crystal decisions
A SEAGATE COMPANY

Holos Analytic System Version 8

Holos Agents

Overview

This document will bring you quickly up-to-speed in the intricate area of Holos Agents; describing the different types, their features and their capabilities.

Please take the time to read from the start. The basics of Agents are explained and a simple example is introduced. More sophisticated techniques such as writing agent components are then explored. For reference information please refer to the Holos documentation set. Although this guide concentrates on the programmatic interface to Holos Agents, the Holos Agents dialog from the Analysis Tools desktop is also briefly described.

Contents

Introduction

Use the Introduction to explain an issue in further detail, provide background information and outline the Body of the paper.

The Basics

What are Agents and what can I do with them?

So, you have heard about Agents but do not know what to do with them, or what they can do for you. Very simply, an Agent is a Holos program that runs as a background task. The program can be executed immediately, or scheduled for a future time and date. The program can do just about anything except I/O which requires user intervention or the client front-end.

Example uses for Agents are overnight data loading and/or analysis, batch printing of reports, repetitive jobs etc; the obvious advantage being that you don't need to remain logged on. But don't limit your thoughts to traditional batch tasks. For example, as I write this two Agents are currently running on my PC, each populating and exporting two different structures while still allowing interactive access to Holos.

Agent Components

There are three distinct components, which together form an Agent: -

- A ***task*** – basically the Holos program which you want to run in the background
- A ***notification method*** – This is how the task will return its results, if any, or indicate that it has finished. (e.g. execute another Holos program, email, popup dialog etc)
- A ***service*** – this manages and runs the Agent and is usually the host batch system (e.g. AT on Windows NT, cron on Unix, etc)

In addition to this, each agent must have a scheduling plan so that the service knows when the Agent should be executed.

When Holos is installed standard tasks, notifications and services are registered to enable Agents to be setup quickly and easily. The standard notification methods are 'email' or 'none' and the standard service is 'local'. This simply means that the host platform's native batch system will be used. The tasks will be discussed later… but first let's see some action!

A Simple Example

One of the standard tasks is the 'process' task. This will submit a Holos HL script to the batch system. The following example shows how to setup the most basic Agent which uses the 'process' task, the 'local' service, has no notification and executes immediately.

Each task, notification, service and schedule plan has an associated record that is used as a means of communicating what needs to be done. Normally the defaults can be used, but look at the task record `process_rec` and the schedule plan record `schedule_rec` below. These tell the process Agent the name of the HL script to execute and that the script is to be executed immediately.

```
!--- AGENT_SHELL.HL

!--- Load the Agent Toolkit.

IF NOT DEFINED( ag$_unload ) THEN
    EXECUTE HOSTFILE "holos_agents:agents.hl"
END IF

!--- Define the task record
!--- and specify the program to be executed.

IF DEFINED( process_rec ) THEN
    DELETE RECORD ~"process_rec"
END IF

RECORD process_rec <SOURCE>
    INCLUDE HOSTFILE "holos_agents:process.recinc"
END RECORD

process_rec.$hl_script = "task1.hl"

!--- Define the service record

IF DEFINED( service_rec ) THEN
    DELETE RECORD ~"service_rec"
END IF

RECORD service_rec <SOURCE>
    INCLUDE HOSTFILE "holos_agents:local_srv.recinc"
END RECORD

!--- Define the notification record.
!--- In this instance its empty.

IF DEFINED( notification_rec ) THEN
    DELETE RECORD ~"notification_rec"
END IF

RECORD notification_rec <SOURCE>
    INCLUDE HOSTFILE "holos_agents:none_notify.recinc"
END RECORD

!--- Define the schedule record, specify the NOW time plan.

IF DEFINED( schedule_rec ) then
    DELETE RECORD ~"schedule_rec"
END IF

RECORD schedule_rec <SOURCE>
    INCLUDE HOSTFILE "holos_agents:agsched.recinc"
END RECORD

schedule_rec.$plan_type = "NOW"

!--- Create an Agent called 'Task n' and invoke it.

WRITE ag$_invoke_agent( "Task", "process", &
  "process_rec",   "local", "service_rec", &
  "none", "notification_rec", "", "schedule_rec" )
```

