Host Access for the Cloud Web Client

3.1.1

Table of contents

Welcome to Host Access for the Cloud Web Client

The Rocket® Host Access for the Cloud web client provides browser-based HTML5 access to 3270, 5250, VT, UTS, ALC, and T27 host applications. The Host Access for the Cloud product eliminates the need to touch the desktop; no software to deploy, patches to apply, or configurations to make. You can provide platform-independent user access to all your host applications.

Connection Settings

Connection Settings

There are common connection settings that are applicable to all host types.

Common Connection Settings

There are additional settings which are specific to your type of host.

3270 and 5250 settings

T27

UTS

VT

ALC

Common Connection Settings

These options are common to all supported host types.

- **Connect at startup**

  By default, sessions are configured to connect to the host automatically when you create or open a session. However, you can set up a session so that it doesn't automatically connect to the host. Choose NO to manually connect to the host.

- **Reconnect when host terminates connection**

  When set to Yes, Host Access for the Cloud attempts to reconnect as soon as the host connection terminates.

- **Protocol**

  From the drop down list, select the protocol you want to use to communicate with the host. To establish a host connection, both the web client and the host computer must use the same network protocol. The available values are dependent on the host to which you are connecting. They are:

Protocol	Description

TN3270	TN3270 is a form of the Telnet protocol, which is a set of specifications for general communication between desktop and host systems. It uses TCP/IP as the transport between desktop computers and IBM mainframes.
TN3270E	TN3270E or Telnet Extended is for users of TCP/IP who connect to their IBM mainframe through a Telnet gateway that implements RFC 1647. The TN3270E protocol allows you to specify the connection device name (also known as LU name), and provides support for the ATTN key, the SYSREQ key, and SNA response handling. If you try to use Telnet Extended to connect to a gateway that doesn’t support this protocol, standard TN3270 will be used instead.
TN5250	TN5250 is a form of the Telnet protocol, which is a set of specifications for general communication between desktop and host systems. It uses TCP/IP as the transport between desktop computers and AS/400 computers.
Secure Shell (VT)	You can configure SSH connections when you need secure, encrypted communications between a trusted VT host and your computer over an insecure network. SSH connections ensure that both the client user and the host computer are authenticated; and that all data is encrypted. Two authentication options are available: **Keyboard Interactive** - You can use this authentication method to implement different types of authentication mechanisms. Any currently supported authentication method that requires only the user's input can be performed with Keyboard Interactive. **Password** - This option prompts the client for a password to the host after a host connection is made. The password is sent to the host through the encrypted channel.
Telnet (VT)	Telnet is a protocol in the TCP/IP suite of open protocols. As a character stream protocol, Telnet transmits user input from character mode applications over the network to the host one character at a time, where it is processed and echoed back over the network.
INT1 (UTS)	Provides access to Unisys 1100/1200 hosts using the TCP/IP network protocol.
TCPA (T27)	Use this protocol to connect to Unisys ClearPath NX/LX series or A Series hosts. TCPA Authentication is the process of verifying user login information. When properly configured, you can request a security credential from your application's credential server and send the credential back to the server. If the credential is valid, your application will be logged in; you do not have to enter a user ID or password. If the credential is not valid however, you will be prompted for a user ID and a password.

MATIP (ALC)	Mapping of Airline Traffic Over Internet Protocol (MATIP) uses TCP/IP for airline reservation, ticketing, and messaging traffic.

- **TLS Security**

  TLS protocols allow a client and server to establish a secure, encrypted connection over a public network. When you connect using TLS, Host Access for the Cloud authenticates the server before opening a session, and all data passed between and the host is encrypted using the selected encryption level.

tip

When TLS security is set to TLS 1.3 or TLS 1.2, you have the option to verify the host name against the name on the server certificate. It is highly recommended that you enable host name verification for all sessions.

The following options are available:

Security Options	Description
None	No secure connection is required.
TLS 1.3	Connect using TLS 1.3. When **Verify server identity** is set to **Yes**, the client checks the server or host name against the name on the server certificate. It is highly recommended that you enable host name verification for all sessions.
TLS 1.2	Connect using TLS 1.2. When **Verify server identity** is set to **Yes**, the client checks the server or host name against the name on the server certificate. It is highly recommended that you enable host name verification for all sessions.

- **Enable emulation tracing**

  You can choose to generate host traces for a session. No is the default. Select Yes to create a new emulation host trace each time the session is launched.

Using Terminal ID Manager

MSS To use Terminal ID Manager, you must have a Terminal ID Manager server configured. See Terminal ID Manager Guide.

Terminal ID Manager provides IDs to client applications at runtime and manages pooled IDs for different host types. An ID is connection data that is unique for an individual host session.

If you decide to use Terminal ID Manager and have configured the Terminal ID Manager server, then you can select from the options below to configure the criteria for acquiring an ID. All criteria must be met in order for an ID to be returned.

note

Keep in mind that by specifying a criterion, you are indicating that the ID should be allocated only when an ID with that specific value is found. The set of criteria selected here must be an exact match of the set of criteria specified on at least one Pool of IDs in Terminal ID Manager before the ID request can succeed.

Terminal ID Manager Criteria

Criterion	Description
Pool name	Include this attribute and enter the name of the pool to limit the ID search to a specified pool.
Client IP address	The IP address of the client machine will be included as part of the request for an ID.
Host address	The address of the host configured for this session will be included as part of the request for an ID.
Host port	The port for the host configured for this session will be included as part of the request for an ID.
Session name	When selected, requires that the ID is configured to be used by this session exclusively.
Session type	The session type (for example, IBM 3270, IBM 5250, UTS, ALC or T27) is always included as part of any request for an ID.

User name	Use this criterion to ensure that only IDs created for exclusive use by specific users will be allocated. The current user's name, which must be found on an ID before it can be allocated, is the name of the user that the session is allocated to at runtime. To configure a session based on user names, a default place holder user name is available: tidm-setup. For the administrator to configure sessions using tidm-setup, the Terminal ID Manager must have IDs provisioned for tidm-setup. You can override the default name with one of your own by modifying the `&ltinstall-dir>/sessionserver/conf/container.properties` file as follows: `id.manager.user.name=custom-username` where `custom-username` is replaced by the name you want to use.
Application name (UTS)	The name of the host application will be used as part of the request for an ID.

To determine the connection attempt behavior if Terminal ID Manager does not successfully allocate an ID to this session, use **If ID is not allocated**:

- **Fail connection attempt** - When selected, the session does not attempt to connect when an ID is not allocated.
- **Allow connection attempt** - When selected, the session attempts to connect when an ID is not allocated. The attempt may be rejected by the host. There are some host types that permit a user to connect without an ID.

  To confirm that Terminal ID Manager can provide an ID using the criterion and value selections you have made, click Test Terminal ID Manager Criteria.
- **Send keep alive packets** - Use this setting to provide a constant check between your session and the host so that you become aware of connection problems as they occur. Choose from the following types of keep alive packets:

This option	Does this ...
None	The default. No packets are sent.
System	The TCP/IP stack keeps track of the host connection and sends keep alive packets infrequently. This option uses fewer system resources than the Send NOP Packets or Send Timing Mark Packets options.

This option	Does this ...
Send NOP packets	Periodically, a No Operation (NOP) command is sent to the host. The host is not required to respond to these commands, but the TCP/IP stack can detect if there is a problem delivering the packet.
Send timing mark packets	Periodically, a Timing Mark Command is sent to the host to determine if the connection is still active. The host should respond to these commands. If a response is not received or there is an error sending the packet, the connection shuts down.

- **Keep alive timeout (seconds)** - If you choose to use either the Send NOP packets or the Send timing mark packets option, select the interval between the keep alive requests set. The values range from 1 to 36000 seconds (1 hour); the default is 600 seconds.

Test Terminal ID Manager Criteria

The Terminal ID Manager provides IDs to client applications at runtime. To confirm that Terminal ID Manager can provide an ID using the criteria and value selections you selected use this test option.

Criteria for the current session are specified on the Connection panel after selecting **Use Terminal ID Manager** from either the Device Name (3270, 5250 host types), the Terminal ID (UTS) field, or the Station ID (T27) field. By default, the selected criteria for the current session are displayed.

Click **Test** to confirm that Terminal ID Manager can provide an ID matching the configured criterion and value selections. The test returns the name of an available ID that satisfies the selected attribute values.

Testing for other criteria and values

You can also use this panel to test criteria different from those associated with the current session.

1. Select any of the session types from the Session type list, and select the criteria you want to test. You can test alternate values that you want to use in a sample Terminal ID Manager request.
2. Click **Test** to confirm that Terminal ID Manager can provide an ID matching the criterion and value selections. The test returns the name of an available ID that satisfies the selected values.

3270 and 5250 connection settings

In addition to the Common connection settings, 3270 and 5250 host types require these specific settings.

- **Terminal model**

  Specify the terminal model (also known as a display station) you want Host Access for the Cloud to emulate. There are different terminal models available depending on the host type.

  If you choose **Custom Model**, you can set the number of columns and rows to customize the terminal model.

- **Use Kerberos automatic sign-on (5250 only)** MSS

  When set to **Yes** the user does not have to enter sign-on credentials. Kerberos automatic sign-on is configured in the MSS Administrative Console > Host Access for the Cloud. In configuring HACloud to use the Kerberos authentication protocol there are terms that you should understand and prerequisites to adhere to in advance of configuring this option. These options are explained in detail in the MSS Administrative Console > Host Access for the Cloud panel documentation, available from the Help button. See the Deployment Guide for more information.

- **Terminal ID** (3270 only)

  When Host Access for the Cloud connects to a Telnet host, the Telnet protocol and the host negotiate a terminal ID to use during the initial Telnet connection. In general, this negotiation will result in the use of the correct terminal ID, and so you should leave this box empty.

- **TLS Security**

  TLS protocols allow a client and server to establish a secure, encrypted connection over a public network. When you connect using TLS, Host Access for the Cloud authenticates the server before opening a session, and all data passed between and the host is encrypted using the selected encryption level. See Common Connection Settings for detailed information on this common setting.

- **Device name**

  If you selected TN3270, TN3270E, or TN5250 as the protocol, specify the device name to use when the session connects to the host. The device name is also known as the host LU or pool. You can also choose to:

  - **Generate a unique device name** - Automatically generates a unique device name.
  - **Use Terminal ID Manager** - Displays additional settings to complete. See Using Terminal ID Manager
  - **Always Prompt the User for ID** - The end user is prompted for the device ID each time a connection is attempted.

- **Prompt the User if ID not Specified** - The end user is prompted the first time a connection is attempted after which the value is saved. The saved value will continue to be used without additional prompting.

If you do not specify a device name for the session, the host dynamically assigns one to the session. A device name that is set within a macro will override this setting.

VT connection settings

In addition to the Common connection settings, VT hosts require additional settings. These settings vary depending on the protocol you are using; Telnet or SSH. The settings are applicable to both protocols unless noted.

