemID or null if ESA can't be configured for this configuration.
Exceptions	None

Example - see `ExternalSystemAdapterBase` base class

```
public String configure(String
externalSystemID, boolean force) throws EsaException, RemoteException {
// Try new parameters
startSync(externalSystemID);
endSync();
return externalSystemID;
}
```

countRange

Signature	int countRange (String externalStructure, String idMin, String idMax)
Description	Counts the number of tickets, with IDs in range of [IDmin;IDmax] (inclusive). This method is a callback for the <code>HelperApi.detectDeleted(ExternalSystemAdapter, String, String, java.util.Date)</code> method and is only called if <code>HelperApi.detectDeleted</code> is called by the ESA
Rationale	This method is only called if the ESA calls the <code>HelperApi.detectDeleted()</code> method. <code>detectDeleted()</code> is based on binary division, checking the numbers of records, whose IDs fall in a range. Therefore, it needs to count the number of such external records through the ESA.
Parameters	▪ structure - structure to check records in ▪ idMin - minimal value of ID range ▪ idMax - maximum value of ID range
Returns	Nothing
Exceptions	None

Example - see `ExternalSystemAdapterBase` base class

```
public int countRange(String
externalStructure, String idMin, String idMax) throws EsaException,
RemoteException {
return 0;
}
```

Create

Signature	ExternalRecord create (String structure, ExternalRecord values)
Description	Creates a record in the external system
Rationale	Sync core calls this method to create a new external record, matching a new Agiloft record.
Parameters	■ structure - structure to create record in ■ values - field and relations of the record to create
Returns	Newly created record. In particular, the ID and Timestamp fields must be filled.
Exceptions	■ EsaRecordException if a record can't be created, ■ EsaRecordDelayedException if ESA wants to delay the creation to form a bulk update.

Example - see SampleEsa class

```
public ExternalRecord create(String structure, ExternalRecord values)
throws RemoteException, EsaException, EsaRecordException {
// Log debug info for troubleshooting
log.debug("create (" + structure + ")");
TableServant servant = name2servant.get(structure);
ExternalRecord result = servant.create(values);
assert result != null;
return result;
}
```

Delete

Signature	void delete (String structure, Date lastSeen, String pk)
Description	Deletes a record in the external system
Rationale	Sync Core calls this method to propagate deletion of an Agiloftrecord to the external system. Note: By default, deletions are not propagated at all. This can be changed within Sync Configuration editor, when editing table mapping in the Field Mapping wizard
Parameters	■ structure - structure to delete record in

	▪ lastSeen - modification time of the record, as seen by sync last time ▪ pk - of the record to update
Returns	Nothing
Exceptions	▪ <code>EsaRecordException</code> if a record cannot be deleted, ▪ <code>EsaRecordDelayedException</code> if ESA wants to delay the delete to form a bulk update. ▪ <code>OptimisticLockFailureException</code> if record is modified after lastSeen timestamp; this means that a record was modified while sync was in progress.

Example - see `SampleEsa` class

```
public void delete(String structure, Date
lastSeen, String pk) throws RemoteException, EsaException, EsaRecordException,
OptimisticLockFailureException {
// Log debug info for troubleshooting
log.debug("delete (" +
structure + ", " + pk + ")");
TableServant servant = name2servant.get(structure);
servant.delete(lastSeen, pk);
}
```

endSync

Signature	<code>void endSync()</code>
Description	Finishes the synchronization session. This call denotes the successful end of a synchronization. ESA may clean up any resources, release connections and so on needed to perform a synchronization. See also Release.
Rationale	Notify ESA that a synchronization is finished. ESA should release “per-synchronization” resources, typically allocated in <code>startSync()</code> by closing connections, freeing internal data structures, and so on.
Parameters	None
Returns	Nothing
Exceptions	<code>EsaException</code> if the ESA fails to free resources.

Example - see SampleEsa class

```

public void endSync() throws EsaException,
RemoteException {
// This ends sync
// Do nothing. Real ESA might disconnect / free resources / etc
// Log debug info for troubleshooting
log.debug("Sync is ended.");
}

```

getAllowedRunModes

Signature	int <code>getAllowedRunModes()</code>
Description	Returns Bit-OR'ed constants from RunModes , restricting synchronization run modes. For example, <code>RunModes.RUN_SYNC_ACTIONS</code> bit-OR <code>RunModes.RUN_EXTERNAL_TRIGGERED</code> would designate that ESA supports non-interactive synchronizations by both Agiloft and External System requests.
Rationale	An ESA can be run in three ways: ▪ Manual - this is when synchronization is manually triggered through the Agiloft GUI - the Sync menu item in the table toolbar. The synchronization itself is always run in the background, but some ESAs such as Google Contacts and Calendar ESA, require user interaction and can only be run in this mode. ▪ By Agiloft actions - Agiloft actions can be executed either by a rule or workflow which are configurable for the admin. For example, it is possible to set up a rule, running a sync action - an action type which runs the synchronization with a given External System ID - every time a record is modified. This allows for 'online' synchronizations. Another possibility is to set up a time-based rule to do batch synchronization. ▪ External Triggered - a synchronization can be triggered by calling the <code>HelperApi.startSync()</code> method from outside Agiloft. For some ESAs, it is impossible to initiate synchronization in any way other than by a request from the External System. In this case, it is advisable to implement the ESA as an HTTPS ESA, listen for an External System request in the system-specific manner and call <code>HelperApi.startSync()</code> when it is time to run sync. In this case <code>HttpXmlEsaTransport</code> should be used as a transport in <code>HelperApi</code>. See also HelperApi Interface. This method tells The sync core which types of operations are suitable for the ESA. Most ESAs are agnostic to the way synchronization is run and support all methods - <code>RunModes.ANY</code>.

Parameters	None
Returns	Bit-OR'ed mask, composed from constants in RunModes class.
Exceptions	None

Example - see ExternalSystemAdapterBase base class

```
public int getAllowedRunModes() throws
EsaException, RemoteException {
return RunModes.ANY;
}
```

getCollections

Signature	Set<ExternalCollection> getCollections (String structureOrCollection, Locale locale)
Description	Gets a list of collections from the given structure.
Rationale	
Parameters	▪ structureOrCollection - a collection to return fields for ▪ locale - locale to use for message localization
Returns	Collections list
Exceptions	None

Example - see SampleEsaclass

```
public Set<ExternalCollection>
getCollections(String structureOrCollection, Locale locale) throws
EsaException, RemoteException {
// Collections are used to represent hierarchical data and are something
// in between of relations and fields. It is a bit advanced concept.
// It is like relation, because it stores other records and is mapped to some EW
// table.
// It is like field, because it is updated at once, as a field
// This ESA doesn't use them.
Set<ExternalCollection> result = new
HashSet<ExternalCollection>();
assert result != null;
return result;
}
```

getCurrentUTCTime

Signature	Date getCurrentUTCTime()
Description	Gets current UTC time of a remote system, if available. If it is impossible to determine current time on the External System machine, this method should return null.
Rationale	Sync subsystem uses this method to compute the time shift between the Agiloft server where the sync subsystem is running and the External System. This is taken into account when checking if the record is modified or not. The shift is applied for all date-time "control" values passed to ESA – that is, timestamps in <code>getModified()</code> / <code>getDeleted()</code>, <code>record.modifiedAt</code>, <code>lastSeen</code> parameter and so on. The shift is not applied for converting record date/date-time/time fields inside <code>ExternalRecord</code> if the record data values are not affected.
Parameters	None
Returns	Current UTC time of a remote system, if available, or null.
Exceptions	None

Example - see `ExternalSystemAdapterBase` base class

```
public Date getCurrentUTCTime() throws
EsaException, RemoteException {
return null; // No time
synchronization
}
```

getDeleted

Signature	Set<String> getDeleted (String structure, Date since)
Description	Gets IDs of records deleted in the external system after a given timestamp. ESA may or may not pay attention to this timestamp. It is acceptable for the ESA to respond to this call with all of the IDs ever deleted in the system - sync will handle this properly, but this will result in more traffic between the ESA and Agiloft. The timestamp corresponds to the time of the last successful invocation of <code>getDeleted()</code>. If an ESA accumulates deletion information, it may always return all accumulated IDs and purge them after successful <code>endSync()</code>.

	In any case, this call may result in a large amount of data to be transferred. If the ESA predicts such traffic, it should return NULL from this method (NOT an empty set!). In this case, Agiloft will make use of the <code>getDeletedPaged</code> method. The ESA may decide how to transfer data at runtime depending on the amount. In any case, Agiloft will call <code>getDeleted</code> first and only call <code>getDeletedPaged</code> if <code>getDeleted</code> returns NULL <code><nullValue>true</nullValue></code>
Rationale	This method provides the sync subsystem with a list of deletions that happened in the External System since the last sync.
Parameters	■ structure - structure to read ■ since - return records deleted since the timestamp
Returns	IDs of records modified after the given timestamp
Exceptions	None

Example - see SampleEsa class

```
public Set<String> getDeleted(String
structure, Date since) throws EsaException, RemoteException {
// Log debug info for troubleshooting
log.debug("getDeleted (" + structure + ", " + since + ")");
TableServant servant = name2servant.get(structure);
Set<String> result = servant.getDeleted(since);
assert result != null;
return result;
}
```

getDeletedPaged

Signature	Cursor getDeletedPaged (String structure, Date after)
Description	Gets deleted records in a paged manner. See <code>getDeleted</code> description. This method is only called if the <code>getDeleted</code> call returned NULL (<code><nullValue>true</nullValue></code>).
Rationale	<code>getDeleted()</code> may return a large amount of data. For an HTTPS ESA, this may cause problems because of network timeouts and proxy server settings. In this case, it is advised to use the <code>getDeletedPaged()</code> method.
Parameters	■ structure - structure to read ■ since - timestamp

Returns	Cursor object, used as a token in readDataPage() method
Exceptions	None

Example - see ExternalSystemAdapterBase base class

```
public Cursor getDeletedPaged(String
structure, Date since) throws EsaException, RemoteException {
throw new EsaException("Not implemented");
}
```

getDetailedReport

Signature	String getDetailedReport ()
Description	A detailed, debug level, progress report or null, if ESA does not provide this feature
Rationale	Provide a debug-level progress report. This report is accessible when clicking 'To view raw log file, click here' in the synchronization progress window.
Parameters	None
Returns	Report text - plain text or HTML markup - or null
Exceptions	None

Example - see ExternalSystemAdapter base class

```
public String getDetailedReport() throws
RemoteException, EsaException {
return null;
}
```

getFieldList

Signature	Set<ExternalField> getFieldList (String structureOrCollection, Locale locale)
Description	Gets a list of fields for a given structure or collection. It is not necessary, but it is allowed, to include ID and timestamp "Modified At" fields here.
Rationale	

	The sync subsystem maintains the mapping between the fields of an AL table and the fields of the mapped external structure. To edit the mapping, select Setup > Sync Configuration , visit the Mapping tab and map a table to a structure, or click the Edit button for already mapped pairs. The list of External Fields in the GUI is the set returned by this ESA method.
Parameters	▪ <code>structureOrCollection</code> - a collection to return fields ▪ <code>locale</code> - locale to use for message localization
Returns	List of external fields of a structure
Exceptions	None

Example - see SampleEsaclass

```

public Set<ExternalField>
getFieldList(String structureOrCollection, Locale locale) throws EsaException,
RemoteException {ResourceBundle i18n = getResources("com.supportwizard.sync.
sampleesa.sampleesa", locale);
Set<ExternalField> result = new HashSet<ExternalField>();
// Switch on the logical structure name (ones, that were returned by
getStructureList())
if(CONTACTS.equals(structureOrCollection)) {
// Return fields for contacts.
// ID field. result.add(new ExternalField(
CONTACT_ID,
// Logical name. This one is used in ExternalRecord, passed to create()/update()
i18n.getString("contacts.id"),
// Localized name, shown to EW administrator when setting up field mappings
XsdType.INT,
// Type. This one is a numeric ID false,
// Not used for unknown records identification.
// If IDs are somewhat persistent among two systems (i.e. John Bull should have ID
#1 in both systems, it should be identifying field
// If IDs are allowed to differ, they usually will and we are better
// to match records on something more meaningful, such as full name
true,
// Required false, true,
// Not updatable except on create
20));
// No longer than 20 digits
// Note we have not specified
that the field is a primary key. PK value is
supplied separately with each record
// and is not a must o
expose the ID field at all.
// You may well do not, if you
don't want to map it and use its value somewhere
in EW
// Simpler full name and email

```

```

fields
result.add(new
ExternalField(CONTACT_FULLNAME,
i18n.getString("contacts.fullname"),
XsdType.STRING,
true, true, true, true,
250));
result.add(new
ExternalField(CONTACT_EMAIL, i18n.getString("contacts.email"),
XsdType.STRING,
true, false, true, true,
250));
} else if(CASES.equals(structureOrCollection)) {
// Fields for cases table
result.add(new
ExternalField(CASE_ID, i18n.getString("cases.id"), XsdType.INT,
false, true, false, true,
20));
result.add(new
ExternalField(CASE_SUMMARY, i18n.getString("cases.summary"),
XsdType.STRING,
true, true, true, true,
250));
}
// Always return something non-null.
assert result != null;
return result;
}

```

getModified

Signature	Set<ExternalRecord> getModified (String structure, Date after)
Description	Gets all records modified in the external system after the given timestamp. For some systems, this may result in a very large amount of data to be transferred, especially for the initial sync. If the ESA predicts such high traffic, it should return null from this method - not an empty set. In this case, Agiloft will make use of the <code>getModifiedPaged</code> method. An ESA may decide how to transfer data at runtime depending on the amount. In any case, Agiloft will call <code>getModified</code> first and only call <code>getModifiedPaged</code> if <code>getModified</code> returns null - <code><nullValue>true</nullValue></code>.
Rationale	To do the synchronization, the Sync subsystem must know which external records have been modified since the last sync and read their content in order to update Agiloft records. Basically this method is equivalent to: <pre>select * from structure where structure.modified >= since;</pre>

	With the special case where "since" is null, which should return all records.
Parameters	■ structure - structure to read ■ after – timestamp, or null if all records are to be returned
Returns	Records modified after given timestamp
Exceptions	None

Example - see SampleEsaclass

```

public Set<ExternalRecord>
getModified(final String structure, final Date after) throws EsaException,
RemoteException {
// Log debug info for troubleshooting
log.debug("getModified (" + structure + ", " + after + ")");
// Though it is not a must, having a "servant" class per table is usually
convenient.
// It is a pure implementation detail, but it is often convenient to organize ESA
in that way,
// avoiding lengthy IFs switches on structure name in in
getModified/getDeleted/create/update/delete methods
// Note: if meta-data is not hardcoded, it is usually a good idea to place
// meta-data reading (getFields(), getRelations(), getCollections()) to "servants"
as well.
TableServant servant = name2servant.get(structure);
Set<ExternalRecord> result = servant.getModified(after);
assert result != null;
return result;
}

```

getModifiedPaged

Signature	Cursor getModifiedPaged (String structure, Date after)
Description	Gets modified records in a paged manner. See the <code>getModified</code> description. This method is only called if the <code>getModified</code> call returned NULL - <code><nullValue>true</nullValue></code>.
Rationale	<code>getModified()</code> may return a large amount of data, especially for initial synchronization, when all records are to be read. For an HTTPs ESA, this may cause problems because of network timeouts and proxy server settings. In this case, it is advised to use <code>getModifiedPaged</code> methods.

Parameters	■ structure - structure to read ■ after – timestamp, or null if all records are to be returned
Returns	Cursor object, used as a token in <code>readDataPage()</code> method
Exceptions	None

Example - see ExternalSystemAdapterBase base class

```
public Cursor getModifiedPaged(String
structure, Date after) throws EsaException, RemoteException {
throw new EsaException("Not implemented");
}
```

getParametersMeta

Signature	List<EsaParameterMeta> getParametersMeta (Locale locale)
Description	Gets localized ESA parameter metadata.
Rationale	Agiloft maintains ESA parameter values through the Sync Configuration wizard. To generate the GUI, Agiloft must know the parameter name, type, etc. This method provides that information. The Sync Configuration wizard ESA Parameters tab is constructed based on the information returned by that method.
Parameters	Locale - locale to use for message localization
Returns	Gets localized ESA Parameters metadata.
Exceptions	None

Example - see SampleEsaclass

```
public List<EsaParameterMeta>
getParametersMeta(Locale locale) throws EsaException, RemoteException {
List<EsaParameterMeta> result = new ArrayList<EsaParameterMeta>();
// Screen names can be localized, see resource properties bundle.
// A simpler ESA may just ignore "locale" parameter and return hard-coded names
ResourceBundle i18n = getResources("com.supportwizard.sync.sampleesa.sampleesa",
locale);
// There is a single parameter we need - where to store our files
result.add(new EsaParameterMeta(WORK_DIR,
// Logical name, used to obtain parameter value
EsaParameterValueType.TEXT,
// String
EsaParameterType.SINGLE,
```

```
// Just one plain value
true,
// Must be set
i18n.getString("workdir.label"),
// Screen name, short
i18n.getString("workdir.hint"),
// Screen hint, explanatory and long
new
EsaParameter("/tmp/sample-esa-data");// Default value
));
// Always return something non-null.
assert result != null;
return
}
```

getProgressReport

Signature	String getProgressReport ()
Description	HTML progress report, to the current moment or null, if ESA does not provide this feature. The report should be provided in fully rendered HTML, starting with an <html>and ending with </html>
Rationale	Provides a nicely-formatted ESA-specific progress screen. If an ESA provides this report, that is, returns non-null from this method, the report HTML text completely replaces the standard progress screen. The ESA should provide the full-featured progress GUI, including automatic page refresh and other GUI elements such as a Close button if necessary. Please see ProgressUIHelper utility class for organizing self-refreshing page. Please also see Example 2 below. ESA may also use this mechanism to interact with the user, providing a fully interactive GUI. In this case, you should return RunModes.MANUAL from the getAllowedRunModes() method.
Parameters	None
Returns	Report HTML text - <html> to </html> - or null
Exceptions	None

Example 1 - ExternalSystemAdapter base class

```
public String getProgressReport() throws
RemoteException, EsaException {
return null;
}
```

Example 2

```
private ProgressUIHelper progressHelper =
new ProgressUIHelper(3000); // refresh every 3 sec
...
public String startSync() {
...
progressHelper.setRefreshing(true); // Enable progress page refreshes
}
public void endSync() {
...
progressHelper.setRefreshing(false); // All done - stop progress page
refreshes
}
public String getProgressReport() throws
RemoteException, EsaException {
return progressHelper.htmlProgressReport("This is a <b>custom</b> Progress
Report!");
}
```

getRelations

Signature	Set<ExternalRelation> getRelations (String structureOrCollection, Locale locale)
Description	Gets a list of relations from the given structure. This should only include relations that are navigable from this object: relations -from- this structure to others.
Rationale	Sync subsystem maintains a mapping of links between Agiloft tables - Linked Field Sets - and relations of the mapped external structures. For example, a table of Widgets might include a Purchased By field, which links to the prospect who purchased that Widget. To see the mapping, edit Sync Configuration > Relations tab . The list of External Relations in the GUI is the set returned by this ESA method.
Parameters	▪ structureOrCollection - a collection to return fields ▪ locale - Locale to use for message localization
Returns	A list of relations from the structure
Exceptions	None

Example - see SampleEsaclass

```
public Set<ExternalRelation>
getRelations(String structureOrCollection, Locale locale)
```

```

throws EsaException, RemoteException
{ResourceBundle i18n = getResources("com.supportwizard.sync.sampleesa.sampleesa",
locale);
Set<ExternalRelation> result = new HashSet<ExternalRelation>();
// There is a single relation - a contact who submits a case
if(CONTACTS.equals(structureOrCollection))
{result.add(new ExternalRelation(
CASE_CUSTOMER,
// Logical name
CONTACTS,
// Logical name of a linked table
i18n.getString("cases. customer"),
// Screen name
false,

// Doesn't hold multiple values
false,

// Not required, can be empty
true));
// Allows postponed updates.
// This option is used to resolve cycles in dependencies
// A rule of the thumb is to allow postponed updates on
// all non-required relations
}
assert result != null;
return result;
}

```

getStructureList

Signature	Set<ExternalStructure> getStructureList (Locale locale)
Description	Gets a list of external data structure names. A 'structure' is a logical data unit, mappable to an Agiloft table. It may be a table, a folder, a class in OODB, or any other type of a data grouping.
Rationale	The sync subsystem maintains the mapping between AL tables and external structures. You may edit this mapping in the Sync Configuration wizard, mappings tab. The list of External Structures in the GUI is the set returned by this ESA method.
Parameters	Locale - locale to use for message localization
Returns	The list of external system structures
Exceptions	None

Example - see SampleEsaclass

```

public Set<ExternalStructure>
getStructureList(Locale locale) throws EsaException,
RemoteException
{
    ResourceBundle i18n =
    getResources("com.supportwizard.sync.samplesesa.samplesesa",
    locale);
    // CONTACTS, CASES are internal "logical" names. They are used
    in all other ESA calls.
    Set<ExternalStructure> structures = new
    HashSet<ExternalStructure>();
    structures.add(new ExternalStructure(CONTACTS,
    i18n.getString("contacts.screenname")));
    structures.add(new ExternalStructure(CASES,
    i18n.getString("cases.screenname")));
    return structures;
}

```

leaseCursor

Signature	void leaseCursor (String cursorID)
Description	Resets cursor expiration timer, if ESA maintains a timer.
Rationale	Allows sophisticated timeout-based resource management inside ESA
Parameters	cursorID - cursor ID
Returns	Nothing
Exceptions	None

Example - see ExternalSystemAdapterBase base class

```

public void leaseCursor(String cursorID)
throws EsaException, RemoteException {
    throw new EsaException("Not implemented");
}

```

leaseSession

Signature	void leaseSession ()
------------------	-----------------------------

Description	Resets the session timeout counter if the ESA has one. If the ESA does use a timeout-based resource de-allocation, it may use this method as an indication that the ESA is still used. Otherwise, the ESA may simply ignore this call.
Rationale	Allows sophisticated resource management within the ESA.
Parameters	None
Returns	Nothing
Exceptions	EsaException if the ESA has already expired due to internal timeout.

Example - see ExternalSystemAdapterBase base class

```
public void leaseSession()
throws EsaException,
RemoteException {
// NO-OP
}
```

needSyncAgain

Signature	boolean needSyncAgain()
Description	Checks whether another synchronization cycle should be run immediately. The method is invoked after <code>endSync()</code> .
Rationale	Sometimes it might be necessary to re-run synchronization again after it just finished. For example, a record update may trigger cascade-updates of other records, possibly after these other records were synchronized. To propagate those changes back to Agiloft, it is necessary to run synchronization once more. In such cases, ESA should return true from this method.
Parameters	None
Returns	True if ESA wants another synchronization round to be run immediately.
Exceptions	None

Example - see ExternalSystemAdapterBase base class

```
public boolean needSyncAgain() throws
RemoteException, EsaException {
return false;
}
```

Read

Signature	ExternalRecord read (String structure, String pk)
Description	Reads a single record in the External System
Rationale	Sync subsystem may need to read a single external record. For example, this method is called if a record was not updated during the last synchronization because of an error.
Parameters	■ structure - read record of that structure ■ pk - read record with that ID
Returns	Read ExternalRecord
Exceptions	EsaRecordException, if a record can't be read/does not exist

Example - see SampleEsa class

```
public ExternalRecord read(String structure, String pk) throws
RemoteException, EsaException, EsaRecordException {
// Log debug info for troubleshooting
log.debug("read (" + structure + ", " + pk + ")");
TableServant servant =
name2servant.get(structure);
ExternalRecord result = servant.read(pk);
assert result != null;
return result;
}
```

readDataPage

Signature	Set readDataPage (String cursorID, int pageIndex)
Description	Gets data from a cursor. This method is used to actually access the data in the cursor.
Rationale	<code>getModifiedPaged()</code> / <code>getDeletedPaged()</code> do not return any data. They allocate the Cursor object on the ESA side. Actual data are read through this method.
Parameters	■ cursorID - cursor ID ■ pageIndex - data page number, 0 based

Returns	Data page consisting of either External Records, if the cursor was created within <code>getModifiedPaged</code> , or record IDs, if the cursor is from <code>getDeletedPaged</code> .
Exceptions	None

Example - see `ExternalSystemAdapterBase` base class

```
public Set readDataPage(String cursorID, int pageIndex) throws
EsaException, RemoteException
{
    throw new EsaException("Not implemented");
}
```

Release

Signature	<code>void release()</code>
Description	Releases the ESA. This call indicates that the sync core will not use the ESA instance anymore and the ESA may therefore free all used resources such as connection factories, and shutdown. This is the last method called on the ESA instance during the sync cycle.
Rationale	Completely releases the ESA. There could be several sync cycles, run on the single ESA instance in sequence. For example, this happens if <code>needSyncAgain()</code> returns true. Every cycle is ended with an <code>endSync()</code> call. In contrast, <code>release()</code> is called only once, after the last <code>endSync()</code>. This allows the ESA to keep some resources during all sync cycles, between first <code>startSync()</code> and <code>release()</code> calls.
Parameters	None
Returns	Nothing
Exceptions	<code>EsaException</code> if the ESA fails to free resources.

Example - see `ExternalSystemAdapterBase` base class

```
public void release() throws EsaException,
RemoteException {
    // NO-OP
}
```

setHelperAPI

Signature	void setHelperAPI (HelperApi helperApi)
Description	Provides the ESA with a means to call the Sync Core. This is the first method called on the ESA after the instance is created. ESA should store a reference to HelperApi in this method.
Rationale	The Helper API interface is fully described in ESA HelperAPI Interface . Its major functions are to provide the actual ESA Parameters values stored inside Agiloft and to help with deletion tracking. It is also used to start externally triggered synchronizations.
Parameters	helperApi - a reference to the sync subsystem Core API interface
Returns	Nothing
Exceptions	None

Example - see `ExternalSystemAdapterBase` base class

```
/**
 * Sets
 * HelperApi to be used by ESA, if necessary
 * @param
 * helperApi
 */
public void
setHelperApi(HelperApi helperApi) {
    this.helperApi = helperApi;
}
```

startSync

Signature	String startSync (String externalSystemID)
Description	Starts a synchronization. This is the very first method called during Sync.
Rationale	Notify ESA that a synchronization is to be started now. The ESA should prepare itself for doing the synchronization by opening connections to the external system, allocating internal data structures and so forth.
Parameters	externalSystemID - External System ID, as defined in the Sync Configuration wizard.
Returns	Sync version, supported by the ESA. This should be one of the SYNC_VERSION_xxx constants.
Exceptions	EsaException if a sync can't be started.

Example - see SampleEsa class

```

public String startSync(String externalSystemID)
throws EsaException,
RemoteException {
this.externalSystemID = externalSystemID;
// Get callback reference
to the sync core
HelperApi syncCoreApi = getHelperApi();
// Read work dir parameter value
List<EsaParameter>
paramValues = syncCoreApi.getParameter(externalSystemID,
WORK_DIR);
assert paramValues.size() == 1; // There must be a single string (parameter
type is TEXT, SINGLE).
String workDirPath =
paramValues.get(0).getStrValue();
assert workDirPath != null;
// Initialize work
dir
workDir = new File(workDirPath);
if(!workDir.exists()) {
workDir.mkdirs();
}
// Create servant classes
name2servant = new
HashMap<String, TableServant>();
name2servant.put(CONTACTS, new ContactsServant());
name2servant.put(CASES, new CasesServant());
// Log debug info
for troubleshooting log.debug("Started sync successfully, work dir is " + workDir.
getAbsolutePath());
// This ESA support sync version 1.1. Please always use the latest you can.
return ExternalSystemAdapter.SYNC_VERSION_1_1;
}

```

syncErrorNotify

Signature	void syncErrorNotify (String message)
Description	Called by Sync to notify the ESA if an error occurs on sync. The exact processing scenario is not defined and depends on the ESA, which may take actions such as notifying the user or logging the error.
Rationale	Notify ESA that sync ends with an error. The ESA may log this or notify the admin somehow.

	Usually, this method is most important for HTTPs ESA that are running as part of another system.
Parameters	Message - error message
Returns	Nothing
Exceptions	None

Example - see ExternalSystemAdapter baseclass

```
public void syncErrorNotify(String message)
throws RemoteException, EsaException {
log.error("Error on sync: " + message);
}
```

Update

Signature	Date update (String structure, Date lastSeen, ExternalRecord values)
Description	Updates a record in the external system
Rationale	Sync subsystem calls this method to propagate changes in an Agiloft record to its External System peer record.
Parameters	■ structure - structure to update record in ■ lastSeen - modification time of the record, as seen by sync last time ■ values - field and relations of the record to update
Returns	New record modification timestamp
Exceptions	■ <code>EsaRecordException</code> if a record can't be updated, ■ <code>EsaRecordDelayedException</code> if the ESA wants to delay the update and run a bulk update. ■ <code>OptimisticLockFailureException</code> if record is modified after lastSeen timestamp. This means that a record was modified while a sync was in progress.

Example - see SampleEsa class

```
public Date update(String structure, Date
lastSeen, ExternalRecord values) throws RemoteException,
```

```

EsaException, EsaRecordException, OptimisticLockFailureException {
// Log debug info for troubleshooting
log.debug("update (" + structure + ", " + values.getId() + ")");
TableServant servant = name2servant.get(structure);
Date result = servant.update(lastSeen, values);
assert result != null;
return result;
}

```

Calls To and From the ESA

This topic describes all of the calls made between the ESA and the Helper API.

Calls to ESA from Sync

The following list describes all calls made by sync to an ESA, with the expected results - not including exceptions.

boolean needSyncAgain()

Sometimes, the ESA may decide there is a necessity to re-run synchronization immediately after sync finish. An example is if update actually updates more than 1 record, so data are changed in a broader scope than the Sync subsystem expects. In this case, the ESA should return true from this method.

Input XML Example

```
<needSyncAgain>  
</needSyncAgain>
```

Output XML Example

```
<value>boolean value  
</value>
```

Cursor getDeletedPaged(String structure, Timestamp after)

Gets deleted records in a paged manner. See `getDeleted()` description. This method is only called if `getDeleted()` call returned NULL. The Cursor object returned contains information about the number of data pages, data page size and so on.

Input XML Example


```

<getDeletedPaged>
<structure>
  structure-name
</structure>
<after>
  timestamp-value
</after>
</getDeletedPaged>

```

Output XML Example

```
<cursor> snippet
```

Cursor getModifiedPaged(String structure, Timestamp after)

Gets modified records in a paged manner. See "getModified()" description. This method is only called if "getModified()" call returned NULL. The Cursor object returned contains information about the number of data pages, data page size and so on.

Input XML Example

```

<getModifiedPaged>
<structure>
  structure-name
</structure>
<after>
  timestamp-value
</after>
</getModifiedPaged>

```

Output XML Example

```
A number of ExternalRecord snippets - <record> tags
```

Date getCurrentTime()

Gets current UTC time of a remote system, if available. Sync uses this method to compute the time shift between Agiloft server and the external system. The shift is taken into account for all datetime control values, passed to the ESA - getModified(), record.modifiedAt, lastSeen timestamps, etc. Note: This shift is not applied when converting record date. Datetime or time fields.

Input XML Example

```
<getCurrentTime>
</getCurrentTime>
```

Output XML Example

```
<value>timestamp value
</value>
```

ExternalRecord read(String structure, String id)

Reads a single record.

Input XML Example

```
<read>
<structure>
  structure-name
</structure>
<id>id-value</id>
</read>
```

Output XML Example

```
ExternalRecord
snippet
```



int countRange(String idMin, String idMax)

Returns a number of external system records, whose external ID is in the given range numerically or alphabetically, whichever is applicable. You should only implement this method if your ESA calls "detectDeleted" HelperApi method.

Input XML Example

```
<countRange>
  <min-id>
    String-value
  </min-id>
  <max-id>
    String-value
  </max-id>
</countRange>
```

Output XML Example

```
<value>integer value
</value>
```

List getParametersMeta(String locale)

Gets localized ESA Parameters meta-data.

Input XML Example

```
<getParametersMeta>
  <locale>String-value</locale>
</getParametersMeta>
```

i Output XML Example

A number of <esa-parameter> tags.

Set getDeleted(String structure, Timestamp after)

Gets IDs of records, deleted in external system after given timestamp For some systems, this may result in a large amount of data to be transferred. If the ESA predicts such traffic, it should return NULL from this method, not an empty set. In this case, Agiloft will make use of getDeletedPaged() method. The ESA may decide how to transfer data at runtime depending on the amount. In any case, Agiloft will call getDeleted() first and only call getDeletedPaged() if "getDeleted()" returns NULL.

i Input XML Example

```
<getDeleted>
  <structure>
    structure-name
  </structure>
  <after>
    timestamp-value
  </after>
</getDeleted>
```

i Output XML Example

```
<id>id-value</id>
...
<id>id-value</id>
```

Or

```
<nullValue>true</nullValue>
```

Set getFieldList(String structureOrCollection, String locale)

Gets a list of fields, for a given structure. ID and ModifiedAt fields should not be included.

Input XML Example

```
<getFieldList>
<structure>
  structure-name
</structure>
<locale>
  lang-value
</locale>
</getFieldList>
```

Output XML Example

A number of ExternalFields snippets - <field> tags

Set getModified(String structure, Timestamp after)

For some systems, this may result in a great amount of data to be transferred, especially for the initial sync. If the ESA predicts such traffic, it should return NULL from this method, not an empty set. In this case, Agiloft will use the getModifiedPaged() method.

The ESA may decide how to transfer data at runtime depending on the amount. In any case, Agiloft will call getModified() first and only call getModifiedPaged() if getModified() returns NULL

Input XML Example

```

<getModified>
<structure>
  structure-name
</structure>
<after>
  timestamp-value
</after>
</getModified>

```

i Output XML Example

A number of ExternalRecord snippets - <record> tags

Or

```
<nullValue>true</nullValue>
```

Set getRelations(String structureOrCollection, String locale)

Gets a list of relations from the given structure. This should only include relations, navigable from this object. A Locale argument can be used to localize relation screen names.

i Input XML Example

```

<getRelations>
<structure>
  structure-name
</structure>
<locale>
  lang-value
</locale>
</getRelations>

```

i Output XML Example

A number of <relation> snippets.

