Collective2 XML AutoTrading Interface

Updated June 27, 2007

Recent changes

June 27 2007:

Must initially connect to autotrade.collective2.com…

not a hard-coded ip address;

Introduced **getsystemhypothetical** command

Feb 26 2007:

Introduced protocol 8.2: minor change in sync packet (now includes systemname) and signal packet (now specifies whether symbol is a mutual fund)

Overview

Collective2 is an online "marketplace" for trading systems for stocks, futures, forex, and options.

A trading system "developer" can sell his trading advice on Collective2. On the most simple level, a system developer types his trade advice (called trade signals) into the Collective2 (C2) Web site. These trades are then tracked by C2 so that the site can build a real-time track record of the trading system.

A trading system "buyer" can subscribe to one or more trading systems on the C2 site. Once a customer subscribes to a trading system, he is immediately informed about trading advice as it occurs, either through email or instant messenger. Further, using Collective2-compatible 3rd-party software, a customer can choose to 'AutoTrade' any of the trading systems he subscribes to. This means that trades recommended by a trading system developer can be placed in a real-life brokerage account.

C2 makes available a pseudo-XML interface for vendors who want to provide auto-trading/auto-execution services on behalf of their customers. This document describes that AutoTrading Interaface.

Other C2 APIs and programming tools available

✓ *In addition to the AutoTrading Interface, Collective2 also makes available a Trade Entry API. Any software or service vendor, or individual trader, may access the API in order to automate the process of 'entering trading advice' on Collective2. Online documentation about the Trade Entry API is available at www.collective2.com/api.*

✓ *As of June 27, 2007, C2 released its C2 Data Services API (C2DS). Any developer can use this API for free. It allows you to access almost the entire C2 database, and to build value-added*

services on top of the data. Examples of things you can access: list of all C2 systems, equity data for systems, stats and meta-stats for systems, trade histories, signal histories, etc. Documentation for the C2DS is available here: http://www.collective2.com/C2DS

The Trade Entry API, C2DS, and C2ATI are separate from each other. You may choose to use one of the interfaces, several, or none.

NOTE TO MULTIPLE-USER SERVICE PROVIDERS
This document, and the API described here, were designed to be used by software which handles one trader at a time. Typically, this type of software will run on a single PC – the PC of the end-user trader. If you are interested in developing a server-based service which handles multiple people simultaneously (example, a gateway or connector to an unsupported broker), please contact C2 and ask for documentation for the Multi-User C2ATI Extension.

General Structure of Collective2 AutoTrading Interface (C2ATI)

C2ATI transactions take place through http protocol, in a psuedo-XML language. It is a "pull" interface – that is, Collective2 never sends unsolicited information to clients. Instead, clients issue an http request and C2 responds. Thus, it is the client's responsibility to continually poll the C2 server.

The general sequence of events is this.

(1) Client logs in with Collective2 account information, and is given a sessionid to use for future requests.

(2) Client requests the latest trading signal (cmd=latestsigs).

(3) Client acknowledges any information contained in the latestsigs packet.

(4) C2 may respond with acknowledgements of its own.

Some general notes:

Session IDs. The sessionid is an unique number that is good for one communication session with the C2 server. After your client software logs out, or after a 20-minute-long period of inactivity (that is, a period in which your client does not communicate with the C2 server using the sessionid), the sessionid becomes invalid. If your client times out, you will receive the error message called <code>SESSION TIMED OUT</code> (see the section below on errors). You need to request a new sessionid each time you start communicating with the C2 server. Then, all subsequent requests to the C2 server must be accompanied by a valid sessionid.

Everything must be acknowledged. At a very general level, communication takes place like this:

YOUR CLIENT SOFTWARE REQUESTS AN UPDATE

C2 SENDS AN XML PACKET WHICH CONTAINS VARIOUS “NODES,” CONTAINING NEW INFORMATION

YOUR CLIENT MUST ACKNOWLEDGE THAT IT HAS RECEIVED AND ACTED ON ANY NEW INFORMATION CONTAINED IN THE PACKET

At any point in the cycle, instead of requesting a new update from the C2 server, your software may report new information to C2 (example: that an order has been filled in your real-life brokerage account). It is up to your client to decide when to report fill information to the C2 server. However, this information is required and should be reported to C2 as soon as the fill is reported to your client by the brokerage software.

C2 WILL ACKNOWLEDGE RECEIPT OF THIS INFORMATION

CYCLE REPEATS

Note that every piece of information must be acknowledged by the other party.

THE PROTOCOL

Communication

1) Logging in and creating a new session

To create a new session and log on, send the following string via http to **autotrade.collective2.com**, using port 7878:

?cmd=login&e=[email]&p=[password]&protoversion=8.2&client=[yourClientName]&h=[host]&build=[version_number_of_your_client_software]

(Note that [brackets] are used only as a convention within this document. No actual [or] characters should be included in your string.)

If you were typing this into a browser (we recommend Firefox for testing XML responses, since response data appears immediately on the screen), you would enter this url:

✓ http://autotrade.collective2.com:7878/?cmd=login&e=[email]&p=[password]&protoversion=8.2&client=[clientname]&h=[host]&build=[version]

An explanation of this string:

Email and password are the email and password assigned to your Collective2 account. The protoversion is the “C2 protocol version” you will be using in the upcoming communication. This document is for Protocol Version 8.2. It’s important that you specify the correct protocol version your client is using, because Protocol

8.2 (for example) contains several new commands and data formats which were previously unavailable. (As long as you specify a protocol number, the C2 server will be sure to handle your client properly, and support only those features which were supported at the time of your software's development.)