Note the line above. The function `ag$_invoke_agent` both creates the Agent and submits it to the batch system. If the Agent is successfully submitted, the function returns the name of the Agent. The syntax is: -

```
ag$_invoke_agent( <agent prefix>, <task>, &
    <task record>, <service>, <service record>, &
    <notification>, <notification record>, &
    <user defaults>, <schedule plan> )
```

Also note that the task, service, notification and schedule plan records must have the `<SOURCE>` attribute applied to them, as shown above. Below is the example HL script that the Agent calls. Because we have no notification, this script simply creates a file, to indicate that the Agent has worked.

```
!--- TASK1.HL

IF NOT DEFINED( hostfile_exists ) THEN
    EXECUTE "holos_support:hostfile_exists.fnc"
END IF

IF hostfile_exists( "task1.out" ) = 1 THEN
    DELETE HOSTFILE "task1.out"
END IF

CHANNEL out HOSTFILE "task1.out" CREATE

WRITE out "I've finished!"

DELETE CHANNEL out
```

Agent Scheduling

Although the above example submits the Agent immediately, many Agents will be scheduled to run at a later time and/or date. So, let's examine the schedule record closer. The file `hostfile "holos_agents:agsched.recinc"` contains the following:-

```
TEXT $plan_type
TEXT $year_plan
TEXT $month_plan
TEXT $day_plan
TEXT $weekday_plan
TEXT $hour_plan
TEXT $minute_plan
```

As of Holos v6.0 there are two schedule plans; NOW and TIMED. If the variable `$plan_type` is set to "NOW" then all other variables in the schedule record are ignored and the Agent is submitted immediately. However, if `$plan_type` is set to "TIMED" then the schedule time will be determined from the settings in the remaining variables. For example, to schedule an Agent for 7:30am on 7th June 1998, the following would be used:-

```
$plan_type = "TIMED"
$year_plan = "1998"
$month_plan = "6"
$day_plan = "7"
$weekday_plan = "ALL"
```

```
$hour_plan = "7"
$minute_plan = "30"
```

NOTE	In Windows NT, scheduling an Agent with a "NOW" time plan will actually schedule the Agent to execute within 2 minutes.

But what if you want to schedule a job to run on Monday, Wednesday and Friday, every other hour between 9:00am and 5:00pm, during January, February, March and June of 1998, 1999 and 2000. Clearly this is an esoteric example, but possible in Holos using the following:-

```
$plan_type = "TIMED"
$year_plan = "1998:2000"
$month_plan = "January:March;6"
$day_plan = "ALL"
$weekday_plan = "1;Wednesday;5"
$hour_plan = "9:17/2"
$minute_plan = "0"
```

Notice how both numbers and text can be used to represent the month and days, ';' is the separator for multiple entries and ':' specifies a range[1]. In the `$hour_plan` variable, '/2' indicates a two hour step. There are two special variables; "ALL" specifies all values that are in range and "LAST" is used only with the `$day_plan` variable to specify the last day of the month. There are many more examples in the on-line help under "Agents/General/Scheduling plan/Timed/Examples".

NOTE	Note that in Holos v5.x the range specifier is '-'. This has been changed in Holos v6 and above to the standard Holos range specifier of ':', though '-' remains for compatibility.

Summary

At this point you now know that the components of an Agent are a ***task***, a ***notification*** and a ***service*** and that it requires a ***scheduling plan***; which can either be "NOW" or "TIMED". You've seen a simple Agent working and know how to submit an HL script using the 'process' task. Let's now examine the other standard tasks.

Using Standard Tasks

There are currently four standard tasks that are registered when Holos is installed: 'process', 'file watch', 'data pattern search' and 'parallel consolidation'.

Each task has a registered name (e.g. 'process'), an HL script which defines it (e.g. `HOSTFILE "holos_agents:process.hl"`) and where appropriate a user exit (e.g. `HOSTFILE "holos_agents:process_ux.hl"`) and a

record for passing parameters (e.g. `HOSTFILE "holos_agents:process.recinc"`).

File Watch

This very simple task is probably one of the commonest uses of Agents. Many large organisations have mainframes where data extracts are run and the resultant files are transferred onto a Unix machine, for example. The 'file watch' task can monitor this file for changes and start a Holos HL script… which could even be another Agent.