VT session configuration options

VT Settings	Description
Terminal ID	This setting determines the response that Host Access for the Cloud sends to the host after a primary device attributes (DA) request. This response lets the host know what terminal functions it can perform. The Host Access for the Cloud response for each Terminal ID is exactly the same as the VT terminal's response; some applications may require a specific DA response. This terminal ID setting is independent of the Terminal type setting. The options are: VT220, VT420, VT100, DEC-VT100, and VT52.
Allow unknown hosts (SSH)	This setting provides the administrator with the ability to decide whether the web client will allow unknown hosts. Options are: **Yes** - Unknown hosts and all SSH connections are permitted. Web client users are not prompted about whether hosts should be trusted. **Ask** - The web client user is prompted whether the host should be trusted when they connect to an unknown host they haven't encountered before. If they choose to trust the host, then its public key is stored in their user preferences and subsequent connections will not elicit a prompt unless the host key changes. **No** - No unknown hosts are permitted. Only those hosts the administrator chooses to trust when configuring the session are permitted. End users are never prompted and the session will either connect or not connect depending on the administrator’s choices.

Suppress banner messages (SSH)	When enabled, the SSH banner is not displayed. This option is useful when recording SSH login macros.
Local Echo (Telnet)	Automatic (default). How Host Access for the Cloud responds to remote echo from a Telnet host: Automatic attempts to negotiate remote echo, but does what the host commands. Yes means Host Access for the Cloud negotiates local echo with the host, but always echoes, while No means Host Access for the Cloud negotiates remote echo with the host, but does not echo.
Renegotiate Echo (Telnet)	No (default). When set to Yes, passwords are not displayed on the local screen, but all other typed text is visible. Host Access for the Cloud supports the Telnet Suppress Local Echo (SLE) option when connected to a host in half-duplex mode. This means that Host Access for the Cloud will suppress character echo to the host computer, and with SLE support Host Access for the Cloud can be instructed to suppress echo locally.
Set Host Window Size	Yes (default). This setting sends the number of rows and columns to the Telnet host whenever they change. This enables the Telnet host to properly control the cursor if the window size is changed.
Host Key Algorithms (SSH)	Sets which host key algorithms to use when connecting to an SSH server. Options are: EC, RSA, DSS (default) - This is the more secure option. Use it during an upgrade to provide more security. RSA, DSS - Select RSA, DSS to continue using an existing key for new SSH sessions.
Request Binary (Telnet)	No (default). Telnet defines a 7-bit data path between the host and the terminal. This type of data path is not compatible with certain national character sets. Fortunately, many hosts allow for 8-bit data without zeroing the 8th bit, which resolves this problem. In some cases, however, it may be necessary to force the host to use an 8-bit data path by selecting this check box.
Send LF after CR (Telnet)	No (default). A "true" Telnet host expects to see a CrNu (carriage return/null) character sequence to indicate the end of a line sent from a terminal. There are some hosts on the Internet that are not true Telnet hosts, and they expect to see a Lf (linefeed) character following the Cr at the end of a line. If you're connecting to this type of Telnet host, select Yes.

Ctrl-break sends (Telnet)	Choose what sequence Ctrl-break sends to the host when pressed. Options are: Telnet break sequence (the default), Interrupt process, or Nothing.
Host Character Set	The default value for the Host character set depends on the type of terminal you are emulating. This setting reflects the current terminal state of VT Host Character Set, which can be changed by the host. The associated default setting, saved with the model is DEC Supplemental.
Auto Answerback	No (default). This setting specifies whether the answerback message (set with the Answerback property) is automatically sent to the host after a communications line connection.
Answerback String	This setting allows you to enter an answerback message if the host expects an answer in response to an ENQ character. The answerback string supports characters with codes less than or equal to 0xFFFF via Unicode escape sequences. The escape sequence begins with \u followed by exactly four hexadecimal digits. You can embed Unicode escape sequences in any string. For example, this embedded \u0045 will be interpreted as this embedded E, since 45 is the hexadecimal code for the character E. To pass Unicode escape sequences to the host, escape the sequence with a leading backslash. For example, to send the string literal \u001C to the host, map a key to \\u001C. Host Access for the Cloud will convert this to the string \u001C when that key is pressed and send the 6 characters of the resulting string to the host.

More information

- TLS descriptions

UTS connection settings

In addition to the Common connection settings, UTS hosts require these additional settings:

UTS INT1 session configuration options

UTS INT1 options	Description
Application	The name of the host application or host operating mode to be accessed. This is the word or phrase that the local machine sends to the host when you first establish communication with the host. If you were using a host terminal, this would be the $$OPEN name of the application.The application name is typically the same as the environment name. However, they can be different. For example, the environment name might be MAPPER, and the application might be UDSSRC. During a terminal emulation session, you would type $ $OPEN MAPPER at the prompt, and INT1 would send UDSSRC to the host once the connection is established.
TSAP	The desired Transport Service Access Point (TSAP), up to 32 characters (such as TIPCSU for TIP connections, RSDCSU for Demand connections).A TSAP is required only if you are connecting to a Host LAN Controller (HLC) or to a Distributed Communications Processor (DCP) in IP router mode. If you're not sure which value to use, contact your host administrator.
Initial transaction	The character, word, or phrase that the local machine will send to the host when communication with the host is first established (up to 15 characters).This parameter is optional and is primarily used with TIP. For example, you might type ^ to run MAPPER. This parameter can also be used to transmit passwords.
Start transaction	When you configure an initial transaction, by default, the data is sent as soon as the session connection is established. You can decide when to send an initial transaction by using a particular string to trigger the initial transaction. For example, to wait for a successful login before sending initial transaction data, type in a string to be used to identify a successful login. You can use this setting in combination with **Send initial transaction**.

Send initial transaction	You can determine when the initial transaction is sent: **Immediately** - Default. **When start of entry (SOE) character is received** - Useful when multi-line transactions must be completed before sending the string. **After specified milliseconds**
Terminal ID	Choose options to specify a terminal ID or to use the Terminal ID Manager. To specify a terminal ID, type it in the **Specify Terminal ID** field. Specify Terminal ID The Terminal ID, a terminal identifier (typically up to 8 alphanumeric characters) to use for the communication session associated with this path. Also known as a TID or PID, each terminal ID should be unique to the host. Prompt the User if ID not Specified The end user will be prompted the first time a connection is attempted after which the value is saved. The saved value will continue to be used without additional prompting. Always prompt the user for ID When you select this option the end user will be prompted for the terminal ID each time a connection is attempted. Use Terminal ID Manager If you choose **Use Terminal ID Manager**, you are prompted to select the Terminal ID attributes you want to use to obtain an ID. See Terminal ID Manager Attributes. To test the attributes, click **Test**.

More information

Terminal ID Manager Attributes

TLS Descriptions

T27 connection settings

Along with the Common connection settings, you can configure these additional T27 connection options:

T27 Connection Settings

T27 options	Description
Terminal type	Select the type of terminal to emulate during the session. T27 emulation supports Unisys TD830, TD830 ASCII, TD830 INTL, and TD830 NDL terminal types.
Request binary	You must enable the Request binary option when you require pass through printing. The default is No. TCPA defines a 7-bit data path between the host and the terminal emulator. This type of data path is not compatible with certain national character sets. However, many hosts allow for 8-bit data without zeroing the 8th bit, which resolves this problem. However, it may be necessary to force the host to use an 8-bit data path; you can do so by selecting this option.
Line width	Select the number of characters the host will send to the client. The default is 80 characters.
TLS security	See **TLS Descriptions** for a description of the various options.

Station ID	Choose an option to specify a station ID or use the Terminal ID Manager. To specify a station ID, choose **Specify Station ID** and type the name in the Station ID field. Each station id should be unique to the host and typically consists of up to eight alphanumeric characters. Prompt the user if ID not specified The end user will be prompted the first time a connection is attempted after which the value is saved. The saved value will continue to be used without additional prompting. Always prompt the user for ID When you select this option the end user will be prompted for the station ID each time a connection is attempted. Use Terminal ID Manager If you select Use Terminal ID Manager, you will see a number of Terminal ID criteria to configure. See **Terminal ID Manager Criteria** for descriptions of the various options. If you do not specify a station id for the session, the host dynamically assigns one to the session.

More information

- TLS Descriptions
- Terminal ID Manager Criteria

ALC connection settings

In addition to the Common connection settings, ALC hosts require these additional settings:

ALC Connection Settings

ALC options	Descriptions
TLS security	See **TLS Descriptions** for a description of the various options.
Character encoding	Choose ASCII, EBCDIC, or IPARS (default) as the code set.
Configuration file	Enter the configuration (CNF) file that associates configuration information appropriate for a specific host type.
Terminal address	Select whether you want to specify the terminal address or use the Terminal ID Manager. **Terminal address** - Specify whether to use 2-byte or 4-byte addressing mode. Although a unique 5-byte address is required when you specify the terminal ID instead of using ID Manager, this option specifies how many bytes of the 5-byte terminal ID address are sent with each message for the purposes of multiplexing. If you specify 2-byte addressing mode, only the last 2 bytes of the ASCU (Agent Set Control Unit) cluster address (A1, A2) are sent. If you specify 4-byte addressing mode, the full ASCU cluster address (H1, H2, A1, A2) is sent. Specify the unique 5-byte terminal address for this session. The terminal address is made up of five 2-hex-digit values in this order: H1, H2, A1, A2, and TA (terminal address). This unique address is usually assigned by the network administrator. **Terminal ID Manager**- Provides IDs to client applications at runtime. If you choose this option, there are additional configuration options to complete. See **Terminal ID Manager Criteria** for descriptions of those options.

More information

- TLS Descriptions
- Terminal ID Manager Criteria

Working with Sessions

All the sessions you have access to are available in the Available Sessions list. Sessions are initially created and configured by your system administrator and accessed through a distributed URL (for example, `https://<clusterdns>/webclient`.

To open a session

1. Select the session and click to open.
2. Interact with your host application using the open session.
3. You can create multiple instances of a configured session.

You can have multiple sessions open at a time and easily switch between them using the tabs arranged across the top of the screen. The current session is always the left-most tab and is indicated by a white background and bold text. Each session remains active for 30 minutes.

Use the toolbar to access the various options available to you as you interact with the session. You can disconnect from a session, close the session, turn on Quick Keys, and access other settings. Some options may be only available once your administrator has granted permission.

Using Quick Keys

The Quick Key terminal keyboard provides a graphical representation of the keys on a host keyboard and gives you quick access to terminal keys.

Click a terminal key on the Quick Key keyboard to send the key to the host. Tool tips, which are available by hovering over a key, provide a description of the mapping.

Quick keys are available for each supported host type and are accessed by clicking the tool bar icon ⌨.

Editing the Screen

note

Each browser handles copy, paste and cut functions differently and in some cases will not support the use of the toolbar buttons or the right-click context menu. It is highly recommended that you use keyboard commands for those functions. Although keyboard commands vary depending on your operating system, in Windows they are: CTRL+C to copy, CTRL+V to paste, and CTRL+X to cut.

