LocDirect API specification

LocDirect API

This document describes the various API commands that can be used with the LocDirect API. The API commands can be sent as XML via the socket API and as XML or JSON via the HTTP API. The commands are described with “pseudo code” where optional fields are surrounded with [ ] and value placeholders are surrounded with < >.

The API can be accessed either via HTTP requests or by using a Socket API. You’ll find information about how the API can be accessed on our support pages. This document only describes the API commands that can be used.

Please note: Where used in this document, the term GUID refers to a string that is formatted as a 36 character long Globally Unique Identifier, for example 9D3E82CB-BF66-C412-94E3-D2A77937EEF5.

StringExport Command

With this command, you can fetch/export strings from LocDirect. This command is quick and works in a similar way as when using the CSV exporter in the LocDirect application client. It's possible to export all kind of string fields and you can choose to filter on custom fields. You can also select the response type which make it possible to stream-read the response.

XML request (Socket API or HTTP API)

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
   <TASK name="StringExport">
      <OBJECT name="String">
         <exportFields/>
      </OBJECT>
      <WHERE>
         <responseType/>
         [<projectName/>]
         [<projectId/>]
         [<folderPaths/>]
         [<identifiers/>]
         [<custom field name to filter on>]
         [<newLineCharacter/>]
      </WHERE>
   </TASK>
</EXECUTION>
```

JSON request (HTTP API)

```
{
  "secId": "<secId>",
  "command": "String.StringExport",

  "data": {
    (see OBJECT-fields description below)
  },

  "where": {
    (see WHERE-fields description below)
  }
}
```

OBJECT-fields	
exportFields	(Required) A semicolon (;) separated list of the fields to be exported. Custom fields are simply just specified with the name of the custom field and translation texts can be specified as *text_<language code>*, for example text_frFR for a French translation. For a full list of fields that can be exported see the list below.

WHERE-fields	
`responseType`	(Required) Specifies the format of the response data. The value can be one of the following: **xml** = XML formatted response. **json** = JSON formatted response. **csv** = CSV formatted response. (Comma Separated Values) **ldc** = Localize Direct Compact data format. **Note:** Using the ldc-format, quotes and other special characters will not be treated in a special way. Instead unique separators will be used. The data for each field is separated with a pair of Unicode 0xAC characters (¬¬) and each string is separated with a pair of Unicode 0xAF characters (¯¯).
projectName	(Required see Text) Project Name. Must be specified unless *projectId* is specified.
projectId	(Required see Text) Unique Project Id (GUID). Must be specified unless *projectName* is specified.
folderPaths	(Required see Text) Specifies the folders to be exported. The folder paths must be valid and start with the root string folder. Use a slash ('/') to separate folders in the path. For example: Strings/Folder/Sub Folder Use semicolon (;) to specify more than one folder path. **Note:** *folderPaths* should always be specified unless *identifiers* is used instead to only export specific strings.
identifiers	(Optional) String Identifier Name of strings to be exported. (If non unique string identifiers are used in the project, then the full path must be specified). This option can be used if only specific strings should be exported. Using a semicolon (;) separator, more than one identifier name may be specified in this field.
Custom field name to filter on.	(Optional) A custom field name that you wish to filter on. For example, to include only strings with a custom filed named "MyCustom" and the value "text", the following element would be used: **<MyCustom>test</MyCustom>.** If instead strings should be excluded on the specified value, then use an exclamation mark before the value, like this: **<MyCustom>!test</MyCustom>**.
newLineCharacter	(Optional) Specifies what new-line character should be used. By default new-lines are returned as \r\n. XML example to instead use \n: `<newLineCharacter>\n</newLineCharacter>`

	JSON example to instead use \n: `"newLineCharacter": "\n"`

List of Fields that can be exported

Export Field Name	**Description (Meaning and response)**
stringId	String Globally Unique Identifier (GUID)
path	String Path
identifierName	String Identifier Name
sourceLanguageText	Source Language Text
Custom Field Name	Custom field name of any custom field that should be exported. **Important:** Please make sure that the custom filed exists in the project, if not, then the response will be incorrect.
text_<language code>	Translation text. This name should be specified as *text_<language code>*, for example text_frFR for a French translation.
status_<language code>	Translation status. This name should be specified as *status_<language code>*, for example status_frFR for a French translation. The response value is a numeric value with the following meaning: 1 = Not translated (Red in client) 2 = Out-of-date (Yellow in client) 3 = Translated (Green in client)
stringType	A numeric value indicating the string type to which this string belongs.
stringTypeName	A string value with the name of the String Type to which this string belongs.
description	String description
childOrder	String child order under it's parent (only automatically set by LD if Dialogue Folders are being used).
maxPixelWidth	Max Pixels Dimensions
maxTextLength	Max Text Dimensions
translationPriority	Numeric value indicating the string translation priority.
requiredContent	Indicates if this String has required content.
tokenRule	Token rule defined for the String.
hasAttachment	Boolean value that indicates if the string has a content attachment.
hasComments	Boolean value that indicates if the string has comments.

lastModified	Date when source string data was last modified (not translations)

List of Character Fields

The list below is names from character objects that also can be exported. These fields are actually not stored on each string, but can still be accessed this way.

Export Field Name	**Description (Meaning and response)**
dialogueCharacterId	GUID of character object associated with string.
dialogueCharacterName	Character Name
dialogueCharacterType	Character Type
characterAge	Character Age
characterDescription	Character Description
characterAccent	Character Accent/ethnicity
characterNotes	Character Notes
characterDefine	Character Define
characterMisc	Character Misc
characterCustom	Character Custom
characterFinalActor	Character Final Actor
characterCharacteristics	Character Charactersistics
characterProcess	Character Process
characterActorMixer	Character Actor Mixer
characterBrand	Character Brand
characterOfficeVoice	Character Office Voice
characterOfficeActor	Character Office Actor
characterStuntActor	Character Stunt Actor
characterRobotVoice	Character Robot Voice
characterOfficeVoiceSettings	Character Office Voice Settings
characterOfficeActorSettings	Character Office Actor Settings
characterFinalActorSettings	Character Final Actor Settings
characterStuntActorSettings	Character Stunt Actor Settings
characterFaceFX	Character Face FX

Example command in XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
   <TASK name="StringExport">
      <OBJECT name="String">
<exportFields>path;identifierName;sourceLanguageText;status_frFR;text_frFR;
MyCustom;</exportFields>
      </OBJECT>
      <WHERE>
            <projectName>My Game</projectName>
            <folderPaths>Strings/Test/A/;Strings/Test2/;</folderPaths>
           <identifiers></identifiers>
           <responseType>csv</responseType>
      </WHERE>
   </TASK>
</EXECUTION>
```

Example command in JSON:

```
{
  "secId": "<secId>",
  "command": "String.StringExport",

  "data": {
           "exportFields":
"path;identifierName;sourceLanguageText;text_frFR;My_Custom_Field;"
  },

  "where": {
    "projectName": "My Game",
    "folderPaths": "Strings/Test2;",
    "responseType": "json",
    "newLineCharacter": "\n"
  }
}
```

Response

CSV

The following is an example of response in CSV:

```
"Strings/Example/","STRING_1","Text 1","2","French Text 1"
"Strings/Example/","STRING_2","Text 2","3","French Text 2"
"Strings/Example/","STRING_3","Text 3","3","French Text 3"
```

LDC

The following is an example of response in LDC:

```
Strings/Example/¬¬STRING_1¬¬Text 1¬¬2¬¬French Text
1¬¬¯¯Strings/Example/¬¬STRING_2¬¬Text 2¬¬3¬¬French Text
2¬¬¯¯Strings/Example/¬¬STRING_3¬¬Text 3¬¬3¬¬French Text 3¬¬¯¯
```

XML

The following is an example of response in XML:

```
    <RESULTSET>
      <DATASETS>
        <currentDateTime>2014-03-14 10:49:22</currentDateTime>
        <UTC_offset>1</UTC_offset>
        <![CDATA[Strings/Example/¬¬STRING_1¬¬Text 1¬¬2¬¬French Text
1¬¬¯¯Strings/Example/¬¬STRING_2¬¬Text 2¬¬3¬¬French Text
2¬¬¯¯Strings/Example/¬¬STRING_3¬¬Text 3¬¬3¬¬French Text 3¬¬¯¯]]>
      </DATASETS>
    </RESULTSET>
```

JSON

The following is an example of response in JSON:

```
{
  "resultset": [
    {"dataset":["Strings/Example/","STRING_1","Text 1","2","French Txt1"]},
    {"dataset":["Strings/Example/","STRING_2","Text 2","3","French Txt2"]},
    {"dataset":["Strings/Example/","STRING_3","Text 3","3","French Txt3"]}
  ]
}
```

StringImport Command

With this command strings can be created, updated and/or moved. This command is efficient when it comes to importing many strings at once. The command operates very much in the same way as the Import Strings feature in the LocDirect application client.