Each piece of software that will connect to the C2 server will need to be assigned a client name (example: "traderobot"). During your software development process, you will use the clientname TESTCLIENT (all caps). It is very important that you contact C2 and tell us that you will be developing and testing C2ATI software. This is because we need to adjust certain server settings that will allow trading signals to be sent to your client. If we do not know this, your software will never receive any new trading signals, and you will spend many hours (weeks?) in frustration, wondering what you are doing wrong.

Finally, the hostname is the hostname of the machine from which you are communicating. Sending the hostname helps prevent multiple clients connecting simultaneously using the same account information.

You will receive in response:

```
<collective2>
<status>OK</status>
<error>
        <code>0</code>
        <url></url>
</error>
<data>
        <session>123456789999</session>
        <first>John </first>
         <last>Doe </last>
        <redirectip>64.68.145.3</redirectip>
        <redirectport>7878</redirectport>
        <pollinterval>3000</pollinterval>
        <servertime>1113226452 </servertime>
        <humantime>2005-04-11 09:34:12</humantime>
</data>
</collective2>
```

If the status is 'OK' then the logon was successful (see possible error codes below). The information returned includes a sessionid, which you will need to use in all subsequent requests. Also, we return the first and last name associated with the Collective2 account.

We also return an IP address and port number to use for all subsequent communication during this session. Keep in mind that the IP and port used in subsequent requests may be different than the initial IP and port used to start the session.

The pollinterval is the duration, in milliseconds, to wait between your server communications (thus 3000 means 3 seconds). The C2 protocol is a pull protocol: that is, you will continuously ask for new information from the server. The server never contacts your client on its own. In order to maintain server responsiveness, you must follow the instructions in the poll interval and delay your server polling by the time specified. In this example, you will set your client to wait 3 seconds between server communications.

Finally, the server returns a time-stamp. You may choose to use this for user-interface time-related matters. The time is delivered in Unix time (seconds since the 1970 epoch, also used by perl), and human time. The time zone of the server is Eastern US.

---

2) Request latest trading signals

Once you have a session id, you will simply continue to poll the server at the polling interval. Typically, you will ask for the latest trading signals that you need to act upon. To do so, make the following request:

?cmd=latestsigs&session=[sessionid]&h=[host]

The latestsigs XML response always looks like this example:

```
<collective2>
<status>OK</status>
<error>
        <code>0</code>
        <url></url>
</error>
<data>

<cancelsiglist>
  <cancelsigid>13867666</cancelsigid>
</cancelsiglist>


<signalsep>13867772</signalsep>

<signalpacket>
  <systemname>vicinoo! trading - Daytrading</systemname>
  <systemidnum>13688760</systemidnum>
  <signalid>13867772</signalid>
  <postedwhen>1113226825</postedwhen>
  <postedhumantime>2005-04-11 09:40:25</postedhumantime>
  <action>STC</action>
  <scaledquant>70</scaledquant>
  <originalquant>700</originalquant>
  <symbol>RIMM</symbol>
  <assettype>stock</assettype>
  <mutualfund>0</mutualfund>
  <ordertype>LIMIT</ordertype>
  <stop>0</stop>
  <limit>76.22</limit>
  <tif>DAY</tif>
  <underlying></underlying>
  <right></right>
  <strike>0</strike>
  <expir></expir>
  <exchange></exchange>
  <marketcode></marketcode>
  <ocagroup></ocagroup>
  <conditionalupon></conditionalupon>
  <commentary></commentary>
  <matchingOpenSigs>
        <match>
                <sigid>1000</sigid>
                <permid>abcd</permid>
        </match>
```

```
</matchingOpenSigs>
</signalpacket>

<ocachanges>
</ocachanges>

<fillinforeceived>
</fillinforeceived>

<recentc2fills>
   <recent>
      <signalid>13808477</signalid>
      <filledago>409812</filledago>
      <filledprice>41.61</filledprice>
   </recent>
   <recent>
      <signalid>13808477</signalid>
      <filledago>409812</filledago>
      <filledprice>41.61</filledprice>
   </recent>
</recentc2fills>

<completetrades>
  <sigid>12001</sigid>
  <sigid>12004</sigid>
  <permid>abcdefgh</permid>
  <permid>ydsrehek</permid>
</completetrades>

<pollinterval>3000</pollinterval>
<servertime>1113226875</servertime>
<humantime>2005-04-11 09:41:15</humantime>
<browser>
</browser>

</data>
</collective2>
```

Explanation:

First, note that every server response carries with it a new pollinterval, which you are required to follow. It is possible that polling intervals grow longer or shorter depending on server loads.

Now, let us review the response packet, block by block.

cancelsiglist – contains a list of signals that have been canceled (or it may be empty). When you receive a signal cancel notice, you will need to acknowledge it (more on this later). Once you acknowledge the cancel, the signal id will no longer appear in the cancelsiglist.

signalsep – this was used by very early client applications to aid in parsing the XML block returns by C2. It can effectively be ignored.

signalpacket contains the signal. The signalid is an unique number. No two signals – even if they belong to different trading systems – will have the same number.

Information of interest within the signal packet:

<action> All signals must be one of the following: BTO, BTC, STO, SSHORT, STC. These mean: Buy To Open, Buy To Cover, Sell to Open, Sell Short, Sell To Close. (STO and SSHORT are equivalents.) Because C2 always marks actions as 'to open' or 'to close' this eliminates a major source of synchronization problems. **If you are trading only one C2 system, you should never 'Close' any quantity larger than a position you have 'Open.' (When multiple systems are traded, this is no longer the case. See "Multiple Systems, One Instrument" in the section called Synchronicity Issues, below.)** C2 will not permit a vendor to use incorrect 'to open' or 'to close' markers when entering trading signals.