It is far more common to encounter problems with the paste function rather than either cut or copy. If the Paste toolbar button is not visible, it is likely that browser security is preventing read access to the system clipboard. Different browsers behave differently when it comes to providing access to the clipboard. However, pasting is almost always available using the keyboard commands, (Control + V on Windows and Command + V on Macs). This assumes you have not remapped those keys. You can also use the browser's built-in paste menu item or button.

To copy from the terminal

1. Highlight the area on the terminal screen that you want to copy.
2. Click **Copy** from the toolbar or select **Copy** from the right-click context menu available within the terminal screen. You can alternatively use the keyboard command, **CTRL+C**.

To paste into the terminal screen

1. Position the cursor where you want to paste content.
2. If the browser supports the paste function, click **Paste** from the toolbar or select **Paste** from the right-click context menu available within the terminal screen. If your browser does not support this functionality, these options will not be available and you should use the keyboard command, **CTRL+V**.

To cut areas from the terminal screen

note

This function is available for all supported terminal types except for VT hosts.

1. Highlight the area on the terminal screen that you want to cut.
2. Click **Cut** from the toolbar or select **Cut** from the right-click context menu available within the terminal screen. You can alternatively use the keyboard command, **CTRL+X**.

More information

Specify Edit Options

Logging Out

In the upper right corner of the screen, open the drop-down list associated with your user name and select **Logout** to stop working with the host application.

Macros

Creating Macros

A macro is a series of keyboard actions that you record and then run. You can use these JavaScript macro programs to automate user interactions with the terminal. You can access and run macros from all supported devices.

Host Access for the Cloud records and saves advanced macros as JavaScript, making it easy to edit and enhance your recorded macros. You can record macros to playback later, run macros at startup or when the session connects or disconnects from the host. You can also write macros from scratch to perform complex tasks that the recorder cannot capture.

Macros are made available to users in two ways; created by an administrator or recorded by users for their own private use. All advanced macros are associated with a session and they all accomplish the same goal, automating host interaction. The only difference between the two flavors is simply who can access them and who manages their creation and availability:

- **Macros created by administrators**

  Administrators create macros when they create the session. They are specific to a session and are available to all users who have access to the session from the Macro icon on the toolbar. Administrators can designate macros to run at startup or when the session connects or disconnects from the host.

- **Macros created by users**

  End-user macros are created by individuals for sessions they are authorized to access. The administrator grants permission to create macros by setting a User Preference Rule. Users can access the session under their own credentials or in a Guest role. Macros that Guest users create are available to all Guest users. Users who are logged in using their own credentials can only see macros that they have created.

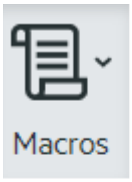
  Advanced macros are listed in alphabetical order in the drop down list available from the toolbar. Macros created by the end-user are listed first and followed by an indicator of three vertical grey dots, which when selected, displays the Edit and Delete options. Macros created by the administrator are listed without the indicator as those macros cannot be modified by the end-user.

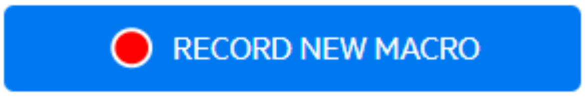
Working with Macros

Follow these steps to record, edit, and run macros.

Record

1. Click the Macro icon on the toolbar, and then click Record New Macro.

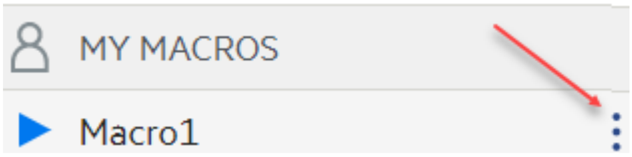




2. Navigate the host application to record the series of steps you want included in the macro.
3. Click ● on the toolbar to stop recording. The red dot pulses to indicate the recording is in process.
4. When prompted, type a name for the macro.

Edit

1. From the Macro drop-down list, select the macro you want to edit.



2. Click the three vertical dots to expand the field.
3. Click Edit to open the Macro Editor (in the left panel).
4. Use JavaScript to make whatever changes are necessary. You can run and save the modified macro using the toolbar icons in the upper panel of the editor.

Run

To run a macro, choose the macro from the drop-down list and click ▶.

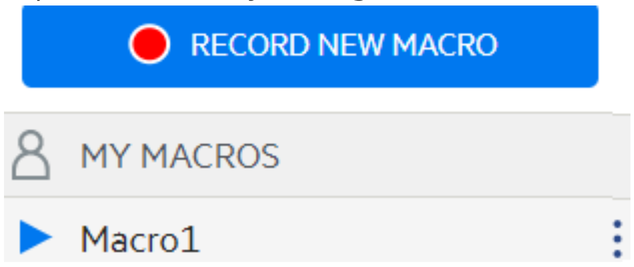
You can also map keys that will automatically trigger an already recorded macro. In the Key Map settings dialog box, choose **Run Macro** from the **Action** drop down list. Choose a macro to associate with the key mapping from the **Value** list.

Stop

You can stop a macro before it completes from either the Macro Editor or the toolbar. Click ■ to stop the macro. To rerun the macro, navigate back to the macro starting screen.

Delete

1. From the Macro drop down list, select the macro you want to delete.
2. Expand the field, by clicking the three vertical dot icon.



3. Click **Delete**.

View

The Macro drop down list is available from the toolbar to all users who have permission to record macros or are accessing a session where macros have been pre-recorded by the administrator for use with that session.

Macros are listed under either **MY MACROS** or **MACROS** depending on how they were recorded.

All users, whether they are logged in using their credentials or as Guest, can see the macros associated with the session. Macros listed under the **MY MACROS** heading are listed in alphabetical order by name and are visible to those users that recorded them. Macros recorded by the administrator and attached to a session are listed alphabetically under **MACROS**.

Debugging Macros

Since macros are written in JavaScript and executed in the browser, the best way to debug and troubleshoot them is by using your web browser's built-in tools. Modern browsers come with a very capable set of tools for debugging JavaScript code. You can place breakpoints, step through code, and output debug information.

hint

JavaScript is case sensitive. Keep that in mind when editing JavaScript code.

To debug a macro:

1. Open the macro for editing. See Working with macros for instructions.
2. Open your browser’s development tools.

Browser	Open debugger
Mozilla Firefox	From the toolbar, open the Menu, and choose Developer. From the Web Developer menu, choose Debugger. The debugger opens in a lower panel.
Google Chrome	From the toolbar, open the Menu, and choose More tools. Choose Developer Tools to open the Debugger.

3. Use one of the these tools in your macro code, and run the code.

   - debugger

     The most thorough approach to debugging is to use the `‘debugger;’` statement. When you insert these statements into your macro code then run it, with the browser’s development tools open, the execution will stop on those lines. You can step through your macro, view the value of local variables and whatever else you need to check.

     You are encouraged to place multiple debugger; statements in your code to help get to the correct line. The asynchronous nature of JavaScript can make stepping through code challenging. This can be offset by using multiple, carefully placed debugger; statements.

     Example 1: `debugger`

```
var hostCommand = menuSelection + ‘[enter]';
debugger;  // <— Browser's debugger will stop here
ps.sendKeys(hostCommand);
```

   - console.log(), alert()

     These two functions are commonly used for debugging JavaScript. While not as flexible as the debugger statement they provide a quick way to output debug information. These functions output the information to the JavaScript “Console” tab in the browser’s developer tools.

     Example 2: `console.log(), alert()`

```
var hostCommand = menuSelection + ‘[enter]';
console.log('Command:' + hostCommand);  // <— Will output the string to
"Console" tab
alert('Command:' + hostCommand); // Will pop up a small window containing
the data
ps.sendKeys(hostCommand);
```

- ui.message()

  The Host Access for the Cloud Macro API provides an ui.message() function that is very similar to JavaScript’s alert() function. You can also use ui.message() to output debug information.

  Example 3: `ui.message()`

```
var hostCommand = menuSelection + ‘[enter]';
ui.message('Command:' + hostCommand);  // <— Will pop up a message window
ps.sendKeys(hostCommand);
```

Keep in mind:

- Stepping and “yields”

  While the yield statements in macros make them easier to understand, they can make the code more challenging to step through with the debugger. Consider either using multiple debugger statements or carefully placed debugger statements of `console.log()` calls to output the right debug information.

Using the Macro API

In Host Access for the Cloud, macros are recorded and written using JavaScript.

The Macro API consists of a set of objects which you can use to interact with the host, wait for screen states, and interact with the user.

About promises and yields

Because JavaScript is single-threaded and uses 'callback functions’ and ‘promises’ to help manage the flow of execution through code, often code can be difficult to follow. Host Access for the Cloud combines the concept of ‘promises' with the ‘yield' keyword so macro code can be organized in a more linear fashion.

- **Promises**

Promises are patterns to help simplify functions that return their result asynchronously, at some point in the future. All ‘wait’ and ‘ui’ functions in the Macro API return promise objects.

- **Yield**

  Macros use the yield keyword to block the execution of the macro until a promise is resolved, or done. So putting yield in front of any ‘wait’ or ‘ui’ function makes the macro execution pause until that function has finished executing. You can place the yield keyword in front of any function that returns a promise, even your own custom functions.

> **note**
>
> The ability to make macro execution block by combining yield with promises is enabled by the `createMacro()` function.

Errors

Errors are handled in macros using a try / catch statement. Some API functions may throw errors if, for example, conditions can’t be met or a timeout occurs. The thrown error is ‘caught’ in the catch statement. You can wrap smaller blocks of code in a try / catch statement to handle errors at a more granular level.

Macro developers can also throw errors by using `'throw new Error('Helpful error message');`

More information

Macro API Objects

Sample Macros

Macro API Objects

You can create macros using the Macro API. By default for use in macros, there are four primary objects available:

- **Session** - the main entry point for access to the host. You use the Session object to connect, disconnect and provide access to the PresentationSpace object.
- **PresentationSpace** - represents the screen and provides many common capabilities such as getting and setting the cursor location, sending data to the host and reading from the screen. It is obtained by calling `session.getPresentationSpace()`.
- **Wait** - provides a simple way to wait for various host states to occur before continuing to send more data or read from the screen. For example, you can wait for the cursor to be located at a

certain position, text to be present in a position on the screen or simply wait for a fixed amount of time. All ‘Wait' function calls require the yield keyword, which is explained below.

- **User Interface** - automatically available in your macro as the “ui” variable. It provides basic user interface capabilities. You can use this object to display data to the user or prompt them for information. All ‘UI' function calls require the yield keyword.

All available objects

See the list of available objects in the right navigation, "On this page." (You may need to expand your browser.)

Attribute

Use the Attribute, along with the AttributeSet, to decode the formatting information present on the data cell.

