

# Seagull - Synchro protocol

## Table of contents

1 Synchro protocol details.....	2
2 Getting started with the synchro lib.....	2
2.1 First try.....	2
2.2 First try explained.....	2
2.3 Commands and parameters.....	4
2.3.1 Commands.....	4
2.3.2 Parameters.....	4
3 Generic configuration.....	5
4 Sync protocol dictionary.....	5
4.1 Types.....	5
4.2 Header.....	5
4.3 Body.....	5
4.4 Dictionary.....	6

## 1. Synchro protocol details

The aim of the synchro protocol is to synchronize several Seagull to each other, as well as to other applications. Inside a scenario, you can mix any protocol with synchro commands to interact with other Seagull instances or application. Those synchro commands can contain parameters that can be extracted from the messages received.

### Note:

An example of a JAVA applet synchronized with Seagull is provided.

## 2. Getting started with the synchro lib

### 2.1. First try

So that you can get familiar with the usage of the synchro lib, here is an example in which Seagull is used with the synchro lib and with another protocol. The Seagull on the left is a server running only with one protocol. The Seagull in the middle is a client running with the same protocol and with the synchro lib. The part on the right is a server to synchronise with. It could be a Seagull server. Here is the schema of this example:

Open one terminal session for each instance of the tool you want to run, as described in the previous section concerning the protocol you are interested into.

In the following section, we will only present the files or part of files containing the synchronisation commands. To build up a complete example, refer to the example with the protocol you are interested into.

### 2.2. First try explained

Here is the script (start\_client.ksh) that launched the client in our example:

```
seagull -conf ../config/conf.client.xml -dico ../config/synchro-dictionary.xml
-scen ../scenario/client.xml -log ../logs/client.log -llevel ET
```

### Note:

On some HP-UX systems, you might need to include the following export in your Seagull script: "export SHLIB\_PATH=/usr/local/bin".

### Note:

```
You can specify 2 dictionaries if needed: one for a protocol and one for the synchronisation library: #!/bin/ksh export
LD_LIBRARY_PATH=/usr/local/bin seagull -conf ../config/conf.client.xml -dico
../config/[protocol-dictionary].xml ../config/synchro-dictionary.xml -scen ../scenario/client.xml
-log ../logs/client.log -llevel ET>
```

This example is based on one server that receives a `CMD_INITSYNCHRO` and answer by a `CMD_INITSYNCHRO` message. Then they exchange `CMD_CALLCREATE` messages. See the dictionary for the complete list of messages that can be exchanged between the client and the server.

The dictionary for the synchronization protocol is "synchro-dictionary.xml", and is specified using the `-dico` parameter on the [command line](#).

The generic configuration (including network and other parameters) is different for the client and the server. The client uses `conf.client.xml` and the server uses `conf.server.xml`. The configuration file is

specified using the `-conf` parameter on the [command line](#).

Here are both files:

conf.client.xml	conf.server.xml
<pre>&lt;?xml version="1.0" encoding="ISO-8859-1"?&gt; &lt;configuration name="Simple IP Client Conf"&gt;    &lt;define entity="transport"     name="trans-ip-v4"     file="libtrans_ip.so"     create_function="create_cipio_instance"     delete_function="delete_cipio_instance"     init-args="type=tcp"&gt;   &lt;/define&gt;    &lt;define entity="channel"     name="channel-1"     protocol="command-synchro-v1"     transport="trans-ip-v4"     open-args="mode=client;dest=127.0.0.1:15000"&gt;   &lt;/define&gt;    ...  &lt;/configuration&gt;</pre>	<pre>&lt;?xml version="1.0" encoding="ISO-8859-1"?&gt; &lt;configuration name="Simple IP Server Conf"&gt;    &lt;define entity="transport"     name="trans-ip-v4"     file="libtrans_ip.so"     create_function="create_cipio_instance"     delete_function="delete_cipio_instance"     init-args="type=tcp"&gt;   &lt;/define&gt;    &lt;define entity="channel"     name="channel-1"     protocol="command-synchro-v1"     transport="trans-ip-v4"     open-args="mode=server;source=127.0.0.1:15000"&gt;   &lt;/define&gt;    ...  &lt;/configuration&gt;</pre>

**Table 1: Example client and server configuration**

**Note:**

Refer to [generic configuration](#) section for more information on the format and possible values.

Now comes the real stuff: the scenario.

First, the scenario source: [synchro.conf.client.xml](#) (synchro.conf.client.xml.html)

And now the commented version:

Scenario	Comments
<pre>&lt;?xml version="1.0" encoding="ISO-8859-1" ?&gt; &lt;scenario&gt;  &lt;counter&gt;   &lt;counterdef name="session-counter" init="0"&gt; &lt;/counterdef&gt; &lt;/counter&gt;  &lt;init&gt;   &lt;send channel="channel-1"&gt;     &lt;command-synchro name="CMD_INITSYNCHRO"&gt;     &lt;/command-synchro&gt;   &lt;/send&gt;   &lt;receive channel="channel-1"&gt;     &lt;command-synchro name="CMD_INITSYNCHRO"&gt;     &lt;/command-synchro&gt;   &lt;/receive&gt; &lt;/init&gt;</pre>	<pre>XML header  Counters definition In this section are initialized the counters that will then be used during the calls. For example here: we initialize the counter that will be incremented every time a call is started.  Beginning of the init scenario  Send a CMD_INITSYNCHRO command-synchro  Receive back a CMD_INITSYNCHRO command-synchro  End of the init scenario.</pre>