<**ocagroup**> Signals may be assigned to an OCA group (one cancels another). If this field is filled in, any signal with the same oca identifier will be cancelled whenever the first signal in the group is filled. As of Protocol 4, OCA groups are handled by the C2 server. This means that you do not need to act on OCA information – instead, cancels will be sent to you for applicable signals. However, we do send OCA information within a signal packet, if it is known at the time the packet is sent, so that the client may act upon it more quickly than waiting for C2 to send a cancel.

<**conditionalupon**> For use with conditional orders. If this field is filled, this order should not be placed at the broker until the conditional order is filled. Your client software is responsible for keeping :stateful" information about conditional orders which must be sent to the broker when another, preceding order is processed.

<mutualfund> New field introduced in protocol 8.2. The value can either be 0 (this is not a mutual fund symbol) or 1 (this is a mutual fund symbol). Mutual funds will continue to have an <assettype> of stock. This new field was introduced because some brokers require different handling for mutual fund orders.

<matchingOpenSigs>

This field appears in Protocols 7 and higher. Starting with Protocol 8, it uses the following format:

Protocol 8: This node only appears within a signal packet for closing orders (that is STC or BTC). It will contain a list of trading signals that were used to open the trade that is now being closed. Only signals that your client has seen and acknowledged can appear in this packet.

Protocol 8 Format:

```
<matchingOpenSigs>
        <match>
                <sigid>1234</sigid>
                <permid>abcd-efg</permid>
        </match>
        <match>
                <sigid>5678</sigid>
                <permid>zxy-ytr</permid>
        </match>

</matchingOpenSigs>
```

Note that each opening signal is described in two ways: using the Collective2 Signalid that was originally reported with the signal (the <sigid> field) and using the permid (the permanent id) that you may have reported when you acknowledged the original signal. Some software developers prefer to use permids to refer to orders because – if the broker uses and reports permanent ids – then your client software will not need to keep track of C2 signalids from session to session. In the example above, Collective2 signal ID 1234 is the same exact order as Perm ID abcd-efg. It is being described in two different ways, for your convenience.

For Protocol 7:
Protocol 7 did not support permIDs. The <matchingOpenSigs> format was simpler. It contained a list of signal id's, separated by a period character (“.” - without the quotes). These signal id's are the signals that were used to open the trade that is now being closed. Only signal id's that your client has seen and acknowledged can appear in this string.

Protocol 7 Example:

A signal packet might contain (in addition to many other lines) the following:

```
<signalid>2222</signalid>
<action>STC</action>
<symbol>IBM</symbol>
…
<matchingOpenSigs>1001.1002.1006.1009</matchingOpenSigs>
```

Here, the matchingOpenSigs string indicates that the signalid 2222, which is meant to close a position in IBM, is closing a position that was opened using the following signalids: **1001, 1002, 1006, 1009.** (In this example, you were instructed to “leg into” a position over four separate trades). Your client software may choose to use this information or not. Technically, this information is not strictly necessary to follow a trading system, but some software developers have requested this feature, so they can know which opening signals match closing signals.

The <**ocachanges**> block is no longer supported in protocol versions 4 and higher.

The <**fillinforeceived**> block:
In the example above, the block is empty. Here is what it looks like when it contains data:

```
<fillack>
        <sigid>1234</sigid>
        <permid>abcd-efg</permid>
        <totalquant>102</totalquant>
</fillack>
<fillack>
        <sigid>4567</sigid>
        <permid>bgshf-hsgsk</permid>
        <totalquant>234</totalquant>
</fillack>
```

This block reports back to your client software the fill information the C2 has received, to date, from your client (in other words, this is a confirmation of the mult2fillconfirm block (described below). Any time there is "new" fill information received from your client, the C2 server will report back to your client (1) That is has received the fill info, and (2) the running total quantity it knows has been filled. The <totalquant> field contains the total number of shares/contracts/lots that C2 knows has been filled.

<**recentc2fills**> contains information about fills that have recently happened on Collective2's hypothetical fill engine. You may choose to use or ignore this information. Typically, this information might be used if you want to convert a limit order to a market order after a set period of time following the moment Collective2 recognizes the fill as taking place. (This is necessary to keep the server and your client in sync.) The <filledago> field is the number of seconds that have elapsed since the hypothetical fill took place.

<scaledquant> and **<originalquant>** - C2 trading system vendors are "given" $100,000 dollars in their hypothetical trading account when they begin their trading system on C2. They scale their trade recommendations accordingly. Thus, unlike many trading signal services, C2's signal vendors are required to provide trade quantity information within their trades. The <originalquant> field contains the original amount recommended by the trading system vendor, based on their current hypothetical account size (which may have increased or decreased from the initial $100,000.

Customers of C2 AutoTrading can specify their "scaling preferences" on the C2 web site. These preferences are set directly by the end user, and are neither known by nor controlled by the system vendor. The <scaledquant> field contains the order quantity that should be used once the customer's scaling preferences have been applied. **Thus the <scaledquant> field is the quantity you should place in the real-life brokerage account.**

<underlying>, <right>, <strike>, <expir> - Are used for equity option orders. Underlying is the stock symbol of the underlying stock. Right is either PUT or CALL. Strike is the option strike price, in decimal form (example 33.5). Expir is the month and year of optiuon expiration in YYYYMM format.

<expir> will also be filled in for futures orders and will contain the expiration month and year of the futures contract in YYYYMM format.

<marketcode> is currently unsupported.

