PEARSON
School Systems

Chancery SMS®

Version 6.6 or higher

cTools Client Validation Guide

June 2007
CSL - 12807

Table of Contents

Design Overview

The client validation feature allows users to attach custom validation code to data entry fields using JavaScript. Also, this feature allows users to validate the value of the data entry field against the value or values of another data entry fields(s) appearing in the same Web page, which are all defined in the Customization Builder in cTools.

Chancery SMS supports several data entry controls.

Existing Controls	Description	Supported
Text Box	Data entry field for entering alphanumeric information.	Yes
Numeric Box	Data entry field for entering numeric information.	Yes
Date Box	Data entry field for entering or selecting date information.	Yes
Drop-down List	Data entry field for selecting an item from the list.	Yes
Check Box	Data entry for checking or un-checking the item.	Yes
List Mover	Data entry field for choosing multiple items from a pool of items.	Yes
Memo		Yes
Collection	Display several fields. A grid may be for display only or for editing. Editable fields maybe a Text Box, Numeric Box, Date Box or Drop-down List. Client validation will be dependent on each of the editable fields' setup.	No
Attachment	Data entry field for entering file names for uploading.	No
Setup List Pair		No
Text Object		No

The client validation support is broken into three processes:

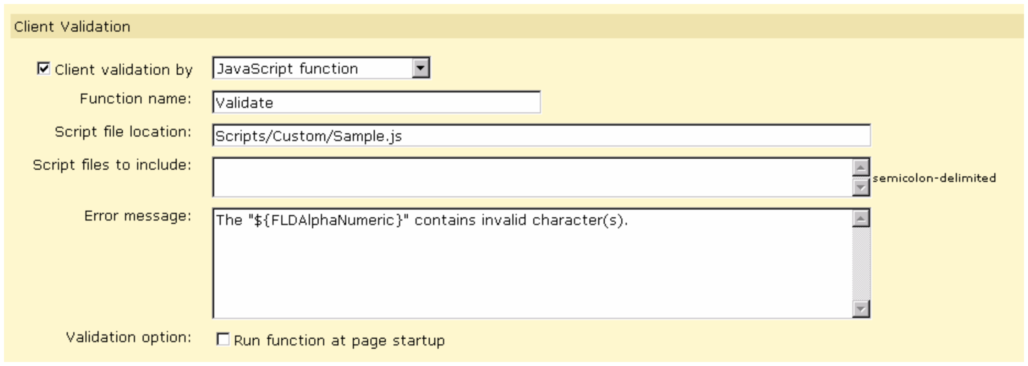
- Setup
- Compilation
- Run-time.

Setup Process

The setup supports two ways of specifying the client validation rule: by JavaScript function, or by JavaScript code block.

The following illustrates the data entry forms for specifying the client validation rule:

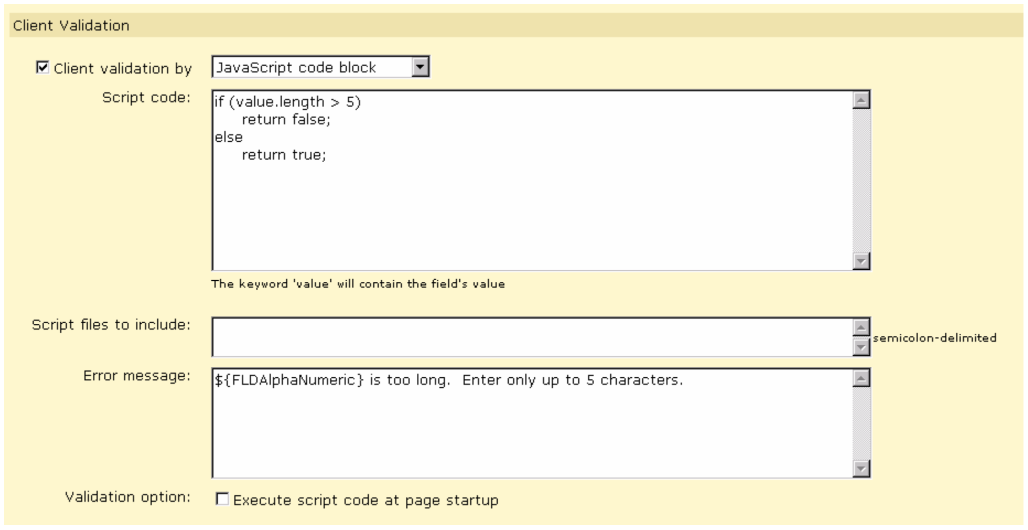
- Specifying client validation by specifying the JavaScript function name to use and the location of the file that contains this function.



Client Validation

☑ Client validation by	JavaScript function
Function name:	Validate
Script file location:	Scripts/Custom/Sample.js
Script files to include:	(semicolon-delimited)
Error message:	The "${FLDAlphaNumeric}" contains invalid character(s).
Validation option:	☐ Run function at page startup

- Specifying client validation by entering JavaScript code block.



Client Validation

☑ Client validation by: JavaScript code block

Script code:

```
if (value.length > 5)
     return false;
else
     return true;
```

The keyword 'value' will contain the field's value

Script files to include: (semicolon-delimited)

Error message: ${FLDAlphaNumeric} is too long. Enter only up to 5 characters.

Validation option: ☐ Execute script code at page startup

Enabling the Data Entry Forms

To enable the data entry forms, you will need to edit the `web.config` file, which you can locate in the ChancerySMS folder. For example, it may be located in c:\inetpub\wwwroot\ChancerySMS. Insert the line as shown below:

```
      <appSettings>
            <add key="TaskSchedulerController.TaskSchedulerMachineName"
value=""/>
            <add key="EligibilityControlRecordsReturnedLimit" value="700"
/>
            <add key="AllowClientValidation" value="true"/>  ← Insert this
line between the appSettings tag
      </appSettings>
```

This solution is temporary. When the data entry forms are decided to be always available, the line above will no longer be applicable.

Specifying Client Validation Rule

Users define client validation using the Customization Builder feature. When adding new fields or editing existing fields, users can specify whether or not to use client validation by turning on the check box appearing to its left. If client validation is enabled, the user then selects which type of client validation they want to use.

Users can create a JavaScript file that contains a library of functions that can be used for client validation, or they can enter the JavaScript code block directly into the *Script code* field.

Specifying JavaScript function client validation

Below describes that data entry form fields for JavaScript function client validation and shows examples on how to enter information into each of the fields:

Field name	Description	Example
Function name	Identifies the JavaScript function to execute when validating.	`MyValidator`
Script file location	The location of the JavaScript file that contains the function.	`Scripts/Custom/Custom.js`
Script files to include	Additional JavaScript files to include that are needed by the function.	`Scripts/Custom/Generic.js;` `Scripts/Custom/Strings.js`
Error message	The error message to display to the user when the function returns false.	`You need to select 2 or more items.`
Validation option	When checked, the function is executed during page load.	`N/A`

Specifying JavaScript code block client validation

Below describes that data entry form fields for JavaScript code block client validation and illustrates examples on how to enter information into each of the fields:

Field name	Description	Example
Script code	Essentially, the body of a function. It returns true if validation is successful; otherwise, return false.	`return (value.length < 2);`
Script files to include	Additional JavaScript files to include that are needed by the script code.	`Scripts/Custom/Generic.js;` `Scripts/Custom/Strings.js`
Error message	The error message to display to the user when the function returns false.	`You need to select 2 or more items.`
Validation option	When checked, the function is executed during page load.	`N/A`

Defining the Function Name and Specifying its Location

If you want to use a JavaScript function as the validator function, you need to follow these rules:

Rule 1: The function MUST declare one parameter. This parameter will contain the value of the field the client validation is attached to.