Set getStructureList()

Gets a list of external data structure names. Structure is a logical data unit, mappable to an Agiloft table; a table, a folder, a class in OODB or any other type of data grouping.

Input XML Example

```
<getStructureList>
</getStructureList>
```

Output XML Example

```
External Structure snippet
```

Set readDataPage(Cursor cursor, int pageIndex)

Gets data from a cursor. This method is used to actually access the data in the cursor.

Input XML Example

```
<readDataPage>
<cursor-id>
  cursor ID
</cursor-id>
<page-index>
  data page index
</page-index>
</readDataPage>
```

i Output XML Example

A number of ExternalRecord snippets - <record> tags -
OR <id>id-value</id>, depending on the cursor type

String startSync(String externalSystemID)

Starts a synchronization. This is the very first method, called during Sync. Return value is the Sync protocol version, supported by the ESA. Currently it must be "1.0"

i Input XML Example

```
<startSync>
  <external-system-id>
    String-value
  </external-system-id>
</startSync>
```

i Output XML Example

```
<value>1.0</value>
```

Timestamp update(String structure, Timestamp lastSeen, ExternalRecord values)

Updates the record in external system, returns update timestamp. Only does the update if the record is not modified since lastSeen timestamp. If the record is modified, throws exception - see below.

i Input XML Example

```
<update>
<structure>
  structure-name
</structure>
```



```

<last-seen>
  timestamp-value
</last-seen>
An ExternalRecord snippet (<record>)
</update>

```

Output XML Example

```
<value>timestamp-value</value>
```

void delete(String structure, Timestamp lastSeen, String pk)

Deletes the record in external system, returns true. Only does the deletion if the record is not modified since lastSeen timestamp. If the record is modified, throws exception - see below.

Input XML Example

```

<delete>
<structure>
  structure-name
</structure>
<last-seen>
  timestamp-value
</last-seen>
<id>id-value</id>
</delete>

```

void endSync()

Returns Bit-ORed constants, restricting synchronization run modes:

- 1 - Run Manually allowed
- 2 - Run by Sync Actions allowed
- 4 - Run by external system request allowed.

For example, 6 - 2 bit-OR 4 - would designate that the ESA supports non-interactive synchronizations by both Agiloft and external system requests

Input XML Example

```
<getAllowedRunModes>
</getAllowedRunModes>
```

Output XML Example

```
<value>
  integer value
</value>
```

void endSync()String configure(String externalSystemID, boolean force)

Configures the ESA to use with an Agiloft instance. This method is called on the ESA when a Sync configuration is created. This also allows a check on ESA availability. The implementation of this method depends on the ESA. Simple ESAs, which don't need to communicate Agiloft back; i.e. neither use Helper API outside of syncn cycle nor initiate Sync by external system requests; may simply ignore this call and return the passed externalSystemID. More complex ESAs should retain these values somewhere and use them to call Agiloft and to authenticate themselves by externalSystemID. If such an ESA is already configured - already has an associated ExternalSystemID - it must return the stored value and only replace it by the new parameters if parameter "force" is true.

Input XML Example

```
<configure>
<external-system-id>
  String-value
</external-system-id>
<force>
  Boolean-value
</force>
</configure>
```



Output XML Example

```
<value>
  String-value
</value>
```

void leaseCursor(Cursor cursor)

Closes the cursor. This call indicates that Agiloft doesn't need the cursor anymore and guarantees that readDataPage will not be called again for this cursor.

Input XML Example

```
<closeCursor>
<cursor-id>
  cursor ID
</cursor-id>
</closeCursor>
```

void leaseSession()

Resets session timeout counter (if the ESA has any).

Input XML Example

```
<leaseSession>
</leaseSession>
```

Calls from ESA to Helper API

The following list describes all calls made to sync by an ESA, with the expected results - not including exceptions.

boolean isKnownID(String externalSystemID, String externalStructure, String id)

Asks Agiloft whether it has a known peer for this external ID.

Input XML Example

```
<isKnownID>
  <external-system-id>
    String-value
  </external-system-id>
  <structure>
    structure-name
  </structure>
  <id>String-value</id>
</isKnownID>
```

Output XML Example

```
Boolean-value
```

Object getParameter(String externalSystemID, String name)

Gets an ESA parameter value, stored on the Agiloft Server.

Input XML Example

```
<getParameter>
  <external-system-id>
    String-value
  </external-system-id>
  <name>
    String-value
  </name>
</getParameter>
```

i Output XML Example

```
<value>
Value. The type depends on the ESA parameter type.
</value>
...
There can be several <value> tags within result
```

Set detectDeleted(String externalSystemID, String externalStructure, Timestamp after)

This is a helper method for deletion detection over auto-incremented IDs. The ESA interface must support an optional countRange method in this case.

i Input XML Example

```
<detectDeleted>
<external-system-id>
  String-value
</external-system-id>
<after>
  Timestamp-value
</after>
<structure>
  structure-name
</structure>
</detectDeleted>
```

i Output XML Example

```
<id>String-value</id>
...
<id>String-value</id>
```

Set enumerateKnownIDs(String externalSystemID, String externalStructure, Timestamp knownBefore)

Asks Agiloft to enumerate external system IDs, known by Agiloft at some moment. the ESA can then check whether records with such IDs exist and compose a list of deleted records.

Input XML Example

```
<enumerateKnownIDs>
  <external-system-id>
    String-value
  </external-system-id>
  <known-before>
    Timestamp-value
  </known-before>
  <structure>
    structure-name
  </structure>
</enumerateKnownIDs>
```

Output XML Example

```
<id>String-value</id>
...
<id>String-value</id>
```

void startSync(String externalSystemID)

This method triggers Sync with an external system, identified by externalSystemID. The method can be used by ESA or any other application, if using HTTPs transport to send a message to Helper Api to start synchronization, based on some event in the external system.

Input XML Example

```
<startSync>
<external-system-id>
  String-value
</external-system-id>
</startSync>
```

void trackRecordDeletion(String externalSystemID, String externalStructure, Timestamp time, String id)

Places a record deletion entry in the internal log associated with this externalSystemID. A timestamp argument can be useful if the ESA chooses to batch reports of record deletions. However, if some deletion reports are delayed, the ESA should flush them as soon as it receives any call on the ESA interface, prior to actually processing the call.

Input XML Example

```
<trackRecordDeletion>
<external-system-id>
  String-value
</external-system-id>
<time>
  Timestamp-value
</time>
<structure>
  structure-name
</structure>
</trackRecordDeletion>
```

Building a Custom ESA

This topic provides some direction for creating a custom External System Adapter (ESA) implementation, based on the Sample ESA provided for free with every Agiloft KB. Follow the steps provided in [Using the Sample ESA](#), replacing the default implementation with your own ESA details.

Read and familiarize yourself with the Javadoc comments provided with the ESA package. The `ExternalSystemAdapter.java` document in `com/supportwizard/sync/interfaces/esa` lists all of the ESA methods and explains their semantics and parameters. In addition, you should read and understand the Sample ESA code.

Create the ExternalSystemAdapter Interface

You could start by using `ExternalSystemAdapterBase` as the base class. The Java file is located at `com/supportwizard/sync/interfaces/esa`. `ExternalSystemAdapterBase` implements auxiliary methods in a default manner suitable for most ESAs. If you need some advanced functionality later, you can always override these default implementations.

1. Based on your setup, decide on a communication style - Command Line or HTTPS
2. Open `JavaExecutableEsa.java`, located at `com/supportwizard/remoteesa`, and look for the following lines:

JavaExecutableEsa

```
// Replace the following line with "new MyESA()"
ExternalSystemAdapter esa = new SampleEsa();
// CL ESA variant
EsaRunner.runCommandLineEsa(esa);
// For HTTPS ESA, comment/uncomment following line (need to supply ewServerURL
and externalSystemID)
// EsaRunner.runHttpsEsa(esa, ewServerURL, externalSystemID);
```

3. Replace the following line: `ExternalSystemAdapter esa = new SampleEsa();` with your new class.
 - a. For a CL ESA, just reassemble the example using `make.sh` or `make.bat` scripts. You now have your own ESA ready to be launched.
 - b. For an HTTPS ESA, comment the line: `EsaRunner.runCommandLineEsa(esa);` and uncomment the line: `// EsaRunner.runHttpsEsa(esa, ewServerURL, externalSystemID);`.
 - c. You will need to make further changes, both because you will need to manage the ESA life cycle yourself, and because you will need to supply the Agiloft server URL and External System ID values.

Implementation Tips

1. Always use logging instead of printing on the System.out. Note that in Agiloft's configuration JBoss log4j is pre-configured to log into cl-esa.log.
2. Using the ESA provided parameters instead of your own private configuration mechanisms will make it easier to use the ESA in practice.
3. Most of the ESA methods should never return null, unless explicitly stated in the Javadoc.
4. If a field is absent in ExternalRecord, this means 'do not change the field value if possible'. If the field is present but null, this means 'make the field blank'.
5. Most important ESA methods such as metadata retrieval and CRUD operations are always framed by startSync() and endSync() calls. You should normally establish connections in startSync() and close them in endSync().
6. If you do cascade updates, return true from the needSyncAgain() method. Otherwise, Agiloft will never know you did the update and will not pick up the changed data.
7. When mapping relations, check that:
 - a. Both ends of a relation are mapped. For example, if you have a relation between Contacts and Cases, you should first map both Contacts and Cases to the Agiloft tables.
 - b. There exists a Linked Field in Agiloft between tables. You can check this at **Setup > Tables > [select table] > Edit**.
 - c. If a relation is required, the linked field should also be required.
 - d. The multiplicity of a linked field and its relation should also match.
 - e. To map external relations to linked field sets in Agiloft you may specify either 'direct' mapping - equivalent to manually entering values into fields - or map the whole set of fields to some relation in an External System. Note that you cannot combine both direct and whole set mappings - if a field is already directly mapped, that entire set of fields becomes unavailable for relation mapping.
8. If you need to run a clean sync - that is, before the systems are already synchronized - do not just delete records in both systems, as this will result in delete propagations on the next sync. Instead, you can optionally delete the records in both systems, then you should click the Reset Records Peering button at the bottom of the Sync Configuration wizard.
9. The Remote Proxy has a timeout to reconnect after the sync finishes. By default, this is 30 seconds and it may cause the Waiting for ESA to connect screen to hang for a short time. The polling period timeout can be changed in the Remoting section of the Sync Configuration wizard.
10. If your ESA reads a large amount of data, consider implementing the ExternalSystemAdapter . getModifiedPaged() and ExternalSystemAdapter . getDeletedPaged() methods.

Delayed Updates

In some cases, an ESA cannot, or should not, immediately perform CRUD operations. A common case is 'bursting', when multiple operations can be served within a single request to the external system. Doing every operation in a single request might be too expensive - it can be ten or even 100 times slower. Another example is updating an External System in an asynchronous manner.

If the ESA decides to delay an update, it should throw `ESaRecordDelayedException` from the CRUD method, supplying a unique structure-scope string token, within a `startSync()/endSync()` period. This token is later passed into `checkDelayedXXX()` methods to get the actual operation result. If an operation failed and an exception should be thrown, the exception should be thrown from the `checkDelayedXXX()` method. It is possible to throw the `ESaRecordDelayedException` again. The number of update attempts is limited by sync to prevent infinite loops.

Localization

Some methods have a Locale argument. The ESA follows the Java convention in naming locales: `<language-code>_<country-code>_<variant>`. Examples are `en_US`, `ru_RU` and `Pt_BR`. See [Localization](#) for more details.

If the ESA doesn't support the required locale or doesn't support localization at all, it should return an American English label and hint.

Troubleshooting Tips

- As a rule of thumb, whenever you experience unexpected behavior check the `esa.log` and `cl-esa.log`. These reside in the current directory of the Remote Proxy, that is, the directory where you run the `java -jar esa.jar` command. The `esa.log` file will hold the remote proxy logging, including the XML message dumps, whereas `cl-esa.log` will only contain logging from your ESA. Most of the time these logs will contain relevant information about the events. ESA logs are very detailed and contain full XML message dumps, as well as debug information.
- Save your configuration before connecting an ESA to the server. Note that it is auto-saved when you click Next on the General tab. If not, you may see the following exception message:

Error connecting to an unsaved configuration

```
anatoly@anatoly
syncHome> java -jar esa.jar
Agiloft ESA Remote Proxy running.
Connecting to http://anatoly:8080 (External System ID is 123) to get sync
parameters...
IO Error (see esa.log for more details):Can't connect to
```

```
http://anatoly:8080/gui2/sync/connect?extsysid=123&acceptssync=false
(http://anatoly:8080);
nested exception is:
com.supportwizard.sync.interfaces.transport.TransportException: Server error:
500
```

This error commonly appears when running the ESA before leaving the Sync Configuration wizard.

- Check that the Remote Proxy connects to the correct server and port, and has the right External System ID. If not, please use the full syntax for the connection: `java -jar esa.jar -ewhost anatoly:8080 -id ewhost` denotes the server name:port, and ID standards for the externalID in the Sync Configuration wizard, at the bottom of the General tab.
- Check that the command line `cmd/c xxxxx\run.bat` is correct.
- Never use `System.out.println` for debugging. You will never see the message because it will be hidden between `RemoteProxy java -jar esa.jar` and your `ESA run.bat` processes. Instead, use log traces.
- Unit test your code. it is difficult to debug CL ESA applications, because the process is automatically started by the RemoteProxy app. It is much easier to debug unit tests.
- Check that your ESA follows the Javadoc standards from the `ExternalSystemAdapter` interface. In particular, check the return values for `create()` and `update()` - these should never return null.

If you still have problems, please contact [Agiloft support](#) and submit a bug report. Please provide the exact steps to reproduce the problem and all the necessary credentials and code.

Additional Areas for Discussion

A few message or data types need special discussion.

Deletions

For many systems, tracking deleted records is problematic. The Agiloft HelperAPI can simplify this task. If you can't easily track deletions in the external system you can either:

- Enumerate all record IDs known by Agiloft and check whether those records actually exist at the moment, or
- Report deletion events to Agiloft when deletion happens in the external system. The latter is usable if you can somehow obtain deletion notifications from the external system.

A more sophisticated approach to ID enumeration with better performance is possible if your system uses never repeating, always increasing auto-incremented IDs. If this is the case, you may process the `countRange` message and send a `detectDeleted` message to `HelperApi`. It will then apply a dichotomy method - binary division - to find deleted IDs. This will be considerably faster than comparing known and currently existing IDs one by one, especially if you can have a fast implement `count A < id < B` operation and the number of deleted records is not large.

Transactions

Sync is transaction-less, because not all external systems support transactions or expose them over integration APIs. If an external system requires transaction usage, the ESA should behave as it would with auto-commit on in SQL, meaning it should execute every operation in a separate transaction.

Collections

Collection fields may hold complex objects, which can be mapped to Agiloft tables. For this reason, `getFields`, `getRelations` and `getCollections` calls should accept both structure names and collection IDs.

Note: An ESA must ensure that structure names and any collection IDs use different name spaces.

Locking - Clashing Modifications

To deal with concurrent data modifications, Sync employs an optimistic locking strategy. When the ESA is asked to modify or delete an external record, it is given a last seen timestamp. If the ESA detects that the record was modified after that time, the ESA should respond with an `OptimisticLockingFailure` exception and not modify or delete the record. If an external system uses the pessimistic locking approach internally, the ESA should wait until the lock is granted first, or else report a record update failure if waiting for the lock is not reasonable.

Currently, delayed updates due to locks are not supported, but this feature is likely to be added in the upcoming releases because it may speed up syncn appreciably. It is a good idea to make your ESA ready to throw some exception when a pessimistic lock is detected - this might be employed in the next Sync release.

When a record update/create/delete error occurs, the ESA should respond with `EsaRecordException`. Such exceptions indicate a single record related error, probably recoverable in the future and do not stop the syncn cycle. On the next cycle, sync will try to re-synchronize the erroneous records.

Cursors

In some setups, `getModified`/`getDeleted` may return a huge amount of data. It may be inefficient or even unfeasible to transfer and process such traffic in one call.

To overcome this, Data Cursors can be employed. A Data Cursor is a paginated stream of data. When requested, cursor "core" information such as ID and page number is transferred from the ESA to Agiloft. Agiloft requests actual pages data from the cursor passing `readDataPage` messages supplying cursor ID and page index.

Cursors are a more advanced technique than the simple return of all data. Their support is optional in the ESA. For the `getModified` function, Agiloft first sends `getModified`, which has a chance to return all data at once. If and only if the `getModified` response is a `USE_PAGES` string and not an empty result, Agiloft sends `getModifiedPaged`, which returns a cursor. The ESA may choose the transferring approach in runtime, depending on the amount of the data.

When a data cursor is fully read, the cursor is closed. However, due to possible communication failures, this may not happen. The ESA should retire cursors in 15 minutes since the last `readDataPage` or `leaseCursor` message.

ESA Parameters

The ESA is likely to have configuration parameters, such as external system host, login, password, etc. While the ESA may maintain them by its own configuration mechanisms, it is often desirable to have them in one place, to simplify administration and maintenance.

For this purpose, Sync may store some ESA parameters within the Agiloft database, providing a simple GUI for changing them. To do this, Sync asks ESA for the names and types of parameters to manage. The ESA provides parameter meta-data such as type, default value, required / not required, and Sync shows them in the configuration GUI, ESA Settings page.

ESA may later request its parameter values during the syncn cycle, using parameter names. See `getParameter` message. Though the number of parameter types is limited they are likely to cover most of the ESA configuration needs and simplify administration greatly. If, for example, your ESA will require some configuration file, it is a good idea to put the local path to it in an ESA parameter.

Parameter names starting with `config` are reserved for getting Sync Configuration settings. Currently, the following are available:

- `config.pollperiod` - polling period
- `config.commandline` - command line to invoke a third-party command line in the ESA.
- `config.esatype` - ESA type name.

It is likely that only `config.pollperiod` is of interest for ESA developers, and only for HTTPs ESAs.

Timestamps

The time resolution for sync is one second, and all timestamps are truncated to the closest second. All timestamps passed to or received from the ESA are in UTC - GMT, Greenwich time. It is the responsibility of the ESA to translate external system timestamps into UTC, taking time zones and daylight savings into effect.

Sync may compute the system clock difference between the external system (ESA) machine and the Agiloft host machine. To do so, sync passes a `getCurrentTime` message to the ESA. If the ESA wants the clock difference to be taken into account, it should respond with the current external system UTC time. Alternatively, the ESA may respond with an empty result, setting clock difference to zero.

If you are sure that the Agiloft host and external system host have their system clocks synchronized, for example via Network Time Protocol, NTP, it is a good idea to turn clock difference computation off by responding with an empty message. Since the Sync time resolution is less precise than NTP, it will never get better results than it could if the system clocks are really synchronized.

Due to limited time difference precision, it is possible that some records may stay unsynchronized. Though the chance of this is very low, it is always better to have machine clocks synchronized and time difference detection off.

If you are unsure that clocks are synchronized, for example, your ESA can be deployed on a number of machines, you should run `getCurrentTime`. The time difference calculation works best if the ESA actually measures time in the middle of a message delivery + message processing + message delivery back loop, but in practice, it is fine to just measure and respond as fast as you can.

The Time difference calculation is performed on each Sync cycle, so if system clocks are corrected between synchronizations, Sync will work correctly. It is however, presumed that system clocks are not changed during synchronization.

Last seen timestamps are always passed just as they were received from the ESA last time and thus are not affected by system clock changes.

ESA HelperAPI Interface

The sync subsystem core provides the ESA with a callback to the HelperApi interface, which might be used to call the sync core. This section describes the methods in this API.

startSync

Signature	- String startSync (String externalSystemID, boolean waitForCompletion) - String startSync (String externalSystemID)
Description	Starts synchronization, whose settings are in the synchronization configuration, identified by External System ID.
Purpose	Provides a way to externally trigger a synchronization. If you use this method in HTTPs ESA, be sure that HelperApi uses HttpXmlEsaTransport.
Parameters	- externalSystemID - External System ID, used to find sync configuration - waitForCompletion (optional) - indicates whether control should be returned immediately, or when synchronization finishes.
Returns	One of SYNC_RUN_ constants. If ended with a SYNC_RUN_ERROR, an error message may follow, separated by space.
Exceptions	None
Example	<pre>//starts external-triggered synchronization HttpXmlEsaTransport hxeTransport = new HttpXmlEsaTransport(new URL (serverHost)); hxeTransport.connect(externalSystemID, false); HelperApiTransmitter helperApi = new HelperApiTransmitter (hxeTransport); helperApi.startSync(externalSystemID, false); // Asynchronous hxeTransport.close();</pre>

getParameter

Signature	List<EsaParameter> getParameter (String externalSystemID, String name)
Description	Gets an ESA parameter value, stored on the Agiloft server. The value is a list, which may contain any number of real values, depending on the parameter type.
Purpose	ESA uses this method to obtain an ESA parameter value. The whole chain of events looks like this: 1. Sync core queries the ESA for the list of parameters it wants the sync core to manage, using the <code>ExternalSystemAdapter.getParameterMeta()</code> call. 2. Agiloft generates a Sync Configuration / ESA parameters screen. 3. The Agiloft admin enters the parameter values in the screen. 4. Agiloft saves the values into its database. 5. At a later time, typically from <code>startSync()</code>, the ESA calls <code>HelperApi.getParameter</code> to get the stored values.
Parameters	▪ <code>externalSystemID</code> - External System ID, used to find sync configuration ▪ <code>name</code> - Name of ESA parameter, as returned from <code>getParameterMeta()</code>
Returns	Stored parameter value
Exceptions	None

getPollingPeriod

Signature	Integer getPollingPeriod (String externalSystemID)
Description	Provides the Polling Period parameter as set in the General tab of the Sync Configuration wizard.
Purpose	This method is only of interest for HTTPs ESAs. When an HTTPs ESA finishes a synchronization, or receives a timeout when getting an XML message, it should reconnect to the Sync subsystem core to make itself available for the next synchronization. The polling period defines the time period to wait before reconnect, as set by the Agiloft admin.
Parameters	<code>externalSystemID</code> - External System ID, used to find the sync configuration.

Returns	Time to wait before reconnect, in seconds, or 0 if the reconnect should occur immediately.
Exceptions	None

Other methods of HelperApi are devoted to tracking deletions. For many systems, this is the most difficult part of creating an ESA because records are simply deleted without a record that they existed. This makes deletion tracking somewhat difficult to implement. In some cases though, the following Helper API can help:

trackRecordDeletion

Signature	void trackRecordDeletion (String externalSystemID, String externalStructure, Date time, String id)
Description	Records a record deletion into an internal log, associated with this externalSystemID.
Purpose	If an ESA, such as HTTPs ESA, receives deletion notifications from the external system, it may forward them to the sync core immediately without the need to maintain an internal list of deleted records. This method does not imply any limitations on the nature of the ID, but requires that ESA receives deletion notifications. Usually this means that the ESA, or a part of it, must always be running.
Parameters	- externalSystemID - External System ID, used to find sync configuration - externalStructure - Structure to which the record belonged - time - Time when a record was deleted - id - ID of the deleted record
Returns	Nothing
Exceptions	None

When it is not possible to subscribe to deletion notifications, it might be possible to find out which records were deleted by comparing a list of all external records ever reported to the Agiloft sync core, that is never synchronized, and the list of records currently existing in the external system.

detectDeleted

Signature	Set<String> detectDeleted (ExternalSystemAdapter esa, String externalSystemID, String externalStructure, Date after)
Description	Detects which records were deleted, using countRange() and binary division approach
Purpose	If the ID is monotonic and never reused, for example an auto-increment DB key, it is possible to find deleted records by finding gaps in the ID sequence.

	To do so, the sync core counts the number of external records that it knows and compares them with the <code>ExternalSystemAdapter.countRange()</code> call result. If the numbers differ, sync core divides the ID range and count records within each range separately via binary division. The process is repeated until the deleted records, e.g., those with missing IDs, are found.
Parameters	▪ <code>esa</code> - ESA instance, used to call <code>countRange()</code> ▪ <code>externalSystemID</code> - External System ID, used to find sync configuration ▪ <code>externalStructure</code> - structure which the record belongs to ▪ <code>after</code> - date as passed to the <code>getDeleted()</code> method
Returns	Set of deleted record IDs
Exceptions	None

isKnownID

Signature	Boolean isKnownID (String externalSystemID, String externalStructure, String id)
Description	Checks whether a record with this ID was ever reported to the sync core, in other words, if it was ever synchronized.
Purpose	If the ID nature does not allow using the <code>detectDeleted()</code> method but the ESA can apply another fast algorithm for comparing known and actual lists, the ESA may check whether the sync core is aware of the record using this method.
Parameters	▪ <code>externalSystemID</code> - External System ID, used to find the sync configuration ▪ <code>externalStructure</code> - Structure which the record belongs to ▪ <code>id</code> - ID of the deleted record
Returns	True if the record was ever synchronized
Exceptions	None

enumerateKnownIDs

Signature	Boolean isKnownID (String externalSystemID, String externalStructure, String id)
Description	Returns a list of records known to sync core, in other words, the records that were ever synchronized.

Purpose	This is the last hope method – ESA may simply take the list of records which were ever synchronized by sync core and compare them with the actual list of records in the External System. While this is the simplest detection method, it can be very slow if there are a lot of records.
Parameters	- externalSystemID - External System ID, used to find sync configuration - externalStructure - structure which the record belongs to - knownBefore - date as passed to the <code>getDeleted()</code> method
Returns	A set of all synchronized record IDs
Exceptions	None

Using the Sample ESA

A working sample External System Adapter (ESA) is directly downloadable from within your Agiloft KB. It also functions as an example for users who would like to develop their own ESA. Users may modify and reuse the sample ESA code for their own use as desired.

While built for illustration purposes, the sample ESA is a fully functional ESA that syncs with a virtual external system which is either co-located or running on an external server.



Note that the Sample Remote Proxy code simulates running on a remote server that contains the remote external system adapter and database. This is why this documentation refers to the external system and its data tables as 'virtual' – there is no actual remote external system database and no actual remote external system data tables. Those 'tables' and 'data' are pre-loaded in the Sample ESA bundle to make this process easier to set up and understand.

To demonstrate how the ESA works, the virtual external system includes two tables: 'Virtual Contacts', with the fields ID, Full Name, and Email; and 'Virtual Cases', with the fields ID and Summary. The AgiloftEmployees table will be mapped to Virtual Contacts and the Agiloft Support Case table will be mapped to Virtual Cases. Actual data is stored in a file directory, using a Java properties format - plain for Employees/Contacts, XML for Cases. Employees/Contacts files use the naming convention `contact_123`, where 123 is the record's value in the ID field. Similarly, Case files use the naming convention `case_123`.

The Sample ESA uses a single parameter, Work Directory (`workDir`), to hold the directory path for storing data files. The value of the Work Directory parameter is input on the ESA Settings screen which appears once a connection gets established, as per Step 11 below. Note that this directory will be a subdirectory of the initial sync Home directory defined in Step 1 of the ESA builds.

Note that the synchronization process is nearly identical for both Windows and Unix systems, with only the directory paths and therefore command line text being unique to each system. However, in the interest of clarity this manual will show the Windows installation process.

Building a Sample ESA

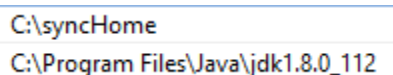
Follow the guided steps to build a sample ESA.

Create a work directory:

1. Create a work directory named 'syncHome' on your local directory, for instance `C:\syncHome`. This will be the directory where data and files are stored.
2. Note: the actual name of the directory is not important, but the rest of this guide will refer to the work directory as `syncHome`.

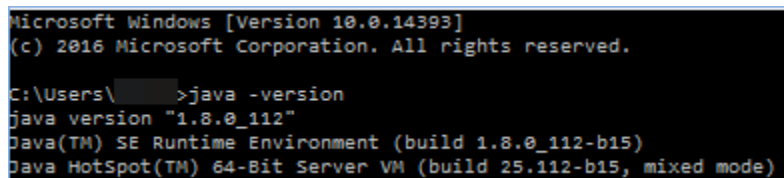
Install the Java Development Kit (JDK):

1. Download and install the most current JDK: <http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>.
2. Note the name of the JDK installer; for instance, jdk-8u112-windows-x64.exe. This will be used to update the PATH variable.
3. Update the PATH variable to include syncHome and the JDK. See: <https://www.java.com/en/download/help/path.xml>.



C:\syncHome
C:\Program Files\Java\jdk1.8.0_112

4. Verify that the JDK was correctly installed:
 - a. Open the Command Prompt window by clicking the Start menu, typing 'cmd' and pressing Enter.
 - b. Type: java -version.



```
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

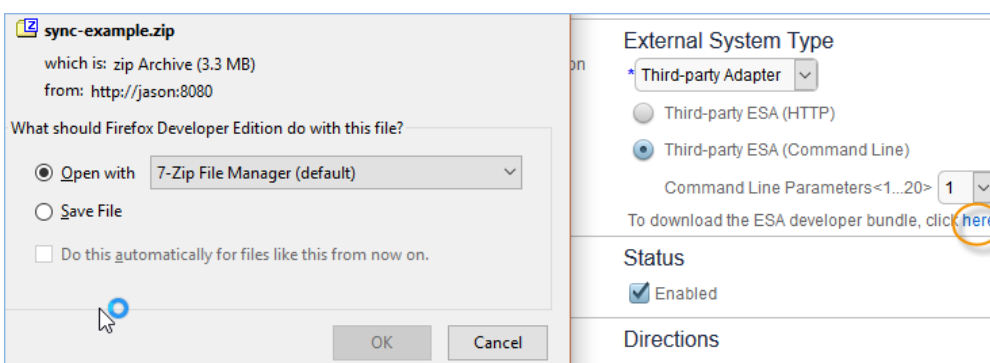
C:\Users\>java -version
java version "1.8.0_112"
Java(TM) SE Runtime Environment (build 1.8.0_112-b15)
Java HotSpot(TM) 64-Bit Server VM (build 25.112-b15, mixed mode)
```

Configure the sync subsystem:

1. In your KB, go to **Setup > Sync > New > External Sync** to open the Sync Configuration wizard.
2. Add the following details:
 - a. Configuration Name - add a descriptive name such as MySync
 - b. external system Type - select Third-party Adapter from the drop-down
 - c. Select the Third-party ESA (Command Line) radio button
 - d. Select 1 for Command Line Parameters
 - e. Status - Enabled
 - f. Directions - Two-way sync
 - g. Conflicts - external system should take precedence
 - h. Remoting - select the external system Adapter runs remotely radio button. This will enable the Download ESA Remote Proxy button.
 - i. The external system ID Prefix is system-generated and will be used by sync to connect to the ESA Remote Proxy.
3. At this point, leave the window open while you complete the next sections of steps to set up the ESA development package. Only when the package is ready should you click Next and proceed.

Download the Sample ESA development package and remote proxy:

1. In the external system Type section of the Sync Configuration wizard, there is a link named "To download the ESA developer bundle, click here." Click the link to download the sync-example.zip file.



2. Extract the zip file contents to your syncHome directory.
3. In the Remoting section of the Sync Configuration wizard, click the Download ESA Remote Proxy button. This downloads the esa.jar file. The jar file is configured specifically for your system and contains your Agiloft server name and external system ID.
4. Save the file to the syncHome directory.

The syncHome directory should now contain the following files and folders, which can be verified using the Command Window. Type 'dir' to view the contents of the directory, or 'dir/p' to show one page on the screen at a time:

File/Directory Name	Description
com (directory)	Contains example sources and a Java support library. The sample ESA code is in the com.supportwizard.sync.sampleesa.SampleEsa class, located in the SampleEsa.java file in the com/supportwizard/sync/sampleesa directory.
commons-logging-1.1.1.jar, FortifyAnnotations.jar, log4j.jar, commons-digester-1.8.jar	Libraries needed for the example.
make.bat, make.sh	Build scripts for Windows and Linux, respectively.
run.bat, run.sh	Run scripts for Windows and Linux, respectively. Run.bat is invoked from Start_ESA.bat in Windows. Run.sh is invoked from Start_ESA.sh in Linux.
META-INF (directory)	META-INF in the example.
log4j.xml	Logging configuration in syncHome/out.
in.txt, out.txt	Input and output XML examples.

esa.log	Handles the remote proxy logging.
cl-esa.log	Handles Agiloft sync subsystem logging.
sync.external systemd	XML messages scheme in syncHome/out.
ESA_Developer_Guide.docx	This document.
Start_ESA.bat	This file needs to be in either the C:\Agiloft folder or its location in the user's PATH environment variable. It is invoked by the sync subsystem to make the connection with the ESA Remote Proxy in Windows.
Start_ESA.sh	This file needs to be in either the /usr/local/Agiloft folder or its location in the user's PATH environment variable. It is invoked by the sync subsystem to make the connection with the ESA Remote Proxy in Linux.

Run the make.bat script:

1. In the Command Prompt window, in the syncHome directory, type `make.bat` and click Enter. This is a script that executes several commands that are necessary for the sample ESA to function correctly.
2. Once the script has run, keep the command prompt window open and return to your browser.

Verify the Start_ESA.bat script:

1. The sync subsystem will call the `Start_ESA.bat` script to execute `run.bat` to launch the Agiloft ESA. First, make sure that `Start_ESA.bat` matches your configuration.
2. Note that the `Start_ESA.bat` file should be in the syncHome directory, and the location added to the user's PATH environment variable.
3. `Start_ESA.bat` expects that in Step 3 above you specified 1 for the Command Line Parameters number in the external system Type section. If you chose a number other than 1, you must edit `Start_ESA.bat` and change the PARAM value to match. If your ESA home directory is not C:\syncHome you must also modify the path of the run.bat file:

```
set
/A PARAM = 1 - or = number of CL Parameters specified during configuration
IF
%1==%PARAM% cmd /c C:\syncHome\run.bat - or C:\your Sync
directory\run.bat
```

Varying the number of command line parameters will allow you to run multiple remote ESAs.