Attribute	Indicates...
PROTECTED	a protected data cell
MODIFIED	a modified data cell
NUMERIC_ONLY	the beginning of a numeric only data cell
ALPHA_NUMERIC	an alpha numeric data cell.
HIGH_INTENSITY	whether the data cell contains high intensity text
HIDDEN	whether the data cell contains hidden text
PEN_DETECTABLE	whether the data cell is pen detectable
ALPHA_ONLY	an alpha only data cell
NUMERIC_SHIFT	the beginning of a numeric shift field
NUMERIC_SPECIAL	the data cell marks the beginning of a numeric special field
KATAKANA_SHIFT	a section of Katakana text
MAGNETIC_STRIPE	the data cell marks the beginning of a magnetic strip field
SIGNED_NUMERIC_ONLY	the data cell is a signed numeric field
TRANSMIT_ONLY	the data cell is a transmit only field

Attribute	Indicates...
FIELD_END_MARKER	the data cell marks the end of a modified field
FIELD_START_MARKER	the data cell marks the start of a modified field
SPECIAL_EMPHASIS_PROTECTED	a special emphasis protected field
TAB_STOP	that the data cell contains a tab stop
REVERSE	the data cell displays in reverse video mode
BLINKING	the data cell contains blinking text
RIGHT_JUSTIFIED	the data cell marks the beginning of a right justified field
LEFT_JUSTIFIED	the data cell marks the beginning of a left justified field
LOW_INTENSITY	the data cell contains low intensity text
UNDERLINE	the data cell contains underlined text
DOUBLE_BYTE	the data cell contains double byte text
COLUMN_SEPARATOR	the data cell contains a column separator
BOLD	the data cell contains bold text
DOUBLE_WIDTH	the data cell marks a double width field
DOUBLE_HEIGHT_TOP	a double height top data cell
DOUBLE_HEIGHT_BOTTOM	a double height bottom data cell
CONTROL_PAGE_DATA	the data cell contains control page data
RIGHT_COLUMN_SEPARATOR	the data cell contains a right column separator
LEFT_COLUMN_SEPARATOR	a data cell containing a left column separator
UPPERSCORE	the data cell contains an upperscore

Attribute	Indicates...
STRIKE_THROUGH	the data cell contains strike through text

AttributeSet

The AttributeSet object allows the user to decode the attributes that are present on the data cell. The AttributeSet object returns values defined in the Attribute object and when used together, you can get formatting information from the data cell.

Method	Description
`contains(attribute)`	Determines if the set contains the specified Attribute. **Parameters** `{Number}` attribute to check **Returns** `{Boolean}` True if the attribute is in the set
`isEmpty()`	Determines if the attribute set is empty. **Returns** `{Boolean}` True if the set is empty.
`size()`	Indicates the number of attributes in a set. **Returns** `{Number}` The attribute count.
`toArray()`	Converts the internal attribute set to an array. **Returns** `{Number[]}` Array of values of attributes in the set.
`toString()`	Converts the internal attribute set to a string. **Returns** `{String} Space-delimited names of attributes in the set.`
`forEach(callback, thisArg)`	Function to iterate over each element in the attribute set. **Parameters** `{forEachCallback}` Callback to perform the specific operation. Called with the name of each attribute in the set. `{Object} thisArg` optional pointer to a context object.

`forEachCallback(string, object)`	A user provided callback function where you provide the behavior, to be used as the callback parameter to forEach.
	Parameters
	`{String} String name of an attribute in the attribute set.`
	
	`{Object} thisArg optional pointer to a context object.`

AutoSignOn

Method	Description
`getPassTicket()`	Obtains a pass ticket to be used for signing onto a mainframe application. Multiple pass tickets may be requested using different application IDs. **Parameters** {String} application ID tells the host which application the sign on is for **Returns** {Promise} fulfilled with the pass ticket key or rejected if the operation fails. The pass ticket obtained from DCAS only works with the current host session and is valid for ten minutes.
`sendUserName()`	Applies the user name contained in the pass ticket to the field at the current cursor location on the current host screen. The user name must be sent before the password. Sending the password first will invalidate the pass ticket, and you will need to get another one. **Parameters** {String} passTicketKey opbtained from getPassTicket **Returns** {Promise} fulfilled if the user name is successfully sent. Rejected if the operation fails.

`sendPassword()`	Applies the password contained in the pass ticket to the field at the current cursor location on the current host screen. The user name must be sent before the password. Sending the password first will invalidate the pass ticket, and you will need to get another one. **Parameters** {String} passTicketKey obtained from getPassTicket **Returns** {Promise} fulfilled if the password is successfully sent. Rejected if the operation fails.

Color

Color constants to use for the DataCell object foreground and background colors.

Color	Description	Numeric Value
BLANK_UNSPECIFIED	No color specified	0
BLUE	Blue	1
GREEN	Green	2
CYAN	Cyan	3
RED	Red	4
MAGENTA	Magenta	5
YELLOW	Yellow	6
WHITE_NORMAL_INTENSITY	Normal intensity white	7
GRAY	Gray	8
LIGHT_BLUE	Light blue	9
LIGHT_GREEN	Light green	10
LIGHT_CYAN	Light cyan	11
LIGHT_RED	Light red	12
LIGHT_MAGENTA	Light magenta	13
BLACK	Black	14

Color	Description	Numeric Value
WHITE_HIGH_INTENSITY	High intensity white	15
BROWN	Brown	16
PINK	Pink	17
TURQUOISE	Turquoise	18

ControlKey

The ControlKey object defines constants for sending cursor control keys and host commands using the sendKeys method. Constants are available for these host types:

- IBM 3270
- IBM 5250
- VT
- UTS

IBM 3270

Key word	Description
ALTVIEW	Alternate view
ATTN	Attention
BACKSPACE	Back space
BACKTAB	Back tab
CLEAR	Clear or clear display
CURSOR_SELECT	Cursor select
DELETE_CHAR	Delete, delete character
DELETE_WORD	Delete word
DEST_BACK	Destructive backspace
DEV_CANCEL	Device cancel

Key word	Description
DOWN	Cursor down
DSPSOSI	Display SO/SI
DUP	Duplicate field
END_FILE	End of field
ENTER	Enter
ERASE_EOF	Erase end of field
ERASE_FIELD	Erase field
ERASE_INPUT	Erase input
FIELD_MARK	Field mark
HOME	Cursor home
IDENT	Ident
INSERT	Insert
LEFT_ARROW	Cursor left
LEFT2	Left two positions
NEW_LINE	New line
PA1 - PA3	PA1 - PA3
PF1 - PF24	PF1 - PF24
PAGE_DOWN	Page down
PAGE_UP	Page up
RESET	Reset, reset terminal
RIGHT2	Right 2 positions
RIGHT_ARROW	Cursor right, right
SYSTEM_REQUEST	System request
TAB	Tab key

Key word	Description
UP	Cursor up

IBM 5250

Key word	Description
ALTVIEW	Alternate view
ATTN	Attention
AU1 - AU16	AU1 - AU16
BACKSPACE	Back space
BACKTAB	Back tab
BEGIN_FIELD	Begin field
CLEAR	Clear or clear display
DELETE_CHAR	Delete, delete character
DEST_BACK	Destructive backspace
DOWN	Cursor down
DSPSOSI	Display SO/SI
DUP	Duplicate field
END_FILE	End of field
ENTER	Enter
ERASE_EOF	Erase end of field
ERASE_FIELD	Erase field
ERASE_INPUT	Erase input
FIELD_EXT	Field exit
FIELD_MINUS	Field minus
FIELD_PLUS	Field plus

Key word	Description
FIELD_MARK	Field mark
HELP	Help request
HEXMODE	Hex mode
HOME	Cursor home
INSERT	Insert
LEFT_ARROW	Cursor left
NEW_LINE	New line
PA1 - PA3	PA1 - PA3
PF1 - PF24	PF1 - PF24
PAGE_DOWN	Page down
PAGE_UP	Page up
[print]	Print
RESET	Reset, reset terminal
RIGHT_ARROW	Cursor right, right
SYSTEM_REQUEST	System request
TAB	Tab key
UP	Cursor up

VT

Key word	Description
BACKSPACE	Back space
BREAK	Break
CLEAR	Clear or clear display
CURSOR_SELECT	Cursor select

Key word	Description
DELETE_CHAR	Delete, delete character
DOWN	Cursor down
EK_FIND	Edit keypad find
EK_INSERT	Edit keypad insert
EK_NEXT	Edit keypad next
EK_PREV	Edit keypad previous
EK_REMOVE	Edit keypad remove
EK_SELECT	Edit keypad select
END_FILE	End of field
ENTER	Enter
F1 - F24	F1 - F24
HOLD	Hold
HOME	Home
INSERT	Insert
KEYPAD_COMMA	Keypad comma
KEYPAD_DOT	Keypad decimal
KEYPAD_ENTER	Keypad enter
KEYPAD_MINUS	Keypad minus
KEYPAD0 - KEYPAD9	Keypad 0 - Keypad 9
LEFT_ARROW	Cursor left
PF1 - PF20	PF1 - PF20
PAGE_DOWN	Page down
PAGE_UP	Page up
RESET	Reset, reset terminal

Key word	Description
RETURN	Return, carriage return
RIGHT_ARROW	Cursor right, right
TAB	Tab key
UDK16 - UDK20	User defined key 6 - User defined key 20
UP	Cursor up

UTS

Key word	Description
BACKSPACE	Back space
BACKTAB	Back tab
CHAR_ERASE	Erases character at the cursor and advances the cursor.
CLEAR_DISPLAY	Clear display
CLEAR_EOD	Clear to end of display
CLEAR_EOF	Clear to end of field
CLEAR_EOL	Clear to end of line
CLEAR_FCC	Clear Field Control Character
CLEAR_HOME	Clear display and cursor home
CONTROL_PAGE	Toggles the control page
DELETE_LINE	Deletes the line containing the cursor and shifts remaining lines up one row
DELIN_LINE	Deletes character under cursor and shifts remaining characters on line to the left.
DELIN_PAGE	Deletes character under cursor and shifts remaining characters on page to the left.
DOWN	Moves the cursor down one line. Wraps at bottom.

Key word	Description
DUP_LINE	Creates a copy of the current line and overwrites the next line with the duplicate.
END_FIELD	Moves the cursor to the end of the current field.
END_PAGE	Moves the cursor to the end of the current page.
EURO	Inserts the Euro character
F1 - F22	Function keys F1-F22
HOME	Moves the cursor to beginning of current page (row 1, col 1)
INSERT	Toggles insert/overwrite mode.
INSERT_IN_LINE	Inserts space at cursor position and shifts the remaining characters on the line to the right. The character in the far right column on the line is discarded.
INSERT_IN_PAGE	Inserts space at cursor position and shifts the remaining characters on the page to the right. The character in the far right column on each line is discarded.
INSERT_LINE	Inserts a new line at the cursor row and shifts the remaining lines down. The last row on the page is discarded.
LEFT_ARROW	Moves the cursor one position to the left wrapping if necessary.
LOCATE_FCC	Finds the next field control character on the screen.
MSG_WAIT	Retrieves messages queued to the terminal.
RETURN	Carriage return
RIGHT_ARROW	Moves the cursor one position to the right, wrapping if necessary.
SOE	Inserts the Start of Entry character
START_OF_FIELD	Moves the cursor to the beginning of the field.
START_OF_LINE	Moves the cursor to column 1 of current line.
TAB	Moves the cursor to the next tab position of the screen.
TOGGLE_COL_SEP	Toggles the column separator attribute.