Rule 2: The function returns a Boolean value, which is either *true* or *false*. If no value is returned, the function is assumed to return *true*.

Rule 3: The location of the file containing the function name must be readable (i.e., read-permission set) by SMS.

Example: `MyJSFile.js`

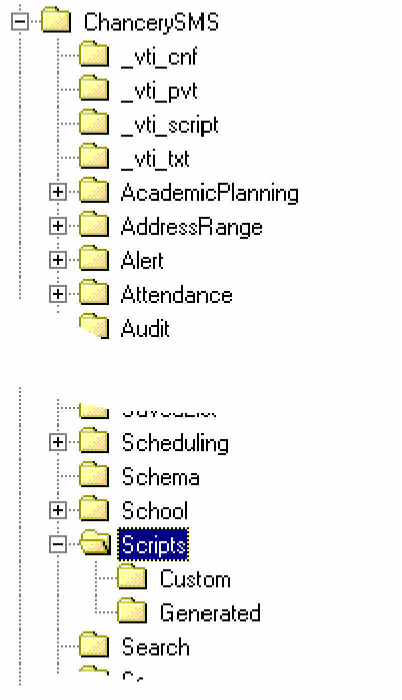
```
function MyValidator1(value)
{
    var num = parseInt(value);

    return (5 < num && num < 10);
}

function MyValidator2(value)
{
    if (value == “${SomeField}”)
    {
        if (value == “NULL”)
            return false;
    }

    return true;
}
```

Recommendation 1: Name the function parameter *value*.

Recommendation 2: Place your custom JavaScript files into the *Scripts/Custom* folder under the *ChancerySMS* root folder.

Knowing that all your custom scripts are located in one location will help ease re-deployment of Chancery SMS into another computer when necessary.

Below shows the sample directory structure with the Custom folder:



If Recommendation 2 is used, the location of `MyJSFile.js`, for example, can be specified either its absolute or relative path as in the following:

- `C:\Inetpub\wwwroot\ChancerySMS\Scripts\Custom\MyJSFile.js`
- `Scripts\Custom\MyJSFile.js`

Defining the Script code

When defining the script code, you simply code the body of a function. This code will be placed inside a function block behind the scene using a generated function name. The rules to follow are:

Rule 1: The word *value* is a keyword and will contain the value associated to the control (see Data Type of *value* in the following section).

Rule 2: The script code returns a Boolean value, which is either *true* or *false*. If no value is returned, the function is assumed to return *true*.

Example:

```
if (value == "${SomeField}")
{
     if (value == "NULL")
          return false;
}
return true;
```

"Value" Data Type

To determine how the handle the "value" of *value*, the table below describes the data type of *value*. Examples are provided to illustrate its usage.

Control type	Value	Example
Text Box	String	`return (value == "Y");`

Numeric Box	String	`function ValidateNumber(value)`
		`{`
		`    var num = parseInt(value);`
		`    return (1 <= num && num <= 10);`
		`}`

Date Box	String	`var d = new Date(value);`
		`return (d.getFullYear() < 2000);`

Drop-down List	String	`return value == "5000"`
		or
		`return (GetCode(value) == "PR");`
		or
		`return (GetDisplayText(value) == "Present");`

Control type	Value	Example

Check Box	Boolean	function MyValidator(**value**)
		{
		     if (**value**)
		     {
		         :
		         :
		     }
		     return true;
		}

List Mover	Array of strings	function ValidateList(**value**)
		{
		     // code cannot be “PR”
		     for (var i = 0; i < **value**.length; i++)
		     {
		          if (GetCode(**value**[i]) == “PR”)
		              return false;
		     }
		
		     return true;
		}

Memo	String	return (**value**.length > 500);

Data Representation

Any customization field that belongs to a page can be accessed within another field’s client validation definition. The fields can be represented by the following:

Notation	Description	Example

${Database field name}	Represents the value of the data entry form element associated to this field.	if (**${FirstName}** == “John”)
		{
		    :
		    :
		}

$ID{Database field name}	Represents the generated control ID of the data entry form element. Use this if you need to reference a control on the page that you want to manipulate.	var ctrl =
		document.getElementById(**$ID{FirstName}**);
		ctrl.style.color = “red”;

Possible Extensions

The following notation can be used to represent other data representation:

Notation	Description	Example

`@VAL{Schema, PropertyName, EntityID}`	Represents the static resolution of the database field value. Use this to render the value of a database field specified with the schema, the property name of the field and the entity ID of interest.	`@VAL{City, Description, 2340}`
		`@VAL{City, Code, 2340}`

`@RVAL{Schema, PropertyName, EntityID}`	Represents the dynamic resolution of the database field value using an RPC mechanism to retrieve its value. Use this to retrieve the value of a database field specified by the schema and property name. Note that this may change as this usage needs to be further designed as the process of data retrieval is asynchronous.	`@RVAL{City, Description, 100}`
		- returns the city description
		
		`@RVAL{SchoolStudent,OwnerObject.Name, 5230}`
		- returns the school name the student belongs to.

Available Special Functions

The following intrinsic functions are readily available for you to use in your client validation code. You do not need to include any special JavaScript files in order to use them.

Function	Purpose	Controls Supported	Sample Usage

GetCode(value)	Retrieves the "code" associated to *value.* The *value* is in this case is the internal ID of the record. For example, the code for Florida city is "FL".	Drop-down list
		List Mover
			`var code = GetCode(value);`
			
			`var code = GetCode(${SomeField});`

GetDisplayText(value)	Retrieves the "description" associated to *value*. The value in this case corresponds to the internal ID of the information. For example, the description or display text for Florida is "Florida".	Drop-down list
		List Mover
			`var code = GetDisplayText (value);`
			
			`var code = GetDisplayText (${SomeField});`

EnableGridAddMenu(gridID, bEnable)
EnableGridEditMenu(gridID, bEnable)
EnableGridDeleteMenu(gridID, bEnable)
EnableGridSelectAllMenu(gridID, bEnable)
EnableGridDeselectAllMenu(gridID, bEnable)
EnableGridViewMenu(gridID, bEnable)
	These functions allow you to manipulate the menu state of all the menu items associated to the Grid	Grid	`var allowEditing = ${AllowEditingFld};`
			`var gridID = $ID{MyGrid};`
			
			`EnableGridAddMenu(gridID, allowEditing);`
			`EnableGridEditMenu(gridID, allowEditing);`
			`EnableGridDeleteMenu(grid,`

Function	Purpose	Controls Supported	Sample Usage
EnableGridChooseColumnsMenu(gridID, bEnable)	object. These menu items are generated along with this object.		`allowEditing);`

Specifying the Script Files to Include

The JavaScript files to include are simply files that either the function name or script code requires because it is calling other JavaScript functions residing in those files. The file names can be specified in either it absolute or relative path form.

Rule 1: File names must be separated using the semi-colon (;).

Rule 2: The file must be readable by SMS (i.e., SMS has read-permission set).