Check the ESA configuration and launch the Agiloft sync subsystem:

1. At this point you should have walked through all previous steps and added all of the settings, with the Sync Configuration wizard open in the browser. In the Sync Configuration wizard, click Next. This launches the Agiloft sync subsystem and begin polling, where it will be waiting for a connection from the external system Remote Proxy ESA.
2. While the system is trying to connect, in the Command Window enter `java -jar esa.jar` and press Enter. This command tells the ESA Remote Proxy to connect to the Agiloft sync subsystem, obtain the command to launch the sample ESA, and run it. You should see a success message similar to the following:

```
C:\syncHome>java -jar esa.jar
EnterpriseWizard ESA Remote Proxy running.
Connecting to http://localhost:8080 (External System ID is 1477996558211@Demo) to
get sync parameters...
Success. Running ESA THIRD_PARTY_COMMAND_LINE
Use Ctrl-C to stop this app.
```

3. When you return to the browser window, the ESA Settings screen will have appeared, where you can set the directory to store the synchronization data files.

Set the data storage directory:

1. In the syncHome directory, create a sub-directory named `sampledata`. This will be where data files are stored following synchronization.
2. In the Work Directory field of the ESA Settings tab, enter `sampledata`.
3. Click Next to open the Mapping screen.

Configure sync table mappings:

1. The Mapping screen contains a list of the tables in the KB, which can be mapped to structures in the external system. Here you can establish the relationships between Agiloft KB tables on the left, and external system ESA tables in the drop-down list on the right. Note that the external system tables are 'virtual' in this example, since there is no real external system or tables.
2. Begin by selecting Virtual Contacts from the Employee drop-down to map the Employees table to the Virtual Contacts external system table, then click Map to open the Field Mapping screen.
3. In the Field Mapping screen, you can define the mappings between fields in your Agiloft table, and in the selected external system table. In this example, create the following mappings between Agiloft and the external system respectively:

- Email - Email. Select "Identifying" for this row.
 - Full Name - Full Name. Select "Identifying" for this row.
 - Employee ID - ID.
4. By default, the Use strict match for identification checkbox is selected.
 5. Click Finish.
 6. Next, select the Support Case table and map it to Virtual Cases, then click Map.
 7. In the Field Mapping screen, create the following mappings:
 - Case ID - ID, with Update in External selected
 - Summary - Summary, with Update in Agiloft, Update in External, and Identifying all selected
 8. Click Finish, then click Next to navigate to the Relation Mapping screen.

Configure the Relation Mappings and Running option:

1. The Relation Mapping screen enables you to map external relations to linked field sets in Agiloft. In this case, the Customer relation will be mapped to the Employee(Cell Phone, User Company, Customer Phone...) linked field set. This will reflect the relationship between Support Case-Virtual Cases and Employee-Virtual Contacts.
2. Click Next to navigate to the Running screen.
3. Select Manual to specify the synchronization process initiation mode.
4. Click Finish.
5. The sample Remote Proxy ESA sync subsystem is configured.

Running Synchronization

This section describes how to perform synchronization between your sample ESA and your Agiloft KB.

1. Open the Support Cases table in the KB.
2. Go to **Actions > Sync > Run MySync**.
3. A dialog screen shows the progress of the synchronization. When the sync has completed, it displays a success message with some links to view the log file and the records which were updated.



If you need to re-sync using a clean synchronization, don't just delete files and re-run synchronization. Doing so may result in propagation of deleted files, if this is permitted in the sync configuration. To reset a full match history, edit your sync record and click the Resets Records Peering button at the bottom of the Sync Configuration wizard.

4. Clicking the affected records link opens the Synchronization Results screen where you can view all of the records which were created, updated or deleted between Agiloft and the external system.

Sample ESA Code Description

The full Sample ESA source code is included in the `SampleEsa.java` file in the `syncHome\supportwizard\sync\sampleesa` directory. See: `com.supportwizard.sync.sampleesa.SampleEsa`. This section will walk through the code in this document, and is Java-implementation specific. However, it is recommended that you understand how the Sample ESA works even if you plan to use another language, since the ESA logic will be the same.

The Javadoc comments for the classes and methods of the ESA are in the `ExternalSystemAdapter.java` document at `syncHome\com\supportwizard\sync\interfaces\esa`.

SampleEsa

```
package com.supportwizard.sync.sampleesa;

import com.supportwizard.sync.interfaces.esa.*;

...

public class SampleEsa extends ExternalSystemAdapterBase {

...
}
```

If you plan to use the Java support libraries, your ESA class should implement the `ExternalSystemAdapter` interface. This single interface contains all of the ESA operations. Some of them are only called under certain circumstances. To have the full implementation of the auxiliary methods, extend the `ExternalSystemAdapter` base class.

Logging

```
public class SampleEsa extends ExternalSystemAdapterBase {

    /**
     * Sample ESA uses log4j logger. You may use any other logger, such as java.
     * util.logging.Logger,
     * but DO NOT USE System.out for logging.
     *
     * Command-line ESA uses System.in and System.out to exchange XML messages with
     * sync core.
     * All non-XML output is omitted, so you will never see it.
     */
    private Logger log = Logger.getLogger(SampleEsa.class);
}
```

The Sample ESA works as a command line (CL) ESA. This means that its standard input and output can be used for exchanging XML messages.

Do not print anything on the standard output. Use the logging facilities instead. By default, all the Sample ESA logging goes into the `cl-esa.log` file, and the Remote Proxy logging goes into the `esa.log` file, both in the home of the `syncHome` directory.

ESA Parameters

The Sample ESA uses a single ESA parameter: a working directory location. In this case the name is `workDir`.

Work directory

```
// File location parameter
private static final String WORK_DIR = "workDir";
```

The following method returns the parameter metadata described for the sync core:

Metadata

```
public
List<EsaParameterMeta>getParametersMeta(Locale
locale) throws EsaException, RemoteException
```

This code inside the `startSync()` method uses the parameter:

startSync()

```
// Read work dir parameter value
List<EsaParameter> paramValues =
syncCoreApi.getParameter(externalSystemID,
WORK_DIR);
assert paramValues.size() == 1; // There must be a
single string (parameter type is TEXT, SINGLE).
String workDirPath = paramValues.get(0).getStrValue();
assert workDirPath != null;
```

The `List<ESAParameter> paramValues... WORK_DIR);` line queries the sync core callback interface for the stored parameter value, passing the parameter name `workDir` as a key.

Sync State

The `startSync()` method sets the ESA into a 'synchronization is running' state.

Starting synchronization

```
public String startSync(String externalSystemID) throws EsaException,
RemoteException {
```

Most of the ESA methods are only called in this state. The ESA may connect to the external system in `startSync()`, for example. The `getParametersMeta` method is called before entering sync state. This is needed to ensure that the ESA parameters are properly configured before entering the sync state.

The sync state is ended with a call to the `endSync()` method:

```
endState()
```

```
public void endSync() throws EsaException, RemoteException {
```

The Sample ESA constructor does a one-time initialization which only sets up the logging. A single instance can be used to run multiple synchronization cycles, so most of the initialization should probably be done by the `startSync()` method.

The ESA parameter values should not be queried in the constructor. The sync core callback interface has not yet been set. Also, it is possible that the ESA parameter is not yet managed because `getParametersMeta` was not called. Therefore, the parameter value query should only be done in the sync state.

Metadata

The ESA returns information about external structures/tables, fields in each table, and relations between them, using the following code:

```
External system structures
```

```
public Set<ExternalStructure>getStructureList(Locale locale)
```

```
...
```

```
public Set<ExternalField>getFieldList(String  
structureOrCollection, Locale locale)
```

```
...
```

```
public Set<ExternalRelation>getRelations(String  
structureOrCollection, Locale locale)
```

Note: the term *structure* is used to highlight the fact that, physically, this could be almost anything: a database table, a file, an object-oriented database, and so on.

`ExternalStructure.name`, returned from `getStructureList`, is a logical structure name. It is later passed to `getFieldList`, `getRelations`, and other calls. Similarly, `ExternalField.name` and `ExternalRelation.id` are local fields and relation names which are used in the `ExternalRecord` structure and passed to CRUD methods.

Data Reads

The ESA is queried on external records using these methods:

Querying the ESA

```
public Set<ExternalRecord>getModified(final String structure, final Date after)
...

public ExternalRecord read(String structure, String pk)

...

public Set<String>getDeleted(String structure, Date since)
```

Please read the methods' Javadoc comments for implementation details.

The `getDeleted` method might be hard to implement: `public Set<String>getDeleted(String structure, Date since)`. See the Javadoc comments for possible strategies.

Note: `structure` is a structure logical name, as returned from the `getStructureList()` method. In addition, if the `after` timestamp in `getModified` is null, this means 'return all records'. Usually it has a null value on the initial synchronization, or after a Reset Records Peering is executed.

Data Updates

External records are manipulated through these methods:

External record manipulation

```
public ExternalRecord create(String structure,
ExternalRecord values)
...

public Date update(String structure, Date lastSeen,
ExternalRecord values)

...

public void delete(String structure, Date lastSeen,
String pk)
```

Please read the methods' Javadoc comments for implementation details.

Note: the `lastSeen` parameter is used to implement optimistic record locking. If this fails, throw `OptimisticLockFailureException`.

Exceptions

Exceptions are reported using a `<result>` tag with a nested `<exception>` element.

```
<exception type="general | record | configuration|alreadyconfigured
|optlockfailed | concurrentdelete">
<message>Exception message to be shown to user</message>
<trace>Exception message to be put to logs</trace>
</exception>
```

Exceptions of type `optlockfailed` and `concurrentdelete` should also have a nested tag:

```
<configured-to>External ID, to which ESA is configured to</configured-to>
```

Exceptions of type `optlockfailed` and `concurrentdelete` should also have nested tags:

```
<external-id>record ID</external-id>
<modified-at>record ID</modified-at>
```

These should contain exception stack trace or some other diagnostic information (FILE/LINE, etc) which would allow ESA developer to investigate the problem better.

- All 'legal' ESA exceptions are derived from the `EsaException` class.
- The generic `EsaException` is fatal and aborts the synchronization.
- `EsaRecordException`, in contrast, does not abort the whole synchronization and just marks a single record as failed.
- `OptimisticLockFailureException` is used to implement optimistic locking for the time between when a record is read, using `getModified` or `read`, and the time it is updated. If it fails, the synchronization cycle will repeat.
- All unexpected exceptions, other than `EsaException`, abort the synchronization.

Locale

Some messages have a Locale argument, which is a locale and country code, as described in the ISO-639 and ISO-3166 - <language-code>- <country-code>- <variant>. Examples are "en_US", "ru_RU" and "pt_BR". If the ESA doesn't support the required locale or doesn't support localization at all, it should return an American English label and hint.

HTTPS and Command Line ESA Types

Two communication styles are currently supported between third-party External System Adapters (ESA) and sync core subsystems for exchanging XML messages:

- HTTPS
- Command-line (CL) standard input/output (STDIO)

In both cases, the same XML messages are passed back and forth, and the ESA operation semantics are identical. However, the ESA implementations themselves could be quite different.

Note: The [ESA Remote Proxy](#) bundled with the Agiloft application converts the command-line ESA into an HTTPS ESA, thus allowing the command-line ESA to run remotely over HTTPS.

Both the CL and HTTPS approaches are quite versatile and both allow for implementations of the ESA in any language using any technique. There are two implementation styles because external systems can be very different and, depending on particulars, one ESA might be easier to implement than the other. For example, if your system is a Windows application and you need to call its Common Object Model (COM) interfaces, a CL ESA would be easier to implement. Since the ESA application is automatically launched by Agiloft when needed, you don't need to write a continually-running Agiloft polling service. The application is quite straightforward in that case: receive the call from the Agiloft sync core, and then do the necessary calls on the COM interfaces. By contrast, if your system is of a client-server type, such as a J2EE server, it might be more natural to implement an HTTPS ESA. This could run on the server side, where you can directly access your back end.

XML message syntax and semantics are the same for both CL and HTTPS ESA types.

This table illustrates some differences between HTTPS and CL ESA:

HTTPS ESA	CL ESA
Life cycle is not under the control of the Agiloft	Automatically launched when needed
Actively polls for messages	Just reads messages from its standard input
Usually an internal part of your system	A separate executable which usually connects to your system
Often remote	Needs a special Remote Proxy application that is shipped with Agiloft in order to be remote

If you plan on implementing an ESA using Java, you can make use of the support library bundled with the ESA example which handles all communication details like parsing, generating XML, exchanging XML messages with the Agiloft sync subsystem and so on. All you need to do is provide a single Java interface `ExternalSystemAdapter` and modify the main `JavaExecutableEsa` class provided in the example. This will result in a ready-to-go CL ESA application. If you are implementing an HTTPS ESA, see [Java ESA Development](#) for more information on the methods that will be called.

HTTPS ESA

An HTTPS ESA is a separate application which runs independently of Agiloft's control; either remotely or co-located on the Agiloft server. It continually polls the Agiloft sync subsystem core for new messages and will stay in a pending state until a synchronization is to be run. The HTTPS ESA exchanges XML messages in a REST-like manner as shown in this table:

URL	Action
GET https://<ewserver:port>/gui2/sync/connect?extsysid=<external system id>&acceptsync=true	Connects to Agiloft sync subsystem, ready to perform synchronizations defined by the sync configuration with <external system id> external system ID. Response contains the connection ID
GET https://<ewserver:port>/gui2/sync/message?connid=<connection id>	Gets next message to the ESA Response contains the XML message to be processed by the ESA
POST https://<ewserver:port>/gui2/sync/message Parameters: ▪ connid=<connection id> ▪ message=XML message	Posts messages to the Agiloft sync subsystem This is used to post replies back to the sync core or to call sync core functions
GET https://<ewserver:port>/gui2/sync/disconnect?connid=<connection id>	Disconnects from sync subsystem

The communication cycle is basically a sequence composed of

- connect
- get-post
- get-post...
- get-post
- disconnect

HTTPS ESAs are usually remote, though this is not a requirement, and their lifecycles are fully independent of the Agiloft server. They are just considered to be running externally and to be polling the Agiloft sync subsystem for new messages at prescribed intervals. If a synchronization is started, it will stay pending until the HTTPS ESA connects and receives sync messages.

If you write server-side software, an HTTPS ESA would generally be the best option. It uses a blocking poll approach, which means that GET requests are blocked on the Agiloft server until a message is ready for the ESA or a timeout occurs.

HTTPS Parameters

An HTTPS ESA needs two parameters before connecting to the Agiloft sync subsystem:

- The Agiloftserver name to connect to
- The external system ID identifying a particular data set - the Agiloft knowledgebase and its particular sync configuration - to be synchronized with

Because these parameters are required before connecting to the Agiloft sync subsystem, the ESA parameters mechanism is not applicable and the parameters should be provided beforehand for the ESA separately. Usually, they are identifiable within the settings of your system.

HTTPS URLs and Transport Types

This type of message transport does not require the ESA to be an application executable by command, and is adapted for Internet communication.

URLs

There are 3 URLs used for HTTP transport. If the Agiloft GUI runs at <https://www.example.com/gui2>, these URLs will be: <https://www.example.com/gui2/sync/connect>, <https://www.example.com/gui2/sync/disconnect>, <https://www.example.com/gui2/sync/message>.

Every request to any URL must contain an `extsysid=<externalSystemID>` parameter, where `<externalSystemID>` matches the external system ID field in the Sync Configuration.

Important: A third-party ESA, using HTTP transport must be aware of its `externalSystemID`.

HTTP response codes should always be examined to check whether transportation was successful. All parameters – GET and POST – should always be URL-encoded. Character encoding is UTF-8.

Connect and disconnect URLs should be accessed when the ESA is about to start/stop receiving messages. These requests don't have any other parameters. The HTTP response to connect will be an `externalSystemID` value passed in the HTTP request. The ESA may check this in addition to the HTTP response code to ensure it has connected successfully.

The message URL can be accessed via a GET HTTP request to get the next message or via a POST HTTPs request to send a message. When GETting message, the XML message is returned as a HTTPs response. When POSTing, the message should be passed as the message HTTPs form parameter.

Cycle

The typical ESA cycle for using HTTPs transport is:

- GET /sync/connect
- GET /sync/message
- Process the message.
- POST /sync/message
- If the message was **not** Release, goto 2
- GET /sync/disconnect
- Wait for some period of time. The Sync Configuration setting Polling Period is accessible as the config . pollperiod parameter which you may request from within sync cycle 1-6. Goto 1

HTTPs transport does not limit the way the ESA is actually implemented. It can be a command-line application, a service, a component within an application server, etc.

Starting and stopping this application and the exact mechanism by which the ESA obtains the external system ID value are outside the scope of this spec, but typically the application is installed just like any other piece of software and reads the ID from a command line parameter or configuration file.

Command Line STDIO

A Command Line ESA (CL ESA) is a command line application, which reads incoming messages from standard input and prints response messages on the standard output. The Command Line ESA application is automatically launched by Agiloft when necessary.

This type of third-party ESA uses standard input and output streams, stdio. In this case, the ESA developer provides his ESA as an executable, command-line application. It need not be a compiled native application; it could even be a script. The two requirements are:

- It must be runnable always using the same command-line
- It will not print anything except XML Sync messages to the standard output

A CL ESA is often easier to write because it doesn't require an always-running server. A CL ESA uses a standard IO (STDIO) interface to exchange XML messages with the Agiloft sync subsystem. XML messages from sync are written to the ESA application Standard Input (STDIN) and responses are read from the application's Standard Output (STDOUT).

The following example demonstrates how a manual CL ESA could look, including the console text and the sample ESA application response:

Command Line ESA Message Exchange

```
//CL console
anatoly@anatoly my-esa> ./Start_ESA
<sync>
```

```

<esa-call call-id="1">
<getCurrentTime/>
</esa-call>
</sync>

//ESA application response
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<sync
xsi:noNamespaceSchemaLocation="sync.xsd" xmlns:xsi="http://www.w3.org/2001
/XMLSchema-instance">
<result response-to="1">
<value>2012-09-14T15:36:40.774Z</value>
</result>
</sync>

//CL console
<sync>
<esa-call call-id="2">
<release/>
</esa-call>
</sync>

//ESA application response
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<sync xsi:noNamespaceSchemaLocation="sync.xsd" xmlns:xsi="http://www.w3.org/2001
/XMLSchema-instance">
<result response-to="2">
<value></value>
</result>
</sync>

//CL console
anatoly@anatoly my-esa>

```

For more information, see [ESA XML message schema](#).

There is a parser and a serializer provided with the Java support library, so you should only need the raw scheme if you are developing an ESA without them.

Line breaks

Although the XML standard says little about line breaks, standard console I/O is line-based. To handle this difference, every `<sync>` and `</sync>` tag which delimit all XML syncn messages must be placed on a line of its own.

Cycle

The typical ESA cycle using STDIO transport is

- Read message, using standard input:
 - Read line. This must always be the XML declaration - `<?xml . . . ?>`
 - Read line. This must always be `<sync>`
 - Read and accumulate all lines before getting `</sync>` line.
 - Parse what was read as XML message
- Process message.
- Write response using standard output
 - Write XML declaration
 - Write `<sync>` line.
 - Write XML message body - a `<call>` or `<result>` element.
 - Write `</sync>` line.
- If the message was NOT release, goto 1.
- Terminate.

For ESA's using STDIO, Sync is responsible for starting the ESA application when necessary.

Remote Proxy

A CL ESA is usually easier to write, but it is often desirable to run an ESA remotely. Since this is not directly possible with CL applications, Agiloft supplies a small application called the Remote Proxy which converts a CL ESA into an HTTPS ESA. The Remote Proxy is a Java Jar file which can be downloaded by clicking the Download ESA Remote Proxy button from the Sync Configuration screen. The downloaded `esa.jar` file will contain both the Remote Proxy code and a settings file which references the sync configuration, namely the Agiloft server name and external system ID of the configuration which were created via the sync GUI. It should then be run in standard Java on the machine of your choice to start polling the Agiloft sync subsystem.

To run the Remote Proxy on an external system host, you should:

1. Download and install the latest Java version, e.g. JRE 1.6, from www.java.com to the external system host, if it isn't already installed.
2. Fill in the command line field in the Sync configuration GUI.
3. Download "esa.jar" by clicking the "Download ESA Remote Proxy" button in the Sync Configuration wizard.
4. Start the ESA Remote Proxy application by running the `"java -jar esa.jar"` command on the external system host.

If you are running a third-party command-line ESA remotely, please make sure the ESA command-line executable can be launched by the ESA Remote Proxy application. In particular, check "execute" permissions and PATH. The ESA Remote Proxy should then launch the ESA application via the command, specified in the Sync Configuration.

ESA Remote Proxy doesn't change the current directory. If you need the current directory to be different from the one containing the esa.jar file, use a path to it within the start up command:

```
java -jar <path>\esa.jar
```

For example:

```
java -jar c:\ewsync\esa.jar
```

When you download esa.jar from the Sync Configuration GUI, it will have the necessary configuration parameters such as ESA type, External System ID and polling period predefined within the jar file. Those parameters can also be changed using command-line arguments in the java -jar command:

Command-line argument	Description
-ewhost	Host and port where Agiloft is run. This parameter is useful if a server "outer" name differs from its "intranet" name
-id	External System ID, see below
-pollprd	Polling period, seconds
-esatype	ESA Name. Valid names are "EXCHANGE" "FILE" "THIRD_PARTY_COMMAND_LINE"

The Remote Proxy should normally always be running on the external machine similar to a daemon. The Remote Proxy will launch and stop a CL ESA at the request of the sync subsystem.



Notes

- The Java archive file is named `esa.jar` because it also contains some inbuilt ESAs shipped with the sync subsystem, and is used in exactly the same way and does not contain any of your code. Normally it will connect to the Agiloft server it was downloaded from and start polling it for messages:

Remote Proxy initiation and sample output

```
c:\syncHome>java -jar esa.jar
Agiloft ESA Remote Proxy running.
Connecting to https://sync.agiloft.com (external system ID is
1386095731639@Demo) to get sync
parameters...
Success. Running ESA THIRD_PARTY_COMMAND_LINE
Use Ctrl-C to stop this app.
```

- In the External Sync setup, you must click Next to save the sync configuration before running the remote proxy.
- You may specify an IP address from which the system can only receive connections. If you are using NAT, you should specify the external IP address as "seen" by Agiloft, rather than the internal address.

When it receives messages, it will run the CL ESA and pass the messages to it. The following illustration depicts a configuration of the Agiloft sync server, a Remote Proxy and a CL ESA. The Remote Proxy and the CL ESA reside in the external machine.

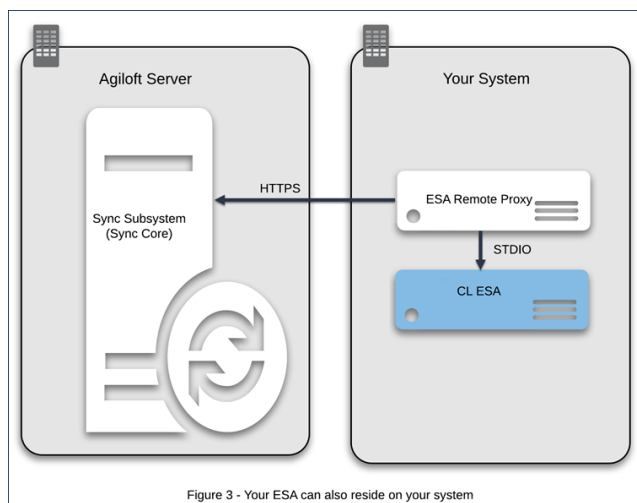


Figure 3 - Your ESA can also reside on your system

Note: the command to run a CL ESA is always Start_ESA.bat on Windows or Start_ESA.sh on Linux.

For maximum security, it is strongly advised that you run your Remote Proxy under a separate account, which is only allowed to run your CL ESA application and does not have permission to do anything else. UNIX users may use a sticky bit to give more rights to the CL application.

Java ESA Development

A number of support libraries have been provided to simplify third-party ESA development. Currently, the only libraries available are for the Java programming language. To develop an ESA in Java, all you need do is to develop a class implementing a single `ExternalSystemAdapter` interface and write about 10 lines to construct your ESA using either STDIO or HTTP transport.

The ESA example package contains a Sample ESA class, `com.supportwizard.sync.sampleesa.SampleEsa`, and a number of other classes which form the ESA Java Support Library. The library API is very simple and only contains two methods for the `com.supportwizard.sync.remoteesa.EsaRunner` class, both of which receive an ESA implementation as a parameter:

Method	Description
- public static void runCommandLineEsa (<code>ExternalSystemAdapter esa</code>) - throws <code>EsaException</code>, <code>InterruptedException</code>, <code>IOException</code>	Runs command line ESA. This method starts by reading standard input, converts received XML messages into ESA calls, dispatches them and prints the XML-ized responses to standard output. This method exits after processing <code>ExternalSystemAdapter.release()</code> call. Normally, the command line ESA application should terminate after this method exits.
- public static void runHttpsEsa (<code>ExternalSystemAdapter esa</code>, <code>URL ewServerURL</code>, <code>String</code> <code>externalSystemID</code>) - throws <code>EsaException</code>, <code>InterruptedException</code>, <code>IOException</code>	Runs HTTPs ESA. This method connects to an Agiloft host, waits for sync message, processes it by calling ESA methods and posts back the reply. This method exits after processing <code>ExternalSystemAdapter.release()</code> call or upon a timeout. Normally, HTTPs ESA should re-invoke it in a cycle, probably with some delay.

Please see [JavaExecutableEsa](#) on using `EsaRunner`.

Usually, you will not need to use any other parts of the Java support library directly. However, if you plan to implement an ESA using a programming language other than Java, or plan to change the way XML messages are passed to and from or the way they are processed, such as by implementing some kind of internal queuing, you may find this useful to explore.

Major classes of interest are:

Class	Description
<code>JavaExecutableEsa</code>	CL ESA 'main' class. To make a minimal change, it is sufficient to replace the line:

	<pre>ExternalSystemAdapter esa = new SampleEsa();</pre> with <pre>ExternalSystemAdapter esa = new<new ESA class here>;</pre> to get a fully working command line ESA.
Transport: - StreamTransport - HttpXmlEsaTransport	These classes are responsible for the XML message flow. StreamTransport passes messages over STDIO streams HttpXmlEsaTransport passes messages over HTTP(S)
- Message - Call - Response	These classes encapsulate particular XML messages. Note: there is a Message.dispatch method, which invokes an ESA method, corresponding to the message.
- XMLGate - MessageSerializer - JaxbParser - JaxbSerializer	These classes handle XML message generation and parsing.
- EsaTransmitter - HelperApiTransmitter	These classes translate ESA and HelperApi calls into XML messages.
EsaReceiver	This class takes messages from Transport and invokes ESA methods according to the message received.

The library classes are supplied in the source form. You are free to reuse them for any Agiloft ESA-related development.

Reusable Classes

The Java ESA Example following contains a number of classes, which can be reused by any Java ESA. These classes are:

- ExternalSystemAdapter interface. Your ESA implementation must include a class, implementing this interface. In fact, this is the only piece of code you must write to get a fully working ESA, based on the example code.
- Message classes, residing in com.supportwizard.sync.interfaces.transport package. These classes encapsulate XML messages to be exchanged.

- Classes of `com.supportwizard.sync.interfaces.transport.xml` and `com.supportwizard.sync.interfaces.transport.xml.jaxb` packages, notably `JaxbParser` and `JaxbSerializer`, parsing and generating XML.
- `StreamTransport` class, implementing STDIO Transport
- `HttpXmlEsaTransport` class, implementing HTTP(s) Transport
- A set of message dispatching classes - `JavaEsaInvoker`, `MessageDispatcher`, etc - which are responsible for calling `ExternalSystemAdapter` when a message arrives, and marshaling the result back to the message exchange stream.

The only things needed to build an ESA using STDIO transport are:

- Write your implementation of `ExternalSystemAdapter` interface
- Replace the line `"FileDirectoryEsa esa = new FileDirectoryEsa();"` in the "main" class with a construction of your class.
- If you wish to compile and jar using the script `make.bat`, you will have to add your package(s) to the Java compiler (javac) call arguments.

You may reuse any classes of the example and handle messages in some other manner, if you choose, but we recommend the method above to facilitate incorporating future changes and fixes.

Java ESA Example and Classes

The following example is a command-line ESA, with an alternative transport class to make an HTTPs ESA, synchronizable with a set of files in some file system directory. For the sake of clarity, some details of XML parsing, logging and message-to-method dispatching are omitted here. The full source files are available in `sync-example.zip`.

File Directory External System

The External System that this example fits holds files in the Java properties file format in either plain or XML properties.

```
"Plain" properties:
#comment
fieldA=value A
fieldB=value B
"XML" properties:
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
<comment>comment</comment>
```

```
<entry key=" fieldA"> value A</entry>
<entry key=" fieldB"> value B </entry>
</properties>
```

The particular type is to be defined by the ESA Parameter format. The set of fields is defined by the ESA Parameter fields.

To test the ESA compilation...

1. Install the latest JDK on your machine.
2. Unzip sync-example.zip to some directory - \$ESA_DIR\$.
3. Run make.bat in that directory. This will compile the sources to \$ESA_DIR\$/out directory and build "fileesa.jar" JAR file.
4. Run "run.bat < in.txt" to test the ESA. This will run the example command-line ESA and feed "in.txt" file to its standard input.

You should get an output such as:

```
E:\sync-example>run.bat < in.txt
<sync>
<result>2009-06-16T19:58:49.062+00:00</result>
</sync>
<sync>
<result></result>
</sync>
```

If you see this, you have compiled the example successfully.

Principal Classes

Two classes are central to this example: `FileDirectoryEsa` and `JavaExecutableEsa`. Other classes in the example can be treated as a Java support library. You don't have to modify them and may use them as is in your own ESA.

Testing the ESA in Sync

To use this ESA in Sync configuration...

1. Uncomment and correct the first line of the run.bat file to read @cd \$ESA_DIR\$, where \$ESA_DIR\$ is the directory where the example is unzipped.
2. Select Third-party command-line in the Sync Configuration Wizard.

3. In the same GUI, put the command `cmd /c $ESA_DIR$\run.bat` in the input box labelled Command Line.
4. Set up the rest of the configuration.

