Datacraft AMX Interface Programming

Guide to this documentation

Introduction

The room booking system programming consists of the following components:

- **The Datacraft API** - This forms part of the datacraft server and provides an interface for external control / applications to control the service. This should be already setup as part of the Datacraft installation and have an address / port definition supplied to you which you will need to input in the code. This will probably be in the form of:
    - datacraft.example.com:8002
    - 192.168.1.10:8002
- **The AMX Duet API to SNAPI interface module** - This is the built Java module which simply interfaces between the above API and the AMX standard SNAPI commands. This handles all communications and sessions between the AMX and the Datacraft API. It will take commands in the standard SNAPI format and also provide feedback. In out programming example though you will not need to do much with this as their already functions defined in the controllers code which does the data processing part of this for you.
- **The Datacraft UI Controller** - This is a module which communicates between the Duet module's virtual device and another virtual device in order to control a room booking panel. All of the logic of how the panel works is done by this module. This will handle all requests of new bookings and the list of current bookings and display them on the room booking touch panel. All other forms of control externally to this, ie from the touch panel in the room, should be routed from this module's virtual device. You will probably want to do this for the extend meeting / meeting end notifications.
- **The main AMX program** - All user code and other modules should be in here.

Example Programming

In the example workspace provided there is a touch panel configuration for a room booking panel. This is defined and passed directly to the Datacraft UI Controller.

The 2nd touch panel configuration is an example AV control panel which you would locate in the meeting room and use to control the system. It's logic is controlled in the main programming or, if you wish, you could create another module to control this.

Included with the code is several pre-built libraries which we have used to make a few things easier. You don't have to use these if you don't want to but you will have to make some changes if you don't use them. For example the UI Kit library makes a few functions available in order to assign a 'key' to a touch panel and then perform commands on it. It's useful because you then don't have to remember all the commands used for TP control.

The UI controller itself has been built as an example to show how to handle the data provided by the Duet module. What it does is pull out, every minute, an array index of meetings for the day, for the defined roomkey. It then can work out and display on the UI, the current and next meetings and also knows how to handle ad-hoc booking requests and the amount of slots of available times should it be available.

The functions used in the UI controller are defined in the various include files. They have functions to send and receive commands to the Duet module and listen for events such as replies for user logons or create meeting requests. Also included are functions to handle the data received and process and sort it.

In a typical request the UI controller will request a list of meetings from the Duet module. The Duet module will request this from the Datacraft API, creating and getting a session ID if needed, and forward it back to the UI controller module. The reply is processed and stored in a struct where it is ready for being used on the touch panel. Once ready the controller will then make a call to update the UI output and work out the current meeting, the next meeting and the remaining minutes of the current meeting. It displays this on the room booking panel and also, if needed, sends the remaining minutes back out on the virtual device. You can then use this to display a message to the meeting host in the meeting room. Included here also is the next meeting info so you can make a choice between offering a 'meeting will end soon warning' and a 'meeting will end soon, do you wish to extend option?'.

Defining the modules

When you are declaring the modules in your main file you must pass the necessary data and components correctly to them.

The AMX Duet API to SNAPI Interface

```
DEFINE_MODULE 'DatacraftHospitality_dr1_0_0' mDatacraft (
      vdvDatacraft_DuetApiInterface,
      datacraftIPAddress )
```

Here we define the Duet module with an instance name of `'mDatacraft'` and pass into the following compenents:

1. The Duet virtual device name which will be defined previously. This must be a device defined in the range of addresses suitable for Duet devices i.e. 41001:1:0
2. A variable char string containing the URL of the Datacraft server and port such as `'192.168.1.11:9003'` or `'demo.datacraftdesign.com:9003'`

The Datacraft UI Controller example

```
DEFINE_MODULE 'Datacraft UI Controller v1-01' mDatacraftUI (
      vdvDatacraft_DuetApiInterface,
      vdvDatacraftController,
      uiBookingPanels )
```

You don't have to use this but it's ready to use if you like and can control several room booking panels and also a single room for AV control. We define it with an instance name of mDatacraftUI and pass into it the following components:

1. The Duet virtual device name (as above).
2. The Virtual Device used as a controller for our controller interface. We can define this as 33001:1:0 or similar.
3. A variable array of user interface devices we wish to use as dedicated room booking panels.