XML request (Socket API or HTTP API)

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
    <TASK name="StringImport">
        <OBJECT name="String">
            <importFields/>
            <fieldData/>
        </OBJECT>
        <WHERE>
            <stringMergeOption/>
            [<projectName/>]
            [<projectId/>]
            [<createFolders>]
            [<deleteEmptyFolders>]
        </WHERE>
    </TASK>
</EXECUTION>
```

JSON request (HTTP API)

```
{
  "secId": "<secId>",
  "command": "String.StringImport",

  "data": {
    "importFields": [ An array of field names ],
    "fieldData": [
         { "dataset": [ An array of field data ] },
         { "dataset": [ An array of field data ] },
         ...
    ]
  },

  "where": {
    "projectName": "<projectName>",
    "createFolders": "<createFolders>",
    "stringMergeOption": "<stringMergeOption>"
  }
}
```

OBJECT-fields	
importFields	(Required) A semicolon (;) separated list of the fields to be imported. The data of these field will then be specified for each string in the **fieldData**-element where separators are being used to separate the stings and the values of the strings. When importing strings there are two fields

	that always is required: *folderPath* and *identifierName*. All other fields are optional. Custom fields are simply just specified with the name of the custom field and translation texts can be specified as *text_<language code>*, for example text_frFR for a French translation.
fieldData	(Required) This is the data for all strings that will be imported. **JSON:** An array of datasest-objects, where each object has an array of field data strings. See example below. **XML:** The data for each field is separated with a pair of Unicode 0xAC characters (¬¬) and each string is separated with a pair of Unicode 0xAF characters (¯¯). The reason for why these special characters are being used, is to allow for more traditional separators like semicolon and comma to be used as field values (in for example the string texts). It's important that the data for each field always end with a end-of-field-separator (even the last one and/or if a field is empty). It's also important to end each string with a end-of-string-separator. For example: Field1¬¬Field2¬¬¯¯.

WHERE-fields	
projectName	(Required see Text) Project Name. Must be specified unless *projectId* is specified.
projectId	(Required see Text) Unique Project Id (GUID). Must be specified unless *projectName* is specified.
stringMergeOption	(Required) The value of this option is very important as it tells what to do if a string with the specified identifier name and folder path already exist: The value can be one of the following numeric values: 0 = Abort import on conflict. 1 = Use all fields from import (delete and create a new string). 2 = Ignore import fields on conflict (skip the import for that string). 3 = Merge fields, imported fields has priority.
createFolders	A Boolean value. If set to true, then folders that are specified in the *folderPath* but doesn't exist in LocDirect will be created. When a

	folder is created it will automatically use the properties of its parent. If not specified, this field will be set to false
`deleteEmptyFolders`	A Boolean value. If set to **true**, then if a string change its parent folder when it's updated and the original folder becomes empty, then this folder will be deleted. The default value for this option is **false**.

Example command:

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
   <TASK name="StringImport">
      <OBJECT name="String">
<importFields>folderPath;identifierName;sourceLanguageText;text_frFR;Custom
1;</importFields>
           <fieldData>Strings/Test2/¬¬ STRING_MSG_RP_SAVED¬¬Replay
saved.¬¬Ralenti enregistr.¬¬A custom
value!¬¬¯¯Strings/Test2/¬¬String_MSG_QUIT_CONFIRM¬¬Are you sure you want to
quit?¬¬Etes-vous sr(e) de vouloir quitter ?¬¬Custom test
text¬¬¯¯Strings/Test2/¬¬String_NO¬¬No¬¬Non¬¬¬¬¯¯</fieldData>
      </OBJECT>
      <WHERE>
           <stringMergeOption>3</stringMergeOption>
           <projectName>My Game</projectName>
      </WHERE>
   </TASK>
</EXECUTION>
```

JSON

```
{
  "secId": "12312312-3123-1231-2312-3123123123123",
  "command": "String.StringImport",

  "data": {

    "importFields": [
           "folderPath",
           "identifierName",
           "sourceLanguageText",
           "text_svSE" ],

    "fieldData": [
      {
        "dataset": ["Strings/Test/","STR_1","Hello there!","Hej på dej!"]
      },
      {
        "dataset": ["Strings/Test/","STR_2","A\nnew-line","En\nny rad"]
      }
    ]
  },

  "where": {
    "stringMergeOption": "3",
    "projectName": "My Game"
  }
}
```

RESULTSET structure

The StringImport command will always return the number of successfully imported strings and the number of failed strings.

Successful Result Example:

XML

```
<RESULTSET>
   <DATASETS>
     <DATASET datatype="dataSet">
       <numberOfInsertedWords>0</numberOfInsertedWords>
       <numberOfUpdatedWords>29</numberOfUpdatedWords>
       <numberOfInsertedStrings>0</numberOfInsertedStrings>
       <numberOfUpdatedStrings>4</numberOfUpdatedStrings>
       <numberOfFailedStrings>0</numberOfFailedStrings>
       <numberOfImportedStrings>3</numberOfImportedStrings>
     </DATASET>
   </DATASETS>
 </RESULTSET>
```

JSON

```
{
  "datasets":[
     {
       "numberOfInsertedWords":"0",
       "numberOfUpdatedWords":"29",
       "numberOfInsertedStrings":"0",
       "numberOfUpdatedStrings":"4",
       "numberOfFailedStrings":"0",
       "numberOfImportedStrings":"3"
     }
  ]
}
```

Unsuccessful Result Example:

XML

```
<RESULTSET>
   <DATASETS>
         ...
   </DATASETS>
   <RESULT>
      <MESSAGE id="7" type="Error">
           A string with identifier STR_123 already exists!
      </MESSAGE>
   </RESULT>
</RESULTSET>
```

JSON

```
{
  "messages": [
    {
      "id": "7",
      "text": "'EMPTY_STRING' already exists!",
      "type":"Error"
    },
    {
      "id": "7",
      "text":"'STRING_2' already exists!",
      "type":"Error"}
  ],

  "datasets": [
    {
     "numberOfInsertedWords": "0",
     "numberOfUpdatedWords": "0",
     "numberOfInsertedStrings": "0",
     "numberOfUpdatedStrings": "0",
     "numberOfFailedStrings": "2",
     "numberOfImportedStrings": "0"
    }
  ]
}
```

GetStrings Command

With this command you can fetch strings from the server. The command has a number of arguments that can be used to specify the scope of strings to fetch.

Important: In almost all cases it's a better to instead use the **StringExport**-command. This command exists for backward compatibility.