External System Adapter Interface

This is an interface which every Java ESA should implement:

```
package com.supportwizard.sync.interfaces.esa;

import com.supportwizard.sync.interfaces.esa.ExternalRecord;
import com.supportwizard.sync.interfaces.esa.Cursor;
import com.supportwizard.sync.interfaces.esa.ExternalRelation;

import java.util.Set;
import java.util.List;
import java.util.Locale;
import java.util.Date;
import java.rmi.Remote;
import java.rmi.RemoteException;

/**
 * An interface of External system connector.
 * Every adapter (Data provider) should implement it.
 *
 * This interface has "throws RemoteException" on each method, in order to be
 * {@link java.rmi.Remote}-compliant. An implementation may or may need use this
 * perk.
 * Such an exception, if thrown, is considered a serious, unrecoverable
 * synchronization error,
 * like a communication link failure or such.
 *
 * Some methods do have "Locale" argument. It follows Java convention in naming
 * locales
 * (<language-code>-<country-code>-<variant>).
 * Examples are "en_US", "ru_RU" and "pt_BR".
 * See http://java.sun.com/j2se/1.4.2/docs/api/java/util/Locale.html for more
 * details.
 *
 * If ESA doesn't support required locale or doesn't support localization at all,
 * it should return (American) English label and hint.
 *
 * @author <a href="mailto:anatoly@supportwizard.com">Anatoly Kochubey</a>
 * @version $Id: ExternalSystemAdapter.java.txt 140548 2013-09-13 11:42:12Z
 * alexg $
 */
```

```

public interface ExternalSystemAdapter extends Remote {

    String SYNC_VERSION_1_0 = "1.0";

    String USE_PAGES = "USE_PAGES";

    /**
     * Starts a synchronization. This is the very first method, called during
     Sync.
     *
     * @param externalSystemID should be associated with a current session (if
     applicable).
     * It is also for simple ESA, which might need to call <ew:shortproductname>
     <span class="product1">EW</span><span class="product2">SaaSWizard</span></ew:
     shortproductname> back, but don't do this outside
     * of synchronization cycle - they may avoid storing externalSystemID
     because of this.
     * @return Sync version, supported by ESA. This should be one of
     SYNC_VERSION_... constants
     */
    String startSync(String externalSystemID) throws EsaException,
    RemoteException;

    /**
     * Resets session timeout counter (if ESA has some). Non-WS ESAs may simply
     ignore this.
     */
    void leaseSession() throws EsaException, RemoteException;

    /**
     * Finishes synchronization session. This call denotes successful end of a
     Sync.
     * ESA may clean up any resources, close connections, etc.
     */
    void endSync() throws EsaException, RemoteException;

    /**
     * Releases ESA (it may free all the resources).
     * This is the last method, called on the ESA interface.
     */
    void release() throws EsaException, RemoteException;

    /**
     * Returns Bit-ORed constants from {@link RunModes}, restricting
     synchronization run modes.
     * For example, ({@link RunModes#RUN_SYNC_ACTIONS} bit-OR {@link
     RunModes#RUN_EXTERNAL_TRIGGERED})
     * would designate that ESA supports non-interactive synchronizations by
     both <ew:shortproductname> <span class="product1">EW</span><span class="product2">
     >SaaSWizard</span></ew:shortproductname> and External System requests
     * @return allowed operation modes, bit-ORed constants from {@link RunModes}

```

```

    */
    int getAllowedRunModes() throws EsaException, RemoteException;

    /**
     * Sometimes, ESA may decide there is a necessity to immediately re-run
    synchronization after
     * sync finish. An example is if update actually updates more than 1 record,
    so data are changed in
     * a more broader scope, than sync subsystem expects. In this case, ESA
    should return true from this method.
     * The method is typically invoked before {@link #endSync()}.
     * @return true if ESA wants another synchronization round to be run
    immediately.
    */
    boolean needSyncAgain() throws RemoteException, EsaException;

    /**
     * Gets current UTC time of a remote system, if available.
     * Sync uses this method to compute time shift between <ew:shortproductname>
    <span class="product1">EW</span><span class="product2">SaaSWizard</span></ew:
    shortproductname> server and External System.
     * The shift is taken into account for all datetime "control" values, passed
    to ESA (getModified(),
     * record.modifiedAt, lastSeen timestamps, etc).
     *
     * <p>Note: This shift is not applied when converting record date/datetime
    /time fields.
     *
     * @return current UTC time of a remote system, if available, or null.
     * @throws EsaException
     * @throws RemoteException
     */
    Date getCurrentUTCTime() throws EsaException, RemoteException;

    /**
     * Configures ESA to use with an <ew:shortproductname> <span class="product1"
    >EW</span><span class="product2">SaaSWizard</span></ew:shortproductname>
    instance.
     * This method is called on the ESA when Sync configuration is created
     * (this also lets to check up ESA availability).
     *
     * The implementation of this method depends on the ESA.
     *
     * Simple ESAs, which doesn't need to communicate (call) <ew:
    shortproductname> <span class="product1">EW</span><span class="product2"
    >SaaSWizard</span></ew:shortproductname> back (i.e. neither use Helper API
     * nor initiates Sync by External System requests) may simply ignore this
    call
     * and return the passed externalSystemID.
     *
     * More complex ESAs should store (persist) those values somewhere and use

```

```

them to call
    * (HelperApi) <ew:shortproductname> <span class="product1">EW</span><span
class="product2">SaaSWizard</span></ew:shortproductname> and to authenticate
themselves (externalSystemID) to it.
    * If such an ESA is already configured (already has an associated
ExternalSystemID)
    * it must return the stored value and only replace it by the new parameters
    * if parameter "force" is true.
    *
    * @param externalSystemID externalSystemID to associate ESA with
    * @param force force setting externalSystemID
    * @return externalSystemID or null if ESA is already configured for another
externalSystemID
    */
    String configure(String externalSystemID, boolean force) throws
EsaException, EsaConfigurationException, RemoteException;

    /**
    * Gets a list of external data structure names.
    * "Structure" is a logical data unit, mappable to <ew:shortproductname>
<span class="product1">EW</span><span class="product2">SaaSWizard</span></ew:
shortproductname> table; a table, a folder,
    * a class (in OODB) or any other type of a data grouping.
    *
    * @return list of external system structures
    * @param locale
    */
    Set<ExternalStructure> getStructureList(Locale locale) throws EsaException,
RemoteException;

    /**
    * Gets a list of fields, for a given structure or collection.
    * ID and "ModifiedAt" fields might be not included.
    * For a collection, the method should return all mappable fields of a
collection members.
    *
    * @param locale
    * @return list of fields
    */
    Set<ExternalField> getFieldList(String structureOrCollection, Locale locale)
throws EsaException, RemoteException;

    /**
    * Gets a list of relations (links) from the given structure.
    * This should only include relations, "navigatable" from this object.
    *
    * @param locale locale
    * @param structure structure or collection to query relations for
    * @return relation list
    */
    Set<ExternalRelation> getRelations(String structureOrCollection, Locale

```


locale) throws EsaException, RemoteException;

```
/**
 * Gets a list of relations (links) from the given structure.
 * This should only include relations, "navigatable" from this object.
 *
 * @param locale locale
 * @param structure structure or collection to query relations for
 * @return collections list
 */
```

Set<ExternalCollection> getCollections(String structureOrCollection, Locale locale) throws EsaException, RemoteException;

```
/**
 * Gets localized ESA Parameters meta-data.
 *
 * @param locale locale
 * @return list of ESA private parameters
 */
```

List<EsaParameterMeta> getParametersMeta(Locale locale) throws EsaException, RemoteException;

```
/**
 * Gets all records, modified in external system after given timestamp.
 * For some systems, this may result in a great amount of data to be
transferred,
 * especially for the initial sync. If the ESA predicts such a traffic, it
should
 * return null from this method (NOT an empty set!).
 * In this case, <ew:shortproductname> <span class="product1">EW</span><span
class="product2">SaaSWizard</span></ew:shortproductname> will make use of {@link
#getModifiedPaged} method.
 *
 * ESA may decide how to transfer data in runtime (depending on their
amount).
 * In any case, <ew:shortproductname> <span class="product1">EW</span><span
class="product2">SaaSWizard</span></ew:shortproductname> will call {@link
#getModified} first and only call {@link #getModifiedPaged}
 * if {@link #getModified} returns null
 *
 * @param structure structure to read
 * @param after timestamp
 * @return records, modified after given timestamp.
 */
```

Set<ExternalRecord> getModified(String structure, Date after) throws EsaException, RemoteException;

```
/**
 * Gets modified records in a paged manner. See {@link #getModified}
description.
 * This method is only called if {@link #getModified} call returned NULL.
```

```

*
* @param structure structure to read
* @param after timestamp
* @return records, modified after given timestamp.
*/
Cursor getModifiedPaged(String structure, Date after) throws EsaException,
RemoteException;

/**
 * Gets IDs of records, deleted in external system after given timestamp.
 * ESA may pay or pay not attention to this timestamp. It is OK for ESA to
respond
 * to this call with ALL deleted IDs, ever deleted in the system - sync will
handle this properly.
 *
 * However, this may result in a greater traffic between ESA and EW.
 *
 * The timestamp corresponds to the the time of the last successful
invocation of getDeleted().
 *
 * If a ESA accumulates deletion information, it may always return all
accumulated IDs and purge
 * them after successful "endSync()".
 *
 * In any case, this call may result in a great amount of data to be
transferred.
 *
 * If the ESA predicts such a traffic, it should return NULL from this
method (NOT an empty set!).
 * In this case, <ew:shortproductname> <span class="product1">EW</span><span
class="product2">SaaSWizard</span></ew:shortproductname> will make use of {@link
#getDeletedPaged} method.
 *
 * ESA may decide how to transfer data in runtime (depending on their
amount).
 * In any case, <ew:shortproductname> <span class="product1">EW</span><span
class="product2">SaaSWizard</span></ew:shortproductname> will call {@link
#getDeleted} first and only call {@link #getDeletedPaged}
 * if {@link #getDeleted} returns NULL
 *
 * @param structure structure to read
 * @param since timestamp
 * @return IDs of record, deleted after timestamp
 */
Set<String> getDeleted(String structure, Date since) throws EsaException,
RemoteException;

/**
 * Gets deleted records in a paged manner. See "getDeleted()" description.
This method is
 * only called if "getDeleted()" call returned NULL.

```

```

    * @param structure structure to read
    * @param since timestamp
    * @return IDs of record, deleted after timestamp
    */
    Cursor getDeletedPaged(String structure, Date since) throws EsaException,
RemoteException;

    /**
     * Resets cursor expiration timer.
     */
    void leaseCursor(String cursorID) throws EsaException, RemoteException;

    /**
     * Closes the cursor. This call indicates that <ew:shortproductname> <span
class="product1">EW</span><span class="product2">SaaSWizard</span></ew:
shortproductname> doesn't need the cursor anymore and guarantees that
"readDataPage" will never be called for this cursor.
     */
    void closeCursor(String cursorID) throws EsaException, RemoteException;

    /**
     * Gets data from a cursor. This method is used to
     * actually access the data in the cursor.
     *
     * @param pageIndex index of the page to retrieve
     * @return data page
     */
    Set readDataPage(String cursorID, int pageIndex) throws EsaException,
RemoteException;

    /**
     * Reads a single record in external system
     * @param structure read record of that structure
     * @param pk read record with that ID
     * @return newly created record. Record must contain at least ID and
ModificationTimestamp information.
     */
    ExternalRecord read(String structure, String pk) throws RemoteException,
EsaException, EsaRecordException;

    /**
     * Creates new record in external system
     * @param values record values
     * @return newly created record. Record must contain at least ID and
ModificationTimestamp information.
     */
    ExternalRecord create(String structure, ExternalRecord values) throws
RemoteException, EsaException, EsaRecordException;

    /**
     * Updates the record in external system.

```

```

    * If the record is modified after <code>lastSeen</code> timestamp,
    * throws OptimisticLockFailureException
    *
    * @param lastSeen last seen timestamp.
    * @param values record values
    * @return update timestamp
    */
    Date update(String structure, Date lastSeen, ExternalRecord values) throws
    RemoteException, EsaException,
        EsaRecordException, OptimisticLockFailureException;

    /**
     * Deletes the record in external system, returns true.
     * If the record is modified after <code>lastSeen</code> timestamp,
     * throws OptimisticLockFailureException
     *
     * @param structure structure to create record in
     * @param lastSeen last seen timestamp.
     * @param pk ID of record to delete
     */
    void delete(String structure, Date lastSeen, String pk) throws
    RemoteException, EsaException,
        EsaRecordException, OptimisticLockFailureException;

    /**
     * Counts a number of tickets, with IDs in range of [IDmin;IDmax]
     (inclusive).
     * This method is a callback for {@link HelperApi#detectDeleted(String,
    String,java.util.Date)}
     *
     * @param idMin min ID, inclusive
     * @param idMax max ID, inclusive
     * @return number of records with matching IDs
     */
    int countRange(String externalStructure, String idMin, String idMax) throws
    EsaException, RemoteException;
}

```

ESA Implementation Base Class

This is a helper class and it is not mandatory for a Java ESA. Instead, a Java ESA may implement the `ExternalSystemAdapter` interface directly. However, most simple ESAs will benefit from using it as a sub-class:

```

package com.supportwizard.sync.interfaces.esa;

import java.rmi.RemoteException;

```

```

import java.util.*;

/**
 * Base class for ESA Implementations. Provides some helpers and stubs some
 * rarely used methods.
 *
 * @author <a href="mailto:anatoly@enterprisewizard.com">Anatoly Kochubey</a>
 */
public abstract class ExternalSystemAdapterBase implements ExternalSystemAdapter
{
    /**
     * Gets resource bundle for a locale. This method doesn't take default
     * system locale into consideration,
     * so if the default english messages are put into <basename>(.properties),
     * they are used whenever
     * passed locale doesn't match
     * @param locale
     * @return
     */
    protected ResourceBundle getResources(String basename, Locale locale) {
        ResourceBundle i18n = ResourceBundle.getBundle(basename, locale, new
        ResourceBundle.Control() {
            @Override
            public Locale getFallbackLocale(String baseName, Locale locale) {
                return null;
            }
        });
        return i18n;
    }

    ////////////////////////////////////////////////////
    // ExternalSystemAdapter default implementation

    /**
     * Default implementation calls startSync() and endSync() to verify paramters
     * @param externalSystemID externalSystemID to associate ESA with
     * @param force force setting externalSystemID
     * @return
     * @throws EsaException
     * @throws RemoteException
     */
    public String configure(String externalSystemID, boolean force) throws
    EsaException, RemoteException {
        // Try new parameters
        startSync(externalSystemID);
        endSync();
        return externalSystemID;
    }

    public void leaseSession() throws EsaException, RemoteException {

```

```

        // NO-OP
    }

    public void release() throws EsaException, RemoteException {
        // NO-OP
    }

    public Date getCurrentUTCTime() throws EsaException, RemoteException {
        return null; // No time synchronization
    }

    //////////////////////////////////////
    // Pages - Not implemented by default

    public Cursor getModifiedPaged(String structure, Date after) throws
EsaException, RemoteException {
        throw new EsaException("Not implemented");
    }

    public Cursor getDeletedPaged(String structure, Date since) throws
EsaException, RemoteException {
        throw new EsaException("Not implemented");
    }

    public Set readDataPage(String cursorID, int pageIndex) throws EsaException,
RemoteException {
        throw new EsaException("Not implemented");
    }

    public void leaseCursor(String cursorID) throws EsaException,
RemoteException {
        throw new EsaException("Not implemented");
    }

    public void closeCursor(String cursorID) throws EsaException,
RemoteException {
        throw new EsaException("Not implemented");
    }

    //////////////////////////////////////
    // Miscellaneous

    public int getAllowedRunModes() throws EsaException, RemoteException {
        return RunModes.ANY;
    }

    public int countRange(String externalStructure, String idMin, String idMax)
throws EsaException, RemoteException {
        return 0;
    }

```

```

        public boolean needSyncAgain() throws RemoteException, EsaException {
            return false;
        }
    }
}

```

FileDirectoryESA

FileDirectoryEsa is an ESA implementation class. Its methods are called on receiving corresponding messages and it may call the Helper Api to make a call back to the Agiloft server.

```

package com.supportwizard.sync.file;

import com.supportwizard.sync.interfaces.esa.*;

import java.rmi.RemoteException;
import java.util.*;
import java.io.File;
import java.io.IOException;
import java.io.FileFilter;

import org.apache.log4j.Logger;

/**
 * This ESA synchronizes <ew:shortproductname> <span class="product1">EW<
 /span><span class="product2">SaaSWizard</span></ew:shortproductname> with files
 in some local file directory (or a shared network path).
 *
 * @author <a href="mailto:anatoly@supportwizard.com">Anatoly Kochubey</a>
 * @version $Id: FileDirectoryEsa.java.txt 140548 2013-09-13 11:42:12Z alexg $
 */
public class FileDirectoryEsa extends ExternalSystemAdapterBase {

    private static final Logger log = Logger.getLogger(FileDirectoryEsa.class);

    //////////////////////////////////////
    // ESA parameter Metas.

    /**
     * Local path to the file directory to sync.
     * There can be multiple directories, separated by local system path
     separator.
     */
    private final String DIRECTORY_PATH = "path";

```

```

/**
 * File formats
 */
private final String FORMAT = "format";
private final String FORMAT_PROPERTIES = "Properties";
private final String FORMAT_XML = "XML";

/**
 * List of "fields" in file, separated by comma. Order is important for CSV
format
 */
private final String FIELD_LIST = "fields";

////////////////////////////////////
// Parameter values

// Directories to search in
private Map<String, File> directories;
// Field names
private List<String> fields;
// File reader/writer
private FileDataGate dataGate;

private String externalSystemID;
private HelperApi helperApi;

public void setHelperApi(HelperApi helperApi) {
    this.helperApi = helperApi;
}

public Date getCurrentUTCTime() throws EsaException, RemoteException {
    return new Date(System.currentTimeMillis());
}

public String startSync(String externalSystemID) throws EsaException,
RemoteException {
    log.debug("Starting sync on File ESA " + externalSystemID);
    this.externalSystemID = externalSystemID;

    List<EsaParameter> dirPath = helperApi.getParameter(externalSystemID,
DIRECTORY_PATH);
    if(dirPath.isEmpty()) {
        throw new EsaConfigurationException("No directories to sync, can't
sync");
    }
    directories = new HashMap<String, File>();
    parseDirectories(dirPath.get(0).getStrValue());

    List<EsaParameter> fields = helperApi.getParameter(externalSystemID,

```



```

FIELD_LIST);
    if(fields.isEmpty()) {
        throw new EsaConfigurationException("Field list is not set, can't
sync");
    }
    parseFields(fields.get(0).getStrValue());

    List<EsaParameter> format = helperApi.getParameter(externalSystemID,
FORMAT);
    if(format.isEmpty()) {
        throw new EsaConfigurationException("Format is not set, can't sync");
    }
    parseFormat(format.get(0).getStrValue());

    log.debug("Started sync on File ESA in dir " + dirPath.get(0).
getStrValue() +
        ", fields " + fields.get(0).getStrValue() + ", formatted as " +
format.get(0).getStrValue());

    return ExternalSystemAdapter.SYNC_VERSION_1_0;
}

/** Parses separated directory list. Repeated dirs are ignored */
private void parseDirectories(String strValue) throws
EsaConfigurationException {
    // Init field list
    directories = new HashMap<String, File>();
    // Parse comma-separated list
    StringTokenizer tokenizer = new StringTokenizer(strValue, File.
pathSeparator);
    while(tokenizer.hasMoreTokens()) {
        String dirName = tokenizer.nextToken().trim();
        if(dirName.length() > 0) {
            File directory = new File(dirName);
            if(!directory.exists()) {
                log.warn("Directory " + directory.getAbsolutePath() + "
doesn't exist, creating");
                boolean success = directory.mkdirs();
                if(!success) {
                    throw new EsaConfigurationException("Can't create
directory " + directory.getAbsolutePath());
                }
            }

            if(!directory.isDirectory()) {
                throw new EsaConfigurationException("Not a directory " +
directory.getAbsolutePath());
            }

            directories.put(directory.getAbsolutePath(), directory);
        }
    }
}

```

```

    }
}

/** Parses comma-separated field list. Repeated fields are ignored */
private void parseFields(String strValue) {
    // Init field list
    fields = new ArrayList<String>();
    // Parse comma-separated list
    StringTokenizer tokenizer = new StringTokenizer(strValue, ",");
    while(tokenizer.hasMoreTokens()) {
        String fieldName = tokenizer.nextToken().trim();
        if(fieldName.length() > 0 && !fields.contains(fieldName)) {
            fields.add(fieldName);
        }
    }
}

private void parseFormat(String strValue) throws EsaConfigurationException {
    if(FORMAT_PROPERTIES.equals(strValue)) {
        dataGate = new PropertyFileGate();
    } else if(FORMAT_XML.equals(strValue)) {
        dataGate = new XmlPropertyFileGate();
    } else {
        throw new EsaConfigurationException("Unimplemented format " +
strValue);
    }
}

public void endSync() throws EsaException, RemoteException {
    log.debug("endSync");
    // Close all open files.
    if(dataGate != null) {
        dataGate.closeAll();
    }
    dataGate = null;
    fields = null;
    directories = null;
}

public void release() throws EsaException, RemoteException {
    log.debug("release");
    endSync(); // Do the same
}

////////////////////
// CRUD

public ExternalRecord read(String structure, String pk) throws
RemoteException, EsaException, EsaRecordException {
    File directory = directories.get(structure);
    if(directory == null) {

```

```

        throw new EsaException("Unknown directory " + structure);
    }

    log.debug("Read file");
    File newFile = new File(directory, pk);
    try {
        return dataGate.read(newFile);
    } catch (IOException e) {
        throw new EsaRecordException("IO error", e, newFile.
getAbsolutePath());
    }
}

    public ExternalRecord create(String structure, ExternalRecord values) throws
RemoteException, EsaException, EsaRecordException {
        File directory = directories.get(structure);
        if(directory == null) {
            throw new EsaException("Unknown directory " + structure);
        }

        log.debug("Creating new file");

        // Do data save
        File newFile = null;
        try {
            // Create new file
            if(values.getId() == null || "".equals(values.getId().trim())) {
                newFile = File.createTempFile("sync", "ew." + dataGate.
getDefaultFileExtension(), directory);
            } else {
                newFile = new File(directory, values.getId());
                if(newFile.exists()) {
                    throw new EsaRecordException("Clashed IDs: File " + values.
getId() + " already exist", values.getId());
                }
                newFile.createNewFile();
            }
            log.debug("Created: " + newFile.getAbsolutePath());

            dataGate.create(newFile, values);
            return dataGate.read(newFile);
        } catch (IOException e) {
            throw new EsaRecordException("IO error", e, newFile == null ?
"NoFile" : newFile.getAbsolutePath());
        }
    }

    public Date update(String structure, Date lastSeen, final ExternalRecord
values) throws RemoteException, EsaException, EsaRecordException {
        File directory = directories.get(structure);
        if(directory == null) {

```

```

        throw new EsaException("Unknown directory " + structure);
    }

    log.debug("Updating file " + values.getId());

    // Do data save
    File file = null;
    try {
        // Find the file
        File[] found = directory.listFiles(new FileFilter(){
            public boolean accept(File pathname) {
                return pathname.getAbsolutePath().equals(values.getId());
            }
        });

        if(found == null) {
            log.debug("Nothing found");
            found = new File[0];
        }

        if(found.length == 0) {
            // File is deleted. Now report as optimistic lock failure
            log.debug("No file found");
            throw new ConcurrentDeleteException(values.getId(), new Date
(System.currentTimeMillis()));
        }

        // File is found

        file = found[0];
        assert file != null;
        if((file.lastModified() / 1000) * 1000 > lastSeen.getTime()) {
            // Optimistic lock failure
            log.debug("File is concurrently modified");
            throw new OptimisticLockFailureException(values.getId(), new Date
(System.currentTimeMillis()));
        }

        // Write data to file
        dataGate.update(file, values);
        return dataGate.read(file).getModified();
    } catch (IOException e) {
        throw new EsaRecordException("IO error", e, values.getId());
    }
}

public void delete(final String structure, final Date lastSeen, final String
pk) throws RemoteException, EsaException, EsaRecordException {
    File directory = directories.get(structure);
    if(directory == null) {
        throw new EsaException("Unknown directory " + structure);
    }
}

```

```

    }

    log.debug("Deleting file " + pk);

    // Find the file
    File[] found = directory.listFiles(new FileFilter(){
        public boolean accept(File pathname) {
            return pathname.getAbsolutePath().equals(pk);
        }
    });

    if(found == null) {
        log.debug("Nothing found");
        found = new File[0];
    }

    if(found.length == 0) {
        // File is deleted. Now report as optimistic lock failure
        log.debug("No file found");
        throw new ConcurrentDeleteException(pk, new Date(System.
currentTimestampMillis()));
    }

    // File is found

    File file = found[0];
    assert file != null;
    if((file.lastModified() / 1000) * 1000 > lastSeen.getTime()) {
        // Optimistic lock failure
        log.debug("File is concurrently modified");
        throw new OptimisticLockFailureException(pk, new Date(file.
lastModified()));
    }

    // Do deletion
    file.delete();
}

////////////////////////////////////////
// READ

    public Set<ExternalRecord> getModified(final String structure, final Date
after) throws EsaException, RemoteException {
        File directory = directories.get(structure);
        if(directory == null) {
            throw new EsaException("Unknown directory " + structure);
        }

        log.debug("getModified(" + structure + ", after: " + after + (after ==
null ? "" : "(" + after.getTime() + ")"));
        try {

```

```

        File[] found = directory.listFiles(new FileFilter(){
            public boolean accept(File pathname) {
                if(pathname.isDirectory()) {
                    return false;
                }
                log.debug("checking modifications on " + pathname.
getAbsolutePath() + ", modifiedAt: " + pathname.lastModified());
                return after == null || (pathname.lastModified() > after.
getTime());
            }
        });

        if(found == null) {
            log.debug("Nothing found");
            found = new File[0];
        }

        Set<ExternalRecord> result = null;
        result = new HashSet<ExternalRecord>();
        for(File file : found) {
            result.add(dataGate.read(file));
        }

        return result;
    } catch (IOException e) {
        throw new EsaException("IO error", e);
    }
}

public Set<String> getDeleted(String structure, Date since) throws
EsaException, RemoteException {
    log.debug("getDeleted(" + structure + ", since: " + since + (since ==
null ? "" : "(" + since.getTime() + ")"));

    File directory = directories.get(structure);
    if(directory == null) {
        throw new EsaException("Unknown directory " + structure);
    }

    // Take all IDs, known by EW
    final Set<String> allKnownIDs = helperApi.enumerateKnownIDs
(externalSystemID, structure, since);
    // Filter out still existing
    directory.listFiles(new FileFilter(){
        public boolean accept(File pathname) {
            if(pathname.isDirectory()) {
                return false;
            }
            allKnownIDs.remove(pathname.getAbsolutePath());
            return false;
        }
    });

```

```

    });

    return allKnownIDs;
}

////////////////////////////////////
// Meta information

    public Set<ExternalStructure> getStructureList(Locale locale) throws
EsaException, RemoteException {
    // Our structures are just directories
    Set<ExternalStructure> result = new HashSet<ExternalStructure>();
    for(File directory : directories.values()) {
        result.add(new ExternalStructure(directory.getAbsolutePath(),
directory.getName()));
    }
    return result;
}

    public Set<ExternalRelation> getRelations(String structure, Locale locale)
throws EsaException, RemoteException {
    // No relations
    return new HashSet<ExternalRelation>();
}

    public Set<ExternalCollection> getCollections(String structureOrCollection,
Locale locale) throws EsaException, RemoteException {
    // No collections
    return Collections.emptySet();
}

    public Set<ExternalField> getFieldList(String structure, Locale locale)
throws EsaException, RemoteException {
    Set<ExternalField> result = new HashSet<ExternalField>();
    for(String fieldName : fields) {
        result.add(new ExternalField(fieldName, fieldName, XsdType.STRING,
false, false, true, true, ExternalField.NO_LENGTH_LIMIT));
    }
    return result;
}

    public List<EsaParameterMeta> getParametersMeta(Locale locale) throws
EsaException, RemoteException {
    ResourceBundle i18n = getResources("com.supportwizard.sync.file.
fileesa", locale);

    List<EsaParameterMeta> result = new ArrayList<EsaParameterMeta>();
    result.add(new EsaParameterMeta(DIRECTORY_PATH, XsdType.STRING,
EsaParameterType.SINGLE, true,
        i18n.getString("param.path.name"), i18n.getString("param.path.
hint"),

```

```

        new EsaParameter(""));
    result.add(new EsaParameterMeta(FORMAT, XsdType.STRING, EsaParameterType.
RADIO, true,
        i18n.getString("param.format.name"), i18n.getString("param.
format.hint"),
        new EsaParameter(FORMAT_PROPERTIES), Arrays.asList(
            new EsaParameter(FORMAT_PROPERTIES),
            new EsaParameter(FORMAT_XML)));
    result.add(new EsaParameterMeta(FIELD_LIST, XsdType.STRING,
EsaParameterType.SINGLE, true,
        i18n.getString("param.fields.name"), i18n.getString("param.
fields.hint"),
        new EsaParameter("EwID, Summary")));
    return result;
}
}

```

FileDataGate

The FileDataGate interface referenced in FileDirectoryEsa is a pure implementation class. The example ESA uses the FileDataGate interface implementation to access plain text and XML property files. This is a FileDirectoryEsa implementation specific interface, not a sync interface.

```

/**
 * Abstract reader-writer of different file formats
 *
 * @author <a href="mailto:anatoly@supportwizard.com">Anatoly Kochubey</a>
 * @version $Id: FileDataGate.java.txt 96044 2010-02-23 08:16:34Z gof $
 */
public interface FileDataGate {

    /**
     * Closes all open handles
     */
    void closeAll();

    /**
     * Updates file with data
     * @param file
     * @param values
     */
    void update(File file, ExternalRecord values) throws IOException;

    /**
     * Reads file data into ExternalRecord
     * @param file

```



```

    * @return
    */
    ExternalRecord read(File file) throws IOException;

    /**
     * Creates new file
     * @param values
     */
    File create(File directory, ExternalRecord values) throws IOException;

```

JavaExecutableEsa

JavaExecutableEsa is the main class of the command-line application. You may modify line 28 - new StreamTransport(true, System.in, System.out) - replacing it with new HttpXmlEsaTransport (externalSystemID, ewHost) to transform the example ESA into an independent ESA, connecting over HTTP. You will have to supply or hard-code ExternalSystemID and Agiloft hostname:port.

This class does several primary things:

- The ESA class instance is constructed in line 25
- The StreamTransport class instance handling **STDIO** transportation is constructed in line 28.
- EsaReceiver is constructed and run in lines 27, 30. EsaReceiver glues together the ESA implementation class and transport.

To convert an example into full-featured ESA, you only have to provide an implementation of the ExternalSystemAdapter interface and modify lines 25 and probably 29 of JavaExecutableEsa.

Other sources, including XML parsing/generating, message dispatching, etc, are available as a reference within example.jar.

```

1 package com.supportwizard.sync.remoteesa;
2
3 import com.supportwizard.sync.interfaces.transport.*;
4 import com.supportwizard.sync.interfaces.transport.EsaReceiver;
5 import com.supportwizard.sync.interfaces.
transport.invokers.JavaEsaInvoker;
6 import com.supportwizard.sync.file.FileDirectoryEsa;
7
8 import org.apache.log4j.Logger;
9
10 import javax.xml.bind.JAXBException;
11 import javax.xml.bind.JAXBContext;
12
13 /**
14  * @author Anatoly Kochubey
15  * @version $Id: $

```

```
16 */
17 public class JavaExecutableEsa {
18
19     private static final Logger log =
20     Logger.getLogger(JavaExecutableEsa.class);
21
22     public static void main(String[] args) {
23     try {
24         log.debug("Start JavaExecutableEsa");
25
26         FileDirectoryEsa esa = new FileDirectoryEsa();
27         JavaEsaInvoker invoker = new JavaEsaInvoker(esa);
28         EsaReceiver receiver = new EsaReceiver(invoker,
29         new StreamTransport(System.in, System.out));
30         esa.setHelperApi(invoker.getHelperApi());
31         receiver.run();
32     } finally {
33         log.debug("Exit JavaExecutableEsa");
34     }
35 }
```

Database Interface

This section describes the steps required to connect via ODBC to a MySQL embedded database instance installed together with the standard Agiloft® installation.

Please note, we do not generally recommend accessing the database directly. The main reasons for this are:

Agiloft is a high-level business-oriented information management system. There are many layers of logic between the raw data in the database and what the user sees and is able to export from the GUI.

The Agiloft database is managed by the application itself, rather than a DBA. This results in "machine-generated" database structures which may not correspond to what is presented in the GUI in any obvious way. The "management" processes occur at runtime and the database can "morph" depending on the actions of the admin user: columns can be added, removed or change type.

Further, during the upgrade process Agiloft may modify the database structures to bring them up-to-date with the definitions that are expected by the code. There is therefore a risk of the external reporting engine failing to work with an updated database.

Finally, certain fields and values, artificial columns, are calculated on-the-fly and are not present in the database at all or are present in different tables.

Reviewing your Configuration File

Please review the configuration file named `EnterpriseWizardConfig.xml` that was created by installer on your host.

- Under Microsoft Windows operating system the file is in your SystemDrive root directory e.g. C :
`\EnterpriseWizardConfig.xml`
- Under Linux and UNIX operating systems the file is in `/etc` directory i.e. `/etc/EnterpriseWizardConfig.xml`
- The file is an XML document, the database related information is stored under "ewDatabase" XML tag.

```
<ewconfiguration>
<ewInstallStatus>2</ewInstallStatus>
<ewHome>/usr/local/EnterpriseWizard</ewHome>
<ewDatabase>
<databaseServer>1</databaseServer>
<databaseServerType>mysql</databaseServerType>
<databaseServerHome>/usr/local/EnterpriseWizard/mysql</databaseServerHome>
<databaseAdminLogin>root</databaseAdminLogin>
<databaseAdminPassword>AsaUR4k</databaseAdminPassword>
<databaseEwLogin>ewuser</databaseEwLogin>
<databaseEwPassword>AsaUR4k</databaseEwPassword>
<databaseReaderLogin>ewreader</databaseReaderLogin>
<databaseReaderPassword>GGha42l</databaseReaderPassword>
<databaseName>sw2_std</databaseName>
<connectionType>1</connectionType>
<databaseAddress>127.0.0.1</databaseAddress>
<databasePort>3333</databasePort>
<socketFile>/var/lib/mysql.sock</socketFile>
</ewDatabase>
...
```

Identifying your database

The examples in this section use a MySQL database instance installed together with the standard Agiloft installation as an embedded database, but the same format is used with other databases.

For example, you will find following two values are specified in the Agiloft configuration file mentioned above:

```
<databaseServer>1</databaseServer>
<databaseServerType>mysql</databaseServerType>
```

Identifying the password for the administrative MySQL user on Agiloft host

In the configuration file mentioned above find the tag that lists the administrator level user and password for the embedded MySQL database.

This user is created during install and the password is generated during install and changed on update. By default the administrative level user is "root".

```
<databaseAdminLogin>root</databaseAdminLogin>  
<databaseAdminPassword>AsaUR4k</databaseAdminPassword>
```

For more information, read how to [Encrypt and Decrypt the Database Password](#).

Note MySQL home directory

In the configuration file mentioned above, find the tag that lists the home directory for the embedded MySQL database server.

By default this is set to `/usr/local/Agiloft/mysql` on Linux.

Create Read Only MySQL Users

Though the standard configuration includes a read-only ewreader MySQL user, it is only allowed to connect to the database from localhost and is re-created during installations and updates with a different password. As such it is not usable for external report engine connectivity.

Notes

If the database is recreated during upgrade this user will be lost and will have to be re-created.

This however only happens when the version of MySQL is upgraded and if the upgrade requires data to be recreated.

If in doubt, verify that external connectivity is retained after upgrade.

Follow These Steps

1. From the Agiloft host log into embedded MySQL instance as administrative level user:

```
host:~ youruser$ /usr/local/EnterpriseWizard/mysql/bin/mysql -uroot
-p --socket=/usr/local/EnterpriseWizard/mysql/ewdbsocket -A sw2_std
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 9033
Server version: 5.0.67-log MySQL Community Server (GPL)
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql>
```

2. Create a new user and grant appropriate read permissions:

```
mysql> create user <USER> identified by '<PASSWORD>';
mysql> grant select on sw2_std.* to <USER>;
mysql> exit
```

i.e

```
mysql> create user myuser identified by 'mypassword';
mysql> grant select on sw2_std.* to myuser;
mysql> exit
```

3. Verify connectivity with the new user by logging into MySQL again:

```
<databaseServerHome>/bin/mysql -u_USER_ -p  
--socket=<databaseServerHome>/ewdbsocket -A sw2_std
```

Identify the Database Listening Port

In the configuration file mentioned above find the tag that lists the port for the embedded MySQL database server. By default the MySQL database instance installed together with Agiloft is listening for connections on port **3333**.

```
<databasePort>3333</databasePort>
```

Verify External Connectivity to the Port

To be able to connect to Agiloft database from an external program the port has to be reachable. One of the ways to verify this is to attempt connecting via telnet:

```
telnet {your_ew_host} {port}
```

You can use Ctrl+] and Q to close the session.

```
host:~ youruser$ telnet 192.168.0.100 3333
Trying 192.168.0.100...
^C
host:~ youruser$
```

If the access to this port is denied by a firewall there will be no response. Press Ctrl+C to stop telnet.

Configure ODBC Connection

Open Database Connectivity (ODBC) is an open standard application programming interface (API) for accessing a database. For more information, see [here](#). This topic describes how to configure an Agiloft database with ODBC.

Install MySQL ODBC Driver on the Client Machine

Download ODBC drivers from mysql.com and install them on your client machine.