System model

The following diagram shows how the program manages the flow of information and requests:

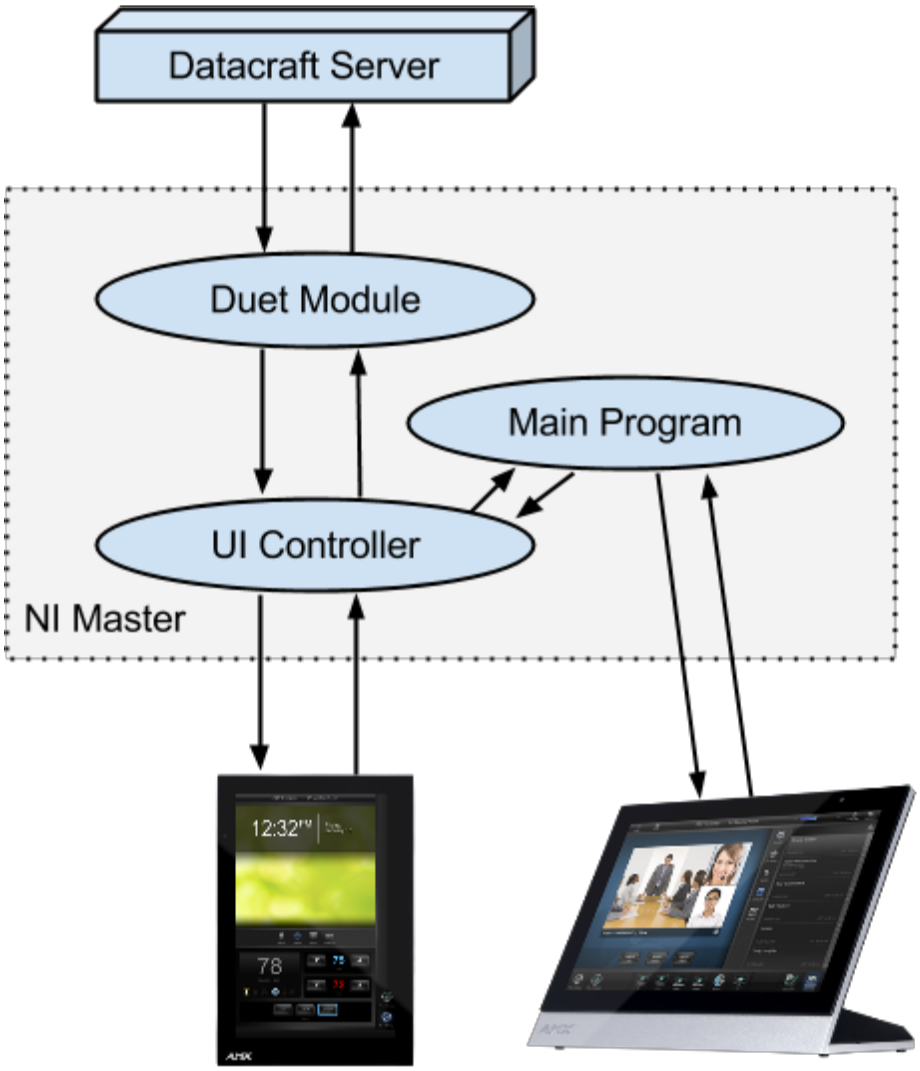

Figure 1 - Programming functionality flow

Controller Commands

In our programming example we see the above model being used to keep the room booking panel code away from the main system code. The only interaction you need is the few commands to the UI Controller for in room notifications of the meeting status and options such as extending the meeting. These commands are detailed here.

Status Update Commands:

For notifications from the controller, the controller uses the following via SNAPI format to send information back to the main program via the virtual device.