<completetrades> was added in Protocols 5 and higher. It is useful only for software clients that do not implement convert-limit-to-market-order logic, but who want to stay "in sync" with a Collective2 trading system. The **<completetrades>** node lists all signalids (and permIDs, as reported by your client) that your client has received which have been completely closed and "flattened" on the Collective2 hypothetical fill engine.

Here is an example of why this information is useful.

> Imagine that a C2 trading system issues Signal # 300: BTO IBM at LIMIT 100.25. You receive this trade signal, and place it at your broker.
>
> Minutes pass. Eventually, you issue a **latestsigs** poll of the C2 server, and you see within the <recentc2fills> node that signal #300 has been filled in Collective2's hypothetical fill engine. Now you have a choice to make. Your own Buy Limit 100.25 order has still not been filled at your real-life brokerage account. What should you do? You (or the user of your software) might specify that a timer should start at the moment you see C2 has filled an order in its hypothetical fill engine. If, after a pre-determined period of time (30 seconds? 3 minutes?) your limit order is still not filled, then you can convert your still-open limit order into a market order. When your market order is filled (as all market orders are filled!) then you will report your opening fill to C2 via the **mult2fillconfirm** command. This will insure that you stay in sync with the C2 trading system.
>
> Now, when the trading system vendor issues closing instructions for IBM, your software client will receive these instructions, because C2 knows you have filled the matching opening order.
>
> Easy enough, you say. But what happens if you (or your software user) chooses not to implement a convert limit-to-market logic? In this case, when C2 fills the opening limit order, your client will see this fact within the <recentc2fills> information that is contained in every latestsigs polling response.
>
> Now, imagine that your client continues to work its own limit order. Minutes pass. Your opening order for IBM still is not filled. Now the C2 trading system vendor issues closing instructions for the IBM position that he has opened on C2. Since you never told C2 that you filled the opening order, you will not receive this closing order. Okay. More time passes. Now the system vendor finally fills

his closing order. You will have no way of knowing that the IBM trade, which you are still "working," has finally closed on C2.

To help you within this scenario is the purpose of the <completetrades> node. If you choose not to use convert-limit-to-market logic, this node will be useful to you. (On the other hand, if you use convert-limit-to-market logic, you can ignore the information in this node.)

When the C2 system vendor ***closes*** his IBM position, the opening signal id (#300 in this example) will appear in the <completetrades> node. This means you should take action, by either canceling your pending open order, or deciding to manage the entire IBM position on your own (since C2 will not send you any further instructions about this position).

If you choose to use the information contained in the <completetrades> node, be sure to acknowledge receipt of each signalid within the node, so that the information is not repeated by the server, and so that new information can take its place. To acknowledge information within the <completetrades> node, see the **ackcomplete** command below.

---

3. Acknowledge when you see a new signal.

If you do not acknowledge the receipt of a signal, it will continue to be sent to you. To acknowledge, send a request like this:

?cmd=confirmsig&sigid=1234&session=11111&h=hostname&quant=400[&localid=12&permid=34abs-ggg765ff]

Note that the portion of the string in the [ ] bracket's is optional. Here you can tell the server either the order id you have assigned to the signal, or the permanent id (the broker's id). Also, you should tell the server what quantity you placed of the order.

About permIDs: Many brokers assign a "permanent ID" (or permID) to each order that is placed. These IDs stay valid from session to session, and from day to day. You are not required to report permIDs to Collective2. However, you may wish to report them, and subsequently use them in all applicable calls to Collective2. The reason for this is that, by using permIDs, you will not have to keep stateful information about Collective2's signalids.

Example:
You receive a new signal from Collective2. It is SignalID #100: Buy 50 shares of IBM at Limit 20.
You place the order at your broker, and you are told that this newly-working order is PermID abcd-egg-1265.

Using cmd=confirmsig, you report to C2 that you received Signal ID 100, and that you are assigning a permID to it.

Days later, you get a fill report from your broker. You are told that Perm ID abcd-egg-1265 has been filled. You can report that information back to C2 (using mult3fillconfirm, below). There is no need to reference a signalid, and no need to keep a permanent stateful table of C2 signalids.

Alternately, all communication to and from C2 can use signalids, if you prefer.

Formats for PermIDs: PermID's must be 40 or fewer characters and more than 2 characters. They can include any ASCII character except ampersand (&). It is your responsibility to strip this character before sending a permID to C2.

---

4. Acknowledge when you see a cancellation.

?cmd=cancelconfirm&sigid=123&session=1234&h=host

or

?cmd=cancelconfirm&permid=abc-gfe-1234&session=1234&h=host

Until you acknowledge the cancellation of the trading signal, the signal will continue to appear in the cancelsiglist block.

---

5. Tell server when your orders have been filled.

You may continue polling the server for latestsigs and then acknowledging new signals and cancellations when you have received them.

In addition to the poll/ack cycle, you must also tell the server when your orders have been filled at the broker. This allows the server to specify proper quantities when giving you closing orders. Fill information should be delivered as soon as it is received from the broker.

To tell the server that your order has been filled:

There are several ways to report brokerage fills to C2. To tell C2 that a broker has filled one of its orders, use the **mult2fillconfirm** command (to report by using C2's internal signalids) or the **mult3fillconfirm** command (to use a broker's permids). Using these commands, you can tell C2 about multiple executions (or various legs of the same order) in one single command. Here is the format:

?cmd=
mult2fillconfirm&session=1234&h=host&filldata=sigid1|sep|quant1|sep|executionID1|sep|priceoffill|s

ep|sigid1|sep|quant1B|sep|executionID1B|sep|priceoffill|sep|sigid2|sep|quant2|sep|executionid2|sep|priceoffil|…sigidN|sep|quantN|sep|executionIDN|sep|priceoffill|&live=1

The five-characters string |sep| acts as a field separator. The filldata= portion of the string is composed of a series of four strings, separated by |sep|. The four components are:
Signal id#
Quantity
Brokerage Execution ID
Fill Price

Keep in mind that a C2 signal (Buy 100 IBM) might actually be filled by a broker in separate pieces, at different prices. You can report each piece separately, either in the same **mult2fillconfirm** (or mult3fillconfirm) command, or stretched across multiple polling intervals.