<http://dev.mysql.com/downloads/connector/odbc/>

Be sure to pick the correct driver version that corresponds to your Operating System. Choose Typical Installation if given a choice by the installer. The installation is rather straightforward, the following articles may be helpful as reference for various OS's.

Windows XP operating system: <http://www.netadmintools.com/art429.html>.

UNIX and Linux: <http://dev.mysql.com/doc/refman/5.0/en/connector-odbc-installation-binary-unix.html>.

Configure ODBC datasource

After installation the ODBC datasource has to be configured for Agiloft database. The following documents describe how to configure the datasource on various OS. <http://dev.mysql.com/doc/refman/5.0/en/connector-odbc-configuration.html>.

Windows XP operating system: <http://www.netadmintools.com/art430.html>.

Linux: <http://www.easysoft.com/developer/interfaces/odbc/linux.html>.

You may want to configure the datasource as System - available to all users.

The important thing to remember is to specify the correct user, password and port.

Be sure to test the connection - both Windows and Linux/UNIX connection managers provide this functionality.

Connect from your client application

Once the DSN has been configured, most applications will display it among others in a list of available DSN's.

Query Tables

You can find out the database-level name of Agiloft tables in Agiloft GUI directly via Table Wizard: **Setup > Table > YOUR TABLE > Edit/General > TableName**. For example the database name for table Support Case in standard Demo KB is "sw_9991f_case".

However, these names may change during upgrades or if you export/import the knowledgebase onto a new server. A more reliable method is to obtain the physical table name from the Logical Table Name that is provided in the GUI: Navigate to **Setup > Table > YOUR TABLE > Edit > General tab**. Use the following query to obtain the physical table name:

```
select a.dbname from swtable a, swproject b where a.name='case' and
a.swprojectid=b.swprojectid and b.name='<Your KB Name>';
```

You should obtain a result such as:

```
+-----+
| dbname |
+-----+
| sw_20288f_case |
+-----+
```

Please be aware that since Agiloft is a business-level information management system not all data available via GUI in a certain Agiloft table can be directly found in the corresponding database table.

1. Some of the data is in a serialized format unsuitable for direct queries or linking, for example multi-value linked fields.
2. When a column of a record in the target table imports a column of a record in a donor table on the level of database this is expressed with indirect links via a linking table.

EW caches the values of imported fields such as the "Updated By" field in the local table to improve performance when searching or reading records, so the data from such fields is available in the target table using the Field DBName shown in **Setup > Table > Edit > Fields tab**.

However if the application requires extra data from the source table, obtaining it becomes more difficult. Since the link is defined on the level of meta-data in order to use such relationship to build a true "join" between two tables an external application would have to repeat the logic of linking Agiloft knowledgebase meta-data with the linking table. Instead please apply the following workaround.

Workaround for single-value single- or multiple-column Linked Fields:

There is one feasible way of querying tables linked in Agiloft directly. It will only work on single-value single- or multiple-column linked fields. In the GUI one has to add the ID field to the set of fields imported from the donor table into the target one. This will cause a column to be added to the target table. Though there is no strict relationship on the database level the values in this column can be used to forge links between two tables in the external reporting tool.

3. Choices values are stored as identifiers in the database. These identifiers may change when the knowledgebase is reconfigured by the administrator. In general there is no explicit guarantee that this internal values will stay the same during the lifetime of the product/knowledgebase.

As such these should be resolved against swchoicelines table bound by the desired value in "choicelinetext" column, linked to swchoices table via "choiceid" column, bound by the name of the search in "name" column and linked to swproject table via swprojectid column, bound by the name of the knowledgebase in the name column. The name of the search can be found in GUI via **Setup > System > Manage Choice Lists > Choice Name**.

The following query obtains the ID that is used to represent the Yes value in the Demo KB for the yes_no choice list:

```
select l.choicelineid from swchoicelines l, swchoices c, swproject p
where l.choicelinetext='Yes' and l.choiceid=c.choiceid and
c.name='yes_no' and c.swprojectid=p.swprojectid and p.name='Demo';
```

Encrypt and Decrypt Database Passwords

The EnterpriseWizardConfig.xml file contains database passwords stored in an encrypted format. In some scenarios, such as creating a read-only MySQL user, you must decrypt the database password. This requires you to encrypt the new root password and then replace the XML tag databaseAdminPassword in EnterpriseWizardconfig.xml. If you need to determine the original password, you can also decrypt the password from EnterpriseWizardconfig.XML.

Note: The commands are platform independent, apart from the direction of the slashes.

- **To encrypt a new password** in the console, run the following command, replacing \$ALHOME with the actual installation directory and newrootpassword with the actual password: `$ALHOME/jre/bin/java -jar $ALHOME/resources/java/lib/encrdecr.jar -e newrootpassword.`
- **To add the encrypted password**, edit EnterpriseWizardconfig.xml and replace the XML tag databaseAdminPassword with the new value: `<databaseAdminPassword>-af6493d3a1c236cc3b5248210e0204e</databaseAdminPassword>.`
- **To decrypt the password** in the console, enter the following command, replacing the given value with the one inside the databaseAdminPassword tag: `$ALHOME/jre/bin/java -jar $ALHOME/resources/java/lib/encrdecr.jar -d 5dcd1311d8854747207a6df87216de44`

The following bash string also performs the action:

```
$(grep -E '\<ewHome>.\?<' /etc/EnterpriseWizardConfig.xml | sed -r 's/\s*<\/?ewHome>
//g')/jre/bin/java -jar $(grep -E '\<ewHome>.\?<' /etc/EnterpriseWizardConfig.xml |
sed -r 's/\s*<\/?ewHome>//g')/resources/java/lib/encrdecr.jar -d $(grep -E
'\<databaseAdminPassword>.\?<' /etc/EnterpriseWizardConfig.xml | sed -r 's/\s*<\/?
databaseAdminPassword>//g')
```

Optimize MSSQL for Development or Test Environments

If you experience problems related to MSSQL in development or cruise environments, use this article to explore options that might optimize your system and resolve the issues. If you are not experiencing any issues, there is no need to perform any of these steps.

 These tips are provided for DBAs to make use of. If you aren't familiar with an option listed, do not use it.

Required or Recommended

Start with these steps, which are required or recommended:

- Make sure that [XA transactions](#) are enabled to correctly process JMS work. This is required.
- We also recommend using a separate MSSQL instance for Agiloft. This prevents inappropriate locks by other products within the same MSSQL instance, and it allows you to reserve the sysadmin role for aluser.

Optional Optimizations

Consider trying any or all of these methods to optimize your system or resolve issues:

- At **Database Properties > Options**, set the recovery model to Simple. If for some reason you must use a full recovery model, configure a periodic backup to avoid constant transaction log growth.
- For versions below MSSQL2016: At **Database Properties > Files**, set Autogrowth to 200Mb+ for data and 64Mb+ to log. Set log autogrowth for tempdb to 64Mb+.
- At **Server Properties > Processors**, set Maximum Worker Threads to 2048 and set Boost SQL Server Priority to On. Note that this might cause some instability.
- For versions below MSSQL2014: At **Server Properties > Memory**, set the maximum server memory to half of server memory.

Finally, it is highly discouraged to use a separated architecture, with JBoss at one server and MSSQL at another. However, if this is required for some reason, consider these tips for managing it:

- Use two virtual machines at one host, or use a host and guest as two machines, with the appropriate network interface. For example, use host-only networking for VMWare.

- Agiloft uses a packet size provided by the MSSQL server, so it's possible to tune network performance by tuning the MSSQL server default packet size at **SSMS > Server > Properties > Advanced > Network > Packet Size**.

Troubleshooting Tools

If none of the other options resolve MSSQL-related issues, you could choose to try these options as well. These options are not recommended for production environments and might affect performance.

 These methods are not safe for production and should be used only after exhausting all other options and taking appropriate precautions.

- Stop all MSSQL services from automatic execution except SQL Server (sqlservr.exe).
- Alter the "sw2_std" database to set auto_shrink on. It is also worth shrinking the tempdb data file (the transaction log cleaned automatically on every MSSQL restart) from time to time.

Custom Scripts

Scripts can extend the function of your knowledgebase by automating events such as moving files to a specific directory on the server. Scripts are executed by creating a script action within a rule or action button.

Script execution

Scripts are automatically executed by the correct interpreter based on their file extension:

- .pl files are executed by the Perl interpreter.
- .py files are executed by the Python interpreter.
- .class files are executed by the Java interpreter.
- .jar files are executed by the Java interpreter. Main class is taken from the Script-Class Manifest entry.
- .exe files are executed as standalone programs.

Script Operations Flow

Agiloft runs scripts in response to some events as defined by Rules specification. Events are always associated with a record from a table and a user action that actually triggered this event. Here is a list of possible events and the order of the resulting actions:

Record created/removed/modified through the GUI user record: a record that has been created/removed/modified and matched our filter.

user action: user record creation/removal/modification.

The order of operations is:

1. Validation is performed on the record at the browser level, for example to confirm that the user has provided value for all required fields.
2. The validity of changes made by the user is checked based on group permissions.

Note: This is an important security feature. Although Agiloft only sends to the browser the fields that the user is permitted to see and only displays the fields for which the user has edit permissions, a hacker might construct their own http stream by hand in an attempt to circumvent this protection. If any of the changes are invalid, the record update is cancelled and an error message is displayed.
3. If the user has changed the workflow state, any associated scripts are called. These scripts may modify fields in the record and act with absolute authority. In other words, they may set field values that the user would not be allowed to set based on his group permissions.
4. The record may trigger business rules that execute automatic actions and any associated scripts. The order in which the business rules are checked is based upon the Priority field in each rule, so if a business rule with high Priority modifies the record, a rule with lower priority will look at the modified record before deciding whether or not it should execute.

Summary based: number of user records matching given filter [operator][threshold], where operator is one of <, >, <=, >=, = and threshold is a number.

user record: all records meeting the filter. User can tell Agiloft to run actions either for all matching records, or only for one of them (see rule settings).

user action: user record creation/removal/modification after which number of matching records exceeded the threshold.

Timer based: periodically executed summary, number of records matching the filter > 0.

user record: the same as in summary based.

user action: undefined.

Linked Fields in Scripts

Since Java scripts have full access to the whole Agiloft functionality, the script author may retrieve whatever linked records he needs. This gives the script developer more flexibility due to no constraints on multiple linked fields or depth of followed links and causes minimal performance impact as long as only the necessary data is retrieved.

However, it is not necessary to fill the linkholders - they will be reconstructed automatically from the values supplied to one or more of the Linked Field data columns. It is important here NOT TO COPY a set of values from the source record and change just some of them, as it may lead to links not being restored correctly.

In perl/exe scripts Linked Fields are exported like any other fields together with the linked record from the donor table. Please note, that the linked record is exported only for a single Linked Field for performance reasons.

To create a link it is enough to set a value to one of the Linked Field data columns, which is done in the same way as for any other field. For example, if you want to change the Assigned To for a record and the Linked Field data column is named `_118_f_login`; you can find this column name in Table wizard on Fields tab in column "Field name"; then put the following to the output file:

```
<field name= "_118_f_user_login"/>
  <value>Umbra</value>
</field>
```

or if you are using Agiloft Perl Script API call

```
EWset::setRecordField('_118_f_user_login', 'Umbra');
```

The script running engine will try to find the best matching record in the donor table and set the Linked Field link to that record. It works the same way for "multiple fields from other table" LF, but you are free to fill only a subset of LF columns. For example, if there are fields User Login and User Phone exported to your table, you may provide a value just for the User Login if you are sure it will result in an exact match.

This will always work for user login, since they are unique across each KB, but other fields could contain repeating values. In this case one has to fill additional fields to achieve a unique match, otherwise the first suitable match will be used.

If no match at all is found, the link will be left empty.

Input File Example

The following information is provided as a guide to the internal implementation of scripts, for use mainly in adding support for additional languages.

Scripts receive several input parameters:

- a set of global variables including current user's record from Contacts, same as used in saved searches and formulas.
- script event source that could be on of the following:
 - WEB_WORKFLOW - script is triggered by workflow while record was edited in GUI
 - WEB_BUSINESS_RULE - script is triggered by rule while record was edited in GUI
 - EMAIL - script is triggered when record was created from inbound email
 - API - script is triggered when record was created or modified with Web Services
 - TIMER_BUSINESS_RULE - when script is triggered by rule based on schedule

current record values. Permissions won't be applied either on reading or on saving. Scripts can also block user action or force user log-out. If a user record contains linked fields, then corresponding records from the linked tables are also provided. For example, if we have a record from Cases table, that contains assigned_to field= "John" which references Contacts table, then John's record from Contacts table will be also included. Only single linked records are exported, i.e. communications tables and any other multiple value LF records will not be exported. This constraint is applied for scripts on perl, .exe scripts and all other languages that interact with Agiloft via .xml files.

[optional] old record values, available when script run in response to "record modified" event. This information allows a script to understand what fields were actually changed by the user, or by preceding rules-triggered actions. Script should output:

- exit code – either in XML or as standard exit code. All other parts of output are optional. See DTD for output XML file.
- [optional] new values for record. Please note, that it is not necessary and even not recommended for some cases, like Linked Fields with multiple columns, to "copy" input records data to output. It is enough to output only the fields your script changes.
- [optional] debug message - will go to Agiloft log.
- [optional] user message - will be shown to the user in the case of non-redirecting exit code - as error message if the script blocks the change or as status message in table view if change accepted by script.
- [optional] redirect-URL to redirect user browser in the case of redirecting exit code.
- [optional] report-message to store in record history to describe script actions upon the record.

More detailed description of input and output formats is provided in the next chapters.

Stringifiers Description

Date, Time, and Datetime fields

Conversion to string:

Date field type is formatted using following pattern: dd/MM/yyyy

Datetime is formatted using the following pattern: dd/MM/yyyy HH:mm:ss

Time is formatted using the following pattern: HH:mm:ss

Conversion is done using current KB timezone and locale.

Parsing from string is as follows:

1. For each type (Date/Datetime/Time) there are list of patterns, which are used to parse.
 - a. Date "dd/MM/yyyy", "MM/dd/yyyy", "MM/dd/yy", "dd/MM/yy", "MMM dd yyyy", "MMM dd yy", "dd MMM yyyy", "dd MMM yy"
 - b. Datetime "dd/MM/yyyy HH:mm:ss", "MM/dd/yyyy HH:mm:ss", "MM/dd/yy HH:mm:ss", "dd/MM/yy HH:mm:ss", "MMM dd yyyy HH:mm:ss", "MMM dd yy HH:mm:ss", "dd MMM yyyy HH:mm:ss", "dd MMM yy HH:mm:ss"
 - c. Time "HH:mm:ss"
2. If the string can't be parsed using any of these patterns then the converter tries to parse its value as a number, which represents time in milliseconds.
3. If finally string can't be parsed - then exception is thrown.
4. Parsed date is converted to KB time zone and current locale.

Elapsed time fields

All values in DB are stored as time in milliseconds since January 1, 1970 00:00:00.000 GMT in the Gregorian calendar. Conversion is dependant on elapsed time column properties. They are:

showDays - use days in date format

showHours - use hours in date format

showMinutes - use minutes in date format

showSeconds - use seconds in date format

decimalHours - means show hours in decimal format, like 3.5

singleInputBox - means show single input box. No input boxes for days, hours, etc.

round - means round to a certain number of digits, for example, setting "round" to 2 would round 5.66666 to 5.67

padding - means show a certain number of digits to the right of the decimal point, for instance, setting "padding" to 3 would display 5.7 as 5.700

All the above properties can be configured from the **Setup > Tables > Edit > Fields screen**.

Conversion to string is quite simple, the algorithm is as follows:

1. Convert from milliseconds to seconds
2. if (showDays) - calculate number of days (value/86400). if it's not single input box - append word "XXX days" to output, otherwise only XXX
3. if (showHours) -
 - a. - if singleinputbox and we already wrote something in output - then append delimiter ":"
 - b. - calculate number of hours (remain_value/3600)
 - c. - if decimal hours enabled - then append formatted value of hours using padding/round, then print out delimiter if single box is enabled or word "hours". Go to step 6.
 - d. - otherwise - go to step 4.
4. if show minutes is enabled - print out floor value of hours, otherwise append rounded value of hours
5. if singlebox is disabled - print out word "hours"
6. if showMinutes is enabled and decimal hours disabled
 - a. - if singlebox is enabled and we already printed out something - print out delimiter ":"
 - b. - calculate and print out minutes.
 - c. - if singlebox disabled - print out word "minutes"
7. if showseconds is enabled and decimal hours disabled
 - a. - if singlebox is enabled and we already printed out something - print out delimiter ":"
 - b. - calculate and print out seconds
 - c. - if singlebox disabled - print out word "seconds"

So, possible values for 1 day 2 hours and 23 min 15 seconds could be:

1: 2:23:15 - when singlebox is enabled

1: 2 - when singlebox is enabled, decimal hours are enabled and show minutes with seconds is disabled.

1 day 2 hours 23 minutes 15 seconds - when singlebox is disabled and we are showing all time units.

1 day 2.0 hours when singlebox is disabled and decimal hours is enabled with round/padding equal to 0 etc. depending on current settings of the column.

Conversion from string to Agiloft internal representation is as follows:

1. if singlebox is enabled then use ":" as delimiter, otherwise space ""
2. Parse whole string and extract string values for each time unit.
3. Iterate through time units and convert/parse string values to numbers.
4. Calculate final value of elapsed time. Note, time must be in milliseconds passed from January 1, 1970 00:00:00.000 GMT Gregorian

Object Type fields

Converted to string as \${RootSubtypeLabel}. {...}{RecordSubtypeLabel}. please, note that subtype labels are used here so you will get localized label value for locale of rule creator.

Record links fields

The external form is (example) "[ID1][ID2][ID3]" where ID1, ID2, ID3 are IDs of linked records.

You can't modify record links by passing new set of linked IDs on save. An empty string represents an empty link set.

Choice fields

Choices are shown as simple text strings (note that choice lines cannot contain commas inside their labels).

Values "\$none", "\$choose_one" and "\$choice_item" are treated as "nothing selected".

Multichoice fields

Multichoice are shown as a text strings separated by commas (hence the choice line itself cannot contain comma inside its label).

Multiple values linked fields

Multiple values are exported in format:

```
<dao_line $recordId1> Some value</dao_line> <dao_line $recordId2> Another value</dao_line>
```

Where \$recordId1 and \$recordId2 are IDs of the donor records these values are from.

Multiple Linked Field values cannot be modified by external scripts.

Perl based Scripts

Using Custom Scripts

Agiloft provides a custom scripting capability that is a powerful tool for validating data, automating business processes and integrating with other back-end programs.

Scripts may be created in any standard programming language such as Perl, Java, C++, or Basic and run by Business Rules at **Setup > Rules** or Workflows at **Setup > Workflow**. These wizards tell you in which directory on the server the script must be placed. For security reasons, it is not possible to upload the script to the server using the GUI; the script must be installed directly on the Unix or Windows file system.

If you would like to use your own Perl distribution with scripts rather than the default distribution, you can define the location of your preferred perl.exe using the [Location of external Perl directory](#) global variable. For example, you might set it to c:\Perl64\bin.

Scripts can:

- Set field values in the record. For example, a script can apply sophisticated logic to automatically assign new Cases to the correct member of staff.
- Logout the user and redirect his or her browser to a specific URL.
- Verify the data being entered by comparing it against data in an external file or database and, if necessary, block the requested change to the record.
- Write to an external program or file or generate email.
- Return personalized messages based on field values, such as "Hi Bob, thank you for the input"
- Accept or reject changes to the record made by the user.

Scripts are provided with the following information:

- Field values of the record that is being modified. For example, Severity.
- Field values that record held prior to the current modification. For example, the old value of Severity.
- Field values from the Contacts table of the user who is modifying the record. For example, Full Name.
- Field values of any records that are linked to the record that is being modified. For example, the Full Name of the person that the record is currently Assigned To.
- Global variables, such as the name of the KnowledgeBase.
- Modification event source. The possible values are: WEB_WORKFLOW, WEB_BUSINESS_RULE, EMAIL, API, TIMER_BUSINESS_RULE.

Perl API

The Perl API makes it easy to write custom scripts. It consists of two modules - [EWGet.pm](#) and [EWSet.pm](#) - for reading input and writing output files.

There are many ready-made functions available, but many scripts only use 6 functions: `EWget: load`, `EWget::getValue`, `EWset::setRecordField`, `EWset::setMessage`, `EWset::setExitAction` and `EWset::save`, and two of these, 'load' and 'save', are identical for all scripts. If you just master the functions `EWget::getValue`, `EWset::setRecordField`, `EWset::setMessage` and `EWset::setExitAction`, you will be able to write very powerful scripts.

Many of the functions take a parameter `$recordName` as an input variable. There is always a record named "current_state", which is the record that is currently being edited. In addition, the script may have available to it "old_state", which provides the old field values of the record and other `$recordNames`, which provide the values of records that are linked to the input record via linked field relationships.

EWget Module

EWget::load(\$fileName)

Loads the input file into internal data structures, you will be calling it like this:

Example

```
my ($input_fname, $output_fname) =@ARGV;  
EWget::load($input_fname);
```

Comments

This is the first thing you must do before any further calls to EWget functions. This call is the same for all scripts.

EWget::getValue(\$recordName, \$fieldname)

Returns the value for the field of the record.

Example

To retrieve the current value of the 'status' field:

```
EWget::getValue('current_state', 'status')
```

```
EWget::getValue('old_state', 'status')
```

Retrieves the old value of the 'status' field, i.e its value prior to any changes made by the user:

Comments

This is the core method for accessing field values, but see also `EWget::record($path)`, it is a more elegant method for retrieving the fields of the 'current_state' record.

EWget::getGlobalVariable(\$variableName)

Returns the value of the specified variable. For example:

Example

The following function retrieves the value of the "my_first_name" field in the contact entry of the user who caused the script to run.

```
printf "my_name= '% s' \n", EWget::getGlobalVariable('my_first_name');
```

Comments

Global variables contain KB-wide settings along with user-specific settings, they are also used in saved searches and formulas.

EWget::getLinkedValue(\$recordName, ...)

Returns value of a field from a linked record.

Parameters:

- name of base record
- name of field from base record
- index of linked record (always=0 at present, because as noted in the getLinkedRecordsNamesList description, only "single" value linked fields are currently exported to scripts)
- name of field of linked record

Example

To find the value of the "email" field of the employee responsible for the given case:

```
my $responsible_person_email=
EWget::getLinkedValue("current_state", "assigned_to", 0,
"email");
```

Comments

This is the core method for accessing the data in linked records.

EWget::record(\$path)

Returns the value of a field in a record.

This function provides a "compact" navigation through the record data for the "current_state" record.

Path format: fieldname [.reffieldname[#index]]

Example

"email" - returns the value of current_state.email

"email.username" - returns the value of the "username" field for the first referenced record from current_state.email

So:

```
EWget::record(email.username)
```

is equal to

```
EWget::getLinkedValue("current_state",  
"email", 0, "username");
```

Comments

As the above example shows, EWget::record() is not really necessary, but it is certainly convenient.

EWget::getCallInitiator()

Returns a string that indicates the action that caused the script to be called.

The value returned is one of:

WEB_WORKFLOW-A workflow action, typically initiated by a user changing the Workflow State in the GUI.

WEB_BUSINESS_RULE-A business rule that is triggered by a change to a record

TIMER_BUSINESS_RULE-A business rule that runs automatically at specified intervals

EMAIL-Inbound email

API-An action triggered by the Web Services API

Comments

This function is useful when you have a single script that responds differently, depending upon the cause of the event. For example, if a ticket is updated via email and meets some special criteria, you might send a confirmation email if the update was performed through email, and provide a message in the GUI if the update was performed using the web interface.

EWget debug functions:

The following functions are mostly used for debugging scripts.

EWget::getGlobalVariablesNamesList()

Returns a sorted array of global variables names.

Example

Names of all available variables

```
printf "all vars: %s\n", join(',', EWget::getGlobalVariablesNamesList());
```

Comments

You will usually know what global variables you want to use in your scripts, so this function is mostly for debugging purposes and to find out what variables are available.

EWget::getFieldsNamesList(\$recordName)

Returns list of field names for record by record's name.

Example

Obtain dump of all the fields' names for "current_state" record

```
printf "fields names: '%s' \n",  
join ('', EWget::getFieldsNamesList("current_state"));
```

Comments

This function is mostly used for debugging since you will usually know what field names you are interested in before writing the script.

EWget::getRecordsNamesList()

Returns an array with the names of all the records provided to the script.

i The following code dumps all the records names from the input file passed to the script.

```
printf "record names: '%s' \n",
join(' ', EWget::getRecordsNamesList());
```

Comments

This function is mostly used for debugging purposes. There always will be a record named "current_state" and possibly one named "old_state" (if the current record is being modified, rather than created). There may also be records that were linked to the current record.

These are named "recXXX", where XXX is an incremental number starting from 1.

EWget::getLinkedRecordsNamesList(\$recordName, \$fieldName)

Returns sorted list of names of records linked via a "linked" field.

Comments

This function is mostly used for debugging because you will probably know the names of the records that you want to access before writing the script.

Only the linked records where the link does not allow multiple values are passed to scripts, so this function will return an array of 0 or 1 entry in a form of "recXXX" string.

EWget::getTable_name_for_record(\$recordName)

Returns the name of the table that a record belongs to. If \$recordName is omitted, it returns the value of the "current_record". For example:

```
printf "table for current record: '%s' \n", EWget::getTable_name_for_record();  
printf "table for rec3: '%s' \n", EWget::getTable_name_for_record('rec3');
```

In most cases you will already know what tables your records are from, but in case of the linked field type "single field from multiple tables" this function will allow you to determine what table the associated record is from and handle it appropriately.

EWset Module

EWset::save(\$outfileName)

Processes changes requested by the user and stores them in an xml file.

You should call this method at the end of your script to make your changes available to Agiloft. For example, the script may have changed the value of a field, this change will be ignored, unless it calls "save" prior to exiting.

Example

If you used

```
my ($input_fname, $output_fname) =@ARGV;
```

at the start of your script to extract commandline params, then use

```
EWset::save($output_fname);
```

to save your changes.

EWset::setRecordField(\$fieldName, \$fieldValue)

Assigns a value to the specified field in the record or sets the field to NULL if the value is undefined.

Example

To set the value of the "email" field in the current record

```
EWset::setRecordField("email",  
"somemail@somesite.com");
```

Comment

This is the core function for change data in a your script.
This function manipulates both common fields and linked fields.

See [Stringifiers Description](#) for full description of acceptable formats for each type of data.

See also [Linked Fields in Scripts](#) chapter for more information.

EWSet::setExitAction(\$exitAction)

Determines what happens when the script exits. Option parameters are enclosed in double quotation marks.

Allowed options are:

- **"AcceptChanges"** - the record is updated in the normal fashion. You can use setMessage to inform user about the details of update.
- **"RejectChanges"** - the modification is rejected. User will be shown the record editing window, if the action was initiated via web interface.
- **"RejectChangesAndExit"** - the modification is rejected and the user is automatically logged out. After logout, user will be redirected to URL specified by setRedirect function call.
- **"AcceptChangesAndExit"** - the record is updated in the normal fashion and the user is automatically logged out. After logout, user will be redirected to URL specified by setRedirect function call.

Alternative to setExitAction

As an alternative to setExitAction, provided for compatibility with languages that do not allow strings to be returned, the script can provide an exit (return) code.

- If the exit code is 0, the record is updated in the normal fashion. You can use setMessage to inform user about the details of update.
- If the exit code is 1, the modification is rejected. User will be shown the record editing window, if the action was initiated via web interface.
- If the exit code is 2, the modification is rejected and the user is automatically logged out. After logout, user will be redirected to URL specified by setRedirect function call.
- If the exit code is 3, the record is updated in the normal fashion and the user is automatically logged out. After logout, user will be redirected to URL specified by setRedirect function call.

Formally speaking this function sets the value for the tag 'exitAction'.

EWset::setMessage(\$userMessage)

Send a message to the user.

This message will be shown to the user if the script is executed because the user has changed a record in GUI. Formally speaking, this method sets the value for the tag 'message'.

i Example

```
EWset::setMessage("Please, select yes to accept
end-user agreement and create new account")
```

When a script blocks a record edit, it is a good practice to use `setMessage` to explain why the change was blocked and what the user should do. Of course, it can also be used to confirm success or provide some other message.

EWset::setRedirect(\$redirectUrl)

Sets the value of the URL that the user should be directed to if the script logs them out of Agiloft

The redirect URL should be absolute and include "http". It will be opened in the same browser window. Formally speaking, this method sets the value of the tag 'redirect'.

i Example

```
EWset::setRedirect("http://www.enterprizewizard.com")
```

It is useless to call `setMessage()` with `setRedirect` since there will be no Agiloft page shown to display the message.

EWset Debug Functions

The following functions are mostly used for debugging scripts.

```
EWset::setDebug($debugMessage)
```

Write to the log file (formally speaking, sets value for tag 'debug').

This message is saved to the Agiloft log. It can be useful for logging the result of a script or to debug the script.

EWset::setReport(\$reportMessage)

Sets a value in the record history

This message is stored in the record history. Formally speaking, this method sets the value for the tag 'report'.

Note: It is sometimes helpful to write a description of what the script did to the data and why.

EWSet::exit_ew (\$exitAction, \$outfileName)

Combination of functions EWSet::setExitAction(\$exitAction) and EWSet::save(\$outfileName) with the same arguments and meanings.

Example

The function is able to get an exitCode directly, as \$exitAction argument, thus the following pairs of values are equivalent:

"AcceptChanges" or 0 or "0"

"RejectChanges" or 1 or "1"

"RejectChangesAndExit" or 2 or "2"

"AcceptChangesAndExit" or 3 or "3"

Perl Scripts and External Programs

Perl scripts and external programs have much in common. External programs, .exe on Windows, runnable [+x] on Unix or Linux, will be executed by Agiloft directly, using `java.lang.Runtime.exec()`. Perl scripts are distinguished by ".pl" extension and will be run using "perl scriptname.pl" command. Agiloft includes a Perl interpreter with the standard installation.

Scripts take 2 command line arguments: names of the input and output files. Input and output files are in XML format. Note that the output file is precreated by Agiloft but is empty.