Key word	Description
TOGGLE_STRIKE_THRU	Toggles the strike-through attribute on the current data cell.
TOGGLE_UNDERLINE	Toggles the underline attribute on the current data cell.
TRANSMIT	Transmits changed field data to the host.
UNLOCK	Sends the UNLOCK key to the host.
UP	Moves the cursor up one row, wrapping if necessary.

DataCell

The DataCell object provides information about a particular position on a terminal screen.

Method	Description
`getPosition()`	Returns the position of this data cell on the screen. **Returns** `{Position}` the position of the data cell on the screen
`getChar()`	Obtains the character associated with the cell. **Returns** `{String}` The character associated with the cell.
`getAttributes()`	Returns the set of attributes specified for this data cell instance. See AttributeSet. **Returns** `{AttributeSet}` The set of attributes for this data cell instance.
`getForegroundColor()`	Returns the foreground color, as defined in the Color object, for this data cell. **Returns** `{Number}` Foreground color for this data cell. The color is defined in the Color object.
`getBackgroundColor()`	Returns the background color, as defined in the Color object, for this data cell. **Returns** `{Number}` Background color for this data cell. The color is defined in the Color object.

`toString`	Converts the internal data cell to a string. **Returns** `{String}` The string representation of a data cell.
`isFieldDelimiter()`	Tests if this cell represents a field delimiter. **Returns** `{Boolean}` True if this cell is a field delimiter, false if otherwise.

Dimension

Represents the size of the screen or screen area.

Method	Description
`Dimension(rows,cols)`	Creates a new Dimension instance. **Parameters** `{Number} rows` screen rows dimension `{Number} cols` screen columns dimension

Field

Use the Field object, along with FieldList, to obtain the information present in a field on the screen.

Method	Description
`getAttributes()`	Returns the set of attributes specified for this field instance. See AttributeSet. **Returns** `{AttributeSet}` The set of attributes for this field.
`getForegroundColor()`	Returns the foreground color for the field. **Returns** `{Number}` Foreground color for this field. These values are defined in the Color object.
`getBackgroundColor()`	Returns the background color of the field. **Returns** `{Number}` Background color for this field. These values are defined in the Color object.

`getStart()`	Returns the starting position of the field. The starting position is the position of the first character of the field. Some host types use a character position to store field level attributes. In this case, the attribute position is not considered the start position. **Returns** `{Position}` Starting position of the field. **Throws** `{RangeError}` For zero length fields.
`getEnd()`	Returns the ending position of the field. The ending position is the position in the presentation space containing the last character of the field. **Returns** `{Position}` Ending position of the field. **Throws** `{RangeError}` For zero length fields.
`getLength()`	Returns the length of the field. For host types that use a character position to store the field attributes, the field length does not include the field attribute position **Returns** `{Number}` Length of the field.
`getDataCells()`	Obtains the data cells that comprise this field. See DataCell. **Returns** `{DataCell[]}` Data cells that comprise this field.
`getText()`	Obtains the text from the field. **Returns** `{String}` field text.
`setText()`	Sets the field text. For certain host types, like VT, the text is transmitted to the host right away, but in other host types, the text is not transmitted to the host until an Aid key is invoked. If the text is shorter than the field, the text is placed in the host field, and the remainder of the field is cleared. If the text is longer than the host field, then as much text as will fit is placed in the field. **Parameter** `{String}` Text to set on the field. **Throws** `{Error}` If the field is protected.
`clearField()`	Clears the current field in an emulation-specific manner. **Throws** `{Error}` If the field is protected or clear is not supported.

`getPresentationSpace()`	Obtains the PresentationSpace that created this field. **Returns** `{PresentationSpace}` Parent of this field instance.
`toString()`	Creates a user-friendly description of the field. **Returns** `{String}` A user readable rendition of the field.

FieldList

Use the FieldList object, along with Field object, to obtain field list information.

Method	Description
`getPresentationSpace()`	Obtains the PresentationSpace that created this field. **Returns** `{PresentationSpace}` Parent of this field instance.
`findField(position, text, direction)`	Returns the field containing the specified text. The search starts from the specified position and proceeds either forward or backward. If the string spans multiple fields, the field containing the starting position is returned. When searching forward the search will not wrap to the top of the screen. When searching backward the search will not wrap to the bottom of the screen. **Parameters** `{Position}` Position from which to start the search. See Position object. `{String}` The text to search for (optional). If not provided, returns the next field to the right of or below the specified position. `{Number}` direction of the search (optional). Use PresentationSpace. SearchDirection constants for this parameter. For example, PresentationSpace.SearchDirection.FORWARD or PresentationSpace.SearchDirection.BACKWARD. If not provided, searches forward. **Returns** `{Field}` containing the string or null if a field meeting the given criteria is not found. **Throws** `{RangeError}` If the position is out of range.

`get(index)`	Obtains the field at the given index. **Parameters** `{Number}` index into the field list. **Returns** `{Field}` located at the specified index. **Throws** `{RangeError}` If the index is out of range.
`isEmpty()`	Determines if the field list is empty. **Returns** `{Boolean}` True if the list is empty.
`size()`	Indicates the number of fields in the list. **Returns** `{Number}` The field count.
`toString()`	Creates a user-friendly description of the field list. **Returns** `{String}` A user readable rendition of the field list.

FileTransfer

Use the FileTransfer object to list and transfer files between the host system and the client.

The Host Access for the Cloud file transfer API abstracts the file path conventions used by different host file implementations. Follow URL or Linux file system path formats when formatting file paths used by the API. For example, `/root/directory/file`.

It is important to observe any rules specific to host systems, such as allowable characters or name lengths.

note

Browsers place significant security restrictions around the ability of Javascript to interact with client file systems.

Method	Description

`getHostFileListing(remotePath)()`	Request a listing of host files. If `RemotePath` is omitted, a file listing for the current remote working directory is shown. **Parameters** `{String}` (optional) If specified will get file listing for specified remote path. If not specified, will get file listing for current remote working directory. **Returns** `{Promise}` Resolves to an array of HostFile objects contained at remoteName. Rejected if the remote path cannot be read.
`sendFile(localFile, remoteName)`	Sends specified file to the host. **Parameters** `{File}` Javascript file object pointing to local file to send. `{String}` (optional) Fully-qualified remote file name as allowed by remote system (Unix, Windows, MVS, VAX). **Returns** `{Promise}` fulfilled with a `HostFile` object representing the sent file on success. Rejected if an error occurred sending the file.
`getDownloadURL(remoteName)`	Constructs a link to download a file from a host system. **Parameters** `{String}` Fully-qualified remote file name as allowed by remote system (Unix, Windows, MVS, VAX). **Returns** `{URL}` that can be used to retrieve the file from the Host Access for the Cloud session server.
`setTransferOptions(options)`	Set transfer options for current FileTransfer session. The transfer options are applied to all future transfers until the session is either exited or overridden by another call to `setTransferOptions`. **Parameters** `{JSON}` see FileTransferOptions for allowed names and values. **Returns** `{Promise}` fulfilled when the call completes. Rejected if an error occurred setting the options.

`cancel()`	Cancels the current transfer in progress. **Parameters** `{String}` Fully-qualified remote file name as allowed by remote system (Unix, Windows, MVS, VAX). **Returns** `{Promise}` fulfilled when the call completes. Rejected if an error occurred canceling the transfer.

FileTransferFactory

A fileTransferFactory object is available to all macros. If file transfers are configured for the session, you can use it to get a reference to a FileTransfer object.

Method	Description
`getIND$File( )`	Returns a FileTransfer object for interacting with the configured Ind$File type for the session. **Returns** `{FileTransfer}` **Throws** `{Error}` If the session hasn't been configured to allow IND$File transfers.

FileTransferOptions

File transfer option object specification. Example:
`fileTransfer.setTransferOptions({ transferMethod : 'ascii' });```

Method	Description

`transferMethod`	`{String}` Allowed values: ‘ascii’ ‘binary’

HostFile

A HostFile object represents a file on the host file system.

Method	Description
`getName()`	Gets the file name. **Returns** `{String}` the file name.
`getParent()`	Gets the parent of this host file. **Returns** `{String}` the parent of this host file. This means different things on different host types. For example on TSO this is the name of the catalog in which the file resides.
`getSize()`	The byte size of the file. **Returns** `{Number}` the size of the file in bytes.
`getType()`	The type of file represented. **Returns**

HostFileType

The HostFileType object defines constants for determining the type of a HostFile object.

Value	Description
FILE	Represents a file on the host system.
DIR	Represents a directory on the host system.

Value	Description
UNKNOWN	Represents a host file of unknown origin.

OIA

Operator Information Area (OIA) interface. The OIA object returns values that are defined in the OIAStatus object.

Method	Description
`getStatus ()`	Returns the set of enabled status flags. See StatusSet. **Returns** `{StatusSet}` Containing the status information.
`getCommErrorCode( )`	Returns the current communication error code. **Returns** `{Number}` the current communication error code. If one doesn't exist, it will be 0.
`getProgErrorCode( )`	Returns the current program error code. **Returns** `{Number}` the current program error code. If one doesn't exist, it will be 0.

OIAStatus

OIAStatus	Description
CONTROLLER_READY	Controller ready
A_ONLINE	Online with a non-SNA connection
MY_JOB	Connected to a host application
OP_SYS	Connected to a SSCP (SNA)
UNOWNED	Not connected
TIME	Keyboard inhibited

OIAStatus	Description
SYS_LOCK	System lock following AID key
COMM_CHECK	Communication check
PROG_CHECK	Program check
ELSEWHERE	Keystroke invalid at cursor location
FN_MINUS	Function not available
WHAT_KEY	Keystroke invalid
MORE_THAN	Too many characters entered in the field
SYM_MINUS	Symbol entered not available
INPUT_ERROR	Operator input error (5250 only)
DO_NOT_ENTER	Do not enter
INSERT	Cursor in insert mode
GR_CURSOR	Cursor in graphics mode
COMM_ERR_REM	Communications error reminder
MSG_WAITING	Message waiting indicator
ENCRYPT	Session is encrypted
NUM_FIELD	Invalid character in numeric only field

Position

Method	Description

`Position(row,col)`	Creates a new Position instance. **Parameters** `{Number}` row screen row coordinate `{Number}` col screen column coordinate

---

PresentationSpace

Use the PresentationSpace object to interact with the terminal screen. Setting and getting the cursor position, sending keys, and reading text are some of the interactions available.