The file to monitor is passed to the 'file watch' task by a record like:-

```
RECORD file_watch_rec <SOURCE>
         INCLUDE HOSTFILE "holos_agents:file_watch.recinc"
END RECORD

file_watch_rec.$file_spec = "datafile.dat"
```

Process

To submit a Holos HL script as an Agent quickly, without writing a specific Agent, use the 'process' Agent. This has already been demonstrated in the previous section. The HL script that the Agent should execute is passed to the 'process' task by a record like:-

```
RECORD process_rec <SOURCE>
    INCLUDE HOSTFILE "holos_agents:process.recinc"
END RECORD

process_rec.$hl_script = "my_prog.hl"
```

Data Pattern Search

The 'data pattern search' task is linked to the data-mining tool in the Analysis Tools Desktop and parameters are normally passed to it via the Data Mining dialogs. Consequently the 'data pattern search' task has no corresponding parameter record or user exit script. However a programmatic interface is available which is accessed differently to the previous tasks… but which is still simple. First, a brief explanation of Agent user exits.

User Exits

A word of warning! Don't be misled by the name "user exit". This code is NOT executed after an Agent has finished. It is executed immediately the Agent is submitted. The main task of a user exit is to setup a default parameter record for the Agent to use if none exists and to verify the data being passed to the Agent, before it executes. For example the 'process' user exit script verifies that the HL script it is asked to execute actually exists.

The 'data pattern search' task is defined by `HOSTFILE "holos_agents:dmin.hl"`. Within this script the macro `dmin$_pattern`

is defined which accepts a record as a parameter. This record contains a plethora of options and parameters. Rather than describe all these, please refer to the Holos on-line help for “Help/Utilities/DMIN.HL”. Instead, the following example will show the complete user exit:-

```
!--- DMIN_UX.HL

RECORD my_search
    TEXT $tuple = "{Calendar.*,Country.USA;Japan;UK, &
                      Sales, Products}"
    TEXT $y_dim = "Country"
    TEXT $x_dim = "Calendar"
    NUMBER $type = 2 ! Look for slopes
    NUMBER $perc = 2 ! Choose mean of series
    NUMBER $change = 0.8 ! Gradient
    NUMBER $step = 15 ! Ramp exists for duration 15 steps
    NUMBER $move = 5              ! Moving average of 5
    NUMBER $fo = 0.5              ! Exponential Smoothing
END RECORD

ag$_user_exit_report( 0 )  ! Exit with success code
```

As the ‘data pattern search’ task has no user exit defined as standard, it must be deregistered and then registered with the new user exit. This is achieved by:-

```
ag$_deregister_task( "data pattern search" )
ag$_register_task( "data pattern search", &
    "holos_agents:dmin.hl", &
    "", "dmin_ux.hl", "", "" )
```

Parallel Consolidation

The ‘parallel consolidation’ task provides a means of performing a parallel consolidation of a compound structure on the specified service, with the results of the consolidation being sent to the notification service.

The details of the structure are passed to the ‘parallel consolidation’ task by a record like:-

```
RECORD compound_str_rec <SOURCE>
    INCLUDE HOSTFILE "holos_agents:msa_agent.recinc"
END RECORD

compound_str_rec.$struct = "all_my_data"
```

The record contains many more variables that can be changed to modify the operation of the parallel consolidation, e.g. the names of the nodes (processors) that can be used, and notify diagnostic results.

Parallel Calculation

The ‘parallel calculation’ task is identical to the ‘parallel consolidation’ task, except that it performs a parallel calculation of a compound structure on the specified service, with the results of the calculation being sent to the notification service.