Input file conforms to the following DTD:

```
<!DOCTYPE input[
<!ELEMENT input project_name, old_state?,
linked_records, global_vars)>
<!ELEMENT project_name (#PCDATA) >
<!ELEMENT old_state (record) >
<!ELEMENT current_state (record) >
<!ELEMENT linked_records (record) *>
<!ELEMENT global_vars (variable) *>
<!ELEMENT record (field) +>
<!ELEMENT variable (value) >
<!--ATTLIST record
table CDATA #REQUIRED
xml: id ID #REQUIRED-->
<!ELEMENT field (value) >
<!--ATTLIST field
name CDATA #REQUIRED
linked_refid IDREFS #IMPLIED-->
<!ELEMENT value (#PCDATA) >
<!--ATTLIST variable
name CDATA #REQUIRED-->
]>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<input>
  <project_name>Demo</project_name>
  <old_state>
    <record table="cases" xml:id="rec1">
      <field name="_118_f_user_login">
        <value>null</value>
      </field>
      <field name="internal_comments">
        <value>null</value>
      </field>
      <field name="f_from">
        <value>null</value>
      </field>
      <field name="bcc">
        <value>null</value>
      </field>
    </record>
  </old_state>
</input>
```

```

</field>
<field name="publication_date">
  <value>31/12/1969 13:00:00</value>
</field>
<field name="assign_date">
  <value>31/12/1969 13:00:00</value>
</field>
<field name="workflow_state">
  <value>null</value>
</field>
<field name="id">
  <value>8</value>
</field>
<field name="creator_full_name">
  <value>null</value>
</field>
<field name="_dao3_link3">
  <value>[0-14]</value>
</field>
<field name="_dao3_link0">
  <value>[]</value>
</field>
<field name="_link_art_column">
  <value></value>
</field>
<field name="attached_files">
  <value></value>
</field>
<field name="published">
  <value></value>
</field>
<field name="_649_f_fullname">
  <value>null</value>
</field>
<field name="parentid">
  <value>null</value>
</field>
<field name="date_">
  <value>null</value>
</field>
<field name="steps_to_duplicate">
  <value>Heres how to duplicate</value>
</field>
<field name="summary">
  <value>test of new fields</value>
</field>
<field name="attachments">
  <value>null</value>
</field>
<field name="owner">
  <value>14</value>

```

```

</field>
<field name="id0">
  <value>null</value>
</field>
<field name="owned_by" linked_refid="rec3">
  <value>admin</value>
</field>
<field name="_dao3_link">
  <value>[]</value>
</field>
<field name="full_name">
  <value>null</value>
</field>
<field name="email_type">
  <value>null</value>
</field>
<field name="message">
  <value>null</value>
</field>
<field name="telephone">
  <value>null</value>
</field>
<field name="_dao3_link1">
  <value>[]</value>
</field>
<field name="_dao3_link2">
  <value>[]</value>
</field>
<field name="company">
  <value>null</value>
</field>
<field name="solution">
  <value>null</value>
</field>
<field name="creator_email">
  <value>|HTML</value>
</field>
<field name="masterid">
  <value>null</value>
</field>
<field name="reply_to">
  <value>null</value>
</field>
<field name="standard_solution">
  <value>No</value>
</field>
<field name="priority">
  <value>Medium</value>
</field>
<field name="type">
  <value>Case.Bug Report</value>

```

```

</field>
<field name="cc">
  <value>null</value>
</field>
<field name="wfstate">
  <value></value>
</field>
<field name="_110_f_user_login">
  <value>null</value>
</field>
<field name="deleteable">
  <value></value>
</field>
<field name="creator">
  <value>14</value>
</field>
<field name="assignto">
  <value>null</value>
</field>
<field name="description">
  <value>Here we are</value>
</field>
<field name="creator_telephone">
  <value>null</value>
</field>
<field name="email">
  <value>|HTML</value>
</field>
<field name="_weighted_rating">
  <value>null</value>
</field>
<field name="subject">
  <value>null</value>
</field>
<field name="created_by">
  <value>null</value>
</field>
<field name="f_to">
  <value>null</value>
</field>
<field name="_649_f_contacts_cell_phone">
  <value>6503638270</value>
</field>
</record>
</old_state>
<current_state>
  <record table="cases" xml:id="rec2">
    <field name="_118_f_user_login">
      <value>null</value>
    </field>
    <field name="internal_comments">

```

```

    <value></value>
  </field>
  <field name="f_from">
    <value>null</value>
  </field>
  <field name="bcc">
    <value>null</value>
  </field>
  <field name="publication_date">
    <value>31/12/1969 13:00:00</value>
  </field>
  <field name="assign_date">
    <value>31/12/1969 13:00:00</value>
  </field>
  <field name="workflow_state">
    <value>null</value>
  </field>
  <field name="id">
    <value>8</value>
  </field>
  <field name="creator_full_name">
    <value></value>
  </field>
  <field name="_dao3_link3">
    <value>[0-14]</value>
  </field>
  <field name="_dao3_link0">
    <value>[]</value>
  </field>
  <field name="_link_art_column">
    <value></value>
  </field>
  <field name="attached_files">
    <value></value>
  </field>
  <field name="published">
    <value></value>
  </field>
  <field name="_649_f_fullname">
    <value>null</value>
  </field>
  <field name="parentid">
    <value>null</value>
  </field>
  <field name="date_">
    <value>null</value>
  </field>
  <field name="steps_to_duplicate">
    <value>Heres how to duplicate</value>
  </field>
  <field name="summary">

```

```

    <value>test of new fields</value>
</field>
<field name="attachments">
    <value>null</value>
</field>
<field name="owner">
    <value>14</value>
</field>
<field name="id0">
    <value>null</value>
</field>
<field name="owned_by" linked_refid="rec3">
    <value>admin</value>
</field>
<field name="_dao3_link">
    <value>[]</value>
</field>
<field name="full_name">
    <value>null</value>
</field>
<field name="email_type">
    <value>null</value>
</field>
<field name="message">
    <value>null</value>
</field>
<field name="telephone">
    <value></value>
</field>
<field name="_dao3_link1">
    <value>[]</value>
</field>
<field name="_dao3_link2">
    <value>[]</value>
</field>
<field name="company">
    <value></value>
</field>
<field name="solution">
    <value></value>
</field>
<field name="creator_email">
    <value>|HTML</value>
</field>
<field name="masterid">
    <value>null</value>
</field>
<field name="reply_to">
    <value>null</value>
</field>
<field name="standard_solution">

```

```

    <value>No</value>
  </field>
  <field name="priority">
    <value>Medium</value>
  </field>
  <field name="type">
    <value>Case.Bug Report</value>
  </field>
  <field name="cc">
    <value>null</value>
  </field>
  <field name="wfstate">
    <value>Closed</value>
  </field>
  <field name="_110_f_user_login">
    <value>null</value>
  </field>
  <field name="deleteable">
    <value></value>
  </field>
  <field name="creator">
    <value>14</value>
  </field>
  <field name="assignto">
    <value>null</value>
  </field>
  <field name="description">
    <value>Here we are</value>
  </field>
  <field name="creator_telephone">
    <value></value>
  </field>
  <field name="email">
    <value>|HTML</value>
  </field>
  <field name="_weighted_rating">
    <value>null</value>
  </field>
  <field name="subject">
    <value>null</value>
  </field>
  <field name="created_by">
    <value>null</value>
  </field>
  <field name="f_to">
    <value>null</value>
  </field>
  <field name="_649_f_contacts_cell_phone">
    <value>6503638270</value>
  </field>
</record>

```



```

</current_state>
<linked_records>
  <record table="contacts" xml:id="rec3">
    <field name="_appendtext_ref_key_created_by">
      <value></value>
    </field>
    <field name="_appendtext_ref_key_city">
      <value></value>
    </field>
    <field name="stop_work">
      <value>21:03:36</value>
    </field>
    <field name="lead_owner">
      <value>null</value>
    </field>
    <field name="_appendtext_ref_key_last_name">
      <value></value>
    </field>
    <field name="fullname">
      <value>Admin Default Admin</value>
    </field>
    <field name="_dao3_link0">
      <value>[]</value>
    </field>
    <field name="_link_art_column">
      <value></value>
    </field>
    <field name="attached_files">
      <value></value>
    </field>
    <field name="parentid">
      <value>null</value>
    </field>
    <field name="made_sale?">
      <value></value>
    </field>
    <field name="probability_of_sale_within_1_month">
      <value>0.0</value>
    </field>
    <field name="_login">
      <value>admin</value>
    </field>
    <field name="time_zone">
      <value></value>
    </field>
    <field name="_appendtext_ref_key_first_name">
      <value></value>
    </field>
    <field name="attachments">
      <value>null</value>
    </field>
  </record>
</linked_records>

```

```

<field name="_106_sw_description">
  <value>Staff</value>
</field>
<field name="probability_of_sale_within_3_months">
  <value>0.0</value>
</field>
<field name="_100_sw_systemteamid">
  <value>0</value>
</field>
<field name="id0">
  <value>null</value>
</field>
<field name="nextcontact">
  <value>null</value>
</field>
<field name="amount_of_sale">
  <value>0.0</value>
</field>
<field name="factors_that_helped_sale">
  <value></value>
</field>
<field name="state">
  <value></value>
</field>
<field name="_appendtext_ref_key_additional_comments">
  <value></value>
</field>
<field name="_105_sw_description">
  <value>&lt;dao_line 1&gt;Staff&lt;/dao_line&gt;&lt;dao_line 7&gt;All&lt;
/dao_line&gt;</value>
</field>
<field name="email_type">
  <value>null</value>
</field>
<field name="message">
  <value>null</value>
</field>
<field name="telephone">
  <value>null</value>
</field>
<field name="rating">
  <value></value>
</field>
<field name="_appendtext_ref_key_id0">
  <value></value>
</field>
<field name="close_date">
  <value>08/07/2003</value>
</field>
<field name="reply_to">
  <value>null</value>

```

```

</field>
<field name="_appendtext_ref_key_subject">
  <value></value>
</field>
<field name="_101_sw_systemgroupid">
  <value>0</value>
</field>
<field name="_appendtext_ref_key_other_decision_makers">
  <value></value>
</field>
<field name="_appendtext_ref_key_email_type">
  <value></value>
</field>
<field name="city">
  <value>unknown</value>
</field>
<field name="website_url">
  <value></value>
</field>
<field name="_appendtext_ref_key__231_f_company_company_name">
  <value></value>
</field>
<field name="picture">
  <value></value>
</field>
<field name="last_name">
  <value>Default Admin</value>
</field>
<field name="lead_source">
  <value></value>
</field>
<field name="actions_taken">
  <value></value>
</field>
<field name="creator">
  <value>14</value>
</field>
<field name="country">
  <value>Unknown Country</value>
</field>
<field name="cell_phone">
  <value>null</value>
</field>
<field name="start_work">
  <value>21:03:36</value>
</field>
<field name="f_to">
  <value>null</value>
</field>
<field name="f_from">
  <value>null</value>

```

```

</field>
<field name="bcc">
  <value>null</value>
</field>
<field name="_appendtext_ref_key_owned_by">
  <value></value>
</field>
<field name="fax">
  <value>null</value>
</field>
<field name="id">
  <value>14</value>
</field>
<field name="_231_f_company_company_name">
  <value>null</value>
</field>
<field name="_dao3_link3">
  <value>[]</value>
</field>
<field name="instant_messaging_nickname">
  <value>null</value>
</field>
<field name="_appendtext_ref_key__login">
  <value></value>
</field>
<field name="date_">
  <value>null</value>
</field>
<field name="first_name">
  <value>Admin</value>
</field>
<field name="zip_code">
  <value>12345</value>
</field>
<field name="_appendtext_ref_key_title">
  <value></value>
</field>
<field name="pager">
  <value>|HTML</value>
</field>
<field name="additional_comments">
  <value></value>
</field>
<field name="owner">
  <value>14</value>
</field>
<field name="_appendtext_ref_key_comments">
  <value></value>
</field>
<field name="_appendtext_ref_key_instant_messaging_nickname">
  <value></value>

```

```

</field>
<field name="_appendtext_ref_key_street_address">
  <value></value>
</field>
<field name="other_decision_makers">
  <value></value>
</field>
<field name="_appendtext_ref_key_date_">
  <value></value>
</field>
<field name="_dao3_link">
  <value>[0-6]</value>
</field>
<field name="owned_by">
  <value>null</value>
</field>
<field name="history">
  <value>Actor: admin Date: Thu Mar 31 15:09:15 MSD 2005 Description:
Record
added; initial values:</value>
</field>
<field name="_groupid">
  <value>14</value>
</field>
<field name="comments">
  <value>null</value>
</field>
<field name="title">
  <value>Admin</value>
</field>
<field name="_appendtext_ref_key__235_f_user_login">
  <value></value>
</field>
<field name="_dao3_link2">
  <value>[0-1]</value>
</field>
<field name="_dao3_link1">
  <value>[0-1, 0-7]</value>
</field>
<field name="home_phone">
  <value>null</value>
</field>
<field name="masterid">
  <value>null</value>
</field>
<field name="_235_f_user_login">
  <value>null</value>
</field>
<field name="password">
  <value>*****</value>
</field>

```

```

<field name="_dao3_link5">
  <value>[]</value>
</field>
<field name="_104_sw_description">
  <value>&lt;dao_line 6&gt;admin&lt;/dao_line&gt;</value>
</field>
<field name="type">
  <value>Contact.User.Employee</value>
</field>
<field name="street_address">
  <value>unknown</value>
</field>
<field name="cc">
  <value>null</value>
</field>
<field name="factors_that_hurt_sale">
  <value></value>
</field>
<field name="deleteable">
  <value></value>
</field>
<field name="_dao3_link4">
  <value>[]</value>
</field>
<field name="email">
  <value>junk@example.com|HTML</value>
</field>
<field name="subject">
  <value>null</value>
</field>
<field name="_appendtext_ref_key_message">
  <value></value>
</field>
<field name="wfstate">
  <value></value>
</field>
<field name="stage">
  <value></value>
</field>
<field name="created_by">
  <value>null</value>
</field>
</record>
</linked_records>
<global_vars>
  <variable name="my_Close_Date">
    <value>2003-07-08 21:19:06.0</value>
  </variable>
  <variable name="my_Created_By">
    <value></value>
  </variable>

```

```

<variable name="my_Fax">
  <value></value>
</variable>
<variable name="my_Stop_Work">
  <value>00:03:36</value>
</variable>
<variable name="my_Group_Name">
  <value>&lt;dao_line 1&gt;Staff&lt;/dao_line&gt;&lt;dao_line 7&gt;All&lt;
/dao_line&gt;</value>
</variable>
<variable name="my_Start_Work">
  <value>00:03:36</value>
</variable>
<variable name="my_Last_Name">
  <value>Default Admin</value>
</variable>
<variable name="my_Actions_Taken">
  <value></value>
</variable>
<variable name="my_groups">
  <value>admin</value>
</variable>
<variable name="my_From">
  <value></value>
</variable>
<variable name="my_Lead_Source">
  <value></value>
</variable>
<variable name="my_Zip_Code">
  <value>12345</value>
</variable>
<variable name="my_deleteable">
  <value></value>
</variable>
<variable name="my_Object_type">
  <value>employees</value>
</variable>
<variable name="my_Pager">
  <value>|HTML</value>
</variable>
<variable name="my_Owned_By">
  <value></value>
</variable>
<variable name="my_Additional_Comments">
  <value></value>
</variable>
<variable name="my_Factors_that_hurt_sale">
  <value></value>
</variable>
<variable name="my_Message">
  <value></value>

```

```

</variable>
<variable name="my_primarygroup_id">
  <value>14</value>
</variable>
<variable name="my_CC">
  <value></value>
</variable>
<variable name="my_First_Name">
  <value>Admin</value>
</variable>
<variable name="my_primarygroup_description">
  <value></value>
</variable>
<variable name="my_primarygroup_dn">
  <value></value>
</variable>
<variable name="my_Made_Sale?">
  <value></value>
</variable>
<variable name="my_Description">
  <value>&lt;dao_line 6&gt;admin&lt;/dao_line&gt;</value>
</variable>
<variable name="my_Amount_of_sale">
  <value>0.0</value>
</variable>
<variable name="my_Date">
  <value></value>
</variable>
<variable name="my_Team_Name">
  <value>Staff</value>
</variable>
<variable name="my_Website_URL">
  <value></value>
</variable>
<variable name="my_Lead_Owner">
  <value></value>
</variable>
<variable name="my_Other_Decision_makers">
  <value></value>
</variable>
<variable name="my_primarygroup_name">
  <value>admin</value>
</variable>
<variable name="my_MasterID">
  <value></value>
</variable>
<variable name="my_Login">
  <value>admin</value>
</variable>
<variable name="my_Workflow_State">
  <value></value>

```



```

</variable>
<variable name="my_Title">
  <value>Admin</value>
</variable>
<variable name="my_Email">
  <value>junk@<ew:productsite><span class="product1">enterprisewizard.com<
/ span><span class="product2">saaswizard.com</span></ew:productsite>|HTML</value>
</variable>
<variable name="my_Home_Phone">
  <value></value>
</variable>
<variable name="my_Rating">
  <value></value>
</variable>
<variable name="my_dn">
  <value></value>
</variable>
<variable name="my_Probability_of_sale_within_1_month">
  <value>0.0</value>
</variable>
<variable name="my_Internal_group_ID">
  <value>14</value>
</variable>
<variable name="my_Attachments">
  <value></value>
</variable>
<variable name="my_Cell_Phone">
  <value></value>
</variable>
<variable name="my_teams">
  <value>all, staff</value>
</variable>
<variable name="my_Time_Zone">
  <value></value>
</variable>
<variable name="my_Full_name">
  <value>Admin Default Admin</value>
</variable>
<variable name="my_login">
  <value>admin</value>
</variable>
<variable name="my_PK_of_owner">
  <value>14</value>
</variable>
<variable name="my_Subject">
  <value></value>
</variable>
<variable name="my_Street_Address">
  <value>unknown</value>
</variable>
<variable name="my_City">

```

```

    <value>unknown</value>
</variable>
<variable name="my_Instant_Messaging_Nickname">
    <value></value>
</variable>
<variable name="my_Telephone">
    <value></value>
</variable>
<variable name="my_PK_of_creator">
    <value>14</value>
</variable>
<variable name="my_ID">
    <value></value>
</variable>
<variable name="my_name">
    <value>admin</value>
</variable>
<variable name="my_Factors_that_helped_sale">
    <value></value>
</variable>
<variable name="my_To">
    <value></value>
</variable>
<variable name="my_Lead_Creator">
    <value></value>
</variable>
<variable name="NULL">
    <value>NULL</value>
</variable>
<variable name="my_Reply_To">
    <value></value>
</variable>
<variable name="my_Company_Name">
    <value></value>
</variable>
<variable name="my_linked_field_Link_Column_Label">
    <value>0</value>
</variable>
<variable name="my_description">
    <value></value>
</variable>
<variable name="my_State">
    <value></value>
</variable>
<variable name="my_Stage">
    <value></value>
</variable>
<variable name="my_Country">
    <value>Unknown Country</value>
</variable>
<variable name="my_ParentID">

```

```

        <value></value>
    </variable>
    <variable name="my_Probability_of_sale_within_3_months">
        <value>0.0</value>
    </variable>
    <variable name="my_Email_Type">
        <value></value>
    </variable>
    <variable name="my_BCC">
        <value></value>
    </variable>
    <variable name="my_Next_Contact">
        <value></value>
    </variable>
    <variable name="my_Comments">
        <value></value>
    </variable>
</global_vars>
</input>

```

For Perl scripts there are interface modules for interaction with Agiloft provided; see full description below.

Output file example:

```

<?xml version= "1.0" encoding="UTF-8" ?>
<!DOCTYPE output[
<!ELEMENT output (exitAction?, debug?, message?, redirect?, report?,
record?) >
<!ELEMENT exitAction (#PCDATA) >
<!ELEMENT debug (#PCDATA) >
<!ELEMENT message (#PCDATA) >
<!ELEMENT redirect (#PCDATA) >
<!ELEMENT report (#PCDATA) >
<!ELEMENT record (field) +>
<!ELEMENT field (value) >
<!ATTLIST field
name CDATA #REQUIRED>
<!ELEMENT value (#PCDATA) >
]>
<output>
<exitAction>AcceptChanges</exitAction>
<debug> Something to be sent to the server logs</debug>
<message> Script accepts your data</message>
<report> Some report for the history fuctionality</report>
<record>
<field name= "assigned_to" >
<value> Umbra</value>
</field>
<field name= "summary" >
<value> some summary here</value>
</field>

```

```
</record>
</output>
```

This file describes the script result, with assigned_to field value set to "Umbra", summary set to "some summary here", changes accepted to be put to database, and "Script accepts your data" message is shown to user.

Both files always use UTF-8 encoding. Input file uses xml: id; see <http://www.w3.org/TR/2004/WD-xml-id-20041109/> for assistance with referring to the linked records. Things like project and table names and values from user record are not translated into the user's language. When converting user record into XML format Agiloft uses so-called Stringifiers from com.supportwizard.util.stringify package.

In short, for every column type there exists a stringifier that can translate the internal data representation into String and vice-versa. For instance, if there is a Choice 'priority' column that contains 'Medium' value, it's internal representation is Medium's ID and choices stringifier will convert this ID into 'Medium' string for scripts, - then some script can change 'Medium' into 'High' and stringifier will convert it back into 'High's ID. This implies that values produced by scripts must be understandable by stringifiers. See the appendix for full stringifiers info.

All of the output file fields are optional, except for <redirect> that must be present if exit code is 2 or 3; see below. If there's no <record>, then it is considered to be unmodified. <debug> only sends it's content to system log. <report> provides data for a report that will appear in the history tab. <message> and <redirect> are covered below.

The main thing that Agiloft expects from a script is its return (exit) code. There are 4 possible values:

- **0 means success.** If <message> is presented, it is shown to the user. <redirect> is ignored, all modifications from <record> take their place.
- **1 leads to user action block.** If <message> is presented, then it will be shown to the user. <record> modifications will be ignored.
- **2 will cause action block, user log-off and redirection.** <message> will be logged, and user will be redirected to <redirect>. <record> modifications will be ignored.
- **3 leads to user log off and redirection, but keeps changes.** This is similar to 2, but action won't be blocked and changes made by script will take place.

It is also possible to use <exitAction> to override the exit codes. Recognized values are:

- AcceptChanges equivalent to exit code of 0
- RejectChanges equivalent to exit code of 1
- RejectChangesAndExit equivalent to exit code of 2
- AcceptChangesAndExit equivalent to exit code of 3

If exitAction is present, it overrides the "Raw" exit code. For instance if <exitAction> is set to AcceptChanges and exit code is 1, changes will be accepted.

Scripts must reside in the scripts directory. By default it is Agiloft.installation.dir}/data/{kb.name}/scripts. When creating script action, user should specify the script filename as action name, like "myScript.pl", for example.

Perl Sample Scripts

The following scripts provide examples of using custom scripting capabilities in a typical Agiloft installation.

Note: The Agiloft distribution includes Perl and will automatically use it to execute scripts with a .pl subscript. If you want to use your own version of perl, you may do so by specifying it on the first line of the script, eg `#!/usr/local/perl`.

```
#####
# Synopsis:
#
# Redirect the user after he/she submits the ticket.
#
# There is no need to specify the Perl interpreter if the script is named *.pl
# because the system recognizes .pl scripts as being Perl based and will
# use the Perl that is included in the distribution.
#
# If you use this script with your own copy of Perl, please note that
# it requires SimpleXML module to be installed
#
#####
use strict;
use warnings;
use EWget;
use EWset;
my ($input_fname, $output_fname) = @ARGV;
EWget::load($input_fname);
#####
#
# Only log the user out if he/she is not a member of the Staff
# group. In other words, we want all members of staff to be able
# to log tickets on behalf of users without being logged out.
#####
my $groups = EWget::getGlobalVariable('my_groups');
my $groupAllowedToClose = "Staff";
if ($groups !~ m/(\^|,)$groupAllowedToClose(,|$)/){
    EWset::setRedirect("http://www.example.com");
    EWset::exit_ew("AcceptChangesAndExit", $output_fname);
} else {
    EWset::exit_ew("AcceptChanges", $output_fname);
}
```

```
#####
# Synopsis:
#
# Redirect the user after he/she submits the ticket.
#
# There is no need to specify the Perl interpreter if the script is named *.pl
# because the system recognizes .pl scripts as being Perl based and will
# use the Perl that is included in the distribution.
```

```
#
# If you use this script with your own copy of Perl, please note that
# it requires SimpleXML module to be installed
#
#####
use strict;
use warnings;
use EWget;
use EWset;
my ($input_fname, $output_fname) = @ARGV;
EWget::load($input_fname);
#####
#
# Only log the user out if he/she is not a member of the Staff
# group. In other words, we want all members of staff to be able
# to log tickets on behalf of users without being logged out.
#####
my $groups = EWget::getGlobalVariable('my_groups');
my $groupAllowedToClose = "Staff";
if ($groups !~ m/(\^|,)$groupAllowedToClose(,|$)/){
    EWset::setRedirect("http://www.example.com");
    EWset::exit_ew("AcceptChangesAndExit", $output_fname);
} else {
    EWset::exit_ew("AcceptChanges", $output_fname);
}

#####
# Synopsis:
# Check if the user belongs to QA group and block closure of the ticket if
# he/she doesn't.
#
# There is no need to specify the Perl interpreter if the script
# is named *.pl because the system recognizes .pl
# scripts as being Perl based and will
# use the Perl that is included in the distribution.
#
# If you use this script with your own copy of Perl, please note that
# it requires SimpleXML module to be installed
#####
use strict;
use warnings;
use EWget;
use EWset;
my ($input_fname, $output_fname) = @ARGV;
EWget::load($input_fname);
my $groups = EWget::getGlobalVariable('my_groups');
my $groupAllowedToClose = "QA";
if ($groups !~ m/(\^|,)$groupAllowedToClose(,|$)/
    && EWget::getValue('current_state', 'wfstate') eq "Closed"
    && EWget::getValue('old_state', 'wfstate') ne "Closed"){
```

```

        EWset::setMessage("You don't belong to group, that can close tickets");
        # Set a field value. In this case, we set the status field
        # to its old value
        EWset::setRecordField('wfstate', EWget::getValue('old_state', 'wfstate'));
        EWset::save($output_fname);
    }
    EWset::exit_ew("AcceptChanges", $output_fname);

```

```

#####
# Synopsis:
# This script performs some common tasks typical of a script.
# You could run this script against the input file example:
# filename.pl input.xml output.xml
#
# There is no need to specify the Perl interpreter if the script
# is named *.pl because the system recognizes .pl
# scripts as being Perl based and will
# use the Perl that is included in the distribution.
#
# If you use this script with your own copy of Perl, please note that
# it requires SimpleXML module to be installed
#
#####
use strict;
use warnings;
# interface modules to interact with input and output xml files
use EWget;
use EWset;
# Retrieve files names from arguments...
my ($input_fname, $output_fname) = @ARGV;
# and load input into internal structures. This is required step before
# accessing input by API calls.
EWget::load($input_fname);
# we will use this to accumulate messages to user
my $message = "";
# Let's retrieve user name to provide personalized message.
# That would be a value of Full Name from
# Contacts table entry for user, editing the ticket.
my $full_name = EWget::getGlobalVariable('my_full_name');
# Let's check if user is a creator of record and prohibit
# the editing if he is not, with respective message.
# The same could be done with permissions, but access
# control through scripts can be more sophisticated.
# For example, you could use EWget::getGlobalVariable('my_start_work')
# and EWget::getGlobalVariable('my_stop_work')
# (if there are such fields in Contacts table) to allow
# modifications only in working time.
open (output_file, ">/tmp/aaaa");
print output_file EWget::getGlobalVariable('my_full_name') . "\n";
print output_file EWget::getValue('current_state', 'created_by') . "\n";

```

```

close (output_file);
if(EWget::getGlobalVariable('my_login')
    ne EWget::getValue('current_state', 'login')){
# We compare here login from global vars that is login of user, who
# edits the record with
# record creator login. Now let's show the message to describe
# the denial reason.
    EWset::setMessage("Sorry, ".$full_name.
        ", you are not the creator of this record, you cannot modify it.");
    EWset::setExitAction("RejectChanges");
# and set exit code to reject the changes
} else {
    # Now let's try to detect some changes in the record.
    # For example, let's make deleteable flag
    # non-editable without blocking other changes.
    if(EWget::getValue('current_state', 'deleteable')
        ne EWget::getValue('old_state', 'deleteable')){
        # We will notify user, that change is not allowed.
        $message="Sorry, ".$full_name.
            ", you are cannot modify deleteable flag".
            ", your change will be ignored. ";
        # Let's return the old value back.
        EWset::setRecordField('deleteable',
            EWget::getValue('old_state', 'deleteable'));
    }
    # Now some more complex techniques - linked
    # records access.
    # Let's find out, for example, the name and
    # phone of record owner
    # if case is "open" ("owned by" is a Linked
    # Field relationship with the contacts table)
    if("Open" eq EWget::getValue('current_state', 'wfstate')){
        # Retreive phone and name from first (index=0) record
        # linked to owned_by field.
        my $owner_phone = EWget::getLinkedValue("current_state",
            "end_user_name", 0, "direct_phone");
        my $owner_name = EWget::getLinkedValue("current_state",
            "end_user_name", 0, "full_name");
        $message = $message."You can contact ".$owner_name.
            ", owner of this support case by number ".$owner_phone." ";
    }
    # Now check if this case assigned to user or team - example of
    # logic over multiple tables Linked Field
    my @linkedRecordNames = EWget::getLinkedRecordsNamesList("current_state",
        "assigned_team");
    # In fact, there is always 0 or 1 element in this array,
    # since multiple table Linked Fields are not exported
    if(EWget::getTableNameForRecord($linkedRecordNames[0]) eq "teams"){
        $message = $message."This issue assigned to ".
            EWget::getLinkedValue("current_state", "assigned_team", 0, "_name").
            " team.";
    }
}

```



```

    }
    # store full message
    EWset::setMessage($message);
    # accept changes to ticket
    EWset::setExitAction("AcceptChanges");
}
# And don't forget to save the output file.
EWset::save($output_fname);

```

Java Scripts

Scripts can also be written in Java, which allows tight integration with Agiloft and direct access to all user data. Note this page refers to scripts written in Java, not JavaScript.

Java scripts are a flexible and performance-effective way to implement your custom logic. You can retrieve only the necessary data and also skip export to xml, running external program and the import results steps. The result is a significant performance boost in the case of complex tables structure with a lot of linked fields.

The script class must implement `com.supportwizard.actions2.interfaces.ExternalScript`.

This interface has only one method: `ScriptOutput runScript (final ScriptInput input)` throws `ActionException`;

Please consult [Javadoc documentation](#) for more details.

Input Data

A `ScriptInput` instance provides input data: project and user table ID 's, user's Seance and the record data.

The record data is passed in a map form - an instance of `com.supportwizard.dml.SWDataMap` class. Specific field values can be accessed by name, as visible in GUI in **Setup > Tables > [Select Table] > Edit > Field**. The script gets one or two instances of the data map - the "old" one and the "new" one. At least one will be present, depending on when the script is invoked - create - only "new", delete - only "old", modify - both.

This is the mapping of specific data types returned for different field types:

- Auto-Increment => class `java.lang.Long`
- Elapsed Time => class `java.lang.Long`
- DAO3 link field => class `com.supportwizard.functionalities.dao3.util.SWDao3LinkHolder`
- Long integer field => class `java.lang.Long`
- Billing field => class `com.supportwizard.functionalities.dao3.util.SWDao3LinkHolder`
- Integer => class `java.lang.Integer`
- Choice => class `com.supportwizard.dictionary.SWChoiceLine`
- Email => class `java.lang.String`
- Email Pager => class `java.lang.String`
- Telephone/Fax => class `java.lang.String`
- Multi-Choice => class `com.supportwizard.dictionary.MultichoiceLines`
- Short Text => class `java.lang.String`
- Password => class `java.lang.String` - however it comes in ***** form for security reasons and is generally useless to a script writer
- Text => class `java.lang.String`
- URL => class `java.lang.String`
- File => class `com.supportwizard.functionalities.blob.SWBlobRefHolder`
- Image => class `com.supportwizard.functionalities.blob.SWBlobRefHolder`

- History => class java.lang.String
- DAO3 multiple field => class com.supportwizard.functionalities.dao3.util.SWDao3MultiValue
- WMI Field => class java.lang.String
- Append Only Text => class com.supportwizard.dictionary.appendtext.AppendOnlyTextContainer
- Floating Point => class java.lang.Double
- Percentage => class java.lang.Double
- Currency => class java.lang.Double
- Date/Time => class java.sql.Timestamp
- Date => class java.sql.Timestamp
- Time => class java.sql.Time
- Compound => class java.lang.String
- Calculation on Multiple Linked Records => class java.lang.Double

A linked field that imports N field from the source table into the target one will be represented by N+1 entries in the "input" Map on the data level.

If a single record is imported each of the N entries corresponding to N fields will have a single value of corresponding type. If a multiple records are imported, each of the N entries corresponding to the N fields will have an instance of SWDao3MultiValue, a list of String values, one for each imported record, with "*_{NULL}_*" value marking the null one. The sequence of values in different multi-value fields is the same for the same set of imported records.

The extra entry mentioned above will be under the name that can be seen in the GUI in **Setup > Table > <yourtable> > Edit > Fields > <yourLF> > Edit > Fields** at the bottom "Linked Column Name". It will contain an instance of the SWDao3LinkHolder - essentially an array of identifiers for the records in the source table(s).

To access the n-th identifier:

```
List<SWDao3LinkHolder.Link> links = linkHolder.getLinks();
SWDao2LinkHolder.Link link = links.get();
Long pk = link.getLinkPK();
```

A script call always starts with the input object and is being called for a single record only. A script however can use the CRUD+S (select) API underlying WebServices and REST access - SimpleAPI.

```
Context jndiContext = new InitialContext();
EWSimpleAPILocalHome localHome = (EWSimpleAPILocalHome)
    jndiContext.lookup("ew/EWSimpleAPI");
EWSimpleAPI EWSimpleAPI = localHome.create();
long[] ids = EWSimpleAPI.EWSelectFromTable("allocation",
    "general_issue=" + general_issue +
    " and specific_issue=" + specific_issue +
    " and default_team = " + line.getId(), seance);
if (ids.length == 0) {
    return blockedScriptOutput(output,
        "There is no default team defined for this combination of Issue Types.");
}
```

For more details on SimpleAPI and other Web Services APIs, please consult the following Javadoc: [WS_API_Javadoc.zip](#). Unzip the file, open index.html, and navigate to EWSimpleAPI on the left pane. Please contact Agiloft Support if you need assistance.

Output data

A ScriptOutput instance describes script output, an instance of this class is intended to be created with the public ScriptOutput - final ScriptInput input - constructor. ScriptOutput can optionally contain modified user record data - don't change ScriptInput.getRecord() instance.

Note: Do not copy any values your script leaves unchanged from the input record to output. Doing so is unnecessary and could have unexpected results if you are filling linked fields with multiple columns.

Modifications will take place if ScriptOutput.getExitCode() is ExternalScript.SUCCESS_CODE. If exit code is equal to BLOCKED_CODE, then the user action will be blocked and the user will get ScriptOutput.getMessage() as an error message; this message will also be shown in case of ACCEPT_CODE. In cases where the user action is undefined - see timer based rules above - nothing will be blocked and the error message will only be logged.

BLOCK_REDIRECT_CODE will also block user action and will force the current user to log-off. The user will be redirected to ScriptOutput.getRedirect(). ACCEPT_REDIRECT_CODE will logoff the current user, but won't block an action and script changes will take place.

Unhandled exceptions propagate to Agiloft where they are caught, logged and displayed to the user depending on the way the script was run. This display is often too verbose and is not really suited for ordinary users.

Thus the recommended way of handling the exceptions is either suppress them in the script itself and instead return a ScriptOutput instance with a blocking exit code and an optional message or wrap in action-related exceptions.

```
private ScriptOutput blockedScriptOutput(ScriptOutput output, String s,
    SWDataMap newRecord, Seance seance) {
    output.setExitCode(ExternalScript.BLOCK);
    output.setMessage(s + " ID:" + newRecord.getSWRecordPK(seance).getID());
    return output;
}
```

When wrapped exceptions can be used to change control flow. A plain ActionException means some system-level error and leads to transaction rollback. Throwing a ActionBlockedException String message amounts to returning BLOCKED_CODE with ScriptOutput.getMessage() being 'message'.

Throwing ActionBlockedException - String message, String redirectURL - will do the same thing as returning BLOCK_REDIRECT_CODE.

General considerations

The call to the script is done in a synchronous manner. The calling code waits. There is a potential to the encompassing business transaction to time out and be rolled back by Agiloft's application server if it takes too long. If this is a frequent case, consider implementing asynchronous mode of operation on the caller side.

A new classloader is created on each script run as well as the script class instance. It is not isolated as far as it delegates to parent classloader in java 2 compliant manner. The script doesn't have to be thread safe since each call is being served by a different instance.

At this moment if a script relies on some 3rd party libraries not available in the Agiloft environment, these have to be repackaged into the script archive.

The Agiloft .jar library files and their locations depend on your Agiloft version:

	2019_01 or later	2018_02 or prior
Directory	{Agiloft.installation.dir}/wildfly/modules/system/layers/base/com/agiloft/main/lib	{Agiloft.installation.dir}/jboss/server/sw/lib/sw/ and {Agiloft.installation.dir}/jboss/server/sw/deploy
Main Library Files	agiloft-core.jar	- SWFunctionalities.jar - SW2Interfaces.jar - SWSeance-ejb.jar
Additional Library Files	- commons*.jar - httpclient*.jar - xml*.jar	- commons*.jar - httpclient*.jar - xml*.jar

The Agiloft jar files you may need while developing are in { Agiloft.installation.dir}/wildfly/modules/system/layers/base/com/agiloft/main/lib. In older versions of Agiloft, you can find them in { Agiloft.installation.dir}/jboss/server/sw/lib/sw and { Agiloft.installation.dir}/jboss/server/sw/deploy.