Example:

```
Scripts/Custom/Generic.js;
c:/inetpub/wwwroot/ChancerySMS/Scripts/Custom/Custom.js
```

Defining the Error Message

The error message is the text that will be displayed when the validation function or script code returns *false*. The text itself is not sufficient to provide a meaningful message. In order to address this, the error message may include specific code in it. For example, a List Mover object whose database field name as it appears in the Customization Builder page is “SelectedGradeLevels” and you may want to display the error message “You selected 10 items. Select between 2 and 6 only.” In order to do this, you will write your error message this way:

```
You selected ${SelectedGradeLevels}.length items.  Select between 2 and 6
only.
```

When the error message is parsed, the `${SelectedGradeLevels}.length` will treated as JavaScript code. Another example is to display the “description” or “display text” of a city list. For example, you may want to display the error message “You selected Burnaby, which is not allowed”. The error message you need to enter is this:

```
You selected GetDisplayText(${CityList}), which is not allowed.
```

The `GetDisplayText(${CityList})` is treated as code and it will be replaced with the actual value upon the display of the error message.

Allowed Script Code inside an Error Message

The following are supported inside an error message:

- `${token}`
- `$ID{token}`
- `GetCode(${token});`
- `GetDisplayText(${token});`

Compilation Process

The compilation occurs once the user clicks on the "Apply Pending Changes" button. When a JavaScript file or JavaScript code block is processed, a .js file is generated and saved into the *Generated* folder. This file will then be referenced by the web pages generated by the Customization Builder. Since both the JavaScript file and JavaScript code block may contain references to other database fields within the same Web page, these fields need to be resolved first by parsing them and replacing them with the appropriate executable code. For example, if a .js file defines this code:

Before:

```
function SomeFunctionName(value)
{
   return value.length > (${FirstName} + ${LastName}).length;
}
```

Then, a new .js file will be generated and placed into the *Generated* folder but with the function above replaced with this:

After:

```
function SomeFunctionName(value, ctrlID, __Values, __CtrlIDs, __DBValues)
{
   return value.length > (__Values[0] + __Values[1]).length;
}
```

If a JavaScript code block is provided, only the body of the function needs to be defined.

Before:

```
return value.length > (${FirstName} + ${LastName}).length;
```

The *value* is treated as a reserved word and it will contain the raw value of the data entry field in which the code block is associated to. In order to call JavaScript functions in another JavaScript file, the location of this file needs to be specified in the *JavaScript files to include* field. The above code block will then be processed and placed into a function in a .js file as follows:

After:

```
function Validate{MetaDataColumnView ID}(value, ctrlID, __Values, __CtrlIDs,
__DBValues)
{
   return value.length > (__Values[0] + __Values[1]).length;
}
```

Where `{MetaDataColumnView ID}` is the entity ID of the IMetaDataColumnView object. This is to ensure uniqueness of the function name and establish a naming pattern that can be easily formed and connected back to the database field to be validated.

The generated JavaScript code is what will be used by cTools. The parameters and descriptions are listed below:

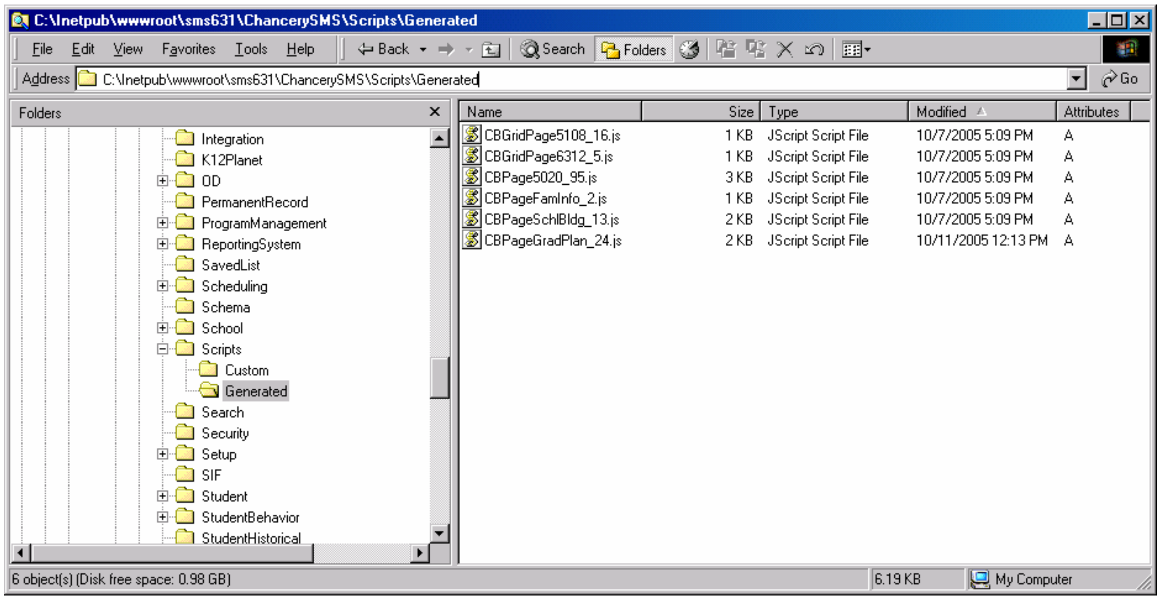
Parameter	Description
value	For script block, this is a reserved word. For function names, the value represents the name used as the parameter. This will contain the value of the control associated to the client validation (see Data Type of *value* table).
ctrlID	Contains the rendered control ID associated to the client validation.
__Values	Contains an array of values for items that uses this notation: ${token}
__CtrlIDs	Contains an array of control IDs for items that uses this notation $ID{token}
__DBValues	Not presently used

Naming convention for the .js file

Each defined Web page will have a corresponding generated .js file, if required. This .js file will contain all the script blocks for the page and all the custom .js file that are used within the page. The name format is as follows:

```
{MetaDataSchemaViewName}_v{PageScriptVersion}.js
```

Here are sample generated files:

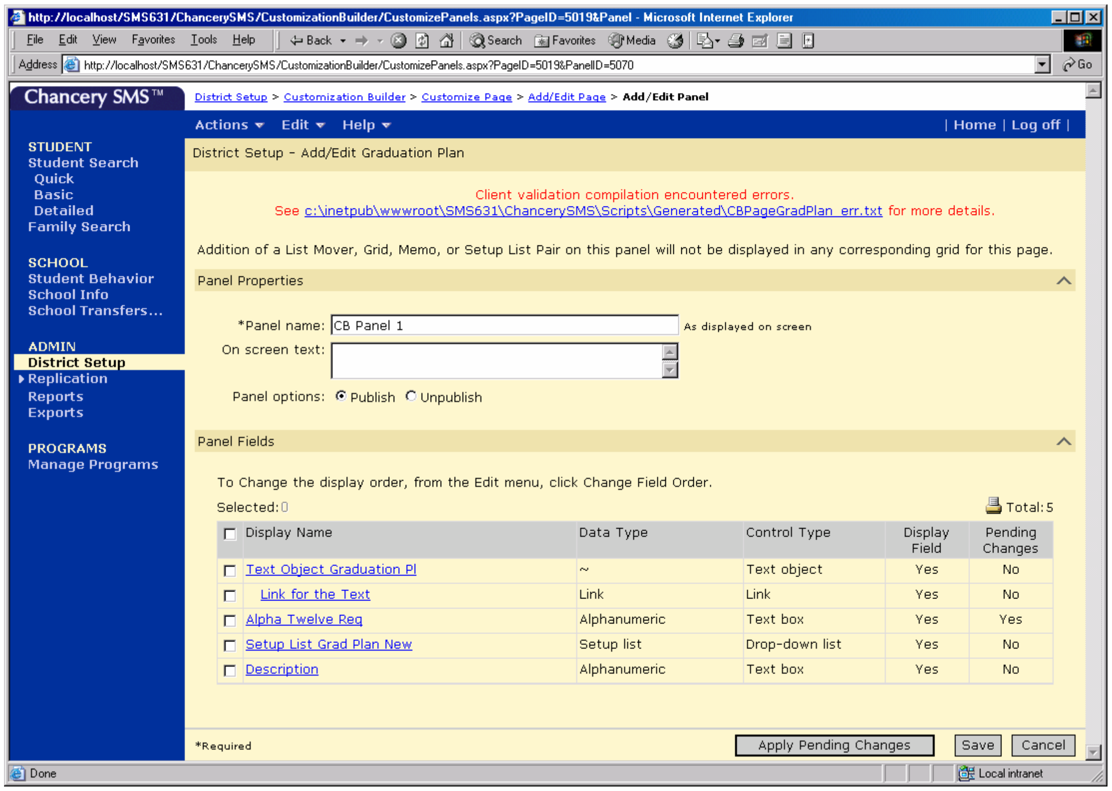


Name	Size	Type	Modified	Attributes
CBGridPage5108_16.js	1 KB	JScript Script File	10/7/2005 5:09 PM	A
CBGridPage6312_5.js	1 KB	JScript Script File	10/7/2005 5:09 PM	A
CBPage5020_95.js	3 KB	JScript Script File	10/7/2005 5:09 PM	A
CBPageFamInfo_2.js	1 KB	JScript Script File	10/7/2005 5:09 PM	A
CBPageSchlBldg_13.js	2 KB	JScript Script File	10/7/2005 5:09 PM	A
CBPageGradPlan_24.js	2 KB	JScript Script File	10/11/2005 12:13 PM	A

6 object(s) (Disk free space: 0.98 GB) 6.19 KB My Computer

Compile Errors

When compilation error occurs, the error messages are logged into a file using this name format: `{MetaDataSchemaViewName}_err.txt` and this file will be located in the Scripts/Generated folder, and then it is presented as a link in the top of the Customization Builder page as shown below:


http://localhost/SMS631/ChancerySMS/CustomizationBuilder/CustomizePanels.aspx?PageID=5019&Panel - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/SMS631/ChancerySMS/CustomizationBuilder/CustomizePanels.aspx?PageID=5019&PanelID=5070
Go
Chancery SMS™
District Setup > Customization Builder > Customize Page > Add/Edit Page > Add/Edit Panel
Actions Edit Help
| Home | Log off |
STUDENT
Student Search
Quick
Basic
Detailed
Family Search
SCHOOL
Student Behavior
School Info
School Transfers...
ADMIN
District Setup
Replication
Reports
Exports
PROGRAMS
Manage Programs
Done
Local intranet

District Setup - Add/Edit Graduation Plan

Client validation compilation encountered errors.
See c:\inetpub\wwwroot\SMS631\ChancerySMS\Scripts\Generated\CBPageGradPlan_err.txt for more details.

Addition of a List Mover, Grid, Memo, or Setup List Pair on this panel will not be displayed in any corresponding grid for this page.

Panel Properties

*Panel name: CB Panel 1 As displayed on screen

On screen text:

Panel options: [✓ Publish] Unpublish

Panel Fields

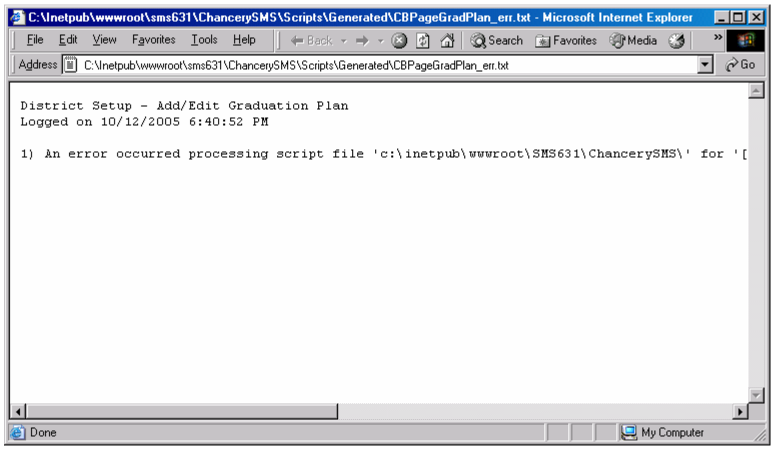
To Change the display order, from the Edit menu, click Change Field Order.

Selected: 0 Total: 5

☐	Display Name	Data Type	Control Type	Display Field	Pending Changes
☐	Text Object Graduation Pl	~	Text object	Yes	No
☐	Link for the Text	Link	Link	Yes	No
☐	Alpha Twelve Req	Alphanumeric	Text box	Yes	Yes
☐	Setup List Grad Plan New	Setup list	Drop-down list	Yes	No
☐	Description	Alphanumeric	Text box	Yes	No

*Required Apply Pending Changes Save Cancel

When you click the link, the content of the file will be displayed:

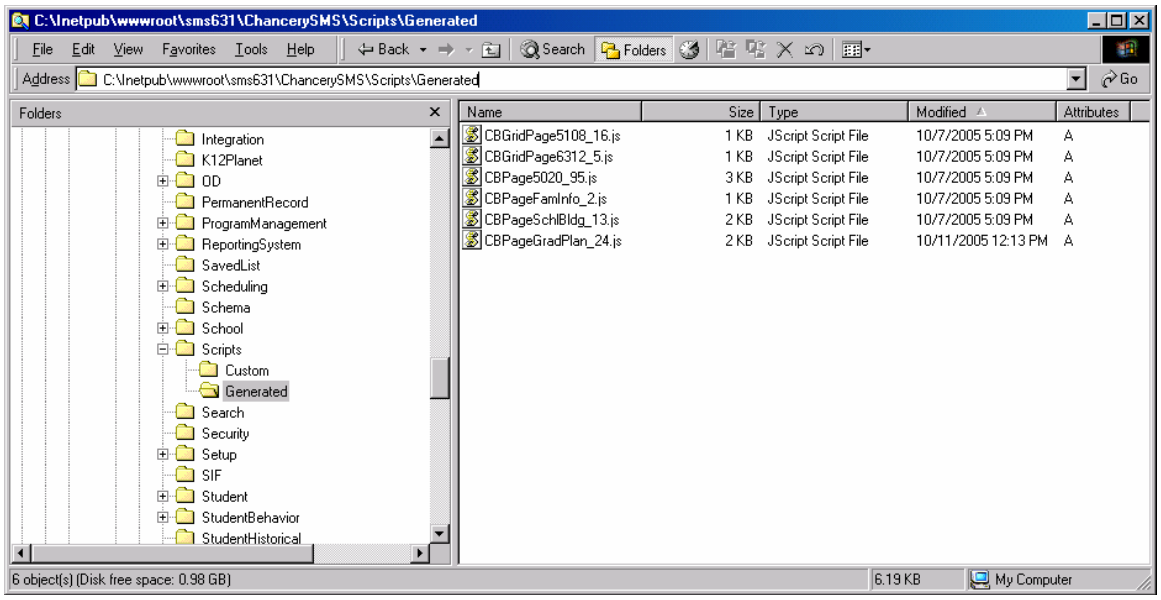

C:\Inetpub\wwwroot\sms631\ChancerySMS\Scripts\Generated\CBPageGradPlan_err.txt - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address C:\Inetpub\wwwroot\sms631\ChancerySMS\Scripts\Generated\CBPageGradPlan_err.txt
Go
Done
My Computer

```
District Setup - Add/Edit Graduation Plan
Logged on 10/12/2005 6:40:52 PM

1) An error occurred processing script file 'c:\inetpub\wwwroot\SMS631\ChancerySMS\' for '[
```

Once you have resolved the problems, the message and the link to the error log file will disappear. If you quit SMS and then restart it, the link message will always display as long as the error log file exists for the page.

