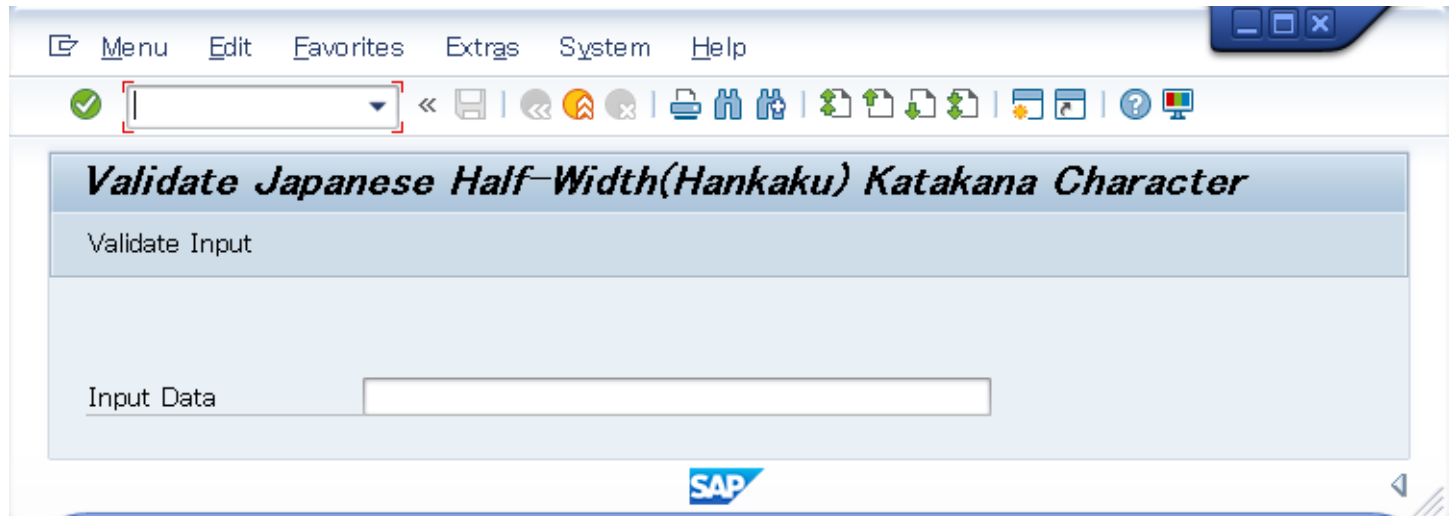


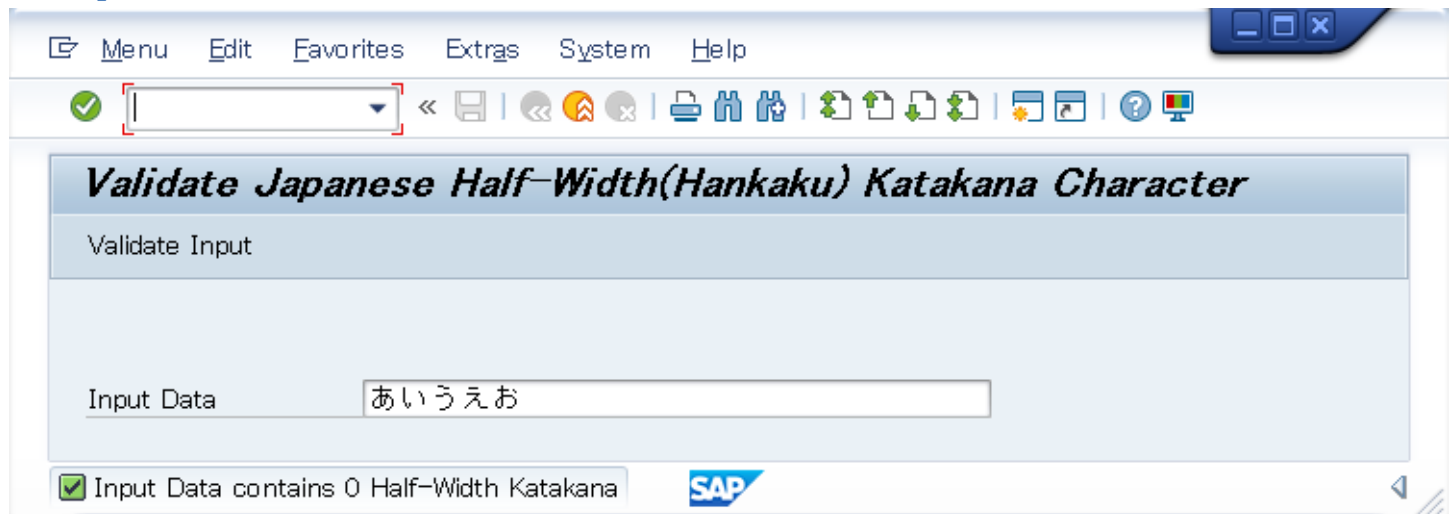
Liquid UI: Validate Japanese Half-Width (Hankaku) Katakana Character