In the example above, you are reporting that C2 signal id 1 has been filled an amount equal to quant1 + quant1B. You are also reporting that signal id 2 has been filled for quant 2, etc. Note that each multfillconfirm string can contain multiple signals, and/or multiple incremental quantities. Each signal and quantity must have an unique ExecutionID associated with it. ExecutionIDs can never be repeated, not even within the same mult2fillconfirm command string. (C2 uses the ExecutionID, and its guaranteed uniqueness, to prevent 'double-calculating' fill amounts. Since all brokers issue unique ExecutionIDs to each piece of a transaction, this is a safe strategy. The ExecutionID may be called a different term by different brokers. At Interactive Brokers, for example, it is called the PermID.)

A few things to note about the **mult2fillconfirm** string:

- A signalid can appear one or more times in the string. The format is (do not use spaces; they are included here for legibility):

  SIGNALID |sep| QUANTFILLED |sep| EXECUTIONID |sep| FILLPRICE … SIGNALID |sep| QUANTFILLED |sep| EXECUTIONID |sep| FILLPRICE (etc).

- And EXECUTIONID must never appear more than once, not even in the same command string. If it does the second instance will simply be ignored by C2, as will all subsequent instances.

- Don't forget about the live=x parameter at the end of the long string.The live=1 or live=0 parameter tells C2 whether the fills being reported are real executions, or are results from demo and simulation accounts. C2 uses this information to help determine trading system robustness, and also to improve its Hypotheticall Fill Engine scheduling algorithm. Thus it is important you report this accurately.

Note that the **mult2fillconfirm** and mult3fillconfirm commands replace the now-deprecated commands **fillconfirm** and **multfillconfirm**.

What is the difference between* mult3fillconfirm *and* mult2fillconfirm ? *(note the numeral 3 instead of 2)

The format for mult3fillconfirm is identical to mult2fillconfirm. The only difference is that, when using mult3fillconfirm, you do not report C2's internal signalids; instead, you report the "permids" that the broker has issued for each order, and which you have reported to C2 earlier in the confirmsig *command. Obviously, for this to work, you need to report to C2 the broker's permids. The advantage of using mult3fillconfirm instead of mult2fillconfirm is that you do not need to keep state information about Collective2 signalids. Instead, when you receive a fill from a broker – perhaps days after you placed the order, and multiple software restarts later – you can report it to Collective2 using the broker's permanent, unique order identifier (which we call the permid). Of course, for this command to work, you must have previously reported the permid to Collective2 when confirming the receipt of the Collective2 signal (i.e. you must have reported a permid when you used the confirmsig command.) Note that permids, in order to be treated as valid by C2, must be more than 3 characters long.*

And yet another way to report fills: *multfillconfirmcumu*

The command **multfillconfirmcumu** follows the mult3fillconfirm format exactly (i.e. it uses permids, not C2's internal signalids). The critical difference is that when you use **multfillconfirmcumu**, you report cumulative fill quantities and cumulative average execution prices – not incremental quantities.

An example of how to use multfillconfirmcumu:
Imagine that C2 issues trading signal #100: Buy 7 shares of IBM at limit 100.25.
You acknowledge placing the order at the broker, and report to C2 that the broker has assigned a permid of 12345|655eeggg|xyz.
Now, at at 10:00 am, 3 shares are filled at price 100.25.
At 11:00 am, the remaining 4 shares are filled at 100.25.

When using **multfillconfirmcumu**, you would first report a quantity of 3. Later, at 11:00 am you would report a quantity of 7. Notice how this differs from the mult3fillconfirm command, which would have required you to report incremental fills. In contrast, **multfillconfirmcumu** requires you to report the running total quantity that has been filled (and the running average execution price).

Important warning about multfillconfirmcumu: Each instance that you use this command requires an unique execution id. Even though you are reporting cumulative totals to C2, the broker will report each leg of an execution to you using an unique execution id. If you do not use unique execution ids when reporting back to C2, subsequent reports will be ignored.

6. Acknowledge when you see a recentc2fill.

Recall that C2 will report in the <recentc2fills> block information about orders that have been filled on C2's hypothetical fill engine. This might be useful, for example, if an order is a limit order, and your client software needs to take certain action when C2 fills the order (perhaps you need to start a timer that counts down to a moment when you convert the still-pending limit order to a market order, so that you can stay 'in sync' with the C2 trading system).

When C2 reports information in the <recentc2fills> block, you must acknowledge that you have received it, or else the information will continue to be sent.

Tell the server that you have received the information about recent fills that have taken place on the C2 hypothetical fill engine:

?cmd=ackc2fill&sigid=123&session=1234&h=host
or
?cmd=ackc2fill&permid=abc-yt65&session=1234&h=host

By acknowledging the c2fill, you will prevent the same information from appearing again and again in the recentc2fills block.

6b. Acknowledge when you see a new signal in the <completetrades> node. (Optional)

This step is optional. If you choose to use it, you must tell the server that you have received the information that a signal is now completely closed on Collective2:

?cmd= ackcomplete&sigid=300&session=1234&h=host

Remember that using the **<completetrades>** node is optional. It is only useful for software clients which do not implement convert-limit-to-market orders, but which want to stay in sync with Collective2 trading systems. (Keep in mind that if you do not ever fill an opening order and thus you do not tell C2 that you have filled it, you will never be sent the matching closing order. The <completetrades> node (described above) allows you to know, nevertheless, when an opening signal you have received has been filled and then closed out to a flat position on C2. When you receive this information, you may choose to cancel any pending opening orders, or to flatten your own position in the instrument.