Run-time Process

Once the compilation is complete, you will need to restart Chancery SMS. Initially during compilation, the compiled code are first stored into the CSL_SMS_COMPILED_CODE database table. Once restarted, the compiled code will be dumped into JavaScript files in the location {ChancerySMS}\Scripts\Generated folder. Below displays several generated JavaScript files whose content originated from the database table.



Name	Size	Type	Modified	Attributes
CBGridPage5108_16.js	1 KB	JScript Script File	10/7/2005 5:09 PM	A
CBGridPage6312_5.js	1 KB	JScript Script File	10/7/2005 5:09 PM	A
CBPage5020_95.js	3 KB	JScript Script File	10/7/2005 5:09 PM	A
CBPageFamInfo_2.js	1 KB	JScript Script File	10/7/2005 5:09 PM	A
CBPageSchlBldg_13.js	2 KB	JScript Script File	10/7/2005 5:09 PM	A
CBPageGradPlan_24.js	2 KB	JScript Script File	10/11/2005 12:13 PM	A

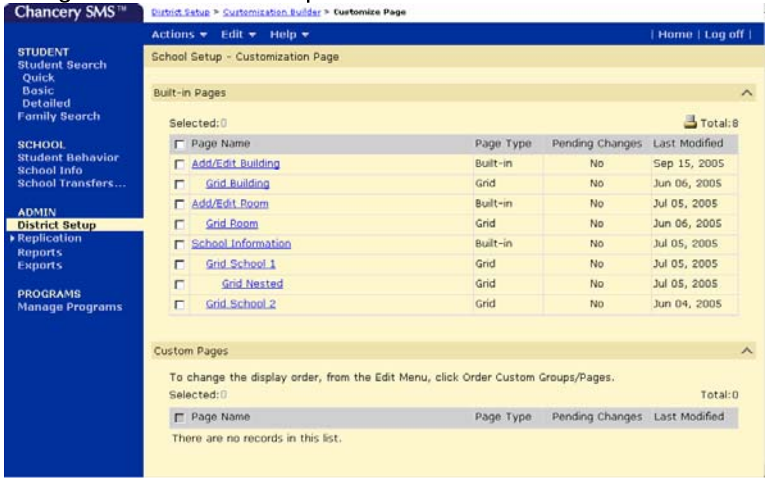
6 object(s) (Disk free space: 0.98 GB) 6.19 KB My Computer

When the web page is rendered, the so-called 'UIFactory' creates the UI elements of the web page. Validator controls are also created as necessary. To support client validation, a new validator called *CslInputClientValidator* will be created. This type of validator will be responsible for resolving the referenced database fields, generating and hooking the call to the JavaScript functions that will perform the actual validation, and setting up the error message to display when the validation fails. Since the validator control will be based on .NET's *BaseValidator*, no special coding is necessary to attach this validation to the main JavaScript validator function that is part of the .NET framework.

Writing JavaScript code for Client Validation

Customization Builder Sample Setup

Navigate to: District Setup > Customization Builder > School Setup



Chancery SMS™

District Setup > Customization Builder > Customize Page

Actions Edit Help | Home | Log off |

STUDENT
Student Search
Quick
Basic
Detailed
Family Search

SCHOOL
Student Behavior
School Info
School Transfers...

ADMIN
District Setup
Replication
Reports
Exports

PROGRAMS
Manage Programs

School Setup - Customization Page

Built-in Pages

Selected:0 Total:8

Page Name	Page Type	Pending Changes	Last Modified
Add/Edit Building	Built-in	No	Sep 15, 2005
Grid Building	Grid	No	Jun 06, 2005
Add/Edit Room	Built-in	No	Jul 05, 2005
Grid Room	Grid	No	Jun 06, 2005
School Information	Built-in	No	Jul 05, 2005
Grid School 1	Grid	No	Jul 05, 2005
Grid Nested	Grid	No	Jul 05, 2005
Grid School 2	Grid	No	Jun 04, 2005

Custom Pages

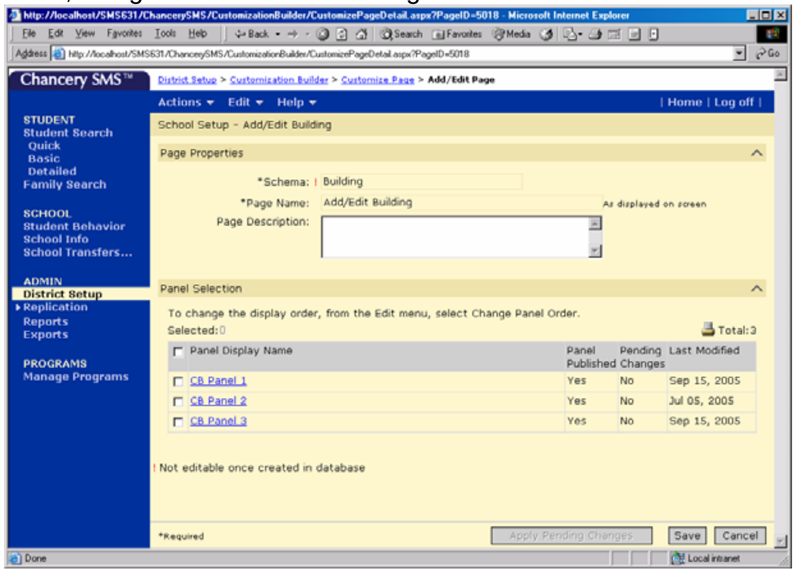
To change the display order, from the Edit Menu, click Order Custom Groups/Pages.

Selected:0 Total:0

Page Name	Page Type	Pending Changes	Last Modified

There are no records in this list.

Then, navigate to Add/Edit Building:



http://localhost/SMS631/ChanceySMS/CustomizationBuilder/CustomizePageDetail.aspx?PageID=5018 - Microsoft Internet Explorer

Address http://localhost/SMS631/ChanceySMS/CustomizationBuilder/CustomizePageDetail.aspx?PageID=5018

Chancery SMS™

District Setup > Customization Builder > Customize Page > Add/Edit Page

Actions Edit Help | Home | Log off |

School Setup - Add/Edit Building

Page Properties

*Schema: ! Building

*Page Name: Add/Edit Building As displayed on screen

Page Description:

Panel Selection

To change the display order, from the Edit menu, select Change Panel Order.