CURRENT_MEETING_INFO

This command will contain information about the current meeting and is only sent if there is a meeting in progress. It will automatically be updated as the room booking panel is updated.

Command paramters:

1. Room ID — `integer`
2. Meeting ID — `integer`
3. Meeting Start Time — `char[]` (timestamp)
4. Meeting End Time — `char[]` (timestamp)
5. Number of minutes remaining — `integer`
6. Meeting State ID — `integer`
7. Meeting Owner Name / Host — `char[]`

NO_CURRENT_MEETING

This command is sent instead of the `CURRENT_MEETING_INFO` command if there is currently no active meeting in progress.

EXTEND_MEETING-TIMES

This command is a repsonse to the `'?EXTEND_MEETING_TIMES'` command. It will come back several times with 2 different sets of parameters so long as the first has a count of more than 0. It gives a list of the available times for you to populate some button options on the main touch panel in the meeting room.

Command paramters:

On the first repsone it will come back with the following:

1. Fixed Command `'COUNT'` — `char[]`
2. Count value — `integer`

So long as count > 0, you will get the command with the following arguments ‘count’ number of times.

1. Fixed Command `'BUTTON'` — `char[]`
2. Index for the button array — `integer`
3. Time as text for button — `char[]`

User Commands:

For you to control the active meeting from the room touch panel you can use the following commands to send to the UI Controller:

START_MEETING

Use this to start the current meeting. You may call this at a point such as when the user starts the system from the room control panel.

Command paramters:

1. Meeting ID `integer`

END_MEETING

Use this to end the current meeting early. You may call this at a point such as when the user performs a system shutdown from the room control panel.

Command paramters:

1. Meeting ID `integer`

?EXTEND_MEETING_TIMES

Use this to get the available extend time options which you can use to populate a button array and present to the user on the room control panel. See the EXTEND_MEETING_TIMES response for it's reply format.

Command paramters:

1. Room ID `integer`

EXTEND_MEETING_SELECT_TIME

Once the button has been selected from the above button array you generated, send the selected button array index back using this command. This will attempt to extend the meeting to the selected time.

Command paramters:

1. Button Index `integer`

Setup Commands

The following commands are used to setup the room booking panel and it's options:

SET_ROOM_ID_FOR_UI

Use this to set the Room ID for the room booking panel at the specified index of touch panel device passed to the Datacraft UI Controller.

Command paramters:

1. Device Index `integer`
2. Room ID `integer`
3. Collection ID `integer`

ENABLE_ADHOC_BOOKING

Set the touch panel device at the specified index to allow ad-hoc bookings from users.

Command paramters:

1. Device Index `integer`
2. Bool Value (1 or 0) `integer`

ENABLE_END_MEETING_BUTTON

Set the touch panel device at the specified index to allow people to end meetings externally on the room booking panel. You may wish to disable this if the system has a touch panel in the room.

Command paramters:

1. Device Index `integer`
2. Bool Value (1 or 0) `integer`

System Requirements

Due to certain limitations of firmware compatibility, the Duet module is not currently compatible with version 4.x masters. This will cause a problem when using upgraded NI Controllers or DVX / DGX masters.

If you are not running version 4 you can use a single processor to host everything on including several room booking panels and also any AV system control. (see Figure 2)

It's recommended that if you want to run several room booking panels that you simply allocate a processor for room booking only. If you want to run AV control and room booking then you should host only the single room booking panel for the room.