Normally, the following classes are enough: SWFunctionalities.jar, SWSeance-ejb.jar. The jars that make the JBoss environment would be under { Agiloft.installation.dir}/jboss/lib, { Agiloft.installation.dir}/jboss/server/lib.

Deployment considerations

Scripts must reside in the scripts directory. By default it is { Agiloft.installation.dir}/data/{kb.name}/scripts.

If your script consists of a single class, then you can simply drop the .class file in the scripts directory. This .class must not belong to any package. If you have several classes, then you must put them in a jar file with a special key in it's Manifest file:

Script-Class: name of the class implementing ExternalScript.

For example: com.mycompany.test.TestExternalScript.

Now your script name will be the name of the script .jar or .class file.

A Java custom script is run by Agiloft within the same JVM. All libraries present in Agiloft application server instance classpath - { Agiloft.installation.dir}/wildfly/modules/system/layers/base/com/agiloft/main/lib, or in older versions, { Agiloft.installation.dir}/jboss/server/sw/lib/, { Agiloft.installation.dir}/jboss/server/sw/lib/sw/, and { Agiloft.installation.dir}/jboss/lib - are available for the script to use. If you add a new library to run a custom Java script, add a reference to the new library in the module.xml file in the /wildfly/modules/system/layers/base/com/agiloft/main directory.

Agiloft runs on Java 6. As for JAXWS - the libraries available to the script will be those distributed with the application server rather than those available from the runtime environment.

Sample Java Scripts

The following scripts provide examples of using custom Java scripting capabilities in a typical Agiloft installation.

```
package com.enterprisewizard.examples;
/*
 * (c) Copyright 2007, <ac:structured-macro ac:name="companyname" ac:schema-
version="1" />, Inc. All Rights Reserved.
 * No part of this document may be stored in a retrievable system,
 * transmitted or reproduced in any way without the prior written
 * permission of EnterpriseWizard.
 */
import com.supportwizard.actions2.interfaces.ExternalScript;
import com.supportwizard.actions2.interfaces.ScriptActionException;
import com.supportwizard.actions2.interfaces.ScriptInput;
import com.supportwizard.actions2.interfaces.ScriptOutput;
import com.supportwizard.dictionary.SWChoiceLine;
import com.supportwizard.dictionary.appendtext.AppendOnlyTextContainer;
import com.supportwizard.dml.SWDataMap;
import org.apache.commons.httpclient.HttpClient;
import org.apache.commons.httpclient.HttpStatus;
import org.apache.commons.httpclient.MultiThreadedHttpConnectionManager;
import org.apache.commons.httpclient.methods.PostMethod;
import java.io.IOException;
import java.io.InputStream;
/**
 * The purpose of the "ReplicateTicket" is to automatically replicate
 * Tickets from one server to another via REST call.
 *
 * This script is run in a script action from a rule invoked on create.
 */
public class ReplicateTicket implements ExternalScript {
    private final static HttpClient httpClient;
    static {
        MultiThreadedHttpConnectionManager connectionManager =
            new MultiThreadedHttpConnectionManager();
        httpClient = new HttpClient(connectionManager);
    }
    public ScriptOutput runScript(ScriptInput input)
        throws ScriptActionException {
        //newRecord represents the Ticket that just have been created.
        final SWDataMap newRecord = input.getNewRecord();
        //PostMethod is used to invoke REST EWCreate operation
        //on the target server
        PostMethod method =
            new PostMethod("http://server2.supportwizard.com:8080" +
                "/ewws/EWCreate?$KB=Support&$login=script&" +
                "$password=example&$lang=en&$table=case");
        String additionalNotes = "";
```

```

//Script uses newRecord to retrieve new Ticket's data
//to construct POST request.
final Object o = newRecord.get("engineering_comments");
if (o != null && o instanceof AppendOnlyTextContainer) {
    additionalNotes = o.toString();
}
addParam("Additional_Notes", additionalNotes, method);
addParam("Build_Number",
    (String) newRecord.get("build_number"), method);
final Object closingType = newRecord.get("closing_type");
if (closingType instanceof SWChoiceLine) {
    final String text = ((SWChoiceLine) closingType).getText();
    addParam("Closing_Type", text, method);
}
addParam("submitter_login",
    (String) newRecord.get("submitter_login"), method);
addParam("Company_0",
    (String) newRecord.get("submitter_company"), method);
addParam("customer_phone", (String) newRecord.get("phone"), method);
addParam("customer_email", (String) newRecord.get("email"), method);
addParam("end_user_name", (String) newRecord.get("f_name"), method);
addParam("Fixed_In_Build_No",
    (String) newRecord.get("fixed_in_build_no"), method);
addParam("For_Product", "EnterpriseWizard", method);
addParam("Old_Ew_Id", getString(newRecord, "id"), method);
addParam("Primary_Team",
    (String) newRecord.get("creator_team"), method);
final Object priority = newRecord.get("priority");
if (priority instanceof SWChoiceLine) {
    final String text = ((SWChoiceLine) priority).getText();
    addParam("Priority", text, method);
} else {
    addParam("Priority", "Medium", method);
}
addParam("Problem_Description",
    (String) newRecord.get("steps_to_duplicate"), method);
final Object published = newRecord.get("published");
if (published instanceof SWChoiceLine) {
    final String text = ((SWChoiceLine) published).getText();
    addParam("Published", text, method);
}
addParam("Solution", (String) newRecord.get("resolution"), method);
final Object server = newRecord.get("server");
String serverTxt = null;
if (server instanceof SWChoiceLine) {
    serverTxt = ((SWChoiceLine) server).getText();
}
String loginTxt = "server: " + serverTxt + "\n";
loginTxt += "knowledgebase: " + newRecord.get("knowledgebase_name")
    + "\n";
loginTxt += "login: " + newRecord.get("admin_group_login") + "\n";

```



```

loginTxt += "password: " + newRecord.get("admin_password");
addParam("Staff_Notes", loginTxt, method);
final Object standardSolution = newRecord.get("standardSolution");
if (standardSolution instanceof SWChoiceLine) {
    final String text = ((SWChoiceLine) standardSolution).getText();
    addParam("Standard_Solution", text, method);
}
addParam("Summary", (String) newRecord.get("summary"), method);
addParam("Type", "case", method);
final Object category = newRecord.get("category");
if (category instanceof SWChoiceLine) {
    final String text = ((SWChoiceLine) category).getText();
    addParam("Type_Of_Issue", text, method);
}
final Object wfstate = newRecord.get("wfstate");
if (wfstate instanceof SWChoiceLine) {
    final String text = ((SWChoiceLine) wfstate).getText();
    addParam("Wfstate", text, method);
} else {
    addParam("Wfstate", "Open", method);
}
//Once all parameters of the request have been filled,
//script uses standard httpClient.executeMethod(method)
//to send the request.
try {
    method.setHttp11(true);
    int i = httpClient.executeMethod(method);
    if (i != HttpStatus.SC_OK) {
        throw new RuntimeException("Error is reported via " +
            "REST interface. Return code: " + i);
    }
    final InputStream asStream = method.getResponseBodyAsStream();
    byte b[] = new byte[1024];
    try {
        while (asStream.read(b) != -1) {
        }
    } finally {
        asStream.close();
    }
} catch (IOException e) {
    // This script doesn't make any effort to recover from Exceptions.
    // Exceptions are re-thrown, thus original and
    // child records won't be created.
    throw new RuntimeException("IOException is reported " +
        "in operation via REST interface.");
} finally {
    method.releaseConnection();
}
//If POST was successfull (HttpStatus.SC_OK),
//script finishes its work by creating a ScriptOutput
//object with normal exit code.

```

```

        final ScriptOutput output = input.createOutput();
        output.setExitCode(ExternalScript.ACCEPT);
        return output;
    }
    private void addParam(String s, String optionOpen, PostMethod req) {
        if (optionOpen != null) {
            req.addParameter(s.toLowerCase(), optionOpen);
        }
    }
    private String getString(SWDataMap newRecord, final String name) {
        final Object value = newRecord.get(name);
        return value == null ? null : value.toString();
    }
}

```

The purpose of the ReplicateTicket is to automatically replicate Tickets from one server to another via a REST call.

```

package com.enterprisewizard.examples;
/*
 * (c) Copyright 2007, EnterpriseWizard, Inc. All Rights Reserved.
 * No part of this document may be stored in a retrievable system,
 * transmitted or reproduced in any way without the prior written
 * permission of EnterpriseWizard.
 */

import com.supportwizard.actions2.interfaces.ExternalScript;
import com.supportwizard.actions2.interfaces.ScriptActionException;
import com.supportwizard.actions2.interfaces.ScriptInput;
import com.supportwizard.actions2.interfaces.ScriptOutput;
import com.supportwizard.datetime.convertor.EwDateTimeConvertor;
import com.supportwizard.dictionary.SWChoiceLine;
import com.supportwizard.dictionary.appendtext.AppendOnlyTextContainer;
import com.supportwizard.dml.SWDataMap;
import org.apache.commons.httpclient.HostConfiguration;
import org.apache.commons.httpclient.HttpClient;
import org.apache.commons.httpclient.HttpStatus;
import org.apache.commons.httpclient.MultiThreadedHttpConnectionManager;
import org.apache.commons.httpclient.methods.PostMethod;

import java.io.IOException;
import java.io.InputStream;
import java.util.Date;
import java.util.Locale;

/**
 * The purpose of the "CreateInEWSupportScript" is to automatically
 * replicate Tickets from one server to another.
 *
 * This script is run in a script action from the rule invoked on create.
 */

```

```

*/
public class CreateInEWSupportScript implements ExternalScript {
    private final static HttpClient httpClient;

    static {
        MultiThreadedHttpConnectionManager connectionManager =
            new MultiThreadedHttpConnectionManager();
        httpClient = new HttpClient(connectionManager);
    }

    public ScriptOutput runScript(ScriptInput input)
        throws ScriptActionException {
        final String EMPTY_STRING = "";
        //newRecord represents the Ticket that just have been created.
        final SWDataMap newRecord = input.getNewRecord();

        //PostMethod is used to invoke REST EWCreate operation
        // on the target server
        PostMethod method = new PostMethod(
            "http://server2.supportwizard.com:" +
                "8080/ewws/EWCreate?$KB=Support&" +
                "$login=script&$password=example&" +
                "$lang=en&$table=case");

        String additionalNotes = EMPTY_STRING;

        //Script uses newRecord to retrieve new Ticket's data
        // in order to construct POST request.
        final Object o = newRecord.get("engineering_comments");
        if (o != null && o instanceof AppendOnlyTextContainer) {
            additionalNotes = o.toString();
        }

        addParam("Additional_Notes", additionalNotes, method);
        addParam("Build_Number", (String) newRecord.get("build_number"),
            method);

        final Object closingType = newRecord.get("closing_type");
        if (closingType instanceof SWChoiceLine) {
            final String text = ((SWChoiceLine) closingType).getText();
            addParam("Closing_Type", text, method);
        }

        addParam("submitter_login",
            (String) newRecord.get("submitter_login"), method);
        addParam("Company_0", (String) newRecord.get("submitter_company"),
            method);
        addParam("customer_phone", (String) newRecord.get("phone"),
            method);
    }

```

```

addParam("customer_email", (String) newRecord.get("email"),
        method);
addParam("end_user_name", (String) newRecord.get("f_name"),
        method);
addParam("Fixed_In_Build_No",
        (String) newRecord.get("fixed_in_build_no"),
        method);
addParam("For_Product", "EnterpriseWizard", method);
addParam("Old_Ew_Id", getString(newRecord, "id"), method);
addParam("Primary_Team", (String) newRecord.get("creator_team"),
        method);

final Object priority = newRecord.get("priority");
if (priority instanceof SWChoiceLine) {
    final String text = ((SWChoiceLine) priority).getText();
    addParam("Priority", text, method);
} else {
    addParam("Priority", "Medium", method);
}
addParam("Problem_Description",
        (String) newRecord.get("steps_to_duplicate"),
        method);

final Object published = newRecord.get("published");

if (published instanceof SWChoiceLine) {
    final String text = ((SWChoiceLine) published).getText();
    addParam("Published", text, method);
}

addParam("Solution", (String) newRecord.get("resolution"), method);
final Object server = newRecord.get("server");
String serverTxt = null;
if (server instanceof SWChoiceLine) {
    serverTxt = ((SWChoiceLine) server).getText();
}
String loginTxt = "server: " + serverTxt + "\n";
loginTxt += "knowledgebase: " +
        newRecord.get("knowledgebase_name") + "\n";
loginTxt += "login: " + newRecord.get("admin_group_login") + "\n";
loginTxt += "password: " + newRecord.get("admin_password");
addParam("Staff_Notes", loginTxt, method);

final Object standardSolution = newRecord.get("standardSolution");
if (standardSolution instanceof SWChoiceLine) {
    final String text =
        ((SWChoiceLine) standardSolution).getText();
    addParam("Standard_Solution", text, method);
}

addParam("Summary", (String) newRecord.get("summary"), method);

```

```

addParam("Type", "case", method);

final Object category = newRecord.get("category");
if (category instanceof SWChoiceLine) {
    final String text = ((SWChoiceLine) category).getText();
    addParam("Type_Of_Issue", text, method);
}

final Object wfstate = newRecord.get("wfstate");
if (wfstate instanceof SWChoiceLine) {
    final String text = ((SWChoiceLine) wfstate).getText();
    addParam("Wfstate", text, method);
} else {
    addParam("Wfstate", "Open", method);
}

//Once all parameters of the request have been filled,
// script uses standard httpClient.executeMethod(method)
// to send the request.
try {
    method.setHttp11(true);
    int i = httpClient.executeMethod(method);
    if (i != HttpStatus.SC_OK) {
        throw new RuntimeException(
            "Error is reported via REST interface. " +
            "Return code: " + i);
    }
    final InputStream asStream = method.getResponseBodyAsStream();
    byte b[] = new byte[1024];
    try {
        while (asStream.read(b) != -1) {
        }
    } finally {
        asStream.close();
    }
} catch (IOException e) {
    // This script doesn't make any effort to recover from Exceptions.
    // Exceptions are re-thrown, thus original and
    // child records won't be created.
    throw new RuntimeException(
        "IOException is reported in " +
        "operation via REST interface.");
} finally {
    method.releaseConnection();
}

//If POST was successfull (HttpStatus.SC_OK),
// script finishes its work by
// creating a ScriptOutput object with normal exit code.
final ScriptOutput output = input.createOutput();

```

```

        output.setExitCode(ExternalScript.ACCEPT);
        return output;
    }

    private void addParam(String s, String optionOpen, PostMethod req) {
        if (optionOpen != null) {
            req.addParameter(s.toLowerCase(), optionOpen);
        }
    }

    private String getString(SWDataMap newRecord, final String name) {
        final Object value = newRecord.get(name);
        return value == null ? null : value.toString();
    }
}

```

The purpose of the CreateInEWSupportScript is to automatically replicate Tickets from one server to another.

```

package com.enterprisewizard.examples;
/*
 * (c) Copyright 2008, EnterpriseWizard, Inc. All Rights Reserved.
 * No part of this document may be stored in a retrievable system,
 * transmitted or reproduced in any way without the prior written
 * permission of EnterpriseWizard.
 */
import com.enterprisewizard.ws.client.demo.*;
import java.net.URL;
/**
 * The purpose of the "QueryAsset" is to demonstrate the current way of
 * working with EnterpriseWizard SOAP Web Services.
 *
 * In particular the interface is generic - exposing and accepting parent
 * object to allow one to use the same web service with different tables.
 * Methods are low level - one has to combine several calls to achieve
 * required. This is also a stateful version of the interface that relies
 * on the underlying protocol support for stateful communication via
 * cookies.
 *
 * This script is run from the commandline, accepts host, port, kbName,
 * user, password and searchValue parameters.
 *
 * The script performs search by SQL on Asset_Table object of the
 * knowledgebase, parses the returned object tree and prints results to the
 * standard output.
 */
public class QueryAsset {
    public static long[] queryAssets(
        final CondeSoapBindingStub ewServiceAPI,

```

```

        final String ipAddress) throws Exception {
// Runs SQL query and returns identifiers of records found
// Never returns null
final long[] ids = ewServiceAPI.EWSelectFromTable("asset_table",
        "ip_address_snmp = '" + ipAddress + "'");
if (ids.length != 1)
    throw new RuntimeException("unexpected result");
// Read the data by identifier
final WSAsset_Table asset =
    (WSAsset_Table) ewServiceAPI.EWRead("asset_table", ids[0]);
// Building, Floor and Location are parts of the Linked Fields set
final WSAsset_TableLocation_Dao3_Link11 location =
    asset.getDAO_Dao3_Link11();
System.out.println("Building = " + location.getNew_Building());
System.out.println("Floor = " + location.getFloor0());
System.out.println("Location = " + location.getNew_City());
// Asset (Device) type is a part of another Linked Fields set
final WSAsset_TableAsset_Definition_Dao3_Link5 assetDefinition =
    asset.getDAO_Dao3_Link5();
System.out.println(
    "Device Type = " + assetDefinition.getAsset_Type());
// Busines Unit is a part of another Linked Field set
final WSAsset_TableProject_Dao3_Link10 build =
    asset.getDAO_Dao3_Link10();
System.out.println("BU = " + build.getBuild_Business_Unit());
// Primary contact is a part of a Linked Field set as well
final WSAsset_TableContacts_Dao3_Link9 escalationContact =
    asset.getDAO_Dao3_Link9();
System.out.println("PrimaryContact = " +
    escalationContact.getEscalation_Contact());
// The following fields are available in the Asset_Table directly
System.out.println(
    "Manufacturer = " + assetDefinition.getManufacturer());
System.out.println("Service = " + asset.getService());
System.out.println("Function = " + asset.getF_Function());
System.out.println("Maintenance = " + asset.getWfstate());
return ids;
}
private static CondeSoapBindingStub getSOAPBinding(String host,
        String kb,
        final String port)

    throws Exception {
// in order to gain access to non-standard maintain
// session parameters one has to upcast the returned
// object to the WS-stack specific SOAP Stub type.
final CondeSoapBindingStub ewServiceAPI =
    (CondeSoapBindingStub) new EWServiceAPIServiceLocator()
        .getconde(new URL("http://" + host + ":" + port +
            "/" + kb + "/EWServiceAPI"));
ewServiceAPI.setMaintainSession(
    true); // enables HTTP session maintainance via cookies

```

```

        return ewServiceAPI;
    }
    public static void main(String[] args) throws Exception {
        final String host = args[0];
        final String port = args[1];
        final String kbName = args[2];
        final String user = args[3];
        final String password = args[4];
        final String searchValue = args[5];
        final String lang = "en";
        final CondeSoapBindingStub stub =
            getSOAPBinding(host, kbName, port);
        stub.EWLogin(kbName, user, password, lang); // We need to log in
        try {
            queryAssets(stub, searchValue);
        } finally {
            stub.EWLogout(); // And logout
        }
    }
}

```

The purpose of the QueryAsset is to demonstrate the current way of working with Agiloft SOAP Web Services. In particular the interface is generic - exposing and accepting parent object to allow one to use the same web service with different tables. Methods are low level - one has to combine several calls to achieve required. This is also a stateful version of the interface that relies on the underlying protocol support for stateful communication via cookies. This script is run from the commandline, accepts host, port, kbName, user, password and searchValue parameters. The script performs search by SQL on Asset_Table object of the knowledgebase, parses the returned object tree and prints results to the standard output.

```

package com.enterprisewizard.examples;
/*
 * (c) Copyright 2008, EnterpriseWizard, Inc. All Rights Reserved.
 * No part of this document may be stored in a retrievable system,
 * transmitted or reproduced in any way without the prior written
 * permission of EnterpriseWizard.
 */
import com.enterprisewizard.ws.client.demo.*;
import com.enterprisewizard.ws.client.ewsupport.WSAsset_Table;
import com.enterprisewizard.ws.api.EWServiceAPI;
import java.net.URL;
/**
 * The purpose of the "QueryAssetNew" is to demonstrate the upcoming changes
 * in EnterpriseWizard Web Services interfaces, in particular "Single Method
 * Operations", "Immediate Read", "Web Service Per Table" concepts.
 *
 * This script is run from the commandline, accepts host, port, kbName,
 * user, password and searchValue parameters.
 *
 * The script performs search by SQL on Asset_Table object of the
 * knowledgebase, parses the returned object tree and prints results to the
 * standard output.
 */

```



```

*/
public class QueryAssetNew {
    public static void queryAssets(final EWSERVICEAPI ewServiceAPI,
                                   final String ipAddress,
                                   final String kbName, final String user,
                                   final String password,
                                   final String lang) throws Exception {
        // A Single Method variant of EWSelect
        // Immediate Read variant of EWSelect
        // Never returns null
        final WSAsset_Table asset = ewServiceAPI.EWSelectAndRead(kbName,
                                                                    user, password, lang, "asset_table",
                                                                    "ip_address_snmp = '" + ipAddress + "' limit 1");
        // Instead one has to check for the id being not null.
        // Most likely will change in the final version of the EW WS SOAP API.
        if (asset.getId() != null) {
            // Building, Floor and Location are parts of the Linked Fields set
            final WSAsset_TableLocation_Dao3_Link11 location =
                asset.getDAO_Dao3_Link11();
            System.out.println("Building = " + location.getNew_Building());
            System.out.println("Floor = " + location.getFloor0());
            System.out.println("Location = " + location.getNew_City());
            // Asset (Device) type is a part of another Linked Fields set
            final WSAsset_TableAsset_Definition_Dao3_Link5 assetDefinition =
                asset.getDAO_Dao3_Link5();
            System.out.println(
                "Device Type = " + assetDefinition.getAsset_Type());
            // Business Unit is a part of another Linked Field set
            final WSAsset_TableProject_Dao3_Link10 build =
                asset.getDAO_Dao3_Link10();
            System.out.println("BU = " + build.getBuild_Business_Unit());
            // Primary contact is a part of a Linked Field set as well
            final WSAsset_TableContacts_Dao3_Link9 escalationContact =
                asset.getDAO_Dao3_Link9();
            System.out.println("PrimaryContact = " +
                               escalationContact.getEscalation_Contact());
            // The following fields are available in the Asset_Table directly
            System.out.println(
                "Manufacturer = " + assetDefinition.getManufacturer());
            System.out.println("Service = " + asset.getService());
            System.out.println("Function = " + asset.getF_Function());
            System.out.println("Maintenance = " + asset.getWfstate());
        }
    }
    private static EWSERVICEAPI getSOAPBinding(String host, String kb,
                                                  final String port)
        throws Exception {
        // The web service used is specific for Asset_Table
        return new EWSERVICEAPIServiceLocator()
            .getconde(new URL(
                "http://" + host + ":" + port + "/" + kb +

```

```

        "/EWServiceAPI.Asset_Table"));
    }
    public static void main(String[] args) throws Exception {
        final String host = args[0];
        final String port = args[1];
        final String kbName = args[2];
        final String user = args[3];
        final String password = args[4];
        final String searchValue = args[5];
        final String lang = "en";
        final EWServiceAPI stub = getSOAPBinding(host, kbName, port);
        queryAssets(stub, searchValue, kbName, user, password, lang);
    }
}

```

The purpose of the QueryAssetNew is to demonstrate the upcoming changes in EnterpriseWizard Web Services interfaces, in particular "Single Method Operations", "Immediate Read", "Web Service Per Table" concepts.

```

package com.enterprisewizard.examples;
/*
 * (c) Copyright 2008, EnterpriseWizard, Inc. All Rights Reserved.
 * No part of this document may be stored in a retrievable system,
 * transmitted or reproduced in any way without the prior written
 * permission of EnterpriseWizard.
 */
import com.enterprisewizard.ws.client.demo.*;
import java.net.URL;
/**
 * The purpose of the "CreateCaseCurrent" is to demonstrate the current way
 * of working with EnterpriseWizard SOAP Web
 * Services.
 *
 * In particular the interface is generic - exposing and accepting parent
 * object to allow one to use the same web service with different tables.
 * This is also a stateful version of the interface that relies on the
 * underlying protocol support for stateful communication via cookies.
 *
 * This script is run from the commandline, accepts host, port, kbName,
 * user, password and several specific field parameters.
 *
 * The script invokes EWCreate from SOAP API to create a Case in EW.
 */
public class CreateCaseCurrent {
    public static long createTicket(final String assetDeviceName,
                                   final int severity,
                                   final String summary,
                                   final Integer serverSerial,
                                   final String agent,
                                   final String ownerUID,
                                   final CondeSoapBindingStub ewServiceAPI)
        throws Exception {

```

```

WCase wcase = new WCase(); // creates an instance of Case object
// Device name is a part of a Linked Fields set, requires an
// instance of a link-class to be constructed
WCaseAsset_Table_Dao3_Link11 asset =
    new WCaseAsset_Table_Dao3_Link11();
asset.setAsset_Device_Name(assetDeviceName);
wcase.setDAO_Dao3_Link11(asset);
wcase.setPriority(
    convert(severity)); // Severity numeric values have to be
// converted into WS enums
wcase.setProblem_Description(summary);
// Customer login and Customer email are part
// of the Linked Field set
final WCaseContacts_Dao3_Link5 reportedBy =
    new WCaseContacts_Dao3_Link5();
reportedBy.setCustomer_Login(ownerUID);
reportedBy.setCustomer_Email(null);
wcase.setDAO_Dao3_Link5(reportedBy);
wcase.setNetcool_Event_Id(serverSerial);
wcase.setSummary(agent);
wcase.setTicket_Category(
    WChoice_Ticketparentchild.OPTION_UNIQUE);
// Choice values
// are set as WS enums
wcase.setType_Of_Issue(
    WChoice_Ticket_New_Type.OPTION_NETCOOL_ALARM);
wcase.setRelated_To(WChoice_Ticket_Related_To.OPTION_ASSET);
// Setting up Query-by-Example, this will search for IOC Team among
// the teams and if found - forge a link and
// import values for fields imported into the Case table as part of
// this Linked Fields relationship
WCaseTeams_Dao3_Link3 assignedTeam = new WCaseTeams_Dao3_Link3();
assignedTeam.setAssigned_Team("IOC Team");
wcase.setDAO_Dao3_Link3(assignedTeam);
// Sets subtype of the record to be created via generic interface.
// Matches the table name for top-level types, but will be
// different for true subtypes.
wcase.setType("case");
return ewServiceAPI.EWCreate("case", wcase);
}
private static CondeSoapBindingStub getSOAPBinding(String host,
                                                    String kb,
                                                    final String port)
    throws Exception {
    // in order to gain access to non-standard maintain session
    // parameters one has to upcast the returned
    // object to the WS-stack specific SOAP Stub type.
    final CondeSoapBindingStub ewServiceAPI =
        (CondeSoapBindingStub) new EWServiceAPIServiceLocator()
            .getconde(new URL("http://" + host + ":" + port +
                "/" + kb + "/EWServiceAPI"));

```

```

        ewServiceAPI.setMaintainSession(
            true); // enables HTTP session maintainance via cookies
        return ewServiceAPI;
    }
    private static WSChoice_Priority convert(int severity) {
        if (severity <= 2) return WSChoice_Priority.OPTION_LOW;
        if (severity == 3) return WSChoice_Priority.OPTION_MEDIUM;
        if (severity == 4) return WSChoice_Priority.OPTION_HIGH;
        if (severity == 5) return WSChoice_Priority.OPTION_CRITICAL;
        return WSChoice_Priority.OPTION_CRITICAL;
    }
    public static void main(String[] args) throws Exception {
        final String host = args[0];
        final String port = args[1];
        final String kbName = args[2];
        final String user = args[3];
        final String password = args[4];
        final int serverity = Integer.parseInt(args[5]);
        final String summary = args[6];
        final int serverSerial = Integer.parseInt(args[7]);
        final String agent = args[8];
        final String ownerUID = args[9];
        final String assetDeviceName = args[10];
        final String lang = "en";
        final CondeSoapBindingStub stub =
            getSOAPBinding(host, kbName, port);
        stub.EWLogin(kbName, user, password, lang);
        try {
            createTicket(assetDeviceName, serverity, summary, serverSerial,
                agent, ownerUID, stub);
        } finally {
            stub.EWLogout();
        }
    }
}

```

The purpose of the CreateCaseCurrent is to demonstrate the current way of working with Agiloft SOAP Web Services. In particular the interface is generic - exposing and accepting parent object to allow one to use the same web service with different tables. This is also a stateful version of the interface that relies on the underlying protocol support for stateful communication via cookies. This script is run from the commandline, accepts host, port, kbName, user, password and several specific field parameters. The script invokes EWCreate from SOAP API to create a Case in Agiloft.

```

package com.enterprisewizard.examples;
/*
 * (c) Copyright 2008, EnterpriseWizard, Inc. All Rights Reserved.
 * No part of this document may be stored in a retrievable system,
 * transmitted or reproduced in any way without the prior written
 * permission of EnterpriseWizard.
 */
import com.enterprisewizard.ws.client.demoonecall.EWServiceAPI;

```

```

import com.enterprisewizard.ws.client.demoonecall.EWServiceAPIServiceLocator;
import com.enterprisewizard.ws.client.demoonecall.WSCase;
import java.net.URL;
/**
 * The purpose of the "CloneTicketNew" is to demonstrate the way to pass
 * results of one WS method to another and challenges associated with
 * this.
 * The script uses "Single Method Operations", "Immediate Read" calls that
 * will be available in upcoming release of <span class="product1">
>EnterpriseWizard</span> Web Services.
 * This script is run from the commandline, accepts host, port, kbName,
 * user, password and searchValue parameters.
 * The script performs search by SQL on Case object of the knowledgebase
 * and submits the returned object right back in to create a clone.
 */
public class CloneTicketNew {
    private static EWServiceAPI getSOAPBinding(String host, String kb,
                                                final String port)
        throws Exception {
        // This script uses generic version of the service, but since
        // we do Single Method Operations and Immediate Read
        // no upcasting and manipulations with parameters of the low
        // level protocol is required.
        return new EWServiceAPIServiceLocator()
            .getDemo(new URL("http://" + host + ":" + port + "/" + kb +
                            "/EWServiceAPI"));
    }
    public static void main(String[] args) throws Exception {
        final String host = args[0];
        final String port = args[1];
        final String kbName = args[2];
        final String user = args[3];
        final String password = args[4];
        final String searchValue = args[5];
        final String lang = "en";
        final EWServiceAPI stub = getSOAPBinding(host, kbName, port);
        // Finds a Case with the summary matching searchValue
        final WSCase original = (WSCase) stub.EWSelectAndRead(kbName, user,
            password, lang, "case", "summary='" + searchValue + "'");
        if (original.getId() != null) {
            System.out.println("e = " + original.getId());
            System.out.println("e = " + original.getSummary());
            original.setSummary("Absolutely new ticket clone");
            // These values are configured to either be filled with
            // defaults or have not write access for anyone, to
            // avoid exceptions they are set to null,
            // i.e. will not be transmitted.
            original.setDate_Closed(null);
            original.setFaq_Feedback(null);
            original.setDate_Assigned(null);
            original.setTime_Created(null);
        }
    }
}

```

```

        original.setDate_Created(null);
        original.setId(null);
        original.setDAO_Dao3_Link2(null);
        original.setSumof_Billable_Hours_11957_(null);
        original.setSumof_Non_Billable_Hours_11958_(null);
        original.setDate_Updated(null);
        // creates a clone case
        final WSCase clone = (WSCase) stub.EWCreateAndRead(kbName,
            user, password, lang, "case", original);
        System.out.println("e = " + clone.getId());
        System.out.println("e = " + clone.getSummary());
    }
}

```

The purpose of the CloneTicketNew is to demonstrate the way to pass results of one WS method to another and challenges associated with this. The script uses Single Method Operations, Immediate Read calls that will be available in upcoming release of Agiloft Web Services. This script is run from the commandline, accepts host, port, kbName, user, password and searchValue parameters. The script performs search by SQL on Case object of the knowledgebase and submits the returned object right back in to create a clone.

```

package com.enterprisewizard.examples;
/*
 * (c) Copyright 2007, EnterpriseWizard, Inc. All Rights Reserved.
 * No part of this document may be stored in a retrievable system,
 * transmitted or reproduced in any way without the prior written
 * permission of EnterpriseWizard.
 */
import com.enterprisewizard.ws.api.EWWSBaseUserObjectMap;
import com.enterprisewizard.ws.impl.EWSimpleImplHelperWrapper;
import com.enterprisewizard.ws.utils.LogHelper;
import com.supportwizard.actions2.interfaces.ExternalScript;
import com.supportwizard.actions2.interfaces.ScriptActionException;
import com.supportwizard.actions2.interfaces.ScriptInput;
import com.supportwizard.actions2.interfaces.ScriptOutput;
import com.supportwizard.dml.SWDataMap;
import com.supportwizard.dml.SWRecordPK;
import com.supportwizard.seance.Seance;
import org.apache.commons.httpclient.HttpClient;
import org.apache.commons.httpclient.HttpStatus;
import org.apache.commons.httpclient.MultiThreadedHttpConnectionManager;
import org.apache.commons.httpclient.NameValuePair;
import org.apache.commons.httpclient.methods.PostMethod;
import org.apache.log4j.Logger;
import java.io.IOException;
import java.io.InputStream;
/**
 * The purpose of the "POSTPush" is to submit the result of modification of
 * a record in <span class="product1">EW</span> to an external host via HTTP
 * POST.
 * This script is run in a script action from the rule invoked on update.