Note that if you do not use the <completetrades> node, and do not acknowledge the information contained within it, the information sent in the node will keep repeating itself, and thus you will not receive all necessary information.

---

7. Repeat the cycle

Once all necessary acknowledgements have been made, you should start the cycle again by requesting latestsigs.

---

8. Log off

?cmd=logoff&session=1234&h=host

Other (optional) commands

You may also choose to use any of the following commands:

requestsystemlist

example: **?cmd=requestsystemlist&sigid=300&session=1234&h=host**

This command is not strictly necessary, but was added at the request of a C2ATI developer. You may want to use it if you are interested in displaying a list of trading systems to which the user is subscribed.
This command allows you to ask C2 for a list of the customer's systems to which has subscribed (or of which he is the owner and thus has AutoTrading permission). The XML response you receive will look like this:

```
<systemList>
        <system>
                <name>Rocket Stocks</name>
                <systemid>12345</systemid>
                <permissions>
                        <asset>
                                <assettype>stocks</assettype>
                                <long>1</long>
                                <short>0</short>
                        </asset>
                        <asset>
                                <assettype>options</assettype>
                                <long>1</long>
                                <short>0</short>
                        </asset>
                        <asset>
                                <assettype>futures</assettype>
```

```
                    <long>1</long>
                    <short>1</short>
                </asset>
                <asset>
                    <assettype>forex</assettype>
                    <long>1</long>
                    <short>1</short>
                </asset>
            </permissions>
        </system>
        … more systems …
</systemList>
```

Note that within each “permissions” node, all assets tradable on C2 are listed. (Currently these are stocks, options, futures, forex). The **long** and **short** nodes indicate whether go-long or go-short permission has been enabled for each asset. (Long=1 means long trades are permitted. 0 means no permission.)

Please note that you should **not** use this information as a filter for whether to place a trade. (I.E. Do not look at the packet above and decide not to place short stock trades.) Collective2’s server will automatically send you the proper trades, depending on trading permissions set by the end-user on the C2 Web site, and so the information contained in the XML packet above is solely for informational purposes (so that you can display information within your user interface). ***The information in this packet may be incomplete or out-of-date, and other factors besides long/short permissions determine whether a trade should really be placed.***

In order to maintain reasonable server loads, please request this information no more frequently than once every 120 seconds. If you make more than one request for “requestsystemlist” within 120 seconds, the information will not be refreshed, and you will receive the same information as the initial request.

---

getsystemhypothetical

example: **?cmd=getsystemhypothetical&systemid=13889808.13202557&session=1234&h=host**

This command returns equity information about trading systems on Colelctive2. Important: the data returned reveals Collective2’s hypothetical system account information. The data does not include any customer/brokerage real-life account data. Only you (a developer writing an interface to a brokerage) can retrieve real-life brokerage account information, and of course you must do that by polling the broker, not C2.

You can request between one and 25 systems at a time. Demarcate each systemid using the period (.), as the example above shows.

Data is returned as in the format below. Most likely, you will be interested only in the “totalequityavail” field. You’ll notice that totalequityavail equals cash plus equity minus marginused.

```
<collective2>
<status>OK</status>
```

```
<error>
<code>0</code>
</error>
<data>
<hypotheticalEquity>
  <system>
        <systemid>13889808</systemid>
        <systemid>Absolute Returns</systemid>
        <totalequityavail>48281.90</totalequityavail>
        <cash>101864.90</cash>
        <equity>-8863</equity>
        <marginused>44720</marginused>
  </system>
  <system>
        <systemid>13202557</systemid>
        <systemid>extreme-os</systemid>
        <totalequityavail>218505.23</totalequityavail>
        <cash>226803.23</cash>
        <equity>894200</equity>
        <marginused>902498</marginused>
  </system>
</hypotheticalEquity>
</data>
</collective2>
```

getallsignals

example: **?cmd=getallsignals&session=1234&h=host**

This command is not strictly necessary, but has been requested by software developers who keep state information about signals. The command getallsignals will return a list of signalids that are still pending, organized by trading system. A signal that is still pending means that it is still “alive” or “working” – i.e. it has not yet been traded, cancelled, or expired.

The XML returned is in the following format:

```
<allPendingSignals>
        <system>
                <systemid>12345</systemid>
                <pendingblock>
                        <signalid>12003</signalid>
                        <signalid>65022</signalid>
                        <signalid>827912</signalid>
                </pendingblock>
        </system>
        <system>
                <systemid>22233</systemid>
                <pendingblock>
                        <signalid>98077</signalid>
```

```
                    <signalid>98079</signalid>
                </pendingblock>
            </system>
</ allPendingSignals>
```

Important: C2 will only send you signals that you have previously acknowledged via confirmsig. It is your responsibility to keep all necessary state information for these signalids.

Synchronicity Issues

The most difficult issues with AutoTrading involve how to stay "in sync" with Collective2. The challenge is that a trading system vendor issues trading instructions based on a hypothetical (imaginary!) account on Collective2. Your AutoTrading software needs to act on these instructions, but your real-life account(s) may or may not receive the same fills and order executions as C2's imaginary accounts. Further, things become even more complicated when Forex trading is introduced, since there is no centralized market for forex trading, and every broker uses different prices. This means that C2's hypothetical account may receive fill prices (or may not receive any fill prices) differently than your real-life accounts.