<pre> &lt;traffic&gt;   &lt;send channel="channel-1"&gt;     &lt;action&gt;       &lt;inc-counter name="session-counter"&gt;&lt;/inc-counter&gt;       &lt;set-value name="user-id-1" format="\$(session-counter)"&gt;&lt;/set-val     &lt;/action&gt;     &lt;command-synchro name="CMD_CALLCREATE"&gt;     &lt;/command-synchro&gt;     &lt;action&gt;       &lt;store name="SESSION-ID" entity="user-id-1"&gt;&lt;/store&gt;     &lt;/action&gt;   &lt;/send&gt;   &lt;receive channel="channel-1"&gt;     &lt;command-synchro name="CMD_CALLCREATE"&gt;     &lt;/command-synchro&gt;   &lt;/receive&gt; &lt;/traffic&gt; &lt;/scenario&gt; </pre>	<pre> Beginning of the traffic scenario Perform actions before sending the next message: - increment the counter defined in the counter section. - set the user id value Send a CMD_CALLCREATE command-synchro Store the session id. Receive back a CMD_CALLCREATE command-synchro End of traffic scenario End of scenario </pre>
---	---

## 2.3. Commands and parameters

### 2.3.1. Commands

Several synchronization commands are available:

- **CMD\_INITSYNCHRO**: generally used to initiate a synchronized session (in the `<init>` section of the scenario)
- **CMD\_ENDSYNCHRO**: generally used to close a synchronized session (optional)
- **CMD\_CALLCREATE**: generally used to send a synchronization command with parameters in the middle of a scenario execution

#### Note:

The usage indicated here for the commands is not mandatory

#### Note:

If needed, more commands can be added in the sync dictionary

### 2.3.2. Parameters

One of the important feature of the synchronization protocol is passing parameters along with synchro commands. The parameters can come from:

- `<restore-from-external (core.html#External+data+management) >` actions (coming from a data file),
- `<restore (core.html#action_restore) >` actions (following a `<store (core.html#action_store) >` action to save the value of a message parameter)

Example: use of two parameters, one from an external data file and one coming from a message received earlier (saved in "PARAM" call variable using `<store (core.html#action_store) >` action:

```
<send channel="channel-1">
```

```
<action>
  <inc-counter name="session-counter"></inc-counter>
  <set-value name="user-id-1"
    format="$(session-counter)"></set-value>
  <restore-from-external field="1" entity="field-bin-1-data"
    begin="1" end="3"></restore-from-external>
  <restore name="PARAM" entity="field-bin-2-data"></restore>
</action>
<command-synchro name="CMD_CALLCREATE">
  <parameter name="field-bin-2-data" value="1234"> </parameter>
</command-synchro>
<action>
  <store name="SESSION-ID" entity="user-id-1"></store>
</action>
</send>
```

"1234" can be replaced by any value, for it will be overwritten by the restore action.

In the default dictionary, there are 4 parameters declared:

- field-bin-1-data
- field-bin-2-data
- field-int-1-data
- field-int-2-data

"bin" parameters are used to pass Octet strings. "int" parameters are used to pass integers.

### 3. Generic configuration

The generic configuration file describes the network environment as well as traffic parameters.

## 4. Sync protocol dictionary

### 4.1. Types

"types" section contains all types needed for the protocol. For synchro, these are:

```
<types>
  <typedef name="String" type="string" unit="octet"> </typedef>
  <typedef name="Integer32" type="signed" size="4" unit="octet"> </typedef>
</types>
```

### 4.2. Header

"header" section contains the description of message header. For synchro, this is:

```
<header name="command-synchro" length="size" type="type">
  <fielddef name="type" size="4" unit="octet"> </fielddef>
  <fielddef name="size" size="4" unit="octet"> </fielddef>
  <fielddef name="user-id-1" size="4" unit="octet"> </fielddef>
  <fielddef name="user-id-2" size="4" unit="octet"> </fielddef>
</header>
```

### 4.3. Body

"body" section contains the description of message body (which naturally comes after the header). For synchro, this is:

```
<body>
  <header name="parameter" length="parameter-length" type="parameter-code">
    <fielddef name="parameter-code" size="4" unit="octet"> </fielddef>
    <fielddef name="parameter-length" size="4" unit="octet"> </fielddef>
```

```
</header>
</body>
```

#### 4.4. Dictionary

"dictionary" section contains all possible parameters and command-synchro. Here is a description for some examples:

```
<dictionary>
  <parameter>
    <define name="field-bin-1-data" type="String">
      <setfield name="parameter-code" value="0x00"></setfield>
    </define>
    <define name="field-bin-2-data" type="String">
      <setfield name="parameter-code" value="0x01"></setfield>
    </define>
    <define name="field-int-1-data" type="Integer32">
      <setfield name="parameter-code" value="0x04"></setfield>
    </define>
    <define name="field-int-2-data" type="Integer32">
      <setfield name="parameter-code" value="0x05"></setfield>
    </define>
  </parameter>

  <command-synchro session-id="user-id-1" out-of-session-id="user-id-2">
    <define name="CMD_INITSYNCHRO">
      <setfield name="type" value="0x00"> </setfield>
    </define>
    <define name="CMD_ENDSYNCHRO">
      <setfield name="type" value="0x01"> </setfield>
    </define>
    <define name="CMD_CALLCREATE">
      <setfield name="type" value="0x02"> </setfield>
    </define>
  </command-synchro>
</dictionary>
```