Like the ‘parallel consolidation’ task, the details of the structure are passed to the ‘parallel calculation’ task by a record like:-

```
RECORD compound_str_rec <SOURCE>
    INCLUDE HOSTFILE "holos_agents:msa_agent.recinc"
END RECORD

compound_str_rec.$struct = "all_my_data"
```

House-keeping

Although Holos has a number of registered tasks, notifications and services when it is installed, any or all of these can be deregistered. Therefore, when using any of the standard tasks, notifications and services, a quick housekeeping check should be performed to ensure that they exit. For example:-

```
!--- Although the following are registered when Holos is
!--- installed, this house-keeping checks that they
!--- haven't been deleted.

IF LOCATE( "process", ag$_list_tasks() ) = 0 THEN
  ag$_register_task( "process", "holos_agents:process.hl",
&
  "", "", "", "" )
END IF

IF LOCATE( "local", ag$_list_services() ) = 0 THEN
    ag$_register_service( "local",  &
        "holos_agents:local_srv.hl", "", &
        "", "", "" )
END IF

IF LOCATE( "none", ag$_list_notifications() ) = 0 THEN
    ag$_register_notification( "none", &
        "holos_agents:none_notify.hl", &
        "", "", "", "" )
END IF
```

Creating New Agent Components

The standard Agent components (i.e. tasks, services and notifications) are useful not only in themselves but also as a guide to developing new components. The standard 'email' and 'none' notifications may not be useful to many users. However, a popup dialog that displayed the agent results file would be much more useful and, as an example, this is the notification that we will now develop.

Specifically, our notification will monitor a notification file, which is a copy of the results file created by the agent task. Once the task is complete, the first 100 lines of the notification file will be displayed as a simple dialog.

Our notification will have the following definition:-

Registered name	dialog popup
Defined	dialog_notify.hl
User Exit	dialog_notify_ux.hl
Record	dialog_notify.recinc

When an Agent is submitted, the sequence of events is that the agent task user exit and notification user exits are executed. All tasks and user exits must be wholly contained in one script. Our notification user exit executes some support files, defines a configuration record for the notification, defines a dialog record to pass information to the dialog and defines two functions; `ag_notify` which displays the dialog and `dialog_notify_user_exit` which initialises the dialog notification method.