We call the complications that arise in this area "synchronicity issues" or "sync issues" for short. All sync issues can be dealt with, but there are always tradeoffs involved. In general, the main tradeoff is between fidelity to the C2 system (i.e. how closely you match the positions on C2) versus fill prices. It's a simple matter to stay in sync with C2 (to always be long when C2 is long, and to be short when C2 is short), but that comes at a cost, and that cost is "slippage" – that is, the difference between the execution prices C2 proclaims and the execution prices your real-life brokerage accounts receive.

Typically, sync issues are handled by having your client software convert limit orders into market orders after C2 has been filled in the limit order, and you have not. This is usually a customer-specified setting: i.e. "Convert limit orders to market orders after XX seconds." We recommend that you allow customers to toggle this setting on-off and modify the timer value on a per-instrument basis, and also on an opening versus closing basis. (You may choose custom values per symbol.) Customers may be willing to accept slippage to exit a trade (and thus stay in sync with C2), but may decide not to accept any slippage on the way into a trade. Their thinking might be: If I miss a trade, then I miss a trade, and that is acceptable. But once the C2 system exits, I want to make sure I get out, also.

Similarly, customers may be willing to convert to market relatively quickly for narrow-spread instruments that are very liquid. They may not want to convert quickly for options (where spreads are much higher, and the conversion to market will introduce large amounts of slippage).

How your client handles synchronicity issues will be a key differentiating factor between C2 AutoTrading clients.

Multiple systems, one instrument. Earlier in this document, we said: “you should never close any quantity greater than you have open….” In fact, this is not strictly true. There are indeed cases in which you may receive an instruction from C2 (Sell to Close 100 IBM) which contains a quantity greater than the quantity you currently own in an open position.

This occurs when your customer has subscribed to two trading systems, each of which trades the same instrument, in different directions.

Example:

System 1: “Rock and Roll Stocks”
9:30 am Buy To Open 100 IBM
9:50 am Sell To Close 100 IBM

System 2: “Stock scalper”
9:40 a.m Sell Short 20 IBM
9:55 a.m. Buy to Cover 20 IBM

In this case, at 9:50 you will own 80 shares of IBM long, and you will then receive a signal from C2 to Sell To Close 100 shares. Your client will need to close out those 80 shares and then go short the remaining 20 shares.

Synchronization Layer

New as of July 29, 2006:

Recently C2 introduced a new set of functions which collectively make up what we call our Synchronization Layer. The Sync Layer allows your client software to periodically request position updates from Collective2. These position updates will describe the positions that Collective2 believes are in your brokerage account. In addition, we now also allow you to tell Collective2 about modifications to these positions. Between these functions, it is now possible to maintain a higher degree of synchronization between C2 and your brokerage account.

How to use the Sync Layer

Simply request a sync packet like this:

?cmd=requestsystemsync&synctype=system&systemid=all&session=1234&h=host

This command returns XML in the following format:

```
<sync>
<system>
  <systemid>20045 </systemid>
  <systemname>Easy as Pie Swing System</systemname>
  <timefiltersecs>1154190903</timefiltersecs>
  <timefilterclock>2006-07-29 12:35:03</timefilterclock>
  <position>
            <symbol>AAPL</symbol>
            <assettype>stock</assettype>
            <quant>350</quant>
            <underlying> </underlying>
            <right></right>
            <strike></strike>
            <expir></expir>
            <exchange></exchange>
            <marketcode>44</marketcode>
  </position>
  <position>
            <symbol>EURUSD</symbol>
            <assettype>forex</assettype>
            <quant>-120000</quant>
            <underlying> </underlying>
            <right></right>
            <strike></strike>
            <expir></expir>
            <exchange> </exchange>
            <marketcode></marketcode>
  </position>
</system>
<system>
<systemid>66666</systemid>
<position>
            <symbol>EURUSD</symbol>
            <assettype>forex</assettype>
            <quant>70000</quant>
            <underlying> </underlying>
            <right></right>
            <strike></strike>
            <expir></expir>
            <exchange> </exchange>
            <marketcode></marketcode>
  </position>
  <position>
            <symbol>@ESU6</symbol>
            <assettype>future</assettype>
            <quant>-3</quant>
            <underlying>@ES</underlying>
            <right></right>
            <strike></strike>
            <expir>200609</expir>
            <exchange>CME</exchange>
            <marketcode></marketcode>
 </position>
<position>
            <symbol>ITTHL </symbol>
            <assettype>option</assettype>
            <quant>unknown</quant>
            <underlying>ITT</underlying>
            <right>call</right>
            <strike>60</strike>
            <expir>200608</expir>
            <exchange></exchange>
```

```
        <marketcode>43</marketcode>
  </position>
</system>
</sync>
```

(Version 8.2 saw introduction of <systemname> field.)

Note that negative quantities are short positions; positive quantities are long positions. All symbols use Collective2 symbology. (For a list of C2 futures symbols, see: www.collective2.com/symbology_futures)

Sync data is returned separately for each system. The xml above shows sync data for two systems. Notice that each system may report a recommended quantity for the same symbol. In the example above, C2 tells you that the first system should hold a short 120,000 position of EURUSD, and the second system should hold a long position of 70,000. This means that when your software looks at the customer's real-life brokerage account, the brokerage account should have a net short 50,000 position.

What to do if you are out of sync?

So you're supposed to be short 50,000… but what if you are not?

Let's imagine that you receive the sync packet above. It tells you (among other things) that your customer's brokerage account should be short 50,000 EURUSD. But when your software communicates with the brokerage, you see a net short position of only 40,000 EURUSD. What should you do?