```

```

* The script re-uses parts of WS/REST internal code to convert internal
* datatypes into POST parameters.
*/
public class POSTPush implements ExternalScript {
    private final static HttpClient httpClient;
    public static final Logger log = Logger.getLogger(POSTPush.class);
    private static final String FAILURE_MESSAGE =
        "REST Push: Failed ot push the change to the remote server";
    static {
        MultiThreadedHttpConnectionManager connectionManager =
            new MultiThreadedHttpConnectionManager();
        httpClient = new HttpClient(connectionManager);
    }
    public ScriptOutput runScript(ScriptInput input)
        throws ScriptActionException {
        // Script gets two copies of the current record - one before change
        // (or null in case of creation) and one
        // after change
        final SWDataMap newRecord = input.getNewRecord();
        // Obtain identification values for the purpose of logging
        final Seance seance = input.getUserSeance();
        final SWRecordPK swRecordPK = newRecord.getSWRecordPK(seance);
        final Long id = swRecordPK.getID();
        final Object type = newRecord.get("type");
        // helper method to aid logging
        final LogHelper logHelper = LogHelper.create("REST Push",
            new String[]{"type", "id"}, new Object[]{type, id});
        final ScriptOutput output = input.createOutput();
        try {
            logHelper.logAttempt(log);
            final EWSimpleImplHelperWrapper implHelperWrapper =
                new EWSimpleImplHelperWrapper();
            // converts internal datastructures to WS/REST object map.
            final EWSBaseUserObjectMap objectMap = implHelperWrapper
                .convertSWDataMap(seance, newRecord, logHelper, log);
            log.info(POSTPush.class.getSimpleName() +
                " invoked for type " + type + " id " + id);
            int i = -1;
            PostMethod method = new PostMethod(
                "http://externalhost/urithatacceptsthe data");
            method.setHttp11(true);
            try {
                objectMap.forEachEntry(new PostProcedure(
                    method)); // convert the object map into POST params
                i = httpClient.executeMethod(method); // perform POST
                try {
                    final InputStream asStream =
                        method.getResponseBodyAsStream(); // read result
                    byte b[] = new byte[1024];
                    try {
                        while (asStream.read(b) != -1) {

```

```

        }
        } finally {
            asStream.close();
        }
    } catch (IOException e) {
        // do nothing, ignore
    }
} finally {
    method.releaseConnection();
}
// report success or failure to EnterpriseWizard
if (i != HttpStatus.SC_OK) {
    logHelper.logExpectedFailure(log,
        new Exception(FAILURE_MESSAGE));
    return blockedScriptOutput(output, FAILURE_MESSAGE,
        newRecord, seance);
} else {
    output.setExitCode(ExternalScript.ACCEPT);
    logHelper.logSuccess(log);
}
return output;
} catch (Throwable e) {
    // Anything that goes wrong is considered unrecoverable
    logHelper.logUnexpectedFailure(log, e);
    return blockedScriptOutput(output, FAILURE_MESSAGE, newRecord,
        seance);
}
}
// Conversion of an entry to a POST parameter
public static class PostProcedure
    implements gnu.trove.TObjectObjectProcedure {
    private PostMethod postMethod;
    public PostProcedure(PostMethod postMethod) {
        this.postMethod = postMethod;
    }
    public boolean execute(Object o, Object o1) {
        String name = (String) o;
        if (o1 instanceof EWWSBaseUserObjectMap) {
        } else if (o1 instanceof EWWSBaseUserObjectMap[]) {
        } else if (o1 instanceof String[]) {
            String[] strings = (String[]) o1;
            for (int i = 0; i < strings.length; i++) {
                String s = strings[i];
                postMethod.addParameter(new NameValuePair(name, s));
            }
        } else if (o1 != null) {
            postMethod.addParameter(
                new NameValuePair(name, o1.toString()));
        }
        return true;
    }
}

```



```

    }
    //Signal to EW that execution was not successful,
    // give the user a chance to correct the data
    private ScriptOutput blockedScriptOutput(ScriptOutput output, String s,
                                              SWDataMap newRecord,
                                              Seance seance) {
        output.setExitCode(ExternalScript.BLOCK);
        output.setMessage(
            s + " ID:" + newRecord.getSWRecordPK(seance).getID());
        return output;
    }
}

```

The purpose of the "POSTPush" is to submit the result of modification of a record in Agiloft to an external host via HTTP POST. This script is run in a script action from the rule invoked on update. The script re-uses parts of WS /REST internal code to convert internal datatypes into POST parameters.

```

package com.enterprisewizard.examples;
/*
 * (c) Copyright 2008, EnterpriseWizard, Inc. All Rights Reserved.
 * No part of this document may be stored in a retrievable system,
 * transmitted or reproduced in any way without the prior written
 * permission of EnterpriseWizard.
 */
import com.enterprisewizard.ws.utils.LogHelper;
import com.enterprisewizard.scriptactions.conde.TicketUpdateScript;
import com.micromuse.response.server.soapserver.*;
import com.supportwizard.actions2.interfaces.ExternalScript;
import com.supportwizard.actions2.interfaces.ScriptActionException;
import com.supportwizard.actions2.interfaces.ScriptInput;
import com.supportwizard.actions2.interfaces.ScriptOutput;
import com.supportwizard.dml.SWDataMap;
import com.supportwizard.dml.SWRecordPK;
import com.supportwizard.seance.Seance;
import org.apache.log4j.Logger;
import java.net.URL;
/**
 * The purpose of the "NetcoolTicketUpdateScript" is to invoke a policy in
 * IBM Tivoli Netcool when a ticket is closed in EnterpriseWizard.
 * A policy can be invoked by calling a "Listener" Web Service provided
 * with each instance of Netcool.
 * This script is run in a script action from the rule invoked on update
 * and also periodically every 5 minutes. The purpose of the periodic run
 * is to pick up those tickets for which for some reason the information
 * was not transmitted successfully i.e. retry if some failure has
 * occurred.
 * The script uses an integer field "submitted_to_netcool" to indicate that
 * the invocation was successful. To avoid cluttering history by recording
 * the success on the very first run the the suggested pattern is to have
 * the field set to 1 (success) by default and set value to 0 only when it
 * fails.
 */

```

```

* The script will only update the field if a change is required.
* To attempt a resend such records can be isolated later by a rule run
* periodically with an IF/THEN condition based on Advanced Filter - the
* field changed value from 1 to 0 in the last X minutes.
*/

```

```

public class NetcoolTicketUpdateScript implements ExternalScript {
    public static final Logger log =
        Logger.getLogger(NetcoolTicketUpdateScript.class);
    public static final String USER_ID = "NetcoolUser";
    public static final String USER_PASSWORD = "NetcoolPassword";
    private static final String POLICY_NAME = "NetcoolPolicyName";
    private static final String ENDPOINT_URL =
        "http://NetcoolHost:NetcoolPort/" +
        "NETCOOL_CLUSTER_NAME_jaxrpc/impact/" +
        "ImpactWebServiceListenerDLIfc";
    private static final String FLAG_FIELD = "submitted_to_netcool";
    public ScriptOutput runScript(ScriptInput input)
        throws ScriptActionException {
        ScriptOutput output = input.createOutput();
        try {
            // Script gets two copies of the current record -
            // one before change (or null in case of creation) and one
            // after change
            SWDataMap newRecord = input.getNewRecord();
            // Obtain EnterpriseWizard ticket id
            Seance userSeance = input.getUserSeance();
            final SWRecordPK swRecordPK =
                newRecord.getSWRecordPK(userSeance);
            Long ewTicketId = swRecordPK.getID();
            //Obtain the id of the Netcool event that correponds
            // to the EnterpriseWizard record
            Integer netcoolEventId =
                (Integer) newRecord.get("netcool_event_id");
            boolean success;
            if (netcoolEventId != null) {
                URL url = new URL(ENDPOINT_URL);
                //Obtain SOAP stub
                ImpactWebServiceListenerDLIfc listener =
                    (new ImpactWebServiceListenerDLLocator())
                        .getImpactWebServiceListenerDLIfcPort(url);
                WSListenerId lid = null;
                //Internal EW class to assist with logging
                final LogHelper logHelper1 = LogHelper.create(
                    "ImpactWebServiceListenerDLIfc.login",
                    new String[]{"userId", "password"},
                    new Object[]{USER_ID, "*hidden*"});
                try {
                    logHelper1.logAttempt(log);
                    //Perform Login to Netcool via WS call
                    lid = listener.login(USER_ID, USER_PASSWORD);
                    if (log.isDebugEnabled()) log.debug("Object id is " +

```

```

        lid.getClientId() + ":" + lid.getObjectId());
    logHelper1.logSuccess(log);
    success = true;
} catch (Throwable e) {
    log.error("Unexpected exception", e);
    logHelper1.logUnexpectedFailure(log, e);
    // we want to finish the script run without failure,
    // to let user go and retry the process later
    success = false;
}
if (success) {
    final LogHelper logHelper2 = LogHelper.create(
        "ImpactWebServiceListenerDLIfc.runPolicy",
        new String[]{"PolicyName", "netcool_event_id"},
        new Object[]{POLICY_NAME, netcoolEventId});
    // In the following piece of code, we will execute
    // a policy called WSListenerTestPolicy.
    // To this policy we send one parameter with name
    // "TicketID" and value of Netcool event id as String.
    // The result of executing this policy
    // will be in the captured and logged.
    WSPolicyUserParameter[] wParams =
        new WSPolicyUserParameter[1];
    WSPolicyUserParameter thisParam =
        new WSPolicyUserParameter();
    thisParam.setName("TicketID");
    thisParam.setValue(netcoolEventId.toString());
    thisParam.setFormat("String");
    wParams[0] = thisParam;
    PolicyExecutionResult[] results;
    try {
        logHelper2.logAttempt(log);
        //Invoke the policy via WS call
        results = listener.runPolicy(lid, POLICY_NAME,
            wParams, true);
        logHelper2.logSuccess(log);
        if (log.isDebugEnabled()) { // log the results
            for (int i = 0; i < results.length; ++i) {
                log.debug("name " + i + ": " +
                    results[i].getName() + " value: " +
                    results[i].getValue());
            }
        }
    } catch (Throwable e) {
        log.error("Unexpected exception", e);
        logHelper2.logUnexpectedFailure(log, e);
        success = false;
    }
}
} else {
    log.warn("IGNORING. No Netcool event id in the record. " +

```

```

        ewTicketId);
        success = true;
    }
    // Set the flag field to 0 or 1 depending on
    // the results and the previous value
    int synced = 0;
    final Object o =
        newRecord.get(FLAG_FIELD); // obtain the current value
    if (o instanceof Integer) {
        synced = (Integer) o;
    }
    if (!success && synced != 0 || success &&
        synced == 0) { // only if the change is required
        final Long tableID = newRecord.getTableID();
        SWDataMap changeRecord = new SWDataMap(tableID);
        changeRecord.put(FLAG_FIELD, synced == 0 ? 1 : 0);
        output.setRecord(changeRecord);
    }
    // signal to EW that script execution was successful
    output.setExitCode(ExternalScript.ACCEPT);
    return output;
} catch (Throwable e) {
    // Anything that goes wrong is considered unrecoverable
    log.error("Unexpected exception", e);
    return blockedScriptOutput(output,
        "Unexpected exception." + e.getMessage());
}
}
//Signal to EW that execution was not successful,
//give the user a chance to correct the data
private ScriptOutput blockedScriptOutput(ScriptOutput output,
    String s) {
    output.setExitCode(ExternalScript.BLOCK);
    output.setMessage(s);
    return output;
}
}

```

The purpose of the NetcoolTicketUpdateScript is to invoke a policy in IBM Tivoli Netcool when a ticket is closed in EnterpriseWizard. A policy can be invoked by calling a Listener web service provided with each instance of Netcool. This script is run in a script action from the rule invoked on update and also periodically every 5 minutes. The purpose of the periodic run is to pick up those tickets for which for some reason the information was not transmitted successfully i.e. retry if some failure has occurred. The script uses an integer field `submitted_to_netcool` to indicate that the invocation was successful. To avoid cluttering history by recording the success on the very first run the suggested pattern is to have the field set to 1 - success - by default and set value to 0 only when it fails. The script will only update the field if a change is required. To attempt a resend such records can be isolated later by a rule run periodically with an IF/THEN condition based on Advanced Filter - the field changed value from 1 to 0 in the last X minutes.

Python Scripts

The Python API makes it easy to write custom scripts. It consists of two modules, `ALget.py` and `ALset.py`, that are used for reading input files and writing output files.

There are several ready-made functions available, but many of the scripts only use the following 6 functions:

- `ALget(filename)`
- `ALget.value()`
- `ALset.setRecordField`
- `ALset.setMessage`
- `ALset.setExitAction`
- `ALset.save`

Mastering the functions listed above will help you to write very powerful scripts. For more details on the `ALget` and `ALset` functions, visit the wiki articles entitled [ALget Common Use](#) and [ALset Common Use](#). You can also find valuable information for writing scripts on the [Script Structure and Basics](#) page, which will help you include all required functions.

Current_State and Old_State

Records that are currently being edited are considered to be in "current_state". There is always at least one record in "current_state", but only one of those current state records can be supported as the input for a given script: the record used as the input is referred to as the current record. You can obtain the current record name directly with an `ALget.currentStateRecordsNamesList()[0]` function.

In addition, the script may be able to use "old_state", which provides the old field values of the records. These are stored in an array that is ordered the same way as the current states, so the old values for the first current state record can be obtained with a similar function and list value, `ALget.oldRecordNamesList()[0]`. These old values are the values before the current edit of the records, assuming the script is being triggered by an edit to a record. These old values are provided so you can easily see, check, and validate user changes to fields and values.

recordName

Many of the functions take a parameter `recordName` as an input variable. If `recordName` is omitted from the input variable, the first of the current state records is used by default, and then is used to populate the output file.

Custom Distribution

If you would like to use your own Python distribution with scripts rather than the default distribution, you can define the location of your preferred python.exe using the [Location of external Python directory](#) global variable. For example, you might set it to something like `c:\Python34`.

ALget Common Use

This helper class allows retrieval of any value related to the record or user. It takes the passed XML file and parses the data structure so class methods can access it.

The discussion here assumes that “get_data” has been defined in the main function as follows:

```
get_data = ALget(inputXMLFile)
```

Retrieving Values

The specific examples call functions of “get_data” to retrieve values. Many types of values can be retrieved from the input file.

A standard field in the current record

The value function retrieves the content from a standard field in the record.

Syntax: `value(fieldName, recordName)`

Value returned: depends on field domain

Example

```
# Get the value in the 'type_of_problem' field. This uses the name of the field,
not the label.
Value1 = get_data.value('type_of_problem')
```

Attached files from the current record

The attachedFiles function retrieves attachment data from a specified field in the record.

Syntax: `attachedFiles(fieldName, recordName)`

Value returned: list of ALAttachedFile objects

Example

```
# Get information about attached files. The files are retrieved as a list of
objects with attributes for name and path.
# These are the only attributes available.
```

```
Value2 = get_data.attachedFiles('attached_files')
Name1 = Value2[0].name
Path1 = Value2[0].path
```

A field related to the user or other global variable

The `globalVariable` function returns the value of a user-related or KB global variable, or returns the default value if the variable is absent or null. The `globalVariablesNamesList` function can be used to find all available variable names.

Syntax: `globalVariable(variableName, default = None)`

Value returned: depends on variable domain

Example

```
# Get the name of the user who triggered the script.
Value3 = get_data.globalVariable('my_full_name')
```

A field from a linked record

The `linkedValue` function retrieves the `linkedRecordFieldName` field value for the `linkedRecordIdx` donor record that is linked to the `recordName` record in the `fieldName` field. So, if you want to use a Contract record to pull data from the assigned Person record, you would use the Contract record in `recordName`, the Contract's linked field as `fieldName`, the Person record in `linkedRecordIdx`, and the data field in the Person record as `linkedRecordFieldName`.

Syntax: `linkedValue(fieldName, linkedRecordIdx, linkedRecordFieldName, recordName)`

Value returned: depends on donor field domain

A multiple linked field in the current record

The `valueMultipleLF` function retrieves the values from the donor record, if the number of values is relatively small. The XML retrieved by this function is limited by the maximum text field size (`max_text_field_size_in_db` global variable), so if there are hundreds of records with large field values, this function might not retrieve all of the data. If you need to iterate over all the linked records in these cases, use the `linkedRecordsNamesList` and `linkedValue` functions.

Syntax: `valueMultipleLF(fieldName, recordName)`

Value returned: `ALMultipleLFContainer` object

Example


```
# Get information about the affected departments, which is a Link to a Single Field
with multiple values enabled.
# This creates an object with built-in functions for checking contained values and
finding records containing a particular value.
#valueMultipleLF is available for all types of linked field sets when multiple
records are allowed, but not related tables or embedded search results.
    Value5 = get_data.valueMultipleLF('affected_departments')
    if Value5.containsValue("Finance")...
        NextID = Value5.findByValue("Finance")

# Retrieve information about the 'assignee' field, which is a link to both the
"teams" and "contacts" tables. This creates a list of objects.
# Then check if the 'assignee' field has a team selected instead of a contact.
    Value6 = get_data.linkedRecordsNamesList("assigned_team")
    if get_data.tableNameForRecord(Value6[0]) == "teams":...
```

Objects Returned

In addition to integer and text values, some ALget functions return special objects.

ALAttachedFile

This object presents an attached file with its file name (name) and location (path).

ALMultipleLFContainer

This object represents multiple imported values for a linked field. There are several available functions for this object type.

- `findByValue(value)`: Input a value to search for and receive a `[recordID, value]` pair if it exists, or `None` otherwise.
- `containsValue(value)`: Input a value to search for and receive `True` if the object contains any lines with the value.
- `items()`: Receive a list of all `[recordID, value]` pairs in the container.
- `itemsMap()`: Receive a dict of `{ donorRecordId: value, ..., donorRecordX: valueX }` for the object.

ALset Common Use

This helper class allows setting a value either related to the record or to the exit conditions of the script action.

The discussion here assume that “set_data” has been defined in the main function as follows:

```
set_data = ALset(outputXMLFile)
```

Setting Values

The specific examples call functions of “set_data” to set values. Many types of values can be set.

A standard field in the current record

The `recordField` function assigns a value to a standard field in the record, or removes the field if the value is undefined.

Syntax: `recordField(name, value)`

Example

```
#Set the 'priority' field based on the variable Value1
set_data.recordField('priority', Value1)
```

Attached files from the current record

These functions add, remove, and replace files in a specified attachment field in the record.

Syntax:

- `addFile(fieldName, fileName, filePath)`
- `delFile(fieldName, fileName)`
- `replaceFile(fieldName, oldFileName, newFileName, newFilePath)`

Example

```
#Add the file "sample.txt" located at C:\tmp to the file "attached_files".
#The general format is ALset.addFile(fieldName, fileName, filePath). The absolute
path must be used.
ALset.addFile('attached_files', 'sample.txt', 'C:\tmp\')
```

```
#Remove the file "old.txt" from the "attached_files" field
ALset.delFile('attached_files', 'old.txt')

#Replace the file "old.txt" in the "attached_files" field with the file "new.txt"
located at C:\tmp. Note that this is only used with file versioning.
replaceFile('attached_files', 'old.txt', 'new.txt', 'C:\tmp\')
```

A message to be displayed to the user

The message function sets a message to be displayed to the user.

Syntax: message(message)

Example

```
#Set the user message based on the variable Value2
set_data.message(Value2)
```

A redirect URL to open after script execution

The redirect function specifies the URL to open after the script execution is completed.

Syntax: redirect(value)

Example

```
#Set a redirect url based on the variable Value3
set_data.redirect(Value3)
```

An exit action

The exitAction function determines whether the record is saved or rolled back, and whether the user is redirected. Set the value parameter to one of the available exit codes:

- 0-ACCEPT, which saves the record
- 1-BLOCK, which rolls back the changes
- 2-BLOCK_REDIRECT, which rolls back the changes and redirects the user
- 3-ACCEPT_REDIRECT, which saves the record and redirects the user

Note that the syntax for the value parameter is an integer.

Syntax: `exitAction(value)`

Example

```
#Set the exit action. The only accepted actions are the 4 shown.
    set_data.exitAction(0)
    set_data.exitAction(3)
    set_data.exitAction(1)
    set_data.exitAction(2)
```

Saving the file once the changes are complete

The save function saves the resulting XML to the fileName file, or to the output file used on object creation if no fileName is specified.

Syntax: `save(fileName)`

Example

```
#Save the output file. This is necessary to make sure the changes are actually
conveyed back to the system.
    set_data.save()
```

ALperf Common Use

ALperf can be used to retrieve system level information like CPU load and disk usage. To instantiate ALperf, import it and call its constructor without passing any arguments:

```
from ALperf import ALperf
perf_data = ALperf()
```

Here is a sample code that makes use of the ALperf module.

Example

```
#####
# Synopsis:
# Store the CPU load in a record of the KB and alert the sysadmin
# if the load is higher than a threshold.
#####
import sys
from ALget import ALget
from ALset import ALset
from ALperf import ALperf
import smtplib
from email.mime.text import MIMEText

def action(inputXMLFile, outputXMLFile):
    get_data = ALget(inputXMLFile)
    set_data = ALset(outputXMLFile)
    perf_data = ALperf()
    cpu_load = perf_data.cpuLoad
    set_data.recordField("field_load", str(cpu_load))
    if cpu_load > 2:
        try:
            body = ("The current load on the server is high.\n"
                    "Please take a look.\n\n"
                    "Thank you,\n")
            msg = MIMEText(body)
            msg['Subject'] = "Alert: Server load is high"
            msg['From'] = "frontdesk@company.com"
            msg['To'] = "sysadmin@company.com"
            s = smtplib.SMTP('agiloft.mail')
            s.sendmail("frontdesk@company.com", "sysadmin@company.com", msg.
as_string())
            s.quit()
        except:
            set_data.recordField("notes", "There was a problem sending email\n")
            set_data.exitAction(ALset.ACCEPT)
            set_data.save()

action(sys.argv[1], sys.argv[2])
```

The sample code above retrieves the **cpuLoad** attribute from the ALperf module.

Here is a list of the attributes available for retrieval:

- **hostName**
- **ipAddress**
- **systemBoot**
- **osType**
- **osDistro**
- **osBuild**
- **gLibc**
- **cpuCount**
- **ramTotal**
- **ramUsed**
- **memAvailable**
- **swapTotal**
- **swapUsed**
- **databaseType**
- **databaseAddress**
- **jbossType**
- **jbossMem**
- **mysqlMem**
- **cpuLoad**
- **topCpuProcess**
- **topMemProcess**
- **appDiskUsed**
- **appDiskFree**
- **dbDiskUsed**
- **dbDiskFree**
- **baseDir**

Here is a list of the methods (functions) available from the module:-

- **getCpuModel()**
- **getJreVersion()**
- **getNginxVersion()**
- **getAppBootTime()**
- **getHeapUsage()**

- `getDbVersion()`
- `getDbOptionValue(dbOption)`
- `getStorageType()`



ALperf.py

Sample Python Scripts

The following scripts provide examples of using custom Python scripting capabilities in a typical Agiloft installation. The Agiloft distribution includes Python and will automatically use it to execute scripts with a .py extension. If you want to use your own version of python, you may do so by setting the KB global variable "Location of external Python directory".

 You must add these functions to every Python script used in Agiloft in order for the script to terminate properly:

```
out.exitAction(ALset.ACCEPT)
out.save()
```

Samples

Click a sample below to see the script text.

```
#####
# Synopsis:
#
# Redirect the user after he/she submits the ticket.
#
# There is no need to specify the Python interpreter if the script is named *.py
# because the system recognizes .py scripts as being Python based and will
# use the Python that is included in the distribution.
#
# If you use this script with your own copy of Python, please note that
# it requires ElementTree module to be installed
#
#####
import sys
from ALget import ALget
from ALset import ALset
def action(inputXMLFile, outputXMLFile):
    input = ALget(inputXMLFile)
    out = ALset(outputXMLFile)
    #####
    #
    # Only log the user out if he/she is not a member of the Staff
    # group. In other words, we want all members of staff to be able
    # to log tickets on behalf of users without being logged out.
    #####
    groups = input.globalVariable('my_groups')
    groupAllowedToClose = "Staff"
    if groupAllowedToClose not in groups.split(","):
        out.redirect("http://www.example.com")
```



```

        out.exitAction(ALset.ACCEPT_REDIRECT)
    else:
        out.exitAction(ALset.ACCEPT)
    out.save()
action(sys.argv[1], sys.argv[2])

```

```

#####
# Synopsis:
# Send new users a confirmation email with their login and password
#
# There is no need to specify the Python interpreter if the script is named *.py
# because the system recognizes .py scripts as being Python based and will
# use the Python that is included in the distribution.
#
# If you use this script with your own copy of Python, please note that
# it requires ElementTree module to be installed
#
#####
import sys
from ALget import ALget
from ALset import ALset
import smtplib
from email.mime.text import MIMEText
#####
# Only send email to members of the Customer group.
# For security reasons, it is unsafe to send password information in
# plain text to privileged groups such as staff or admin.
#####
def action(inputXMLFile, outputXMLFile):
    input = ALget(inputXMLFile)
    out = ALset(outputXMLFile)
    groups = input.valueMultipleLF("f_group")
    me = "support@CompanyName.com"
    you = input.value("email")
    if groups and groups.containsValue("Customer") and you:
        body = ("Dear " + input.value("full_name") + ",\n"
               "Thank you for registering with our support system.\n\n"
               "You have chosen the following\n"
               "login: " + input.value("_login") + "\n"
               "password: " + input.value("password") + "\n"
               "Thank you,\n"
               "CompanyName Support staff\n")
        msg = MIMEText(body)
        msg['Subject'] = "Welcome to CompanyName Support"
        msg['From'] = me
        msg['To'] = input.value("email")
        s = smtplib.SMTP('localhost')
        s.sendmail(me, [you], msg.as_string())
        s.quit()
    out.exitAction(ALset.ACCEPT)

```

```

        out.save()
    action(sys.argv[1], sys.argv[2])

```

```

#####
# Synopsis:
# Check if the user belongs to QA group and block closure of the ticket if
# he/she doesn't.
#
# There is no need to specify the Python interpreter if the script is named *.py
# because the system recognizes .py scripts as being Python based and will
# use the Python that is included in the distribution.
#
# If you use this script with your own copy of Python, please note that
# it requires ElementTree module to be installed
#####
import sys
from ALget import ALget
from ALset import ALset
def action(inputXMLFile, outputXMLFile):
    input = ALget(inputXMLFile)
    out = ALset(outputXMLFile)
    groups = input.globalVariable('my_groups')
    groupAllowedToClose = "QA"
    if groupAllowedToClose not in groups.split(",") and input.value("wfstate")
== "Closed" and input.value("wfstate", input.oldRecordsNamesList()[0]) !=
"Closed":
        # Set a field value. In this case, we set the status field
        # to its old value
        out.recordField("wfstate", input.value("wfstate", input.
oldRecordsNamesList()[0]))
        print("Restore state")
        out.exitAction(ALset.ACCEPT)
        out.save()
    action(sys.argv[1], sys.argv[2])

```

```

#####
# Synopsis:
# This script performs some common tasks typical of a script.
# You could run this script against the input file example:
# filename.py input.xml output.xml
#
# There is no need to specify the Python interpreter if the script is named *.py
# because the system recognizes .py scripts as being Python based and will
# use the Python that is included in the distribution.
#
# If you use this script with your own copy of Python, please note that
# it requires ElementTree module to be installed
#
#####

```

```

import sys
# these are interface modules to interact with input and output xml files
from ALget import ALget
from ALset import ALset
def action(inputXMLFile, outputXMLFile):
    # Load the XML files into internal structures. This is a required step
    before interacting with input using API calls.
    input = ALget(inputXMLFile)
    out = ALset(outputXMLFile)
    # we will use this to accumulate messages to the user
    message = ""
    # Let's retrieve the user's name to provide a personalized message.
    # That would be the Full Name from the Contacts table entry for the user
    editing the ticket.
    full_name = input.globalVariable('my_full_name')
    # Let's check if user is a creator of the record and prohibit editing if
    not, with an appropriate message.
    # The same could be done with permissions, but access control through
    scripts can be more sophisticated.
    # For example, you could use input.globalVariable('my_start_work') and input.
    globalVariable('my_stop_work')
    # (if there are such fields in Contacts table) to allow modifications only
    within the user's working hours.
    print("Editor name:" + input.globalVariable('my_full_name'))
    print("Creator name:" + input.value('created_by'))
    if input.globalVariable('my_login') != input.value('login'):
        # We compare the login of the user who edits the record (via the global
        variable) with
        # the login of the record's creator. Note that this example would be for
        the Support Cases table,
        # in which the Creator Login variable is simply "login"). If
        they're not the same, we add a message to
        # the message variable we defined earlier, to describe the
        reason for denial.
        message += "Sorry, " + full_name + ", you are not the creator of this
        record, you cannot modify it."
        # Then we set the exit code to reject the changes.
        out.exitAction(ALset.BLOCK)
    else:
        # Now let's try to detect some changes in the record.
        # For example, let's make deleteable flag non-editable without blocking
        other changes.
        # The second parameter of value() is optional. Passing this
        argument in will refer to the version just before
        # the user's edit. If the current (edited) version doesn't
        equal that version, the user must've changed it.
        if input.value('deleteable') != input.value('deleteable', input.
        oldRecordsNamesList()[0]):
            # We will notify user that the change is not allowed.
            message += "Sorry, " + full_name + ", you cannot modify the
            deleteable flag, your change will be ignored."

```

```

        # Let's return the old value back.
        out.recordField('deleteable', input.value('deleteable', input.
oldRecordsNamesList()[0]))
        # Now some more complex techniques - linked records access.
        # Let's find out, for example, the name and phone number of the
record owner
        # if a Support Case is "open" ("owned by" is a Linked Field
relationship with the contacts table)
        if "Open" == input.value('wfstate'):
            # Retrieve phone and name from first (index=0) record linked to
owned_by field.
            owner_phone = input.linkedValue("end_user_name", 0,
"direct_phone")
            owner_name = input.linkedValue("end_user_name", 0, "full_name")
            message += "You can contact " + owner_name + ", owner of this
support case by phone at " + owner_phone + "."
            # Now check if this case is assigned to a team as opposed to a user
            # NOTE: this example is outdated because the assigned_team field
in support cases only links to the Teams table,
            # but the methods used here are interesting regardless.
            linkedRecordNames = input.linkedRecordsNamesList("assigned_team")
            if input.tableNameForRecord(linkedRecordNames[0]) == "teams":
                message += "This issue assigned to " + input.linkedValue
("assigned_team", 0, "_name") + " team."
            # Accept changes to ticket
            out.exitAction(ALset.ACCEPT)
            # Store the full message we've been accumulating in the variable "message"
using the message() method, so the user can read it all.
            out.message(message)
            # And don't forget to save the output file.
            out.save()
# Retrieve input and output XML files from sys.argv and pass them into our
function!
action(sys.argv[1], sys.argv[2])

```

Script Structure and Basics

As with all scripts, the script file itself should be placed in the scripts folder under /Agiloft/data/[kb name]/scripts. To run the script a script action should be created with the name of the script file as the action name. The script action checks to see if the file exists in the proper location when it is saved.

To use the ALget and ALset functions they must be imported from the modules. All of the Agiloft sample scripts start with importing just these functions. The standard sys module should also be imported.

```
import sys
# interface modules to interact with input and output xml files
from ALget import ALget
from ALset import ALset
```

The rest of the script can consist of a single function unless other references are needed. The Agiloft sample scripts define a single function called 'action' and calls that function at the end.

```
def action(inputXMLFile, outputXMLFile):
    function parts here
    .
    .
    .
action(sys.argv[1], sys.argv[2])
```

The Agiloft python scripts are always called with 2 arguments: an XML file representing the current and former record states, and a second XML file for the version of the records which should be saved to the database when the script is complete.

The following lines are required to generate XML at the end of the script:

```
out.exitAction(ALset.ACCEPT)
out.save()
```

A normal Agiloft python script involves reading values, history, and user data from the input file and saving any changes to the output file.

The first step in dealing with these files is to load them into Python objects that can be later referenced.

```
# Load input into internal structures. This is required step before
# accessing input by API calls.
input = ALget(inputXMLFile)
out = ALset(outputXMLFile)
```

The sample scripts and examples assume the scripts are structured this way and refer to "input" and "out" when using ALget and ALset.

Reset Record ID Numbers

This Bash script is applicable for the MySQL/MSSQL platform. It will reset the ID for each table of the KB to 1 or the MaxID+1 if there exist some records in the table with MaxID being the ID for the last record.

Replace "Demo" with the actual KB name before executing it through the Admin Console at **Setup > Debugging > Bean shell**.

Note: If demo records are deleted using the Admin Console delete demo data functions, the "deleted" records leave a "null" record in the SQL DB, and this will block any attempt to reset ID numbers or to import directly to that ID number. Those empty SQL records can be deleted by support or developers.

```
String projectName = "Demo";
import com.supportwizard.db.*;
import com.supportwizard.dictionary.*;
import com.supportwizard.utils.*;
import com.supportwizard.utils.db.*;
import java.sql.*;

StringBuilder log = new StringBuilder();
HomesGetter homes = new HomesGetter();
SWProjectHome projectHome = homes.getHome(SWProjectHome.class);
SWTableHome tableHome = homes.getHome(SWTableHome.class);
SWInternal_SQL sql = new SWInternal_SQL();

sql.dbConnect();
try {
    for
        (SWTable table :
        tableHome.findByProjectAndType(projectHome.findByName(projectName).getSWProjectID(),
        SWTable.USER_OBJECT)) {
        String autoincColumnDBName = null;
        for (SWColumn column : table.getSWColumns()) {
            if (column.isAutoIncrement()) {
                autoincColumnDBName = column.getDBName();
                break;
            }
        }
        if (autoincColumnDBName == null) continue;
        ResultSet
            rs = sql.getConnection().createStatement()
                .executeQuery("select max("+autoincColumnDBName+") from "+table.getDBName());
        int autoinc = 0;
        if (rs.next()) autoinc = rs.getInt(1);
        autoinc++;
        log.append("setting autoinc="+autoinc+" for table="+table.getDBName()+"."
            +autoincColumnDBName+"\n");
        if (DBSyntaxHelper.isMSSQL)
            sql.getConnection().createStatement().execute("DBCC CHECKIDENT ('[" + table.
                getDBName() + "]', RESEED, " + autoinc + ")");
        else
```

```

sql.getConnection().createStatement().execute("alter table "+table.getDBName()+"
auto_increment=" + autoinc);
}
} finally {
sql.dbDisconnect();
}
return log;

```