User Interface



The screenshot shows the SAP Liquid UI interface for validating Japanese Half-Width (Hankaku) Katakana characters. The interface includes a menu bar with options: Menu, Edit, Favorites, Extras, System, and Help. Below the menu bar is a toolbar with various icons. The main area has a title bar that reads "Validate Japanese Half-Width(Hankaku) Katakana Character". Below the title bar is a section labeled "Validate Input". Underneath this section is an input field labeled "Input Data" which is currently empty. The SAP logo is visible in the bottom right corner.

Example 1: No Half-width Katakana



The screenshot shows the same SAP Liquid UI interface as before, but with the input field "Input Data" containing the Japanese characters "あいうえお". The status bar at the bottom of the window displays a green checkmark and the text "Input Data contains 0 Half-Width Katakana". The SAP logo is visible in the bottom right corner.

Example 2. Contains Half-width Katakana

The screenshot shows a SAP dialog box titled "Validate Japanese Half-Width(Hankaku) Katakana Character". The dialog has a menu bar with "Menu", "Edit", "Favorites", "Extras", "System", and "Help". Below the menu bar is a toolbar with various icons. The main area of the dialog is light blue and contains the text "Validate Input". Below this is a text input field labeled "Input Data" containing the Japanese text "あかいきうくえけお". At the bottom of the dialog, there is a status bar with a green checkmark icon and the text "Input Data contains 5 Half-Width Katakana". The SAP logo is visible in the bottom right corner of the dialog.

Validate Japanese Half-Width(Hankaku) Katakana Character

Validate Input

Input Data: あかいきうくえけお

☒ Input Data contains 5 Half-Width Katakana

SAP

Liquid UI Code [Script]

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Author: Synactive, Inc. [1065 E. Hillsdale Blvd, Foster City, CA, 94404, USA]
// Email: support@guixt.com; sales@guixt.com;
// Contact: 650.341.3310
// Version: 1.0.0.0
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//Validate Japanese Half-Width(Hankaku) Katakana Character
//User interface

title("Validate Japanese Half-Width(Hankaku) Katakana Character");

del("P[User menu]");
del("P[SAP menu]");
del("P[SAP Business Workplace]");
del("P[Other menu]");
del("P[Add to Favorites]");
del("P[Delete Favorites]");
del("P[Change Favorites]");
del("P[Move Favorites down]");
del("P[Move Favorites up]");
del("P[Create role]");
del("P[Assign users]");
del("P[Documentation]");

clearscreen();

inputfield([2,2], "Input Data", [2,20], {"name":"z_input", "size":40});

pushbutton([TOOLBAR], "Validate Input", "?", {"process":validateHalfWidthKatakana});

//Function to validate the converted unicode result
function validateHalfWidthKatakana(){
    onscreen ""
    //Logic to split the input data into an array by actual characters
    var splited_ary = getSplitedTextAry(z_input);

    var regular_char_counter = 0;
    var half_width_char_counter = 0;
    var full_width_char_counter = 0;
    var other_char_counter = 0;
    var cur_unicode = "";

    //Logic to calculate how many Half-Width Katakana/regular character/number/syntax in the input data
    for(var k=0; k<splited_ary.length; k++){
        if(splited_ary[k].length > 1){
            cur_unicode = getUnicode(splited_ary[k]); //Logic to get Unicode for each character