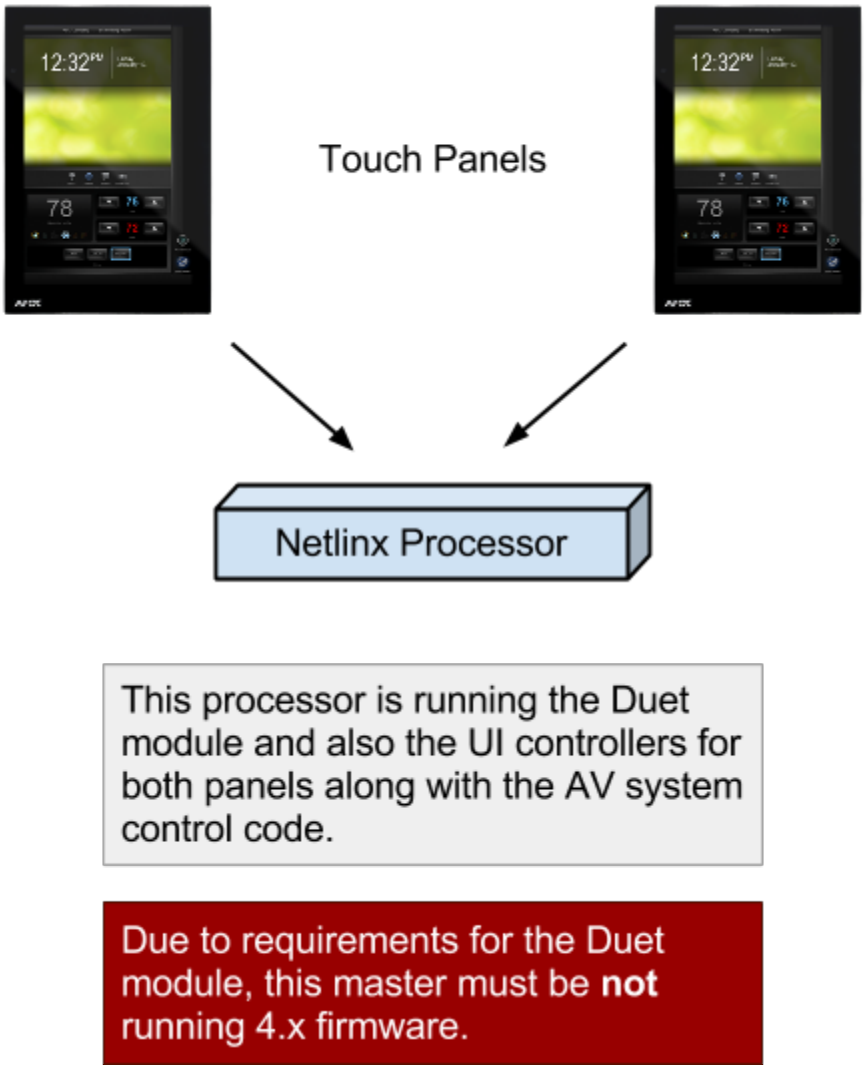

Figure 2 - Processor configuration for v3.x NI Masters

If you have a processor running v4.x firmware such as a DVX then you need to run the Duet module on another processor which has v3.x firmware. You then simply add a URL connection between the 2 processors, making sure they both have different system IDs, and define the Duet module on the remote processor.