Selected:0 Total:3

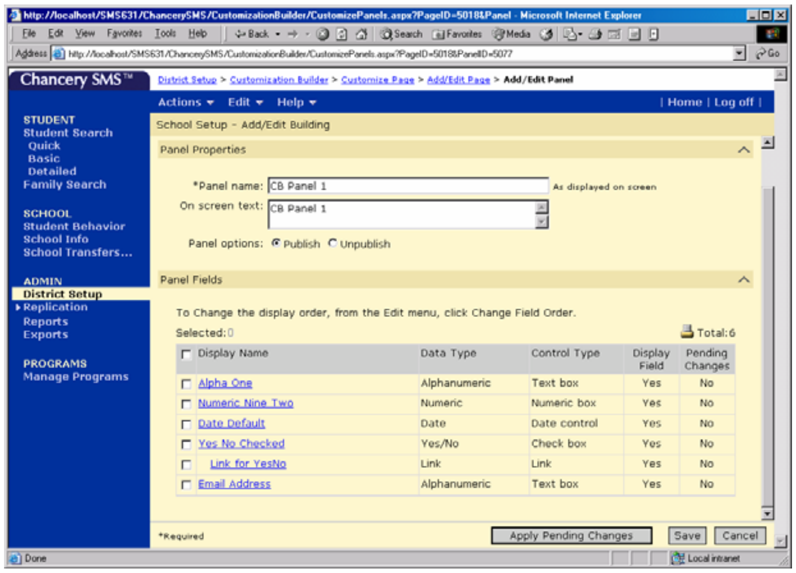
Panel Display Name	Panel Published	Pending Changes	Last Modified
CB Panel 1	Yes	No	Sep 15, 2005
CB Panel 2	Yes	No	Jul 05, 2005
CB Panel 3	Yes	No	Sep 15, 2005

! Not editable once created in database

*Required

Apply Pending Changes Save Cancel

Navigate to CB Panel 1


http://localhost/SMS631/ChanceryySMS/CustomizationBuilder/CustomizePanels.aspx?PageID=5018&Panel - Microsoft Internet Explorer
Address http://localhost/SMS631/ChanceryySMS/CustomizationBuilder/CustomizePanels.aspx?PageID=5018&PanelID=5077
Chancery SMS™
District Setup > Customization Builder > Customize Page > Add/Edit Page > Add/Edit Panel
Actions Edit Help
Home Log off
STUDENT
Student Search
Quick
Basic
Detailed
Family Search
SCHOOL
Student Behavior
School Info
School Transfers...
ADMIN
District Setup
Replication
Reports
Exports
PROGRAMS
Manage Programs
Apply Pending Changes
Save
Cancel
Done
Local intranet

School Setup - Add/Edit Building

Panel Properties

*Panel name: CB Panel 1 As displayed on screen

On screen text: CB Panel 1

Panel options: [✓ Publish] Unpublish

Panel Fields

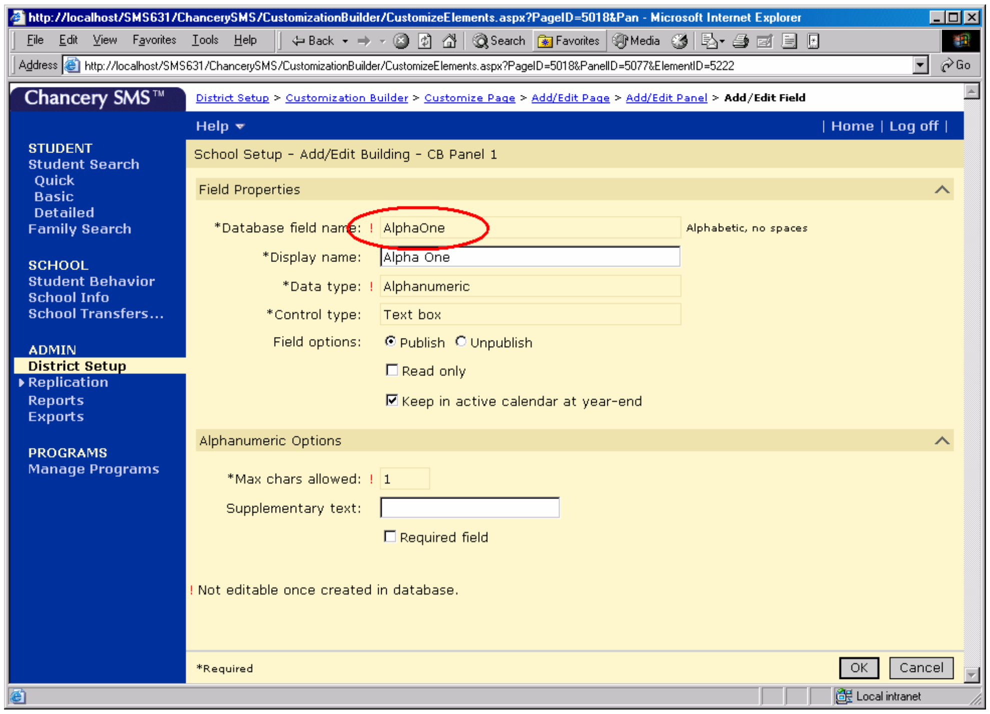
To Change the display order, from the Edit menu, click Change Field Order.

Selected:0 Total:6

Display Name	Data Type	Control Type	Display Field	Pending Changes
Alpha One	Alphanumeric	Text box	Yes	No
Numeric Nine Two	Numeric	Numeric box	Yes	No
Date Default	Date	Date control	Yes	No
Yes No Checked	Yes/No	Check box	Yes	No
Link for YesNo	Link	Link	Yes	No
Email Address	Alphanumeric	Text box	Yes	No

*Required

Above shows the data fields for CB Panel 1. The validation will be hooked to *Alpha One* and *Numeric Nine Two* fields. This is done in the Setting up section. Note that the id 5222 happens to correspond to *Alpha One* and 5223 to *Numeric Nine Two* fields in my database.


http://localhost/SMS631/ChanceryySMS/CustomizationBuilder/CustomizeElements.aspx?PageID=5018&Pan - Microsoft Internet Explorer
Address http://localhost/SMS631/ChanceryySMS/CustomizationBuilder/CustomizeElements.aspx?PageID=5018&PanelID=5077&ElementID=5222
Chancery SMS™
District Setup > Customization Builder > Customize Page > Add/Edit Page > Add/Edit Panel > Add/Edit Field
Help
Home Log off
STUDENT
Student Search
Quick
Basic
Detailed
Family Search
SCHOOL
Student Behavior
School Info
School Transfers...
ADMIN
District Setup
Replication
Reports
Exports
PROGRAMS
Manage Programs
OK
Cancel
Local intranet

School Setup - Add/Edit Building - CB Panel 1

Field Properties

*Database field name: ! AlphaOne Alphabetic, no spaces

*Display name: Alpha One

*Data type: ! Alphanumeric

*Control type: Text box

Field options: [✓ Publish] Unpublish

☐ Read only

☑ Keep in active calendar at year-end

Alphanumeric Options

*Max chars allowed: ! 1

Supplementary text:

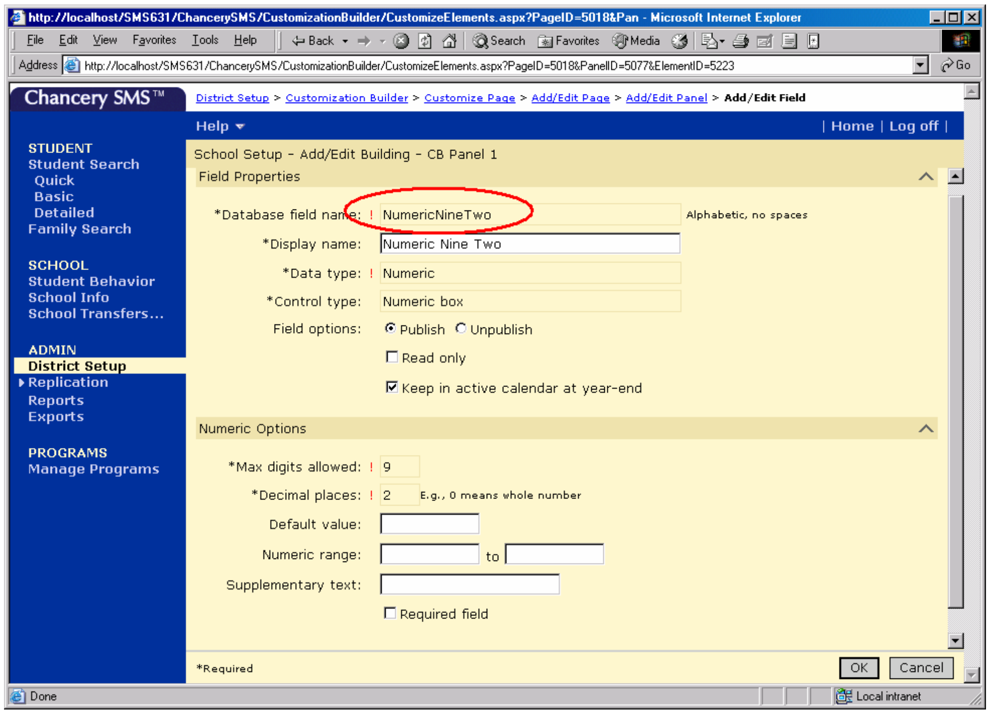
☐ Required field

! Not editable once created in database.

*Required

The above shows the page to edit the *Alpha One* data field. In order to reference this data field in your JavaScript, you need to use the Database field name, for example, ${AlphaOne}. This example will return you the value entered by the user. To get the reference to the control itself, use $ID{AlphaOne}. Take a look at the Test.js in the next section to get an idea how to manipulate controls in a Web page.

The following diagram illustrates the *Numeric Nine Two* data field:


http://localhost/SMS631/ChancerySMS/CustomizationBuilder/CustomizeElements.aspx?PageID=5018&Pan - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/SMS631/ChancerySMS/CustomizationBuilder/CustomizeElements.aspx?PageID=5018&PanelID=5077&ElementID=5223
Go
Chancery SMS™
District Setup > Customization Builder > Customize Page > Add/Edit Page > Add/Edit Panel > Add/Edit Field
Help
| Home | Log off |
STUDENT
Student Search
Quick
Basic
Detailed
Family Search
SCHOOL
Student Behavior
School Info
School Transfers...
ADMIN
District Setup
Replication
Reports
Exports
PROGRAMS
Manage Programs
OK
Cancel
Done
Local intranet

School Setup - Add/Edit Building - CB Panel 1

Field Properties

*Database field name: ! NumericNineTwo — Alphabetic, no spaces

*Display name: Numeric Nine Two

*Data type: ! Numeric

*Control type: Numeric box

Field options: [✓ Publish] Unpublish

☐ Read only

☑ Keep in active calendar at year-end

Numeric Options

*Max digits allowed: ! 9

*Decimal places: ! 2 — E.g., 0 means whole number

Default value:

Numeric range: to

Supplementary text:

☐ Required field

*Required

Setting up

Sample Script – the manual way

```
-- Example 1: Client validation type is by function call.

UPDATE
    CSL_SMS_WORKING_ELEMENT
SET
    CLIENT_VALIDATION_TYPE = 2,
    SCRIPT_FILE_LOCATION =
'c:\inetpub\wwwroot\sms631\chancerysms\Scripts\Custom\Test.js',  -- Must be full
path
    SCRIPT_FUNCTION_NAME = 'MyTestValidator',
    VALIDATION_ERROR_MESSAGE = 'The value ${AlphaOne} must not be X.'
WHERE
    ID_SMS_WORKING_ELEMENT = 5222

-- Example 2: Client validation type is by script block

UPDATE
    CSL_SMS_WORKING_ELEMENT
SET
    CLIENT_VALIDATION_TYPE = 1,
    SCRIPT_BLOCK = 'var val = ${NumericNineTwo};  if (val.length > 0 &&
parseInt(val) == 3) { return false; } else { return true; }',
    VALIDATION_ERROR_MESSAGE = 'The value of AlphaOne is ''${AlphaOne}'' and
NumericNineTwo is ''${NumericNineTwo}''.'
WHERE
    ID_SMS_WORKING_ELEMENT = 5223
```

Notice that in Example 2, the script code references another data field belonging to the same CB page. *You can reference any data fields in CB as long as they are all part of the CB page.*

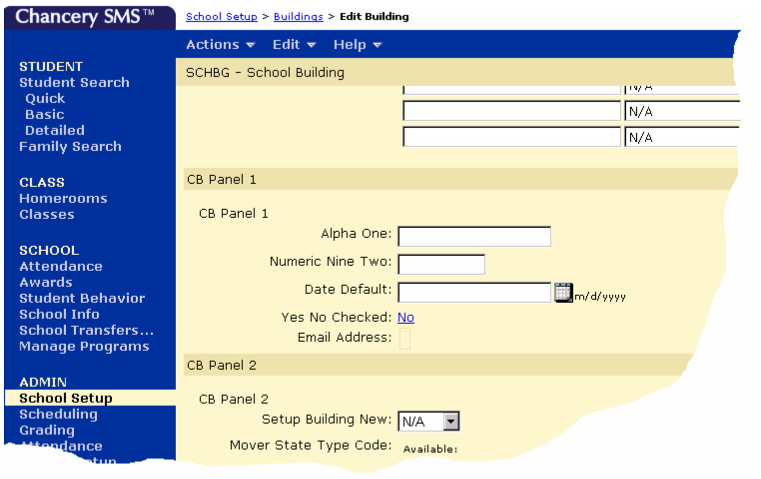
The sample Test.js file

In the sample code above, this JavaScript file is located in `c:\inetpub\wwwroot\sms631\chancerysms\Scripts\Custom\Test.js`.

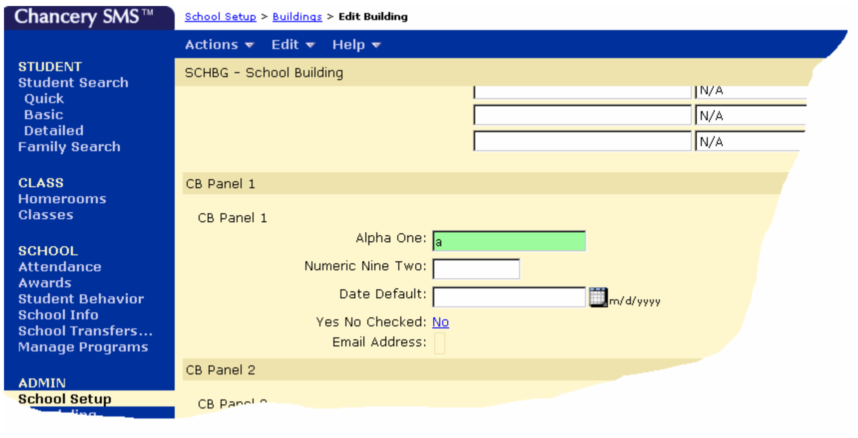
```
function MyTestValidator(value)
{
   var ctrlID = $ID{AlphaOne};

   var objCtrl = document.getElementById(ctrlID);

   if (${AlphaOne} != "X")
   {
        objCtrl.style.backgroundColor = "palegreen";
       return true;
   }
   else
   {
        objCtrl.style.backgroundColor = "tomato";
       return false;
   }
}
```