Method	Description
`getCursorPosition()`	Returns a Position instance representing the current cursor position. An unconnected session has a cursor position of 0,0. **Returns** `{Position}` current cursor location
`setCursorPosition(position)`	Moves the host cursor to the specified row and column position. For some hosts, such as VT, the host may constrain the movements of the cursor. **Parameters** `{Position}` Position new cursor position. **Returns** None **Throws** `{RangeError}` If the position is not valid on the current screen.
`isCursorVisible()`	Tests that the cursor is currently visible in the presentation space. The cursor is considered not visible if the session is not connected. **Returns** `{Boolean}` True if the cursor is visible. False if the cursor is not visible.

`sendKeys(keys )`	Transmits a text string or ControlKey to the host at the current cursor position in the presentation space. If the cursor is not in the desired position, then use `setCursorPosition` function first. The text string can contain any number of characters and ControlKey objects. For example: "myname" + `ControlKey.TAB` + "mypass" + `ControlKey.ENTER` will transmit a user ID, tab to the next field, transmit a password, and then transmit the Enter key. If you need to transmit a square bracket, double the brackets ([[or]]). **Parameters** `{String}` keys text and/or control keys to transmit
`getText(start, length)`	Returns a string representing a linear area of the presentation space. No new line characters are inserted if row boundaries are encountered. **Parameters** `{Position}` start position from which to retrieve text `{Number}` length the maximum number of characters to return. If the length parameter causes the last position of the presentation space to be exceeded then only those characters up to the last position will be returned. **Returns** `{String)` representing a linear area of the presentation space which may be empty if the session is not connected. **Throws** `{RangeError}` If the position or length are not valid on the current screen.
`getSize()`	Gets the dimensions of the screen as a Dimension object. **Returns** `{Dimension}` Containing the number of rows and columns. The screen size is [row:0, col:0] if the session is not connected.

`getDataCells(start, length)`	Returns DataCell instances where the first member will be for the position specified by the start parameter. The maximum number of DataCell instances in the list is specified by the length parameter. **Parameters** `{Position}` start the first position on the host screen in which to retrieve DataCell instances. See Position. `{Number}` length of the maximum number of DataCell instance to be retrieved. If not specified, returns DataCells from the start position to the end of the screen. **Returns** `{DataCell[]}` Instances which may be empty if the session is not connected. If position is not specified, returns all DataCells. If length is not specified, returns DataCells from the start position to the end of the screen. **Throws** `{RangeError}` if start or length are out of range.
`getFields()`	Returns a list of the fields in the presentation space. If the host type does not support fields or the current screen is not formatted then the return value will always be an empty list. See FieldList. **Returns** `{FieldList}` of host defined fields in the presentation space.

Session

The session object is the main entry point for interacting with the host. It contains functions for connecting, disconnecting, and obtaining the PresentationSpace object.

Method	Description
`connect()`	Connects to the configured host. If needed, use , `wait.forConnect()` to block macro execution until the session is connected. **Returns** `None`
`disconnect()`	Disconnects from the configured host. If needed, use `wait.forDisconnect()` to block macro execution until the session is connected. **Returns** `None`

`isConnected()`	Determines whether the connection to the host is connected. **Returns** `{Boolean}` true if host connection is established; false if not.
`getPresentationSpace()`	Provides access to the PresentationSpace instance for this session. **Returns** `{PresentationSpace}` instance associated with this session.
`getDeviceName()`	Returns the device name for a connected session or an empty string if the session is disconnected or doesn't have device name. **Returns** `{String}` The connected device name.
`getType()`	Returns the type of host session. See SessionType. **Returns** `{String}` The type of host session.
`setDeviceName()`	Provides a means to modify the device name on a session instance. **Parameters** `{String}` name Device name to use when connecting to a host. **Throws** `{Error}` If an attempt is made to set the device name while the session is connected.
`getOIA()`	Provides access to the OIA instance for this session. **Returns** `{OIA}` Associated with this session

SessionType

Constants used to identity the type of host to which the connection is being made. See Session object.

Available host types:

IBM_3270

IBM_5250

VT

ALC

UTS

T27

StatusSet

You can use the StatusSet object to decode the OIA status. The StatusSet object returns values defined in the OIAStatus object and when used together, you can get status information from the OIA.

Method	Description
`contains(statusFlag)`	Determines if the set contains the specified status flag from OIAStatus constants. **Parameters** `{Number}` statusFlag status to check **Returns** `{Boolean}` True if the status flag is present in the set.
`isEmpty()`	Determines if the status set is empty. **Returns** `{Boolean}` True if the set is empty.
`size()`	Indicates the number of status flags in the set. **Returns** `{Number}` The status count
`toArray()`	Converts the internal status set to an array. **Returns** `{Object []}` Array of status flags in the set.
`toString()`	Converts the internal status set to a string. **Returns** `{String}` Space delimited names of status flags in the set.
`forEach(callback, thisArg)`	Function to iterate over each element in the status set. **Parameters** `{forEachCallback}` Callback to perform the specific operation. Called with the name of each status in the set.

`forEachCallback(string, thisArg)`	A user-provided callback function where you provide the behavior, to be used as the callback parameter to forEach. **Parameters** `{String} String` The name of a status in the status set. `{Object} thisArg` Optional pointer to a context object.

User Interface

The user interface object provides functions for interacting with the user, prompting for and displaying basic information. The UI object is made automatically available in your macro as the “ui” variable”.

note

Important: All UI functions require the ‘yield’ keyword in front of them. This allows the macro to block execution until the conditions of the UI function have been met.
`[parameter]` denotes an optional parameter.

Method	Description
`prompt(message, [defaultAnswer], [mask])`	Prompts the user for information in the user interface. **Parameters** `{String} message title to display to the user`. Default: blank String. `{String} defaultAnswer to use if user leaves it blank`. Default: blank String `{Boolean} mask indicates whether to hide the prompt` (as with a password) **Returns** `{Promise}` Fulfilled when the user closes the dialog window. Returns the users input on “OK” or null on “Cancel”.

`message([message])`	Display a message in the user interface.
	Parameters
	`{String} message to display to the user.`
	`Default: blank String.`
	
	`Returns`
	
	`{Promise}  Fulfilled when the user closes the message window.`

Wait

Use the wait object to wait for a particular session or screen state. For example, you can wait until the cursor is found at a particular location or text is present at a certain location before continuing with the macro execution.

Wait functions are often used in conjunction with asynchronous functions such as connect() and sendKeys().

note

All functions take timeouts as an optional parameter and have a default time out value of 10 seconds (10000ms).
Important: All wait functions require the 'yield' keyword in front of them. This allows the macro to block execution until the conditions of the wait function are met.
`[parameter]` denotes an optional parameter.

Method	Description
`setDefaultTimeout(timeout)`	Sets the default timeout value for all functions. **Parameters** `{Number}` default timeout to use for all wait functions in milliseconds. **Returns** `{None}` **Throws** `{RangeError}` If the specified timeout is less than zero.

`forConnect([timeout])`	Waits for a connect request to complete. **Parameters** `{Number}` in milliseconds. **Returns** `{Promise}` Fulfilled if the session is already connected or when connection occurs. Rejected if the wait times out.
`forDisconnect([timeout])`	Waits for a disconnect request to complete. **Parameters** `{Number}` timeout in milliseconds. **Returns** `{Promise}` Fulfilled if the session is already disconnected or when it finally disconnects. Rejected if the wait times out.
`forFixedTime([timeout])`	Waits unconditionally for fixed time. Time is in milliseconds (ms). **Parameters** `{Number}` timeout in milliseconds. **Returns** `{Promise}` Fulfilled after time elapses.
`forScreenChange([timeout])`	Waits for the host screen to change. This function returns when a screen update is detected. It makes no guarantees about the number of subsequent updates that may arrive before the screen is complete. Waiting repeatedly until the screen contents match some known stopping criteria is advisable.) **Parameters** `{Number}` timeout in milliseconds. **Returns** `{Promise}` Resolved if the screen change. Rejected if the wait times out.
`forCursor(position, [timeout])`	Waits for the cursor to arrive at the specified position. **Parameters** `{Position}` The position specifying the row and column **Returns** `{Promise}` Fulfilled if the cursor is already located or when it is finally located. Rejected if the wait times out.

`forText(text, position, [timeout])`	Wait for text located at a specific position on the screen **Parameters** `{String}` text to expect `{Position}` position specifying the row and column `{Number}` timeout in milliseconds **Returns** `{Promise}` Fulfilled if the text is already at the specified position or whenever it is located. Rejected if the wait times out. **Throws** `{RangeError}` if the position is not valid.
`forHostPrompt(text, column, [timeout])`	Waits for a command prompt located at a particular column on the screen. **Parameters** `{String}` text prompt to expect `{Number}` column where cursor is expected `{Number}` timeout in milliseconds. **Returns** `{Promise}` Fulfilled if the conditions are already met or when the conditions are finally met. Rejected if the wait times out. **Throws** `{RangeError}` if the column is out of range.

`forHostSettle( [settleTime], [timeout])`	**NOTE:** `wait.forHostSettle` should only be used when other more targeted wait functions are insufficient. Monitors incoming screen data and resolves settleTime ms after the last update **and** the keyboard is unlocked. This function is useful when data arrives in multiple packets and you want to be sure the whole screen has been received before carrying on. **Parameters** `{Number}` time to wait after the last update to make sure more data doesn't arrive unexpectedly. The default is 200 milliseconds. `{Number}` timeout in milliseconds. **Returns** `{Promise}` Fulfilled when the settle time has elapsed after receipt of the last screen update and the keyboard is unlocked.

Sample Macros

To help you create successful macros that take advantage of all the capabilities of the Macro Editor, these samples are available as a starting point.

Basic Host Interaction

This sample illustrates basic host interaction, including:

- Sending data to the host
- Waiting for screens to display
- Using the `yield` keyword to wait for asynchronous functions
- Reading text from the screen
- Displaying basic information to the user
- Handling error basics

All macros have the following objects available by default:

1. **session** - main entry point for access to the host. Can connect, disconnect and provides access to the PresentationSpace.

   The PresentationSpace object obtained from the session represents the screen and provides many common capabilities, such as getting and setting the cursor location, sending data to the host, and reading from the screen.