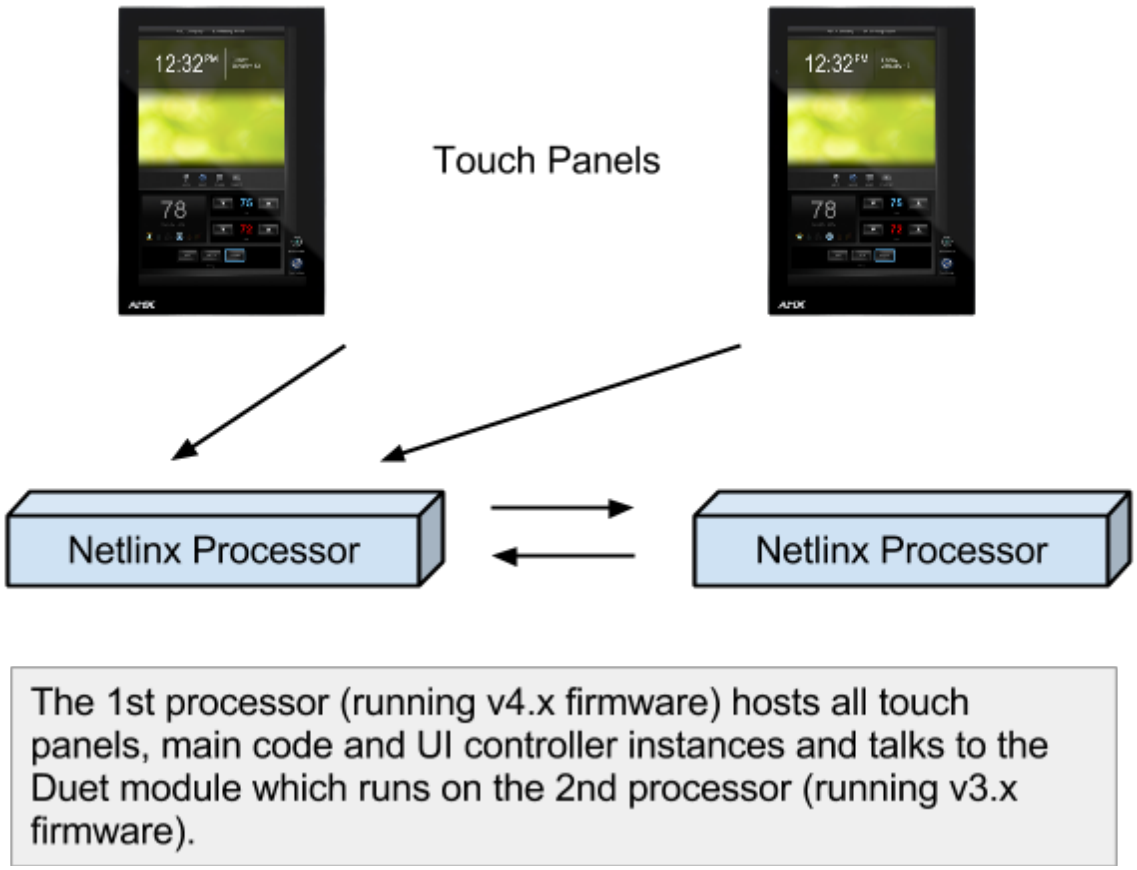

Figure 2 - Processor configuration for v4.x NI Masters

When using a configuration of this type you don't need a 2nd processor for every v4.x processor. You could install multiple of instances of the Duet module on the v3.x processor and then link this with multiple systems.

Datacraft API Functions

This documents the following functions included in 'DatacraftTouchPanelAPI.axi' which is the include used in the UI Controller. In theory you shouldn't have to use these commands as it is all done

User Functions

The following user functions can be called by the user. In the case of our example code they are called by the UI Controller code when needed. Most of them will usually follow by a call to an event function *(see further on for section on Events)*

DC_StartSession

This is called to start a session between the Duet module and the Datacraft API.

Required Arguments: **None**

Returns: **None**

Expected event call:

- **DCEvent_SessionAquired**

DC_UserLogin

Call to authenticate a user. In our example we only use a username and the password is not required.

Required Arguments:

1. **user** `char[]` username for the login
2. **pass** `char[]` password for the login

Returns: **None**

Expected event calls:

- **DCEvent_UserLoginSuccess** called on successful user login
- **DCEvent_UserLoginFail** called if login error has occured
- **DCEvent_AuthenticationError** called after 3 unsuccessful logins

DC_GetUserDetails

This is called to start a session between the Duet module and the Datacraft API. You can call this from the resulting user id returned by the **DC_UserLoginSuccess** event.

Required Arguments:

- **userid** `integer` unique user id of the user

Returns: **None**

Expected event call:

- **DCEvent_UserDetails**

DC_GetRoomDetails

This is called to start a session between the Duet module and the Datacraft API. You need the room id from the Datacraft setup. In our example we add this from main by calling a function to our UI controller at startup.

Required Arguments:

- **roomid** `integer` unique room id of the room

Returns: **None**

Expected event calls:

- **DCEvent_RoomDetails**

- **DCEvent_DefRoomLayout**

DC_GetBookings

Call to get a list of bookings within a specified time period.