XML request (Socket API or HTTP API)

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="GetStrings">
          <OBJECT name="String">
               [<identifierName/>]
               [<text/>]
               [<stringId/>]
               [<sourceLanguageText/>]
               [<stringType/>]
               [<childOrder/>]
               [<folderPath/>]
               [<folderId/>]
               [<description/>]
               [<folderDescription/>]
               [<lastModified/>]
               [<folderName/>]
               [<requiredContent/>]
               [<audioStatus/>]
               [<textImageStatus/>]
               [<dependentFileStatus/>]
               [<custom field name/>]
          </OBJECT>
          <WHERE>
               [<projectName/>]
               [<projectId/>]
               [<compactData/>]
               [<lastModified/>]
               [<languageCode/>]
               [<folderName/>]
               [<folderId/>]
               [<recursive/>]
               [<identifierName/>]
               [<stringId/>]
               [<stringType/>]
               [<excludeOutOfDate/>]
               [<includeStatus/>]
               [<custom field name/>]
          </WHERE>
  </TASK>
</EXECUTION>
```

JSON request (HTTP API)

```
{
  "secId": "<secId>",
  "command": "String.GetStrings",
  "data": {
    (see xml example above)
  },

  "where": {
    (see xml example above)
  }
}
```

OBJECT-fields	
identifierName	(Optional) String Identifier Name (If non unique string identifiers are used in the project, then the full path must be specified).
text	(Optional) String or translation text
stringId	(Optional) String Id as a GUID
stringType	(Optional) String type
sourceLanguageText	(Optional) Source Language Text
childOrder	(Optional) String order
folderPath	(Optional) Folder Path
folderId	(Optional) Folder Id as a GUID
description	(Optional) Description
folderDescription	(Optional) Folder description
lastModified	(Optional) Date and time when the string was last modified.
`requiredContent`	(Optional) Required content specification. Specifies the required content by a value that is a logical OR of the following (depending on the requirements for the string): 0x1 = Audio file required 0x2 = Text Image required 0x4 = Dependent file required. Example: A value of 7 means audio, text and dependent file is required. A value of 1 means that an audio file is required.
`audioStatus`	(Optional) Audio status. Can be one of: 0 = No audio file 2 = Out of date 4 = Final Note: This status will be returned as an attribute to the text-element.
`textImageStatus`	(Optional) Text Image status. Can be one of: 0 = No text image file 2 = Out of date 4 = Final

	Note: This status will be returned as an attribute to the text-element.
`dependentFileStatus`	(Optional) Dependent File status. Can be one of: 0 = No dependent file 2 = Out of date 4 = Final Note: This status will be returned as an attribute to the text-element.
<custom field name>	(Optional) Name of a custom field to be fetched. More than one custom field name can be specified in the OBJECT-element.

WHERE-fields	
projectName	Project Name. Must be specified unless *projectId* is specified.
projectId	Unique Project Id (GUID). Must be specified unless *projectName* is specified.
compactData	(Optional) If set to true, then data will be returned in a compact way instead of XML formatted. *See description below "Compact Data".*
lastModified	(Optional) Specifies that only strings modified after the specified date and time should be fetched. The time must be specified in the server time zone and the format must be: yyyy-mm-dd hh:mm:ss **Note:** The GetStrings-command will always return an element called <CurrentDateTime> that holds the server time for when the strings of this command was fetched from the server. This time can then be used as lastModified-argument to fetch strings that's been changes since last time this command was invoked.
languageCode	(Optional) One or many language codes of languages to fetch. The language is a 4 character string that identifies the language, for example "enUS" means American English and "frFR" means French. For the source language "sourceLanguageText" can be used. To specify more than one languageCode, use semicolon (;) as separator. For example: enUS;frFR **Note:** If this argument isn't specified as a WHERE-argument then all languages will be fetched.

folderPath	(Restricted Optional) Specifies that only strings in the specified folders should be fetched. The folder paths must be valid and start with the root string folder. Use a slash ('/') to separate folders in the path. For example: Strings/Folder/Sub Folder Use semicolon (;) to specify more than one folder path. **Note:** *folderPath* should only be used if *folderId* isn't specified. (Don't use both *folderId* and folderPath as they are alternatives to each other and the one will override the other depending on the order they appear in the command).
folderId	(Restricted Optional) Specifies that only strings in a specific folder should be fetched. The folderId must be a valid GUID. If more than one folder needs to be specified, then use a semicolon (;) as separator. **Note:** *folderId* should only be used if *folderPath* isn't specified. (Don't use both *folderId* and folderPath as they are alternatives to each other and the one will override the other depending on the order they appear in the command).
recursive	(Optional) This value can either be set to "true" or "false" (default is "false"). The value specifies whether or not strings should be fetched recursively under the folders specified either using the <folderId> or <folderPath> element. (Recursive in this case means that strings in sub-folders also will be fetched.)
identifierName	(Optional) Specifies one or many strings to be fetched using identifier name. Use semicolon (;) to specify more than one identifier name.
stringId	(Optional) Specifies one or many strings to be fetched using string id (GUID). Use semicolon (;) to specify more than one string id.
stringType	(Optional) String Type.
excludeOutOfDate	(Optional) This value can be set to either “true” or “false”. If it’s set to “true”, then all translations that are out-of-date will be excluded from the result.
includeStatus	(Optional) This value can be set to either “true” or “false”. If it’s set to “true”, then the translation status for each translation will be returned. If non-compact data sets is used, then the status will be returned as an attribute to the <text> element. If compact data sets are used, then an additional status-column will be added for each language. The status can be one of the following values: 0 = Source Language 1 = Not translated

	2 = Out-of-date 3 = Translated (up-to-date)
<custom field name>	(Optional) Custom field and value to filter on.

RESULTSET structure

The elements in the RESULTSET-structure may vary depending on the fields specified in the OBJECT-element of this command.

```
<RESULTSET>
     <DATASETS>
          <Strings>
               <currentDateTime>2011-08-22 13:25:30</currentDateTime>
               <UTC_offset>1</UTC_offset>
               <String>
                     <identifierName>STR_APPLY</identifierName>
                     <text language="esES">Aplicar</text>
                     <text language="enUS">Apply</text>
                     <text language="frFR">Appliquer</text>
                     <text language="deDE">Anwenden</text>
                     <text language="itIT">Applica</text>
                     [<Additional fields specified in OBJECT element>]
               </String>

               ...

          </Strings>
     </DATASETS>
     <RESULT/>
</RESULTSET>
```

Strings element	
currentDateTime	The server date time for when the strings where fetched from the server.
UTC_offset	The server UTC (Coordinated Universal Time) offset (in hours).

String elements (default)	
identifierName	String Identifier Name
text	String text for the language specified by the *language*-attribute.

The result node will contain MESSAGE-elements if something went wrong.

Compact Data

If the WHERE-argument <compactData> is set to "true", the returning string data will be returned in a more compact way compared to when returned as XML formatted data. Instead of having String xml elements with child elements for each column, all of the data will be returned in one single xml element called <data>. Each of the returning strings will be separated with pairs of 0xAF (Unicode characters: ¯¯) and the data of each string will be separated with pairs of 0xAC (Unicode characters: ¬¬).

An additional XML element called <metadata> will be returned together with the <data> element. The <metadata> element will describe the name and the order of the data returned in the <data> element.

This is an example of what the data may look like when using compact data:

```
<RESULTSET>
<DATASETS>
  <Strings>
    <metadata>identifierName¬¬text_enEN¬¬text_deDE¬¬</metadata>
     <data>STR_APPLY¬¬Apply¬¬Anwenden¯¯STR_SEARCH¬¬Search¬¬Suchen¯¯</data>
     ...

  </Strings>
</DATASETS>
</RESULTSET>
```

In the example above, two strings are returned. Note that the metadata element will only describe one single "row of data", as all returning strings will be returned this way (eg. the same columns and the same order of the columns).
Note that the *language*-attribute that is used when string data is returned as XML formatted now instead is described in the metadata as text_<language code><contry code> (eg. text_enUS for American English and text_deDE for German).

UpdateStrings Command

With this command you can update the text or meta-data of one or many strings.

Important: In almost all cases it's a better to instead use the **StringImport**-command. This command exists for backward compatibility.

XML request (Socket API or HTTP API)

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="UpdateStrings">
          <OBJECT name="String">
               [<identifierName/>]
               [<sourceLanguageText/>]
               [<text_languageCode/>]
               [<description/>]
               [<custom field name/>]
               [<order>]
               [<stringType>]
               [<retranslate>]
               [<requiredContent/>]
               [<translationStatus_languageCode/>]
               [<audioStatus_languageCode/>]
               [<textImageStatus_languageCode/>]
               [<dependentFileStatus_languageCode/>]
          </OBJECT>
          <WHERE>
               [<projectName/>]
               [<projectId/>]
               [<identifierName/>]
               [<stringId/>]
          </WHERE>
     </TASK>
</EXECUTION>
```

Example how to update a string:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="UpdateStrings">
          <OBJECT name="String">
               <sourceLanguageText>testing</sourceLanguageText>
          </OBJECT>
          <WHERE>
               <projectId>31FB05E1-F3B0-7AEF-0508-46162DFE987D</projectId>
               <identifierName>Strings_1</identifierName>
          </WHERE>
     </TASK>
</EXECUTION>
```

JSON request (HTTP API)