2. **wait** - provides a simple way to wait for various host states before continuing to send more data or read from the screen.
3. **UI** - provides basic user interface capabilities. Display data to the user or prompt them for information.

```
// Create a new macro function
var macro = createMacro(function*(){
'use strict';

// All macros have the following objects available by default:
// 1. session - Main entry point for access to the host. Can connect,
disconnect and provides access to the PresentationSpace.
//    The PresentationSpace object obtained from the session represents the
screen and provides many common capabilities such as getting and setting the
//    cursor location, sending data to the host and reading from the screen.
// 2. wait - Provides a simple way to wait for various host states before
continuing to send more data or read from the screen.
// 3. ui - Provides basic User Interaction capabilities. Display data to the
user or prompt them for information.

// Declare a variable for reading and displaying some screen data.
// As a best practice all variables should be declared near the top of a
function.
var numberOfAccounts = 0;

// Start by obtaining the PresentationSpace object, which provides many
common screen operations.
var ps = session.getPresentationSpace();

try {
    // Can set and get the cursor location
    ps.setCursorPosition(new Position(24, 2));

    // Use the sendKeys function to send characters to the host
    ps.sendKeys('cics');

    // SendKeys is also used to send host keys such as PA and PF keys.
    // See "Control Keys" in the documentation for all available options
    ps.sendKeys(ControlKey.ENTER);

    // Wait for the cursor to be at the correct position.
    // The wait object provides various functions for waiting for certain
states to occur
    // so that you can proceed to either send more keys or read data from the
screen.
    yield wait.forCursor(new Position(24, 2));

    // You can mix characters and control keys in one sendKeys call.
    ps.sendKeys('data' + ControlKey.TAB + ControlKey.TAB + 'more data' +
ControlKey.ENTER);
```

```
    // The "yield" keyword must be used in front of all "wait" and "ui" 
function calls.
    // It tells the browser to pause execution of the macro until the
    // (asynchronous) wait function returns.  Consult the documentation for 
which functions
    // require the yield keyword.
    yield wait.forCursor(new Position(10, 26));
    ps.sendKeys('accounts' + ControlKey.ENTER);

    // Can also wait for text to appear at certain areas on the screen
    yield wait.forText('ACCOUNTS', new Position(3, 36)) ;
    ps.sendKeys('1' + ControlKey.ENTER);

    // All wait functions will timeout if the criteria is not met within a 
time limit.
    // Can increase timeouts with an optional parameter in the wait functions 
(in milliseconds)
    // All timeouts are specified in milliseconds and the default value is 10 
seconds (10000ms).
    yield wait.forCursor(new Position(1, 1), 15000);
    ps.sendKeys('A' + ControlKey.ENTER);

    // PS provides the getText function for reading text from the screen
    numberOfAccounts = ps.getText(new Position(12, 3), 5);

    // Use the ui object to display some data from the screen
    ui.message('Number of active accounts: ' + numberOfAccounts);

    // The try / catch allows all errors to be caught and reported in a 
central location
} catch (error) {
    // Again we use the ui object to display a message that an error occurred
    yield ui.message('Error: ' + error.message);
}
//End Generated Macro
});

// Run the macro and return the results to the Macro Runner
// The return statement is required as the application leverages
// this to know if the macro succeeded and when it is finished
return macro();
```

User Interaction

This sample illustrates how to use the provided API methods to prompt the user for input or alert them with a message.

```
var macro = createMacro(function*(){
  'use strict';

  // The "ui" object provides functions for prompting the user for
information and displaying information

  // Declare variables for later use
  var username;
  var password;
  var flavor;
  var scoops;

  //Begin Generated Macro
  var ps = session.getPresentationSpace();

  try {
    // Prompt the user to enter their name and store it in a variable.
    // Note that 'yield' keyword is needed to block execution while waiting
for the user input.
    username = yield ui.prompt('Please enter your username');

    // Prompt the user to enter a value with a default provided to them.
    flavor = yield ui.prompt('What is your favorite flavor of ice cream?',
'Chocolate');

    // Prompt the user to enter private information by using the 'mask'
option and the input field will be masked as they type.
    // If a parameter is not used, 'null' can be used to specify that it
isn't to be used.
    // Here we illustrate that by specifying that we don't need to show a
default value .
    password = yield ui.prompt('Please enter your password', null,
true);

    // The prompt function returns null if the user clicks the 'Cancel'
button instead of the 'OK' button.
    // One way to handle that case is to wrap the call in a try/catch block.
    scoops = yield ui.prompt('How many scoops would you like?');
    if (scoops === null) {
      // This will exit the macro.
      return;
      // Alternatively could throw an Error and have it be caught in the
"catch" below
     }
     // Use the collected values to order our ice cream
    ps.sendKeys(username + ControlKey.TAB + password + ControlKey.ENTER);
    yield wait.forCursor(new Position(5, 1));
    ps.sendKeys(flavor + ControlKey.TAB + scoops + ControlKey.ENTER);

     // Display a message to the user.  Using the 'yield' keyword in front of
the call will block
     // further execution of the macro until the user clicks the 'OK' button.
    yield ui.message('Order successful.  Enjoy your ' + scoops + ' scoops of
' + flavor + ' ice cream ' + username + '!');
    } catch (error) {
     // Here we use the ui object to display a message that an error occurred
    yield ui.message(error.message);
    }
    //End Generated Macro

});

return macro();
```

Paging Through Data

This sample illustrates how to page through a variable number of screens and process the data on each screen.

```
 // Create a new macro function.
var macro = createMacro(function*(){
  'use strict';

  // Create variable(s) for later use
  var password;
  var accountNumber;
  var transactionCount = 0;
  var row = 0;

  // Obtain a reference to the PresentationSpace object.
  var ps = session.getPresentationSpace();

  try {
    // Enter Username and Password to log on to the application.
    yield wait.forCursor(new Position(19, 48));
    ps.sendKeys('bjones' + ControlKey.TAB);

    yield wait.forCursor(new Position(20, 48));
    password = yield ui.prompt('Password:', null, true);
    ps.sendKeys(password);
    ps.sendKeys(ControlKey.ENTER);

    // Enter an application command.
    yield wait.forCursor(new Position(20, 38));
    ps.sendKeys('4');
    ps.sendKeys(ControlKey.ENTER);

    // Going to list transactions for an account.
    yield wait.forCursor(new Position(13, 25));
    ps.sendKeys('2');
    // Input an account number. Hard coded here for simplicity.
    yield wait.forCursor(new Position(15, 25));
    accountNumber = yield ui.prompt('Account Number:', '167439459');
    ps.sendKeys(accountNumber);
    ps.sendKeys(ControlKey.ENTER);

    // Wait until on account profile screen
    yield wait.forText('ACCOUNT PROFILE', new Position(3, 33));

    // Search for text that indicates the last page of record has been
reached
    while (ps.getText(new Position(22, 12), 9) !== 'LAST PAGE') {

      // While the last page of record has not been reached, go to the next
page of records.
```

```
      ps.sendKeys(ControlKey.PF2);
      yield wait.forCursor(new Position(1, 1));

      // If the cursor position does not change between record screens, and
there is no text
      // on the screen you can check to confirm a screen is updated, you may
wait for a
      // fixed time period after an aid key is sent for the screen to settle.
      // For example:
      // yield wait.forFixedTime(1000);

      // For each of the rows, increment the count variable if it contains
data.
      for (row = 5; row <= 21; row++) {

        // There are 2 columns on the screen. Check data on column 1.
        // In this example we know that if there is a space at a particular
        // position then there is a transaction.
        if (ps.getText(new Position(row, 8), 1) !== ' ') {
          transactionCount++;
        }
        // Check data on column 2.
        if (ps.getText(new Position(row, 49), 1) !== ' ') {
          transactionCount++;
        }
      }
    }

    // After going through all record pages, display the number of records in
a message box.
    yield ui.message('There are ' + transactionCount + ' records found for
account ' + accountNumber + '.');

    // Log out of the application
    ps.sendKeys(ControlKey.PF13);
    ps.sendKeys(ControlKey.PF12);

    // The try / catch allows all errors to be caught and reported in a
central location
  } catch (error) {
    // Here we use the ui object to display a message that an error occurred
    yield ui.message(error.message);
  }
});

// Here we run the macro and return the results to the Macro Runner
// The return statement is required as the application leverages
// this to know if the macro succeeded
return macro();
```

Invoking a Web Service

This sample illustrates how to make an AJAX / REST call directly from a macro to a web service. You can integrate data from your host application into the web service call or from the web service into your host application.

In this example, we are calling the Verastream Host Integrator (VHI) CICSAcctsDemo REST service. However, you can easily adapt the code to call any web service. You are not limited to VHI.

In the example, the call goes through a proxy configured in the session server (shown below) to avoid a “Same Origin Policy” complication. If you are using a web service that supports Cross-origin Resource Sharing (CORS) and are using a modern browser, the proxy is unnecessary.

Since the jQuery library is available in macros, you may use the $.post() function directly to invoke REST services.

This example also demonstrates how to wrap a jQuery REST call in a new Promise. The promise returned from the custom function below allows "yield" to be used in the main macro code. This allows the main macro execution to wait until the service call is complete before continuing.

```
 var macro = createMacro(function*() {
  'use strict';

  // Create a few variables for later user
  var username;
  var password;
  var accountNumber;
  var accountDetails;

  // Create a function that will make an AJAX / REST call to a VHI Web
Service.
  // Could be adjusted to call any web service, not just VHI.
  // If not using CORS, the request will likely need to pass through a
  // proxy on the session server. See sample notes for more information.
  /**
   * Hand-coded helper function to encapsulate AJAX / REST parameters, invoke
the
   * REST service and return the results inside a Promise.
   * @param {Number} acctNum to send to the REST query.
   * @param {String} username to access the REST service.
   * @param {String} password to access the REST service.
   * @return {Promise} containing $.post() results that are compatible with
yield.
  */
  var getAccountDetails = function (acctNum, username, password) {
    var url = "proxy1/model/CICSAcctsDemo/GetAccountDetail";
    var args = {"filters": {"AcctNum": acctNum}, "envVars": {"Username":
username, "Password": password}};

    // Wrap a jQuery AJAX / HTTP POST call in a new Promise.
    // The promise being returned here allows the macro to yield / wait
    // for its completion.
    return Promise.resolve($.post(url, JSON.stringify(args)))
      .catch(function (error) {
      // Map errors that happen in the jQuery call to our Promise.
      throw new Error('REST API Error: ' + error.statusText);
    });
  };

  // Begin Generated Macro
  var ps = session.getPresentationSpace();
  try {
    // Could interact with the host here, log into a host app, etc...
    // Gather username and password
    username = yield ui.prompt('Username:');
    password = yield ui.prompt('Password:', null, true);
    accountNumber = yield ui.prompt('Account Number:');
    if (!username || !password || !accountNumber) {
      throw new Error('Username or password not specified');
    }

    // Invoke external REST service, and yields / waits for the call to
complete.
    accountDetails = yield getAccountDetails(accountNumber, username,
password);

    // We now have the data from our external service.
    // Can integrate the data into our local host app or simply display it to
the user.
    // For this sample we simply display the resulting account details.
    if (accountDetails.result && accountDetails.result.length > 0) {
      yield ui.message(accountDetails.result[0].FirstName + ' $' +
accountDetails.result[0].AcctBalance);
    } else {
      yield ui.message('No records found for account: ' + accountNumber);
    }
  } catch (error) {
    // If an error occurred during the AJAX / REST call
    // or username / password gathering we will end up here.
    yield ui.message(error.message);
  }
});

// Run our macro
return macro();
```

Cross Origin Scripting Proxy Support

If you have web services that do not support CORS, any AJAX/REST calls will fail if they attempt to access a server other than the one where the Host Access for the Cloud application originated. This is a browser security feature.

The Host Access for the Cloud server provides a way to explicitly proxy to trusted remote servers.