Client Validation in Action

Client Validation by Function Call



When users enter a valid value as validated according to the validator function provided, you will get this result:



SCHBG - School Building

N/A
N/A
N/A

CB Panel 1

CB Panel 1

Alpha One: a

Numeric Nine Two:

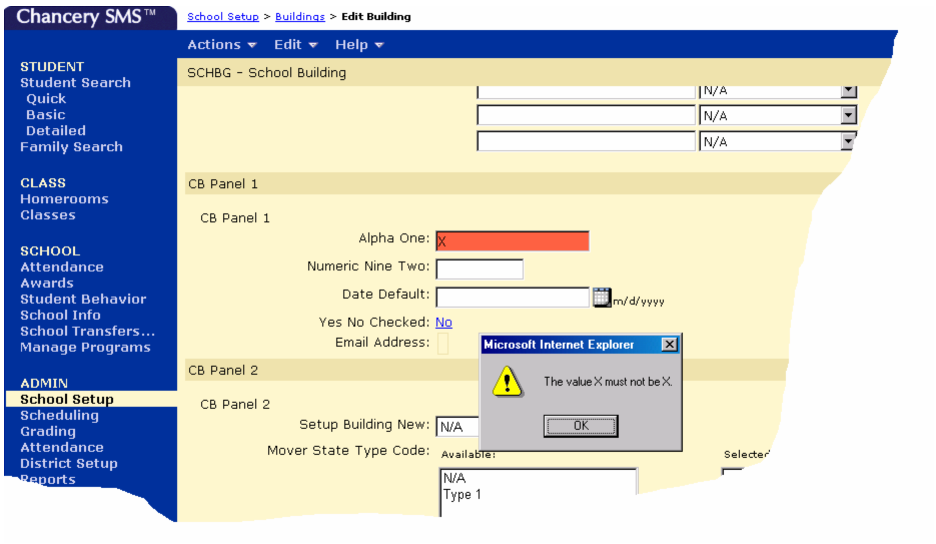
Date Default: m/d/yyyy

Yes No Checked: No

Email Address:

CB Panel 2

When users enter “X” in the Alpha One field, the validation error message displays as illustrated:



SCHBG - School Building

N/A
N/A
N/A

CB Panel 1

CB Panel 1

Alpha One: X

Numeric Nine Two:

Date Default: m/d/yyyy

Yes No Checked: No

Email Address:

Microsoft Internet Explorer

The value X must not be X.

OK

CB Panel 2

CB Panel 2

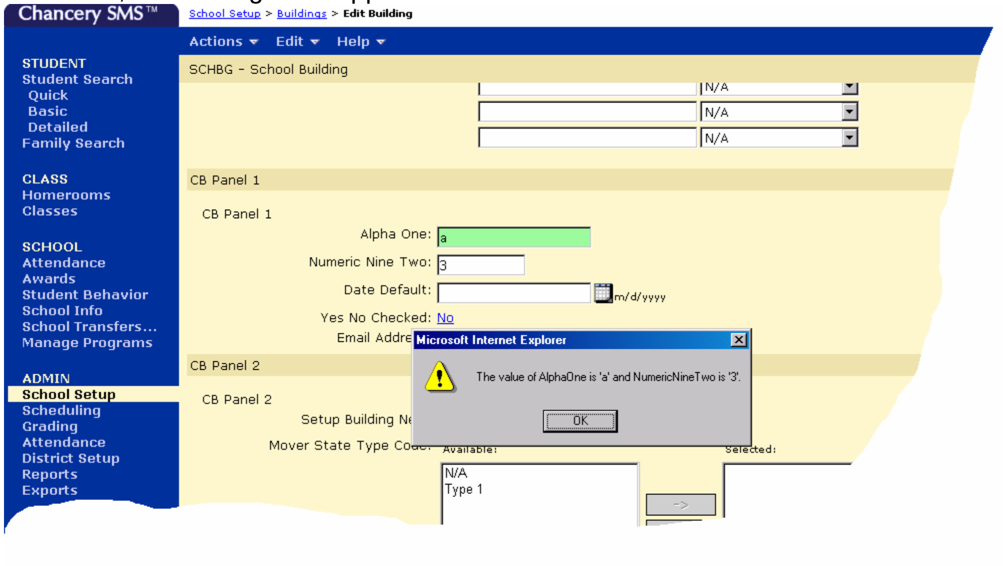
Setup Building New: N/A

Mover State Type Code: Available: Selected

N/A
Type 1

Client Validation by Script Block

If users type “3” in the Numeric Nine Two field, the validation error message appears below; otherwise, no message box appears.



Alpha One: a
Numeric Nine Two: 3
Date Default: m/d/yyyy
Yes No Checked: No

The value of AlphaOne is 'a' and NumericNineTwo is '3'.

Calling a Back-End Stored Procedure using JavaScript

As an enhancement to the JavaScript validation feature stated in this document, you now have the ability to call a back-end stored procedure inside the JavaScript block. The main purpose is to allow the execution of more complex business rules available at the server side (in the form of stored procedures), and then update the UI items based on the values returned by those business rules. For instance, the programmer can decide to disable, update or even hide a particular UI Item depending on what value is returned from the stored procedure. This feature is particularly useful for clients who have complex customization requirements to their UI pages which implementation are not possible through the cTools feature itself.

Here is how it works:

Call the JavaScript function named ‘ExecuteStorProc’ (see signature below) located in the generic.js file.

function ExecuteStorProc((string)storProcName, (string)handler, (string) handlerParams)

The ExecuteStorProc function takes the following 3 parameters and doesn’t return any value.

storProcName – the name of the stored procedure to call. If the stored procedure takes parameters, their names and values need to be appended to the stored procedure name itself and delimited by the ‘~’ character. Example: storprocName~para1Name~para1Value~para2Name~para2Value…
handler – the name of the JavaScript function that receives the returned value from the stored procedure. This is needed as the call to the stored procedure is done asynchronously and requires a handler (in the form of a JavaScript function) to capture the results from the stored procedure on completion.
handlerParams – this parameter is optional and is used if additional values (other than the ones returned from the stored procedure) are required to be sent to the handler. A good example would be the id of the UI control to be updated with the value returned by the stored procedure.

Examples:

```
With no parameters:
ExecuteStorProc("StorProcName", "handler", "handlerParams")

With one parameter:
ExecuteStorProc("StorProcName~ParameterName~ParameterValue", "handler",
"handlerParams")

With multiple parameters:
ExecuteStorProc("StorProcName~ParameterName1~ParameterValue1~ParameterName
2~ParameterValue2...", "handler", "handlerParams")
```

As mentioned above, when the stored procedure completes its execution it will call the specified handler function. The handler function needs to have the following signature:

function [handlerFunctionName]((object)returnedValue, (object)handlerParams)

returnedValue – The value returned from the stored procedure. The returned value will be in the form of an array if the stored procedure is to return a collection of data.
handlerParams – The handlerParams value passed to the ExecuteStorProc function.