            if( (cur_unicode >= "3000" && cur_unicode <= "30ff") || //Punctuation, Hiragana, Katakana
                (cur_unicode >= "FF00" && cur_unicode <= "FF9F") || //Full-width Roman, Half-width Katakana
                (cur_unicode >= "4E00" && cur_unicode <= "9FAF") || //CJK (Common & Uncommon)
                (cur_unicode >= "3400" && cur_unicode <= "4DBF")){ //CJK Ext. A (Rare)
                if(cur_unicode >= "FF61" && cur_unicode <= "FF9F"){//It's a Half width katakana (Hankaku)
                    half_width_char_counter++;
                } else{
                    //It's either Punctuation, Hiragana, Katakana, Full-width Roman, CJK, or CJK Ext. A
                    full_width_char_counter++;
                }
            } else {
                other_char_counter++; //Unhandled characters
            }
        } else {
            regular_char_counter++; //It's regular character/number/syntax
        }
    }
    message("S:Input Data contains " + half_width_char_counter + " Half-Width Katakana");

    enter("?");
}

```

```

//Function to return an array with splitted text
function getSplittedTextAry(str){
    var result_ary = [];
    var ref_str = str;

    var converted_str = encodeURIComponent(str);    //Converted the string becomes encoded result
    var converted_str_ary = [];
    var ref_str_ary = [];

    //Loop until the converted string becomes nothing
    while(converted_str.length > 0){
        //If Unicode character is found in the string
        if(converted_str.indexOf("%") > -1){
            //If Unicode character is not from the first character
            if(converted_str.indexOf("%") != 0){
                converted_str_ary.push(converted_str.slice(0,converted_str.indexOf("%")));
                ref_str_ary.push(ref_str.slice(0,converted_str.indexOf("%")));

                ref_str = ref_str.slice(converted_str.indexOf("%"));
                converted_str = converted_str.slice(converted_str.indexOf("%"));
            }
            //When Unicode character is from the first character
            else {
                //Every actual character are 18 char long after encoded
                converted_str_ary.push(converted_str.slice(0,18));
                //Every actual character are 3 char long before encoded
                ref_str_ary.push(ref_str.slice(0,3));

                converted_str = converted_str.slice(18);    //Subtract the string
                ref_str = ref_str.slice(3);    //Subtract the string
            }
        }
        //When Unicode character is not found in the string
        else {
            converted_str_ary.push(converted_str);    //Push the rest of the string
            ref_str_ary.push(ref_str);    //Push the rest of the string
            converted_str = "";    //Clear string
        }
    }

    //Reform the return string
    for(var k=0; k<converted_str_ary.length; k++){
        if(converted_str_ary[k].indexOf("%") < 0){
            //If it's a regular character/string
            for(var j=0; j<ref_str_ary[k].length; j++){
                //Push individual characters into the array
                result_ary.push(ref_str_ary[k].charAt(j));
            }
        } else {
            //If it's a multi-byte character
            result_ary.push(ref_str_ary[k]);    //Push entire multi-byte character into the array
        }
    }

    return result_ary;
}

```

```

//Function to return Unicode in string
function getUnicode(str){
    var result_ary = [];

    //Logic to get the actual byte value for each unencoded character
    for(var k=0; k<str.length; k++){
        println("===>>" + str.charCodeAt(k).toString(2).substring(24,32) + "<===");
        result_ary.push(str.charCodeAt(k).toString(2).substring(24,32));
    }

    //Logic to form multi-byte data become 16-bit hex value
    switch(result_ary.length){
        case 1: // U+00000000 - U+0000007F 0xxxxxxx
            var ucode = "";
            break;

        case 2: // U+00000080 - U+000007FF 110xxxxx 10xxxxxx
            var ucode = "";
            break;

        case 3: //U+00000800 - U+0000FFFF 1110xxxx 10xxxxxx 10xxxxxx
            var c1 = parseInt(result_ary[0],2);
            var c2 = parseInt(result_ary[1],2);
            var c3 = parseInt(result_ary[2],2);

            var b1 = (c1 << 4) | ((c2 >> 2) & 0x0F);
            var b2 = ((c2 & 0x03) << 6) | (c3 & 0x3F);
            var ucode = ((b1 & 0x00FF) << 8) | b2;
            break;

        case 4: // U+00010000 - U+001FFFFF 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
            var ucode = "";
            break;

        case 5: // U+00200000 - U+03FFFFFF 111110xx 10xxxxxx 10xxxxxx 10xxxxxx
            var ucode = "";
            break;

        case 6: // U+04000000 - U+7FFFFFFF 1111110x 10xxxxxx 10xxxxxx 10xxxxxx
            var ucode = "";
            break;
    }

    println("===>>Unicode="+ucode.toString(16).toUpperCase()+"<===");

    return ucode.toString(16).toUpperCase(); //Return Unicode value in hex
}

```