- Open `..\<install_dir>\sessionserver\microservice\sessionserver\service.yml` for editing.
- In the `env` section add:

```
name: zfe.proxy.mappings
value: proxy-path=proxy-to-address
```

Where `proxy-path` refers to the desired url-mapping and `proxy-to-address` refers to the URL where the call will be proxied.

- In this example:

```
name: zfe.proxy.mappings
value: proxy1=http://remote-vhi-server:9680/vhi-rs/
```

Calls made to `<server:port>/proxy1` will be proxied to `http://remote-vhi-server:9680/vhi-rs/`

- Multiple proxy mappings can be specified using a comma to separate the individual proxy mappings
- Keep in mind that even when a REST server supports CORS headers, some older browsers may not, so this example may still be relevant.

hint

Your `service.yml` file may be replaced whenever you redeploy Host Access for the Cloud. Always back up your files.

Working with Data Cells and Attributes

This macro illustrates how to use DataCells and AttributeSet to inspect a given row/column on the screen for text and attributes. In this sample you can see:

- How to get a collection of DataCells for a given position and length.
- How to iterate through DataCells to build up a text string
- How, for comparison, you can also do a similar thing using getText().
- And finally, how to work with attributes, get a string listing, or determine whether specific ones are set at a given screen location.

```
var macro = createMacro(function*() {
    'use strict';

    // Obtain the PresentationSpace for interacting with the host
    var ps = session.getPresentationSpace();

    // Declare variables for later use
    var cells;
    var text;
    var attrs;

    // Set the default timeout for "wait" functions
    wait.setDefaultTimeout(10000);

    // Sample macro for working with DataCells and Attributes
    try {
        yield wait.forCursor(new Position(24, 2));

        // Get DataCells from the presentation space.
        // Row 19, col 3 is the prompt, 35 characters long
        // "Choose from the following commands:"
        cells = ps.getDataCells({row:19, col:3}, 35);
        text = '';

        // You can display text using getText
        yield ui.message("Screen text: " + ps.getText({row:19, col:3}, 35));

        // Or you can assemble the text from the DataCells at each position
        for(var index = 0; index < cells.length; index++) {
            text = text.concat(cells[index].getChar());
        }
        // And display the text
        yield ui.message("Cells text: " + text);

        // Get the attributes for the first DataCell (cell[0])
        attrs = cells[0].getAttributes();

        // Display whether we have any attributes on the data cell
        yield ui.message("Attribute set is empty: " + attrs.isEmpty());

        // Display how many attributes are set
        yield ui.message("Number of attributes: " + attrs.size());

        // Display which attributes are set
        yield ui.message("Attributes: " + attrs.toString());
        // Now display whether the high intensity attribute is set
        yield ui.message("Is high intensity: " +
                         attrs.contains(Attribute.HIGH_INTENSITY));

        // Now display whether the underline attribute is set
        yield ui.message("Is underline: " +
                         attrs.contains(Attribute.UNDERLINE));

        // Now display whether alphanumeric, intensified and pen-detectable
attributes are set
        yield ui.message("Is alphanumeric, intensified and pen-detectable: "
+
                         attrs.containsAll([Attribute.ALPHA_NUMERIC,
Attribute.HIGH_INTENSITY, Attribute.PEN_DETECTABLE]));

        // Now display whether underline, intensified and pen-detectable
attributes are set
        yield ui.message("Is underline, intensified and pen-detectable: " +
                         attrs.containsAll([Attribute.UNDERLINE,
Attribute.HIGH_INTENSITY, Attribute.PEN_DETECTABLE]));
    } catch (error) {
        yield ui.message(error);
    }
    //End Generated Macro
});

// Run the macro
return macro();
```

Using Fields and Field Lists

This macro sample illustrates how to use common functions to interact with the fields in the Macro API. For example, how to get field text, view field information, and use field.setText as an alternative to sendKeys to interact with the host.

note

Due to browser considerations, `ui.message` collapses strings of spaces down to a single space. The spaces are preserved in the actual JavaScript.

```
var macro = createMacro(function*() {
    'use strict';

    // Obtain the PresentationSpace for interacting with the host
    var ps = session.getPresentationSpace();

    // Declare variables for later use
    var fields;
    var field;
    var searchString = 'z/VM';

    // Set the default timeout for "wait" functions
    wait.setDefaultTimeout(10000);

    // Sample macro for working with FieldList and Fields
    try {
        yield wait.forCursor(new Position(24, 2));

        // Get the field list.
        fields = ps.getFields();

        // Run through the entire list of fields and display the field info.
        for(var index = 0; index < fields.size(); index++) {
            field = fields.get(index);

            yield ui.message("Field " + index + " info: " +
field.toString());
        }

        yield ui.message("Now, find a field containing the text '" +
searchString + "'");
        field = fields.findField(new Position(1, 1),
searchString);

        if(field !== null) {
            yield ui.message("Found field info: " + field.toString());
            yield ui.message("Found field foreground is green? " +
(Color.GREEN === field.getForegroundColor()));
            yield ui.message("Found field background is default? " +
(Color.BLANK_UNSPECIFIED === field.getBackgroundColor()));
        }

        // Now, find command field and modify it.
        field = fields.findField(new Position(23, 80));
        if(field !== null) {
            field.setText("cics");
        }

        yield ui.message("Click to send 'cics' to host.");
        ps.sendKeys(ControlKey.ENTER);

        // Wait for new screen; get new fields.
        yield wait.forCursor(new Position(10, 26));
        fields = ps.getFields();

        // Find user field and set it.
        field = fields.findField(new Position(10, 24));
        if(field !== null) {
            field.setText("myusername");
        }

        // Find password field and set it.
        field = fields.findField(new Position(11, 24));
        if(field !== null) {
            field.setText("mypassword");
        }

        yield ui.message("Click to send login to host.");
        ps.sendKeys(ControlKey.ENTER);

        // Wait for new screen; get new fields.
        yield wait.forCursor(new Position(1, 1));
        fields = ps.getFields();

        // Find command field and set logoff command.
        field = fields.findField(new Position(24, 45));
        if(field !== null) {
            field.setText("cesf logoff");
        }

        yield ui.message("Click to send logoff to host.");
        ps.sendKeys(ControlKey.ENTER);

    } catch (error) {
        yield ui.message(error);
    }
    //End Generated Macro
});

// Run the macro
return macro();
```

Automatic Sign-On Macro for Mainframes

In this example the Autosignon object is used to create a macro that uses the credentials associated with a user to obtain a pass ticket from the Digital Certificate Access Server (DCAS).

```
var macro = createMacro(function*() {
    'use strict';

    // Obtain the PresentationSpace for interacting with the host
    var ps = session.getPresentationSpace();

    // Variable for login pass ticket
    var passTicket;

    // Login application ID
    var appId = 'CICSV41A';

    // Set the default timeout for "wait" functions
    wait.setDefaultTimeout(10000);

    // Begin Generated Macro
    try {
        yield wait.forCursor(new Position(24, 2));

        // Obtain a pass ticket from DCAS.
        passTicket = yield autoSignon.getPassTicket(appId);

        ps.sendKeys('cics');
        ps.sendKeys(ControlKey.ENTER);

        yield wait.forCursor(new Position(10, 26));

        // Replace generated username with sendUserName(passTicket) ...
        yield autoSignon.sendUserName(passTicket);

        // ps.sendKeys('bvtst01' + ControlKey.TAB + ControlKey.TAB);
        ps.sendKeys(ControlKey.TAB + ControlKey.TAB);

        yield wait.forCursor(new Position(11, 26));

        // Replace generated password with sendPassword(passTicket) ...
        yield autoSignon.sendPassword(passTicket);

        // var userInput3 = yield ui.prompt('Password:', '', true);
        // if (userInput3 === null) {
            // throw new Error('Password not provided');
        // }
        // ps.sendKeys(userInput3);
        ps.sendKeys(ControlKey.ENTER);

        yield wait.forCursor(new Position(1, 1));
```

```
            yield ui.message('Logged in. Log me off.');
            ps.sendKeys('cesf logoff');
            ps.sendKeys(ControlKey.ENTER);
        } catch (error) {
            yield ui.message(error);
        }
        //End Generated Macro
    });

    // Run the macro
    return macro();
```

Using File Transfer (IND$File)

This series of sample macros demonstrate how to use the File Transfer API to retrieve a list of files, download a file, and upload a file to a 3270 host.

note

You must be logged in and at a ready prompt before running these macros.

- List files
- Download file
- Upload file

List files

This macro demonstrates how to use the File Transfer API to retrieve a list of files on a 3270 host using IND$File transfer. The IND$File transfer object is retrieved from the file transfer factory and then used to obtain an array of HostFile objects from either TSO or CMS.

```
var macro = createMacro(function*() {
    'use strict';

    try {
        var fileTransfer = fileTransferFactory.getInd$File();
        var hostFiles = yield fileTransfer.getHostFileListing();

        yield ui.message('Found ' + hostFiles.length + ' files');
        if (hostFiles.length > 0) {
            var firstFile = hostFiles[0];
            var msg1 = 'The catalog name is ' + firstFile.getParent() + '.
';
            var msg2 = 'The first file is ' + firstFile.getName();
            yield ui.message(msg1 + msg2);
        }
    } catch (error) {
        yield ui.message(error);
    }
});

// Run the macro
return macro();
```

Download file

This macro shows how to use the File Transfer API to download a file from a 3270 host using IND$File transfer. The IND$File transfer object is retrieved from the file transfer factory. In this example, the transfer method is set to ASCII to demonstrate use of the setTransferOptions function.

The sample macro downloads the first file returned from a call to getHostFileListing by creating a download URI with a call to the getDownloadUrl function. The macro can be used in either a CMS or TSO environment but the choice must be specified on the first line or the code modified slightly for the intended system.

```
var hostEnvironment = 'CMS';  // 'TSO'
// Construct file path, ie catalog/file.name or catalog/partition/file
function getPath (fileNode) {
    var prefix = fileNode.getParent() ? fileNode.getParent() + '/' : '';
    return prefix + fileNode.getName();
}

var macro = createMacro(function*() {
    'use strict';

    try {
        var fileTransfer = fileTransferFactory.getInd$File();

        // The transferMethod options are 'binary' and 'ascii'
        fileTransfer.setTransferOptions({transferMethod: 'ascii'});

        // This demo retrieves the first file returned in the listing
        var hostFiles = yield fileTransfer.getHostFileListing();
        var firstHostFile = hostFiles[0];

        if (hostEnvironment === 'CMS') {
           yield wait.forText('Ready', new Position(1,1), 5000);
        }

        // Download
        // If you already know the path of the file you want, just pass that
to getDownloadURL()
        var downloadUrl =
fileTransfer.getDownloadURL(getPath(firstHostFile));

        // This changes the browser location. You may experience different
results on different browsers
        window.location = downloadUrl;

        // If you want to read the file contents into a variable instead of
downloading
        // it, you can use jQuery
        // var fileContents = yield $.get(downloadUrl);

    } catch (error) {
        yield ui.message(error);
    }
});

// Run the macro
return macro();
```