```
{
  "secId": "<secId>",
  "command": "String.UpdateStrings",

  "data": {
    (see xml example above)
  },

  "where": {
    (see xml example above)
  }
}
```

OBJECT-fields	
identifierName	(Optional) New String Identifier Name (If non unique string identifiers are used in the project, then the full path must be specified).
sourceLanguageText	(Optional) New Source Language Text.
`text_<languageCode>`	(Optional) New String translation text to be updated. The *languageCode* section of the name should be replaced with the code of the language of the translation, e.g. text_frFR or text_esES.
description	(Optional) Description
<custom field name>	(Optional) New Custom field value to be updated.
order	(Optional) New String order (only applies to Dialogue Strings)
stringType	(Optional) String Type
retranslate	(Optional) If the source language text is changed and it has already been translated, then this parameter can be used to indicate whether or not translations need to be retranslated. The value can be either *true* or *false*. Not specifying this parameter (not including it in the command) is equal to setting it to *false*.
`requiredContent`	(Optional) Required content specification. Specify the required by a value that is a logical OR of the following (depending on the requirements for the string): 0x1 = Audio file required 0x2 = Text Image required 0x4 = Dependent file required. Example: A value of 7 means audio, text and dependent file is required. A value of 1 means that an audio file is required.
`translationStatus_<languageCode>`	(Optional) Updates the status of the translation with the specified language code. The status

	may be set to one of the following. 2 = Out-of-date 3 = Translated (up-to-date) **Note:** Status 0 (meaning Source Language) and 1 (meaning no translation) may not be set using this command.
`audioStatus_<languageCode>`	(Optional) Audio status. Can be one of: 0 = No audio file 2 = Out of date 4 = Final
`textImageStatus_<languageCode>`	(Optional) Text Image status. Can be one of: 0 = No text image file 2 = Out of date 4 = Final
`dependentFileStatus_<languageCode>`	(Optional) Dependent File status. Can be one of: 0 = No dependent file 2 = Out of date 4 = Final

WHERE-fields	
projectName	(Restricted Optional) Project Name. Must be specified unless *projectId* is specified.
projectId	(Restricted Optional) Unique Project Id (GUID). Must be specified unless *projectName* is specified.
identifierName	(Restricted Optional) String Identifier Name (If non unique string identifiers are used in the project, then the full path must be specified). Using a semicolon (;) separator, more than one identifier name may be specified in this field.
stringId	(Restricted Optional) Unique String Id (GUID). Must be specified unless *identifierName* is specified. Using a semicolon (;) separator, more than one string id may be specified in this field.

RESULTSET structure

If successfully updated, then the result set will consist of a RESULT-element with no elements, else there will be a MESSAGE-element under the RESULT-element.

Successful Result Example:

```
<RESULTSET>
     <DATASETS />
     <RESULT />
</RESULTSET>
```

Unsuccessful Result Example:

```
<RESULTSET>
     <DATASETS />
     <RESULT>
          <MESSAGE id="1" type="Error">
            Found no String matching the specified projectId and
            identifierName!
          </MESSAGE>
     </RESULT>
</RESULTSET>
```

InsertStrings Command

With this command you can insert new strings.

Important: In almost all cases it’s a better to instead use the **StringImport**-command. This command exists for backward compatibility.

XML request (Socket API or HTTP API)

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="InsertStrings">
          <OBJECT name="String">
               <identifierName/>
               [<sourceLanguageText/>]
               [<text_languageCode/>]
               [<description/>]
               [<custom field name/>]
               [<order/>]
               [<stringType/>]
               [<requiredContent/>]
               [<audioStatus_languageCode/>]
               [<textImageStatus_languageCode/>]
               [<dependentFileStatus_languageCode/>]
          </OBJECT>
          <WHERE>
               [<projectName/>]
               [<projectId/>]
               [<folderName/>]
               [<folderId/>]
          </WHERE>
     </TASK>
</EXECUTION>
```

Example how to insert a source language string text:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="InsertStrings">
          <OBJECT name="String">
               <identifierName>Strings/Test/str_test</identifierName>
               <sourceLanguageText>testing</sourceLanguageText>
          </OBJECT>
          <WHERE>
               <projectName>Test Project</projectName >
          </WHERE>
     </TASK>
</EXECUTION>
```

JSON request (HTTP API)

```
{
  "secId": "<secId>",
  "command": "String.InsertStrings",

  "data": {
    (see xml example above)
  },

  "where": {
    (see xml example above)
  }
}
```

OBJECT-fields	
identifierName	String Identifier Name (If non unique string identifiers are used in the project, then the full path must be specified).
sourceLanguageText	(Optional) Source Language Text.
`text_<languageCode>`	(Optional) String translation text to be inserted. The *languageCode* section of the name should be replaced with the code of the language of the translation, e.g. text_frFR or text_esES.
description	(Optional) Description
<custom field name>	(Optional) Custom field to be inserted.
order	(Optional) String order (only applies to Dialogue Strings)
stringType	(Optional) String Type
`requiredContent`	(Optional) Required content specification. Specify the required by a value that is a logical OR of the following (depending on the requirements for the string): 0x1 = Audio file required 0x2 = Text Image required 0x4 = Dependent file required. Example: A value of 7 means audio, text and dependent file is required. A value of 1 means that an audio file is required.
`audioStatus_<languageCode>`	(Optional) Audio status. Can be one of: 0 = No audio file 2 = Out of date 4 = Final
`textImageStatus_<languageCode>`	(Optional) Text Image status. Can be one of: 0 = No text image file 2 = Out of date 4 = Final
`dependentFileStatus_<languageCode>`	(Optional) Dependent File status. Can be one of: 0 = No dependent file 2 = Out of date 4 = Final

WHERE-fields	
projectName	(Restricted Optional) Project Name. Must be specified unless *projectId* is specified.
projectId	(Restricted Optional) Unique Project Id (GUID). Must be specified unless *projectName* is specified.
folderName	(Optional) Specifies the name of the folder under which the strings should be inserted. If a path was specified in the *identifierName* (object-element) then there's no need to specify the *folderId* element in the command.
folderId	(Optional) Specifies the unique folder id (GUID) under which the strings should be inserted. If a path was specified in the *identifierName* (object-element) then there's no need to specify the *folderId* element in the command.

RESULTSET structure

If successfully inserted, then the result set will consist of a RESULT-element with no elements, else there will be a MESSAGE-element under the RESULT-element.

Successful Result Example:

```
<RESULTSET>
     <DATASETS />
     <RESULT />
</RESULTSET>
```

Unsuccessful Result Example:

```
<RESULTSET>
     <DATASETS />
     <RESULT>
          <MESSAGE id="1" type="Error">
            Failed to insert the string message
          </MESSAGE>
     </RESULT>
</RESULTSET>
```

DeleteStrings Command

This command will delete strings from LocDirect.

XML request (Socket API or HTTP API)

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="DeleteStrings">
          <OBJECT name="String"/>
          <WHERE>
               [<projectName/>]
               [<projectId/>]
               [<identifierName/>]
               [<stringId/>]
          </WHERE>
     </TASK>
</EXECUTION>
```

Example how to delete strings:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="DeleteStrings">
          <OBJECT name="String"/>
          <WHERE>
               <projectName>Test Project</projectName >
               <identifierName>str_test1;str_test2</identifierName>
          </WHERE>
     </TASK>
</EXECUTION>
```

JSON request (HTTP API)

```
{
  "secId": "<secId>",
  "command": "String.DeleteStrings",

  "data": {},

  "where": {
     (see xml example above)
  }
}
```

OBJECT-fields (Non)

WHERE-fields	
projectName	(Restricted Optional) Project Name. Must be specified unless *projectId* is specified.
projectId	(Restricted Optional) Unique Project Id (GUID). Must be specified unless *projectName* is specified.
identifierName	(Restricted Optional) String Identifier Name (If non unique string

	identifiers are used in the project, then the full path must be specified). Must be specified unless *stringId* is specified. Using a semicolon (;) separator, more than one identifier name may be specified in this field.
stringId	(Restricted Optional) Unique String Id (GUID). Must be specified unless *identifierName* is specified. Using a semicolon (;) separator, more than one string id may be specified in this field.

RESULTSET structure

If successfully deleted, then the result set will consist of a RESULT-element with no elements, else there will be a MESSAGE-element under the RESULT-element.

Successful Result Example:

```
<RESULTSET>
     <DATASETS />
     <RESULT />
</RESULTSET>
```

Unsuccessful Result Example:

```
<RESULTSET>
     <DATASETS />
     <RESULT>
          <MESSAGE id="1" type="Error">
            Failed to delete string message
          </MESSAGE>
     </RESULT>
</RESULTSET>
```

MoveStrings Command

With this command you can move strings from one folder to another.

Note: To move or change the order of a Dialogue String, use the StringImport-command instead.

XML request (Socket API or HTTP API)

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="MoveStrings">
          <OBJECT name="String">
               [<folderName/>]
               [<folderId/>]
          </OBJECT>
          <WHERE>
               [<projectName/>]
               [<projectId/>]
               [<identifierName/>]
               [<stringId/>]
          </WHERE>
     </TASK>
</EXECUTION>
```

Example how to move strings:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="MoveStrings">
          <OBJECT name="String">
               <folderName>Strings/Test/</folderName>
          </OBJECT>
          <WHERE>
               <projectName>Test Project</projectName >
               <identifierName>str_test1;str_test2</identifierName>
          </WHERE>
     </TASK>
</EXECUTION>
```

OBJECT-fields	
folderName	(Restricted Optional) Path to the folder into which the string will be moved. Must be specified unless *folderId* is specified.
folderId	(Restricted Optional) Unique folder id (GUID) of the folder into witch the string will be moved. Must be specified unless *folderName* is specified.

WHERE-fields	
projectName	(Restricted Optional) Project Name. Must be specified unless *projectId* is specified.
projectId	(Restricted Optional) Unique Project Id (GUID). Must be specified unless *projectName* is specified.

identifierName	(Restricted Optional) String Identifier Name (If non unique string identifiers are used in the project, then the full path must be specified). Must be specified unless *stringId* is specified. Using a semicolon (;) separator, more than one identifier name may be specified in this field.
stringId	(Restricted Optional) Unique String Id (GUID). Must be specified unless *identifierName* is specified. Using a semicolon (;) separator, more than one string id may be specified in this field.

RESULTSET structure

If successfully moved, then the result set will consist of a RESULT-element with no elements, else there will be a MESSAGE-element under the RESULT-element.

Successful Result Example:

```
<RESULTSET>
     <DATASETS />
     <RESULT />
</RESULTSET>
```

Unsuccessful Result Example:

```
<RESULTSET>
     <DATASETS />
     <RESULT>
          <MESSAGE id="1" type="Error">
            Failed to move string message
          </MESSAGE>
     </RESULT>
</RESULTSET>
```

ChangeStringFolderStatus Command

With this command you can change the folder status of String Folders. The states that can be set is whether or not strings in the folder is ready for translation and whether or not the source language text is locked. The command can change the state of several folders at a time and also set the states recursively.

XML request (Socket API or HTTP API)

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
  <TASK name="ChangeStringFolderStatus">
     <OBJECT name="Folder">
         [<ReadyForTranslation/>]
         [<SourceLanguageLocked/>]
         [<applyOnChildFolders/>]
     </OBJECT>
     <WHERE>
         [<projectName/>]
         [<projectId/>]
         [<folderName/>]
         [<folderId/>]
     </WHERE>
  </TASK>
</EXECUTION>
```

Example how to use the command:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
  <TASK name="ChangeStringFolderStatus">
     <OBJECT name="Folder">
         <ReadyForTranslation>true</ReadyForTranslation>
         <applyOnChildFolders>true</applyOnChildFolders>
     </OBJECT>
     <WHERE>
         <projectName>Test Project</projectName>
         <folderName>Strings/Folder1;Strings/Folder2</folderName>
     </WHERE>
  </TASK>
</EXECUTION>
```

JSON request (HTTP API)

```
{
  "secId": "<secId>",
  "command": "Folder.ChangeStringFolderStatus",

  "data": {
    (see xml example above)
  },

  "where": {
    (see xml example above)
  }
}
```

OBJECT-fields	
ReadyForTranslation	(Optional) *true* or *false* depending on if strings in the specified folders should be ready for translation.
SourceLanguageLocked	(Optional) *true* or *false* depending on if strings in the specified folders should have the source language text locked.
applyOnChildFolders	(Optional)) If set to *true* then the folder states will also be set on all child folders. Default is false (if this element isn't specified).

WHERE-fields	
projectName	(Restricted Optional) Project Name. Must be specified unless *projectId* is specified.
projectId	(Restricted Optional) Unique Project Id (GUID). Must be specified unless *projectName* is specified.
folderName	(Restricted Optional) Path to the folder(s) to be affected. Must be specified unless *folderId* is specified.
folderId	(Restricted Optional) Unique folder id (GUID) of the folder to be affected. Must be specified unless *folderName* is specified.
applyOnChildFolders	(Optional) If set to *true* then the folder states will also be set on all child folders. Default is false (if this element isn't specified).

RESULTSET structure

If successfully updated, then the result set will consist of a RESULT-element with no elements, else there will be a MESSAGE-element under the RESULT-element.

Successful Result Example:

```
<RESULTSET>
     <DATASETS />
     <RESULT />
</RESULTSET>
```

Unsuccessful Result Example:

```
<RESULTSET>
     <DATASETS />
     <RESULT>
          <MESSAGE id="1" type="Error">
               Missing folderIDs or folder paths!
          </MESSAGE>
     </RESULT>
</RESULTSET>
```

CreateStringFolder Command

With this command you can create new string folders.

XML request (Socket API or HTTP API)

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="CreateStringFolder">
          <OBJECT name="String">
               <folderName/>
               [<description/>]
               [<folderSubType/>]
               [<folderStatus/>]
          </OBJECT>
          <WHERE>
               [<projectName/>]
               [<projectId/>]
               [<parentFolderName/>]
               [<parentFolderId/>]
          </WHERE>
     </TASK>
</EXECUTION>
```

Example how to create a new string folder:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="CreateStringFolder">
          <OBJECT name="String">
               <folderName>New Folder</folderName>
               <description>A description</description>
          </OBJECT>
          <WHERE>
               <projectName>Test Project</projectName>
               <parentFolderName>Strings/Folder/</parentFolderName>
          </WHERE>
     </TASK>
</EXECUTION>
```

JSON request (HTTP API)

```
{
  "secId": "<secId>",
  "command": "String.CreateStringFolder",

  "data": {
    (see xml example above)
  },

  "where": {
    (see xml example above)
  }
}
```

OBJECT-fields	
folderName	Folder name. Must be unique under its parent

	folder.
description	(Optional) Description
folderSubType	(Optional) Specifies whether the folder is a string or dialogue folder. If not specified, then this will be a string folder. 0 = String Folder 1 = Dialogue Folder
folderStatus	(Optional) Specifies the default status of the folder. Use the following bits to specify a value for the folder status (default is 1): 0x1 = Strings not ready for translation. 0x2 = Source language text locked.

WHERE-fields	
projectName	(Restricted Optional) Project Name. Must be specified unless *projectId* is specified.
projectId	(Restricted Optional) Unique Project Id (GUID). Must be specified unless *projectName* is specified.
parentFolderName	(Restricted Optional) Specifies the name and path of the folder under which the new string folder should be created. Must be specified unless *folderId* is specified.
parentFolderId	(Restricted Optional) Specifies the unique folder id (GUID) under which the new string folder should be created. Must be specified unless *parentFolderName* is specified.

RESULTSET structure

If successfully inserted, then the result set will return the folderId of the created folder (see example below), else there will be a MESSAGE-element under the RESULT-element.

Successful Result Example:

```
<RESULTSET>
  <DATASETS>
    <DATASET datatype="Folder">
      <folderId>98903985-8954-10D3-7956-9AC31A27DCE1</folderId>
    </DATASET>
  </DATASETS>
  <RESULT />
</RESULTSET>
```

Unsuccessful Result Example:

```
<RESULTSET>
     <DATASETS />
     <RESULT>
          <MESSAGE id="0" type="Error">
            Reason for the failure message.
```