There are several ways to handle this scenario, but here is what we recommend. First, cancel all pending limit or stop orders at the broker for EURUSD. Once those cancellations are confirmed, issue a market order to sell an additional 10,000 EURUSD. When this iorder is filled, you will not report it back to C2. (The reason is that it is impossible to report: the order is not tied to any specific system id or signal id. It is a "make good" order that C2 does not need to know about or care about.)

How often should you poll for sync information?

We recommend no more frequently than once every two minutes.

What is Timefiltersecs and **timefilterclock**

Timefiltersecs and **timefilterclock** both describe the time (New York Time – GMT-5) *before which* these positions were valid. In the example above, as of the exact moment *before* the clock became 12:35:03, these were the positions Collective2 assumed were in your brokerage account. This data is typically not used by C2ATI software, but is made available.

Why do I see <quant> = *unknown*?

When requesting synctype=system requests, you may sometimes see a quant field that contains the word *'u*nknown*' (no quotes)*.

This indicates to your software that you should not take any sync-related action for the symbol in question. You may receive an unknown quant for several reasons. The most important one is that the asset in question is in "sync-flux." This means that C2 is in the process of (or has very recently) noticed some activity for the trading system/symbol combination in question. An example of this is that C2 may have very recently filled an order for the symbol. In this case, the fill notification will not have had a chance to work its way to your client. Without the sync-flux filter, we would be faced with a large number of oscillating sync adjustments, as C2 reports a desired quantity, then you fill the quantity, then C2 reports an adjustment, and then you adjust, etc. By observing the quant=unknown status, we can eliminate most extraneous synchronization orders.

Finally, quant=unknown may also appear in cases when the markets for the symbol in question is closed. If the market is closed, you will not really be able to act on sync information anyway: if you send a market order to adjust the quantity of the symbol in the customer's account, it can't be filled, anyway. And if you're not careful, you will request the sync xml 120 seconds later, and issue a second, and then a third, and then a fourth market order! By the time the market opens, your customer may purchase thousands of incorrect shares of stock.

In summary, it's very important to honor the quant=unknown field. If you see this, you should not attempt to sync the asset in question, for any trading system.

Alerts and error messages

Alerts

C2 may need to deliver a message to the user of your client software. It will do this by sending the URL of a Web page containing an important alert. You may choose to parse the web page yourself and deliver the message within it to your customer, or you may simply open a Web browser to the requested page. This feature will never be used for marketing or advertising purposes, and will be used only for legitimate important communications (information about service outages, scheduled maintenance, etc.)

When C2 wants to send an alert to your customer, it will include a <browser> node within the returned XML datapacket. This node may appear in any returned <collective2>…</collective2> data packet. It is most likely to appear after your client calls cmd=latestsigs. The format is as follows:

<collective2>

…. Other information: new signals, fill acknowledgement, etc. ….

<browser>
<url>http://www.collective2.com/alertxxx.htm</url>
<type>REQUIRED</type>
</browser>

</collective2>

Currently, the C2 server will send only **REQUIRED** messages. These are messages which are important and thus must be relayed to your customer. In the future, we may choose to send **OPTIONAL** messages, which will contain less important information (upcoming upgrades, new features, etc) and thus may be ignored at your customer's preference.

Errors

If there is an error condition, the C2 server will change the <status> field from **OK** to **error**. Thus, the <collective2> packet will look like this:

```
<collective2>
        <status>error</status>
        <error>
                <code>ERRORTYPE</code>
                <url></url>
        </error>
</collective2>
```

Known error types:

MISSINGORINCORRECTPARAMS – Only sent during login, if your client sends an invalid protocol version, client type, or host id.

INCORRECTEMAILPW – Sent during login, if Collective2 email and password are unknown or do not match.

OLDCLIENT – This client/protocol version are no longer supported. The URL field will be filled in with a page where new software can be downloaded or an explanation can be issued. If you want to force this error message when an "old" version of your software checks in, please let Collective2 know which build numbers you want to trigger this flag.

NOSUBSCRIPTIONS – The C2 customer (identified by the email and password during login) has no valid subscriptions to trading systems, and thus will not receive any signals. The URL field will be filled in with a URL that takes the customer to a page where he can sign up for trading systems with free trial periods.

SESSION TIMED OUT – Your client did not contact the server for over 20 minutes. The session id is now invalid. You must re-login and request a new session id.

NEEDAGMT – Before an end-user can use the C2 AutoTrading platform, he is required to read and electronically sign an Agreement in which he acknowledges the risks of AutoTrading. No trades will be sent until this agreement is signed. When you receive a NEEDAGMT message, you will also be given a <url>. You should open a browser window and go to that URL. There, the customer will be able to read and (if he chooses) sign the agreement. In the meantime, you can continue polling C2, and will continue receiving the NEEDAGMT message. As soon as the customer signs the agreement, this error message will no longer be sent.

Other important information:

For trading information to appear, you need to be subscribed to a trading system, and you must have turned on AutoTrading Permissions for the trading system. This is done through the Collective2 Web site. There is currently no API available to set trading permissions for end-user customers.

And finally…

The C2 ATI is a work in progress. My goal is to make it as useful for programmers and traders as I can. I encourage software developers to build C2-compatible software, and to make money doing so (either by trading or selling the software).

Some of the best ideas and features in Collective2 have come from users and developers. If there is something that is missing – something that would make your life easier, or your software more powerful, I encourage you to email me and let me know. I try to be as responsive as possible, and I may surprise you by how quickly I can act on a valuable request. So please do not hesitate to contact me with questions or suggestions.

- Matthew Klein, Founder and Programmer, Collective2 Corporation - matthew@collective2.com.