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.

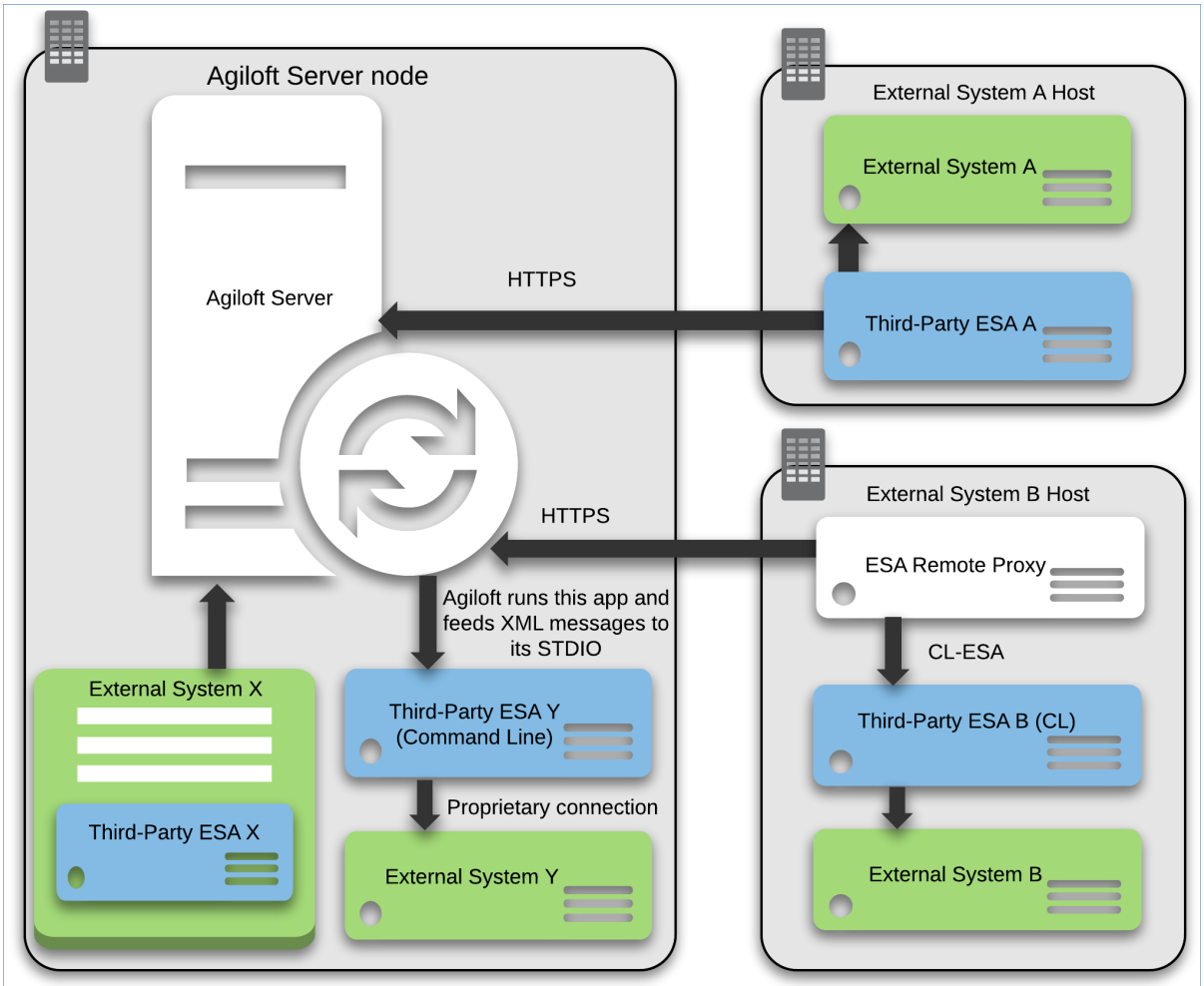

Illustration of possible ESA configurations

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 sync 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(Helper API)`.

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.

Authorization

To set up or run a sync using an API call, an `authorizationkey` request header is required. The value for this is found at **Setup > Sync > External Sync** under the Secure API Call section of the Sync record when you click Generate Keys.

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 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.

