```
var lineBreaker = "\n";
var rules = [validateTable, validateField];
var beforeSendValidate = function(CURRENT_STATE, NEXT_STATE){
     var errorMsg = "";
     if(CURRENT_STATE == NEXT_STATE){
          return;
     }

     var fields = requiredFields.getFields();
     for(var i=0;i<fields.length; i++){
          if(fields[i].activities.indexOf(CURRENT_STATE) >=
0){
               console.log("fields[i] : "+fields[i]);
               var selector = (fields[i].name.indexOf("___")
> 0) ?
                              '[name^="'+fields[i].name+'"]'
                              : '[name="'+fields[i].name+'"]';
               $(selector).each(function(){
                    for(rule in rules){
                         console.log("rules[rule] :
"+rules[rule]);
                         var validation = rules[rule](this,
selector);
                         if(validation.status != "success"){
                              if(validation.status ===
"error"){
                                   errorMsg +=
validation.message;
                              }
                              break;
                         }
                    }
               });
          }
     }

     //VALIDAÇÃO CUSTOMIZADA DOS CAMPOS
     errorMsg += validateCustomField();

     if(errorMsg != ""){
          throw errorMsg;
     }
}

function validateCustomField(){
     var errorMsg = "";
     var customFields = new Fields();
```

```
     if(CURRENT_STATE == INICIO_0 || CURRENT_STATE ==
INICIO){

          if($("input[name='rdTipoInclusao']:checked").val()
== undefined){
               errorMsg += "Campo Tipo de inclusão é
obrigatório!"+ lineBreaker;
          }

/*        var indices = retornaIndices("tablePropostas");

          if(indices.length == 0){
               errorMsg += "Informe ao menos uma proposta
para a solicitação! "+ lineBreaker;
          }else{

               for (var i in indices){


     if($("input[name='progRegre___"+indices[i]+"']:checked")
.val() == undefined){

                         errorMsg += "Campo Regime de
Tributação é obrigatório! "+ lineBreaker;

                    }


     if(!validacaoJson($("#jsonFundos___"+indices[i]).val()))
{
                         errorMsg += "Informe ao menos um
fundo para a proposta na linha "+indices[i]+"! "+
lineBreaker;
                    }

               }

          }*/


     }

     var fields = customFields.getFields();
     for var i=0;i<fields.length; i++){
          if(fields[i].activities.indexOf(CURRENT_STATE) >=
0){
```

```
                console.log("fields : "+fields[i]);
                var selector = (fields[i].name.indexOf("___")
> 0) ?
                        '[name^="'+fields[i].name+'"]'
                        : '[name="'+fields[i].name+'"]';
                $(selector).each(function(){
                    for(rule in rules){
                        console.log("rules[rule] :
"+rules[rule]);
                        var validation = rules[rule](this,
selector);
                        if(validation.status != "success"){
                            if(validation.status ===
"error"){
                                errorMsg +=
validation.message;
                            }
                            break;
                        }
                    }
                });
            }
        }
        console.log("return error message custom: " + errorMsg);

        return errorMsg;
}

/**
 * Regra de negócio: Valida se pai x filho tem ao menos um
registros.
 * @param el: Elemento sendo avaliado.
 * @returns validation Validation.
 */
function validateTable(el){
        var validation = new Validation();
        if($(el).prop("tagName") === "TABLE" ){
            validation.status = "ignore";
            if($(el).find("tr").size() <= 2){
                validation.message = "Insira ao menos um
registro no pai x filho: "

        +getLabel($(el).attr("name"))+lineBreaker;
                validation.status = "error";
            }
        }
        return validation;
```

```
}

/**
 * Regra de negócio: Valida campos, sejam radio, check,
inputs mesmo que em pai x filho.
 * @param el: Elemento sendo avaliado.
 * @param selector: Seletor usado para pegar o elemento.
 * @returns validation Validation.
 */
function validateField(el, selector){
     var validation = new Validation();
     if((

      ["radio","checkbox"].indexOf($(el).attr("type")) >= 0
                && $(selector+':checked').size() <= 0
          )
          || (el.value == "" && el.tagName != "SPAN")){
          validation.message = "Campo
"+getLabel($(el).attr("name"))+" é obrigatório!"+lineBreaker;
          validation.status = "error";
     }
     return validation;
}



/**
 * Classe validação.
 * @attribute status: success, ignore ou error.
 * @attribute messagem: Messagem de erro.
 */
function Validation(){
     this.status = "success";
     this.message = "";
}

var requiredFields = new Fields();
/**
 * Classe campos.
 * @attribute fields: success, ignore ou error.
 * @method addField: Adiciona objetos compostos por String
name e Array activities.
 * @method getFields: Recupera campos.
 */
function Fields(){
     this.fields = [];
     this.addField = function(name, arrayActivities){
```

```
        this.fields.push({"name":name,"activities":arrayActiviti
es});
    }
    this.getFields = function(){
        return this.fields;
    }
}

/**
 * Inclui o * indicativo de campo obrigatório nos labels a
partir do nome do campo.
 * @param name: name do campo. Utiliza o name para ser
compativel com os campos do tipo radio.
 * @param isRequired: é obrigatório? booleano.
 * @returns void.
 */
function setRequired(name, isRequired){
    name = name.split("___")[0];
    (isRequired) ?
$("[for='"+name+"']").addClass('required')
        : $("[for='"+name+"']").removeClass('required');
}

/**
 * Retorna label baseado no name do campo, verificando
atributo "for" da label.
 * @param name: name do campo.
 * @returns label: texto do label.
 */
function getLabel(name) {
    name = name.split("___")[0];
    return $("[for='"+name+"']").html();
}

function getClosestByElement(element, selector){
    var max = 10;
    returnElement = undefined;
    while(returnElement == null && max >= 0){
        if($(element).is(selector)){
            returnElement = element;
        }else{
            element = element.parent();
        }
        max--;
    }
```

```
        return returnElement;
    }
```