Required Arguments:

- **roomid** integer — unique room id of the room
- **timezone** char[] — timezone eg “GMT Standard Time”
- **periodstart** char[] — time of the start of the period
- **periodend** char[] — time of the end of the period

Returns: **None**

Expected event calls:

- **DCEvent_RoomBookins** — called with the count of the result
- **DCEvent_DefineRoomBooking** — repeatedly called to the count of the above value

DC_AddBooking

Call to get a list of bookings within a specified time period.

Required Arguments:

- **roomid** integer — unique room id of the room
- **timezone** char[] — timezone eg “GMT Standard Time”
- **starttime** char[] — start time of the meeting
- **endtime** char[] — end time of the meeting
- **userid** integer — result user id from the login function
- **collectionid** integer — collection id of the group the room id belongs in
- **title** char[] — title of the meeting to be created

Returns: **None**

Expected event calls:

- **DCEvent_AddBookingOk** — called when the booking is created
- **DCEvent_AddBookingError** — called if there is an error

DC_AddBookingEx

As above but extended arguments.

Required Arguments:

- **roomid** integer — unique room id of the room
- **timezone** char[] — timezone eg “GMT Standard Time”
- **starttime** char[] — start time of the meeting
- **endtime** char[] — end time of the meeting
- **userid** integer — result user id from the login function
- **collectionid** integer — collection id of the group the room id belongs in
- **title** char[] — title of the meeting to be created
- **meetingtypeid** integer — type of meeting to create other than the default

Returns: **None**

Expected event calls:

- **DCEvent_AddBookingOk** — called when the booking is created
- **DCEvent_AddBookingError** — called if there is an error

DC_MeetingStart

Call to start a meeting. If this is not called after the meeting start time has passed then the Datacraft system may auto end the meeting and shrink the end time to let others book the room.

Required Arguments:

- **userid** `integer` userid - optional, can be 0
- **meetingid** `long` meetingid of the meeting to start

Returns: **None**

Expected event calls:

- **DCEvent_MeetingState**

DC_MeetingEnd

Call to end the meeting early.

Required Arguments:

- **userid** `integer` userid - optional, can be 0
- **meetingid** `long` meetingid of the meeting to start

Returns: **None**

Expected event calls:

- **DCEvent_MeetingState**

DC_MeetingExtend

Call to end the meeting early.

Required Arguments:

- **userid** `integer` userid - optional, can be 0
- **meetingid** `long` meetingid of the meeting to start
- **newendtime** `char[]` requested change of end time

Returns: **None**

Expected event calls:

- **DCEvent_MeetingExtended**

DC_GetMeetingTypes

Call to get a list of meeting types. End of list is indicated by index=0.

Required Arguments: **None**

Returns: **None**

Expected event calls:

- **DCEvent_DefineMeetingType**

DC_GetMeetingStates

Call to get a list of meeting states. End of list is indicated by index=0.

Required Arguments: **None**

Returns: **None**

Expected event calls:

- **DCEvent_DefineMeetingType**

Event Functions

The following event functions are called by the virtual device event listener and pass pack data from the Duet module. They are usually called as replies to user functions.

DCEvent_AuthenticationError

Occurs after 3 login attempts or other user login failure

Arguments passed from event listener:

- **code** integer — error code returned by the Datacraft API
- **description** char[] — description of the error

Returns: **None**

DCEvent_SessionAquired

Occurs after 3 login attempts or other user login failure

Arguments passed from event listener:

- **userid** integer — will be the userid of “apiuser”
- **permission** integer
- **admin** integer

Returns: **None**

DCEvent_UserLoginSuccess

Occurs after successful login

Arguments passed from event listener:

- **username** char[] — the name of the user
- **userid** integer — user id of the user logged in
- **persmission** integer — permission value of the user
- **costcode** char[]
- **costcodeid** integer

Returns: **None**

DCEvent_UserLoginFail

Occurs if user login function was not successful

Arguments passed from event listener:

- **username** char[] — the name of the user

Returns: **None**

DCEvent_RoomDetails

Occurs after successful call to DC_GetRoomDetails function

Arguments passed from event listener:

- **roomid** integer — room id of the room
- **name** char[] — the short name of the room
- **longname** char[] — long version of the room name
- **floor** char[] — details of the room floor
- **capacity** integer — capacity of the room in number of persons
- **phone** char[] — phone number of the phone in the room
- **maxchangeid** integer

- **chartinterval** integer the size of the meeting segments in minutes

Returns: **None**

DCEvent_UserDetails

Occurs after successful call to DC_GetUserDetails function

Arguments passed from event listener:

- **userid** integer user id of the user logged in
- **name** char[] the name of the user
- **initials** char[] initials of the user
- **agentid** integer agent id of the user
- **costcode** char[]
- **costcodeid** integer
- **siteid** integer
- **department** char[]
- **phoneno** char[]
- **emailaddr** char[]
- **room** char[]

Returns: **None**

DCEvent_DefRoomLayout

Occurs after Room Details event

Arguments passed from event listener:

- **index** integer
- **roomid** integer
- **layoutid** integer
- **name** char[]
- **maxcapacity** integer
- **mincapacity** integer

Returns: **None**

DCEvent_RoomBookings

Occurs after successful call to DC_GetRoomBookings with details of the query result

Arguments passed from event listener:

- **roomid** integer room id of the queried room
- **periodstartdate** char[]
- **periodstarttime** char[]
- **periodenddate** char[]
- **periodendtime** char[]
- **count** integer amount of meetings in the result

Returns: **None**

DCEvent_DefineRoomBookings

Occurs after successful call to DC_GetRoomBookings for each meeting that is received by the query

Arguments passed from event listener:

- **index** integer receives 0 to mark the end of the query
- **roomid** integer room id of the room
- **meetingid** long meeting id of the meeting in the index
- **startdate** char[]

- **starttime** char[]
- **enddate** char[]
- **endtime** char[]
- **title** char[] the title of the meeting
- **ownername** char[] the meeting host's name
- **agaentname** char[]
- **agentphone** char[]
- **meetingstate** integer the state of the meeting
- **meetingtype** integer the type of the meeting
- **hiptype** integer
- **hipid** long

Returns: **None**

DCEvent_MeetingState

Occurs after a call to DC_MeetingStart/End advising the new status of the meeting

Arguments passed from event listener:

- **meetingid** long id of the meeting
- **state** integer state id of the meeting
- **statename** char[] state name

Returns: **None**

DCEvent_MeetingExtended

Occurs after a call to DC_MeetingExtend with the result of the request

Arguments passed from event listener:

- **meetingid** long id of the meeting
- **state** integer state id of the request
  - 0 = Active
  - 1 = Cancelled
  - 2 = Waiting
  - 3 = Denied

Returns: **None**

DCEvent_AddBookingOk

Occurs after a successful call to DC_AddBooking. In our example we call DC_GetBookings to update our index of meetings and refresh the UI.

Arguments passed from event listener:

- **meetingid** long id of the meeting
- **cancelstate** integer state id of the meeting
  - 0 = Active
  - 1 = Cancelled
  - 2 = Waiting
  - 3 = Denied

Returns: **None**

DCEvent_AddBookingError

Occurs after a unsuccessful call to DC_AddBooking.

Arguments passed from event listener: **None**

Returns: **None**

DCEvent_DefineMeetingState

Occurs after successful call to DC_GetMeetingStates for each state type that is received by the query

Arguments passed from event listener:

- **index** integer receives 0 to mark the end of the query
- **stateid** integer state id of the state
- **name** long name of the meeting state
- **colour** integer
- **ordinal** integer

Returns: **None**

DCEvent_DefineMeetingState

Occurs after successful call to DC_GetMeetingStates for each type of meeting type that is received by the query

Arguments passed from event listener:

- **index** integer receives 0 to mark the end of the query
- **typeid** integer type id of the meeting type
- **name** long name of the meeting type
- **colour** integer
- **ordinal** integer

Returns: **None**