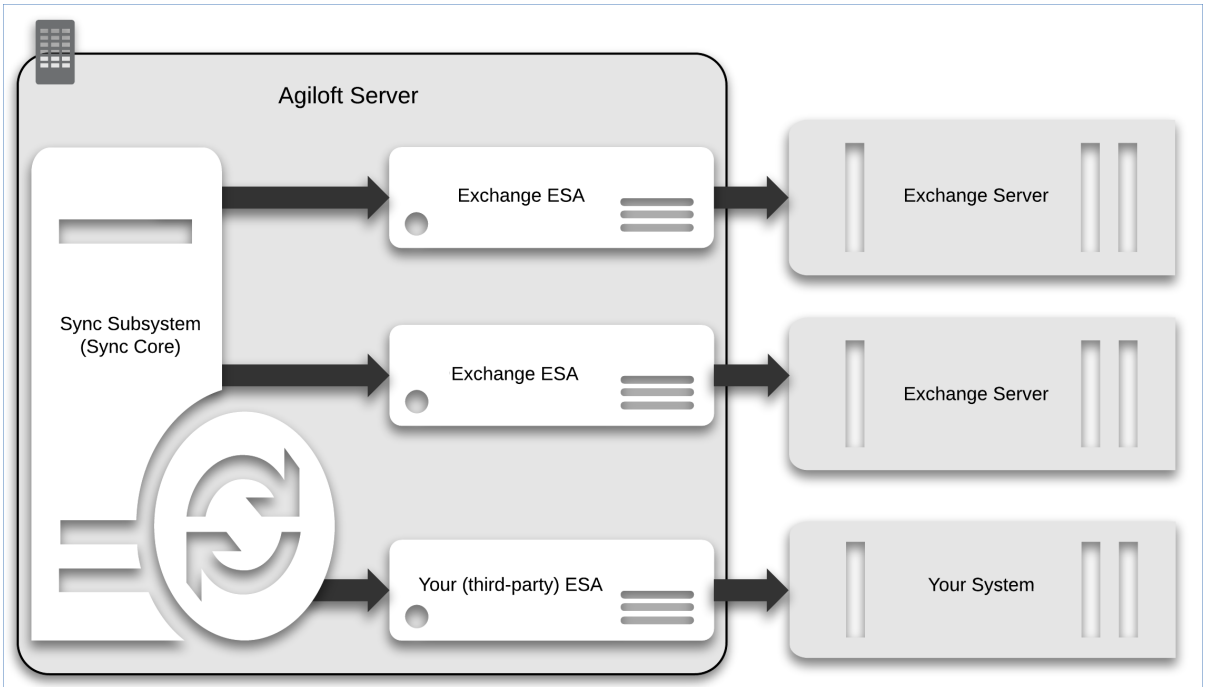

Figure 2 - Multiple ESAs can reside on and connect to the Agiloft server

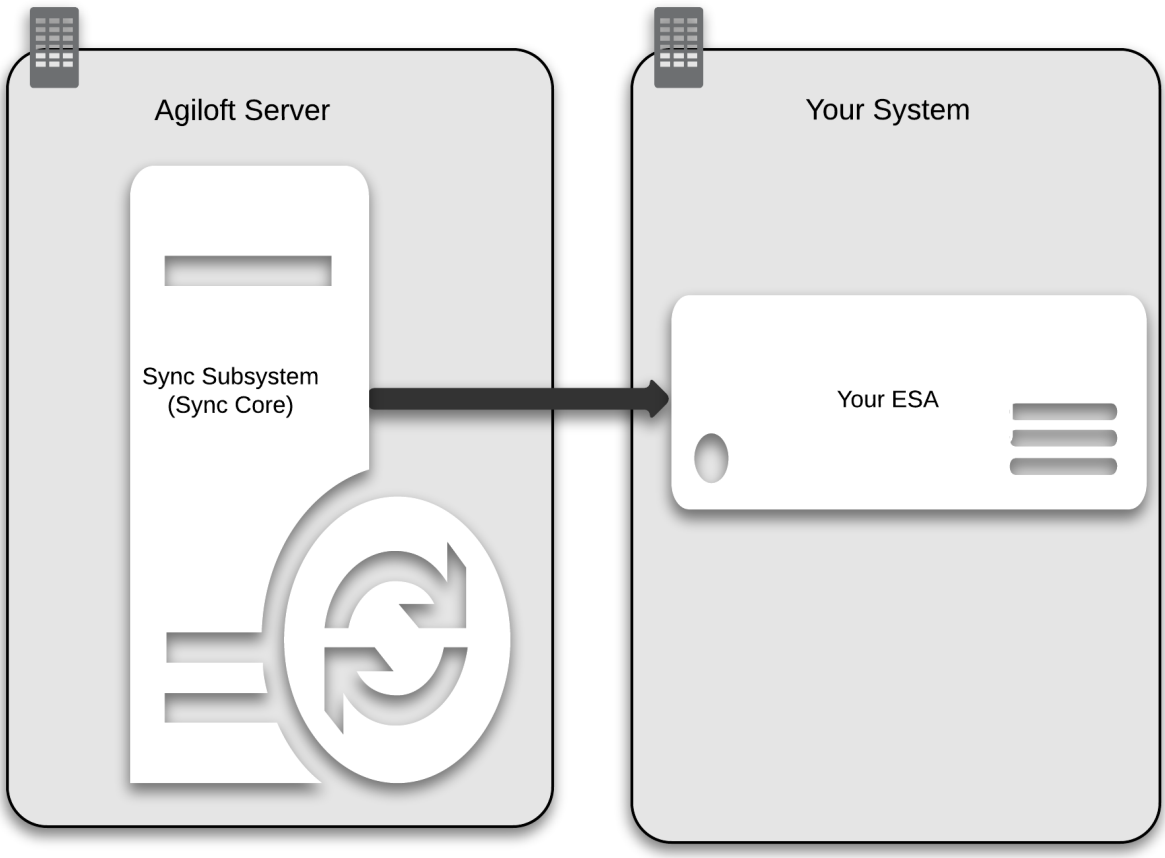

Figure 3 - Your ESA can also reside on your system

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 RecordListResultMessaqe( modified records);
      break;
    }...
    Case update:
    {
      // Update a record
      result = new RecordResultMessaqe( 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)
```