The function `dialog_notify_user_exit` is called by the user exit and this function defines an event that monitors the notification file for modification. When the agent task completes, "dialog_notify.hl" is executed which defines and calls the function `dialog_notify`. This function copies the agent results file to the notification file. At this point the event identifies a change to the notification file and calls the function `ag_notify`, which displays the first 100 lines of the results file as a dialog. The code is below.

```
!--- DIALOG_NOTIFY_UX.HL

IF NOT DEFINED( gen_name ) THEN
    EXECUTE "holos_support:gen_name.fnc"
END IF

IF NOT DEFINED( hostfile_exists ) THEN
    EXECUTE "holos_support:hostfile_exists.fnc"
END IF

IF NOT DEFINED( get_info ) THEN
    EXECUTE "holos_support:get_info.mcr"
END IF

IF NOT DEFINED( ev$_allocate ) THEN
    EXECUTE "holos_support:events.fnc"
END IF

!--- We need to leave this record and function in memory as
!--- it will be called when the notify file appears
```

```
IF DEFINED( notify_dlgrec ) THEN
    DELETE RECORD ~"notify_dlgrec"
END IF

IF DEFINED( ag_notify ) THEN
    DELETE FUNCTION ~"ag_notify"
END IF

!--- Dialog record for displaying results information

RECORD notify_dlgrec
    TEXT 'ResultsList.Class' = "List_1"
    TEXT 'ResultsList.Value' = ""
    TEXT 'AgentLabel.Value' = ""
END RECORD

CHANNEL in

!--- Given an agent and results file, this function will
!--- populate the dialog with the first 100 lines of the
!--- log.


FUNCTION ag_notify( TEXT file_name, TEXT agent_name, &
                    NUMBER event_no)

    TEXT message = ""
    TEXT line
    NUMBER max_lines <NO MODIFY> = 100
    NUMBER counter = 0

    CHANNEL in HOSTFILE ~file_name READ

    _STATUS = 0

    READ in line

    LOOP WHILE _STATUS = 0 AND counter <= max_lines
        counter = counter + 1
        message = message + "," + line
        READ in line
    END LOOP

    IF _STATUS = 0 THEN
        line = ",,,ONLY THE FIRST "
        ADD max_lines <FIXED 0>, line
        line = line + " LINES OF THE RESULT ARE SHOWN"
        message = message + line
    END IF

    CHANNEL in

    message = message - ","

    !--- Populate dialog with agent results

    notify_dlgrec.'ResultsList.Value' = message
    notify_dlgrec.'AgentLabel.Value' = agent_name

    CALL DIALOG "ag_dlg.hds.notify" USING RECORD &
                    ~"notify_dlgrec"

    DELETE HOSTFILE ~file_name

    EVENT event_no DELETE

END FUNCTION

!--- Configuration record for the dialog notification user
!--- exit.
```

```
RECORD dialog_notify_user_exit_info<SAVE, SOURCE>
!Requires SAVE&SOURCE

    INCLUDE "dialog_notify.recinc"      !TEXT notify_file

END RECORD

!--- This function is called when an agent using 'dialog
!--- popup' notification is invoked. It executes prior to
!--- the main agent task running.

FUNCTION dialog_notify_user_exit( TEXT reason, &
                                  TEXT agent_name)

  TEXT notify_file
  TEXT dir

  IF reason = ag$_ux_invoke THEN

  !--- If no information record has been passed then we
  !--- will create one

    IF ag$_user_exit_info = "" THEN

      !--- Need to create a new file to use in
      !--- Notification (use gen_name until a new name
      !--- is created)

      get_info directory "dialog_notify_user_exit.dir" TEXT
      notify_file = dir + "dlg_not" + gen_name() + ".tmp"

      LOOP WHILE hostfile_exists( notify_file ) <> 0
        notify_file = dir + "dlg_not" + gen_name() + ".tmp"
      END LOOP

      ag$_user_exit_info = "dialog_notify_user_exit_info"

      dialog_notify_user_exit_info.notify_file =
notify_file

    ELSE

      DELETE RECORD ~"dialog_notify_user_exit_info"

    END IF

    IF ev$_allocate( TRUE ) = 0 THEN

      ERROR "No event available to monitor agent
finishing."
      ERROR "Agent will not be scheduled"

      !--- Tell Agent framework we are NOT happy
      ag$_user_exit_report( 1 )

    ELSE

      !--- Initialise dialog calling function and set up &
      !--- event to monitor the file

      EVENT ev$_allocate MODIFY HOSTFILE ~notify_file, 5
      WHENEVER EVENT ev$_allocate ag_notify( agent_name, &
                   notify_file, ev$_allocate )

      !--- Tell Agent framework we are happy
      ag$_user_exit_report( 0 )

    END IF

  ELSE
```

```
    !--- Not ag$_invoke_ux so just continue
    ag$_user_exit_report( 0 )

  END IF

END FUNCTION


dialog_notify_user_exit( ag$_user_exit_reason,
ag$_user_exit_agent_name )

DELETE FUNCTION dialog_notify_user_exit

!--- DIALOG_NOTIFY.HL

FUNCTION dialog_notify( TEXT agent_name, TEXT notify_info,
&
                        TEXT results_file )

    TEXT file_name

    COPY TEXT ~notify_info.notify_file file_name

    !--- Remove the notify file if it exists
    IF hostfile_exists( file_name ) = 1 THEN
        DELETE HOSTFILE ~file_name
    END IF

    !--- Copy the results file to the dialog file then
    !--- remove the results file
    IF hostfile_exists( ag$_fwk_results_file ) = 1 THEN
        COPY HOSTFILE ~ag$_fwk_results_file, &
               HOSTFILE file_name
        DELETE HOSTFILE ~ag$_fwk_results_file
    END IF

    !--- Indicate notification went OK
    ag$_notify_complete()

END FUNCTION


dialog_notify( ag$_fwk_agent_name, ag$_fwk_notify_info, &
               ag$_fwk_results_file )

DELETE FUNCTION dialog_notify

!!! Holos Dialog Script AG_DLG.HDS !!!
DialogSpec
  Dialog
    Name notify
    Origin Holos
    Caption "Holos"
    Font "MS Sans Serif" - 8 -
    Location 1;39
    Size 345;320
    Control
      Name ResultsList
      Type List
      Justify T+L;-;0
      Location 6;36
      Size 333;248
    End Control
    Control
      Name AgentLabel
      Type Label
      Text "Label1"
      Justify M+C;M+C;4
      Location 5;9
```

```
      Size 333;18
    End Control
    Control
      Name Button1
      Type Button
      Text "OK"
      Justify T+L;-;0
      Location 134;287
      Size 65;26
      OnClick
        Dismiss(True)
      End OnClick
    End Control
  End Dialog
End DialogSpec
```

To register and deregister our new notification method, use the following:-

```
! De-register dialog notification
  ag$_deregister_notification( "dialog popup" )

! Register dialog notification
  ag$_register_notification( "dialog popup", &
     "dialog_notify.hl", "", "dialog_notify_ux.hl", "", ""
)
```

Agent Definitions

The above code demonstrates the use of some variables and agent definitions that are available for use.

It is also possible to interrogate the agent repository about registered components and invoked Agents. For example `ag$_list_agents()` returns a comma-separated list of invoked Agents and `ag$_list_tasks()` returns a list of registered Agent definitions. The agent framework provides a number of definitions that are available for read-only use. For example `ag$_fwk_agent_name` contains the name of the Agent and `ag$_fwk_results_file` contains the name of the file which contains the output from the Agent task.

Agent Dialog

The Agent Dialog System provides an intuitive, graphical user interface to setting up standard and custom agents. It can be invoked from the agent icon on the Analysis Tools desktop, or programmatically by:-

```
EXECUTE "holos_agents:agents_dialog.hl"
```

```
ag$_manage_invoke()
```

Once entered, a dialog will list all current agents with the option of displaying their properties, deleting them or creating a new agent. When creating a new agent a dialog prompts you for the task, notification method and service required… with a configuration button available for each component, depending on the choice made. The schedule plan is defined by a further dialog where the choice can be made to execute the agent now, daily, on a certain day/date and time, monthly, quarterly, annually etc. The Agent Dialog System makes it easy.

Environment Variables

There are a number of directories associated with Agents that have been assigned to environment variables. Knowing these variables, and consequently where to look, may help you in tracking down any problems. The examples below are for a Windows NT server installation.

```
holos_agents=c:\holos\server\user\agents\
holos_agent_ctl=c:\holos\agents\control\
holos_agent_def=c:\holos\agents\def\
holos_agent_notify=c:\holos\agents\notify\
holos_agent_srv=c:\holos\agents\services\
holos_agent_tasks=c:\holos\agents\tasks\
holos_agent_temp=c:\holos\agents\temp\
```

The `holos_agent_tasks`, `holos_agent_notify` and `holos_agent_srv` directories contain the definitions of the standard tasks,

notification methods and services respectively. The `holos_agent_ctl` directory stores locking information while the `holos_agent_def` directory stores the `.agf` and `.agr` definition files. The `holos_agent_temp` directory is a temporary filestore and the `holos_agents` directory is the main Agents code directory.

Reminder

If you've managed to read this in one go, congratulations! A lot of information has been given to you but this section will remind you of the important points, with tips if everything isn't going quite as planned.

An agent comprises three components:-

1. ***Task*** - What the agent does
2. ***Notification*** – Where the agent tells you it has finished
3. ***Service*** – How the agent executes

Additionally, an Agent must have a ***scheduling plan*** to tell it when to execute. This is done via a scheduling record which is created by defining a record and INCLUDE-ing `HOSTFILE "holos_agents:agshed.recinc"`.

Remember that all relevant task, notification, service and scheduling records must be defined with the `<SOURCE>` attribute.

Don't be confused by the term "user exit". The "user exit" actually executes immediately BEFORE the agent does and is used to set any appropriate parameters.

Agents can fail for a variety of reasons. Assuming that there are no coding errors with the agent itself, the common problems are:-

- Your userid does not have sufficient privilege to use the host batch system. Many customers will have a user account, developer account and an administration account for each application. If appropriate, use the administration account to submit the agent.

The Agent sub-system was not installed correctly. Check that all the Holos environment variables map to the correct drive. If there are problems the Agent sub-system can be re-installed using the install script in the `holos_agent_def:` directory. Please refer to the Holos Installation Guide before manually running this script.

Contacting Crystal Decisions for Technical Support

Along with this document, and the *Crystal Holos User Manuals*, we recommend that you visit our Technical Support web site for further resources and sample files. For further assistance, visit us at the web sites below.

Technical Support web site:

http://support.crystaldecisions.com/homepage/

Answers By Email Support:

http://support.crystaldecisions.com/support/answers.asp

Phone Support:

Tel: (0208) 231 0606

Tel: 1 877 779 8326 N.America