```
        </MESSAGE>
    </RESULT>
</RESULTSET>
```

UpdateStringFolder Command

With this command you can update a string folder. This command can also be used to move a string folder by updating the parentFolderId.

XML request (Socket API or HTTP API)

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="UpdateStringFolder">
          <OBJECT name="String">
               [<folderName/>]
               [<parentFolderId/>]
               [<description/>]
               [<folderStatus/>]
          </OBJECT>
          <WHERE>
               <folderId/>
          </WHERE>
     </TASK>
</EXECUTION>
```

Example how to update a folder:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="UpdateStringFolder">
          <OBJECT name="String">
               <folderStatus>1</folderStatus>
               <description>A description</description>
          </OBJECT>
          <WHERE>
               <folderId>E7659155-36FF-9C45-59BC-80F0AB1A8B3F</folderId>
          </WHERE>
     </TASK>
</EXECUTION>
```

Example how to move a folder:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="UpdateStringFolder">
          <OBJECT name="String">
     <parentFolderId>B64632C2-0FC3-A40A-DF26-116DBBBB1723</parentFolderId>
          </OBJECT>
          <WHERE>
               <folderId>E7659155-36FF-9C45-59BC-80F0AB1A8B3F</folderId>
          </WHERE>
     </TASK>
</EXECUTION>
```

JSON request (HTTP API)

```
{
  "secId": "<secId>",
  "command": "String.UpdateStringFolder",
  "data": {
    (see xml example above)
  },

  "where": {
    (see xml example above)
  }
}
```

OBJECT-fields	
folderName	(Optional) Folder name. Must be unique under its parent folder. **Please note:** Only use this if you wish to rename a folder.
description	(Optional) Folder description
folderStatus	(Optional) Specifies the default status of the folder. Use the following bits to specify a value for the folder status (default is 1): 0x1 = Strings not ready for translation. 0x2 = Source language text locked.
parentFolderId	(Optional) Use this if you wish to move the folder to another parent new folder.

WHERE-fields	
folderId	Unique Folder Id (GUID) of the folder to be updated.

CreateUser Command

With this command you can create a new user.

XML request (Socket API or HTTP API)

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="CreateUser">
          <OBJECT name="User">
               <userName/>
               <password/>
               <fullName/>
               [<clientType>]
               [<memberOfGroup/>]
               [<memberOfProject/>]
               [<emailAddress>]
               [<description/>]
               [<languageCode>]
               [<countryCode>]
          </OBJECT>
          <WHERE>
               [<parentFolderName/>]
               [<parentFolderId/>]
          </WHERE>
     </TASK>
</EXECUTION>
```

Example how to create a new user:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="CreateUser">
          <OBJECT name="User">
               <userName>John</userName>
               <password>123</password>
               <fullName>John Smith</fullName>
               <memberOfGroup>Test Project Users</memberOfGroup>
               <memberOfProject>Test Project</memberOfProject>
               <clientType>Developer</clientType>
          </OBJECT>
          <WHERE>
               <parentFolderName>Domain Users</parentFolderName>
          </WHERE>
     </TASK>
</EXECUTION>
```

JSON request (HTTP API)

```
{
  "secId": "<secId>",
  "command": "User.CreateUser",

  "data": {
    (see xml example above)
  },

  "where": {
      (see xml example above)
   }
}
```

OBJECT-fields	
userName	User name. Must be unique under its parent folder.
password	User password
fullName	Full user name
clientType	(Optional) User account client type. The value can be one of the following: Administrator, Developer, Translator, Loc Manager, Loc QA, Builder If not specified, then 'Developer' will be used.
memberOfGroup	(Optional) Name of one or many groups that this user is member of. Group names should be separated with a semicolon (;). **Note:** Make sure the group names are unique on the server.
memberOfProject	(Optional) Name of projects that this user is member of. Project names should be separated with a semicolon (;). **Note:** Make sure the project names are unique on the server.
emailAddress	(Optional) User Email Address
description	(Optional) User description.
languageCode	(Optional) If this is a translator user account, then a language code should be specified.
countryCode	(Optional) If this is a translator user account, then a country code should be specified.

WHERE-fields	
parentFolderName	(Restricted Optional) Specifies the name of the folder under which the new user should be created. This is just the name of the folder NOT a path (as user folders exists outside of projects). Make sure that the name of the user folder is unique before using this parameter. Must be specified unless *parentFolderId* is specified.
parentFolderId	(Restricted Optional) Specifies the unique folder id (GUID) under which the new user folder should be created. Must be specified unless *parentFolderName* is specified.

RESULTSET structure

If successfully inserted, then the result set will return the userId of the created user (see example below), else there will be a MESSAGE-element under the RESULT-element.

Successful Result Example:
```
<RESULTSET>
  <DATASETS>
    <DATASET datatype="User">
      <userId>98903982-8957-10D9-7954-9AC31A37DCE4</userId>
    </DATASET>
  </DATASETS>
  <RESULT />
</RESULTSET>
```

Unsuccessful Result Example:
```
<RESULTSET>
     <DATASETS />
     <RESULT>
          <MESSAGE id="0" type="Error">
            Reason for the failure message.
          </MESSAGE>
     </RESULT>
</RESULTSET>
```

UpdateUser Command

With this command you can update a user. It’s also possible to move a user to a different folder using this command.

XML request (Socket API or HTTP API)

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="UpdateUser">
          <OBJECT name="User">
               [<userName/>]
               [<password/>]
               [<fullName/>]
               [<clientType>]
               [<emailAddress>]
               [<description/>]
               [<parentFolderName/>]
               [<parentFolderId/>]
               [<memberOfGroup/>]
               [<memberOfProject/>]
               [<languageCode>]
               [<countryCode>]
          </OBJECT>
          <WHERE>
               [<userName/>]
               [<userId/>]
          </WHERE>
     </TASK>
</EXECUTION>
```

Example how to update a user:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="UpdateUser">
          <OBJECT name="User">
               <password>123</password>
               <fullName>John Smith</fullName>
               <clientType>Administrator</clientType>
          </OBJECT>
          <WHERE>
               <userName>John Smith</userName>
          </WHERE>
     </TASK>
</EXECUTION>
```

Example how to move a user to another user folder:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="UpdateUser">
          <OBJECT name="User">
               <parentFolderName>Admin Users</parentFolderName>
          </OBJECT>
          <WHERE>
               <userName>John Smith</userName>
          </WHERE>
```

```
    </TASK>
</EXECUTION>
```

JSON request (HTTP API)

```
{
  "secId": "<secId>",
  "command": "User.UpdateUser",

  "data": {
    (see xml example above)
  },

  "where": {
    (see xml example above)
  }
}
```

OBJECT-fields	
userName	(Optional) New name of user. Must be unique under its parent folder.
password	(Optional) User password
fullName	(Optional) Full user name
clientType	(Optional) User account client type. The value can be one of the following: Administrator, Developer, Translator, Loc Manager, Loc QA, Builder If not specified, then 'Developer' will be used.
memberOfGroup	(Optional) Name of one or many groups that this user is member of. Group names are separated with a semicolon (;). **Note:** Make sure the group names are unique on the server.
memberOfProject	(Optional) Name of projects that this user is member of. Project names are separated with a semicolon (;). **Note:** Make sure the project names are unique on the server.
emailAddress	(Optional) User Email Address
description	(Optional) User description.
parentFolderName	(Optional) Name of the folder to where the user should be moved. As an alternative the *parentFolderId* can be specified.
parentFolderId	(Optional) Id of the folder to where the user should be moved. As an alternative the *parentFolderName* can be specified.
languageCode	(Optional) If this is a translator user account,

	then this field can be used to update the language code.
countryCode	(Optional) If this is a translator user account, then this field can be used to update the country code.

WHERE-fields	
userName	(Restricted Optional) Name of the user to be updated. Must be specified unless *userId* is specified.
userId	(Restricted Optional) Id (GUID) of the user to be updated. Must be specified unless *userName* is specified.

RESULTSET structure

If successfully updated, then the result set will consist of a RESULT-element with no elements, else there will be a MESSAGE-element under the RESULT-element.

Successful Result Example:

```
<RESULTSET>
     <DATASETS />
     <RESULT />
</RESULTSET>
```

Unsuccessful Result Example:

```
<RESULTSET>
     <DATASETS />
     <RESULT>
          <MESSAGE id="<code>" type="<type>">
               Message text
          </MESSAGE>
     </RESULT>
</RESULTSET>
```

DeleteUser Command

With this command you can delete an existing user.

XML request (Socket API or HTTP API)

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="DeleteUser">
          <OBJECT name="User"></OBJECT>
          <WHERE>
               [<userName/>]
               [<userId/>]
          </WHERE>
     </TASK>
</EXECUTION>
```

Example how to delete a user:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="DeleteUser">
          <OBJECT name="User"></OBJECT>
          <WHERE>
               <userName>Some User</userName>
          </WHERE>
     </TASK>
</EXECUTION>
```

JSON request (HTTP API)

```
{
  "secId": "<secId>",
  "command": "User.DeleteUser",

  "data": {
     (see xml example above)
  },

  "where": {
     (see xml example above)
  }
}
```

WHERE-fields	
userName	(Restricted Optional) Name of the user to be deleted. Must be specified unless *userId* is specified.
userId	(Restricted Optional) Specifies id of the user to be deleted. Must be specified unless *userName* is specified.

RESULTSET structure

If successfully deleted, then a message will be returned that the user has been deleted, else there will be message with information on why it failed.

GetUsers Command

This command will list all users that the user making the request have access to (typically an administrator account).

XML request (Socket API or HTTP API)

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="GetUsers">
          <OBJECT name="User"/>
          <WHERE/>
     </TASK>
</EXECUTION>
```

JSON request (HTTP API)

```
{
  "secId": "<secId>",
  "command": "User.GetUsers",
  "data": {},
  "where": {}
}
```

RESULTSET structure

This is an example of a response of this request:

```
    <RESULTSET>
      <DATASETS>
        <DATASET>
          <userId>F85C5BC9-723F-A47F-F6F0-980874CF382F</userId>
          <userName>Admin</userName>
          <fullName>Administrator</fullName>
          <emailAddress />
          <description>Administrator account<description>
          <hasRootAccess>true</hasRootAccess>
          <clientType>Administrator</clientType>
          <languageCode />
          <countryCode />
          <imageURL />
          <GroupIds>10EEAB0A-70A5-14F9-14E0-971C7A10A6D8;1FAD4FBE-2095-
8D4A-81AC-3844B077F15B;</GroupIds>
          <Groups>Admins;Developers;</Groups>
          <ProjectIds />
          <Projects />
          <Folder>Domains/My Domain/Domain Users/</Folder>
        </DATASET>
        <DATASET>
          <userId>80B80203-6BB6-32EB-C1FC-66B5DB8BE8C6</userId>
          <userName>Alice</userName>
          <fullName>Alice Anderson</fullName>
          <emailAddress />
          <description />
          <hasRootAccess>false</hasRootAccess>
          <clientType>Developer</clientType>
          <languageCode />
        <countryCode />
        <imageURL />
        <GroupIds>1FAD4FBE-2095-8D4A-81AC-3844B077F15B;</GroupIds>
        <Groups>Developers;</Groups>
        <ProjectIds>8AD58808-6C65-2013-ABB4-40E5573E5611;</ProjectIds>
        <Projects>My Game;</Projects>
        <Folder>Domains/My Domain/Domain Users/</Folder>
      </DATASET>
    </DATASETS>
</RESULTSETS>
```

CreateGroup Command

With this command you can create a new group.

XML request (Socket API or HTTP API)

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="CreateGroup">
          <OBJECT name="Group">
               <groupName/>
               [<description/>]
               [<users/>]
               [<memberOfProject/>]
          </OBJECT>
          <WHERE>
               [<parentFolderName/>]
               [<parentFolderId/>]
          </WHERE>
     </TASK>
</EXECUTION>
```

Example how to create a new user:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="CreateGroup">
          <OBJECT name="Group">
               <groupName>Developers</groupName>
               <users>developer1;developer2;</users>
               <memberOfProject>My Game</memberOfProject>
          </OBJECT>
          <WHERE>
               <parentFolderName>Domain Groups</parentFolderName>
          </WHERE>
     </TASK>
</EXECUTION>
```

JSON request (HTTP API)

```
{
  "secId": "<secId>",
  "command": "Group.CreateGroup",

  "data": {
    (see xml example above)
  },

  "where": {
    (see xml example above)
  }
}
```

OBJECT-fields	
groupName	Name of the group. Must be unique under its parent folder.
description	Group description

users	(Optional) Name of one or many users that should be members of this Group. User names should be separated with a semicolon (;).
memberOfProject	(Optional) Name of projects that this group should be member of. Project names should be separated with a semicolon (;). **Note:** Make sure the project names are unique on the server.

WHERE-fields	
parentFolderName	(Restricted Optional) Specifies the name of the folder under which the new group should be created. This is just the name of the folder NOT a path (as group folders exists outside of projects). Make sure that the name of the group folder is unique before using this parameter. Must be specified unless *parentFolderId* is specified.
parentFolderId	(Restricted Optional) Specifies the unique folder id (GUID) under which the new group folder should be created. Must be specified unless *parentFolderName* is specified.

RESULTSET structure

If successfully inserted, then the result set will return the userId of the created user (see example below), else there will be a MESSAGE-element under the RESULT-element.

Successful Result Example:

```
<RESULTSET>
  <DATASETS>
    <DATASET datatype="Group">
      <groupId>98903982-8957-10D9-7954-9AC31A37DCE5</groupId>
    </DATASET>
  </DATASETS>
  <RESULT />
</RESULTSET>
```

Unsuccessful Result Example:

```
<RESULTSET>
     <DATASETS />
     <RESULT>
          <MESSAGE id="0" type="Error">
            Reason for the failure message.
          </MESSAGE>
     </RESULT>
</RESULTSET>
```

UpdateGroup Command

With this command you can update a group. It's also possible to move a group to a different folder using this command.

XML request (Socket API or HTTP API)

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="UpdateGroup">
          <OBJECT name="Group">
               [<groupName/>]
               [<description/>]
               [<parentFolderName/>]
               [<users/>]
               [<memberOfProject/>]
          </OBJECT>
          <WHERE>
               [<groupName/>]
               [<groupId/>]
          </WHERE>
     </TASK>
</EXECUTION>
```

Example how to update a group:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="UpdateGroup">
          <OBJECT name="Group">
               <users>Developer1;Developer2;Developer3;</users>
               <memberOfProject>My Game;My New Game;</memberOfProject>
          </OBJECT>
          <WHERE>
               <groupName>Developers</groupName>
          </WHERE>
     </TASK>
</EXECUTION>
```

Example how to move a group to another user folder:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="UpdateGroup">
          <OBJECT name="Group">
               <parentFolderName>My Game Groups</parentFolderName>
          </OBJECT>
          <WHERE>
               <groupName>Developers</groupName>
          </WHERE>
     </TASK>
</EXECUTION>
```

JSON request (HTTP API)

```
{
  "secId": "<secId>",
  "command": "Group.UpdateGroups",

  "data": {
    (see xml example above)
  },

  "where": {
    (see xml example above)
  }
}
```

OBJECT-fields	
groupName	(Optional) New name of the group. Must be unique under its parent folder.
description	(Optional) User description.
Users	(Optional) Name of one or many users that should be members of this group. User names should be separated with a semicolon (;).
memberOfProject	(Optional) Name of projects that this group is member of. Project names should be separated with a semicolon (;). **Note:** Make sure the project names are unique on the server.
parentFolderName	(Optional) Name of the folder to where the group should be moved. As an alternative the *parentFolderId* can be specified.
parentFolderId	(Optional) Id of the folder to where the group should be moved. As an alternative the *parentFolderName* can be specified.

WHERE-fields	
groupName	(Restricted Optional) Name of the group to be updated. Must be specified unless *groupId* is specified.
groupId	(Restricted Optional) Id (GUID) of the group to be updated. Must be specified unless *groupName* is specified.

RESULTSET structure

If successfully updated, then the result set will consist of a RESULT-element with no elements, else there will be a MESSAGE-element under the RESULT-element.

Successful Result Example:

```
<RESULTSET>
     <DATASETS />
     <RESULT />
</RESULTSET>
```

Unsuccessful Result Example:

```
<RESULTSET>
    <DATASETS />
    <RESULT>
        <MESSAGE id="<code>" type="<type>">
            Message text
        </MESSAGE>
    </RESULT>
</RESULTSET>
```

DeleteGroup Command

With this command you can delete an existing group.

XML request (Socket API or HTTP API)

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="DeleteGroup">
          <OBJECT name="Group"></OBJECT>
          <WHERE>
               [<groupName/>]
               [<groupId/>]
          </WHERE>
     </TASK>
</EXECUTION>
```

Example how to delete a group:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="DeleteGroup">
          <OBJECT name="Group"></OBJECT>
          <WHERE>
               <groupName>Some Group</groupName>
          </WHERE>
     </TASK>
</EXECUTION>
```

JSON request (HTTP API)

```
{
  "secId": "<secId>",
  "command": "Group.DeleteGroup",

  "data": {
     (see xml example above)
  },

  "where": {
     (see xml example above)
  }
}
```

WHERE-fields	
groupName	(Restricted Optional) Name of the group to be deleted. Must be specified unless *groupId* is specified.
groupId	(Restricted Optional) Specifies id of the group to be deleted. Must be specified unless *groupName* is specified.

RESULTSET structure

If successfully deleted, then a message will be returned that the user has been deleted, else there will be message with information on why it failed.

GetGroups Command

This command will list all groups that the user making the request have access to (typically an administrator account).

XML request (Socket API or HTTP API)

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="GetGroups">
          <OBJECT name="Group"/>
          <WHERE/>
     </TASK>
</EXECUTION>
```

JSON request (HTTP API)

```
{
  "secId": "<secId>",
  "command": "Group.GetGroups",
  "data": {},
  "where": {}
}
```

RESULTSET structure

This is an example of a response of this request:

```
    <RESULTSET>
      <DATASETS>
        <DATASET>
          <groupId>10EEAB0A-70A5-14F9-14E0-971C7A10A6D8</groupId>
          <groupName>Admins</groupName>
          <description />
          <UserIds>31CAD629-955D-1582-2991-EC521EBDE824;4FC90A6A-33C9-5650-
F0BA-6E686052E322;63296D6B-7697-C76D-9719-50A4CFBD8D3D;</UserIds>
          <Users>Frank Smith;Kalle Kula;Admin;</Users>
          <ProjectIds>8AD58808-6C65-2013-ABB4-40E5573E5611;</ProjectIds>
          <Projects>My Game;</Projects>
          <Folder>Domains/My Domain/Domain Groups/</Folder>
        </DATASET>
        <DATASET>
          <groupId>569DFEBC-1798-D0C2-D9E6-17157F46158F</groupId>
          <groupName>My Group</groupName>
          <description>This is my group</description>
          <UserIds>80B80203-6BB6-32EB-C1FC-66B5DB8BE8C6;AD86E482-43A1-53C2-
5738-E761188D140B;</UserIds>
          <Users>Alice;Emily;</Users>
          <ProjectIds>ED6E70BF-4591-049F-21EE-902168A7717C;</ProjectIds>
          <Projects>My Project;</Projects>
          <Folder>Domains/My Domain/Special Groups/</Folder>
        </DATASET>
      </DATASETS>
    </RESULTSET>
```

CreateUserFolder Command

With this command you can create a new user folder. This command requires administrative privileges and access rights to the domain where the folder should be created.

XML request (Socket API or HTTP API)

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="CreateUserFolder">
          <OBJECT name="Folder">
               <folderName/>
               [<description/>]
          </OBJECT>
          <WHERE>
               [<domainName/>]
               [<domainId/>]
          </WHERE>
     </TASK>
</EXECUTION>
```

JSON request (HTTP API)

```
{
  "secId": "<secId>",
  "command": "Folder.CreateUserFolder",

  "data": {
    (see xml example above)
  },

  "where": {
    (see xml example above)
  }
}
```

OBJECT-fields	
folderName	Folder name. Must be unique under its parent folder.
Description	(Optional) Description

WHERE-fields	
domainName	(Restricted Optional) Domain Name. Must be specified unless *projectId* is specified. If the folder should be created in the root, then the name “root” can be used.
domainId	(Restricted Optional) Unique Domain Id (GUID). Must be specified unless *domainName* is specified.

RESULTSET structure

If successfully inserted, then the result set will return the folderId of the created folder (see example below), else there will be a MESSAGE-element under the RESULT-element.

Successful Result Example:

```
<RESULTSET>
  <DATASETS>
    <DATASET datatype="Folder">
      <folderId>98903985-8954-10D3-7956-9AC31A27DCE1</folderId>
    </DATASET>
  </DATASETS>
  <RESULT />
</RESULTSET>
```

Unsuccessful Result Example:

```
<RESULTSET>
     <DATASETS />
     <RESULT>
          <MESSAGE id="0" type="Error">
            Reason for the failure message.
          </MESSAGE>
     </RESULT>
</RESULTSET>
```

CreateGroupFolder Command

With this command you can create a new group folder. This command requires administrative privileges and access rights to the domain where the folder should be created.

XML request (Socket API or HTTP API)

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="CreateGroupFolder">
          <OBJECT name="Folder">
               <folderName/>
               [<description/>]
          </OBJECT>
          <WHERE>
               [<domainName/>]
               [<domainId/>]
          </WHERE>
     </TASK>
</EXECUTION>
```

JSON request (HTTP API)

```
{
  "secId": "<secId>",
  "command": "Folder.CreateGroupFolder",

  "data": {
     (see xml example above)
  },

  "where": {
     (see xml example above)
  }
}
```

OBJECT-fields	
folderName	Folder name. Must be unique under its parent folder.
description	(Optional) Description

WHERE-fields	
domainName	(Restricted Optional) Domain Name. Must be specified unless *projectId* is specified. If the folder should be created in the root, then the name “root” can be used.
domainId	(Restricted Optional) Unique Domain Id (GUID). Must be specified unless *domainName* is specified.

RESULTSET structure

If successfully inserted, then the result set will return the folderId of the created folder (see example below), else there will be a MESSAGE-element under the RESULT-element.

Successful Result Example:

```
<RESULTSET>
  <DATASETS>
    <DATASET datatype="Folder">
      <folderId>98903985-8954-10D3-7956-9AC31A27DCE1</folderId>
    </DATASET>
  </DATASETS>
  <RESULT />
</RESULTSET>
```

Unsuccessful Result Example:

```
<RESULTSET>
     <DATASETS />
     <RESULT>
          <MESSAGE id="0" type="Error">
            Reason for the failure message.
          </MESSAGE>
     </RESULT>
</RESULTSET>
```

ListStringHistory Command

With this command you can list history events in a project that affected strings, translations and folders.

XML request (Socket API or HTTP API)

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="ListStringHistory">
          <OBJECT name="History"/>
          <WHERE>
               <dataId/>
               [<limit>]
               [<limitDateTime>]
          </WHERE>
     </TASK>
</EXECUTION>
```

JSON request (HTTP API)

```
{
  "secId": "<secId>",
  "command": "History.ListStringHistory",

  "data": {},

  "where": {
    (see xml example above)
  }
}
```

WHERE-fields	
projectName	(Restricted Optional) Project Name. Must be specified unless *projectId* is specified.
projectId	(Restricted Optional) Unique Project Id (GUID). Must be specified unless *projectName* is specified.
limit	(Optional) Maximum number of rows (returning datasets) that this call should return. (E.g. if set to 10, then only the 10 latest history rows will be returned). If not specified then all rows will be returned.
limitDateTime	(Optional) Events that occurred since this date or time. The format of this value can be one of the following: yyyy yyyy-MM yyyy-MM-dd yyyy-MM-dd hh yyyy-MM-dd hh:mm yyyy-MM-dd hh:mm:ss For example to get all events since 5 PM the 21:st of Sep 2019: **2019-09-21 17:00**

XML example how to use this command:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
  <TASK name="ListStringHistory">
    <OBJECT name="History" />
      <WHERE>
        <projectName>My Project</projectName>
        <limit>1000</limit>
        <limitDateTime>2019-09-18</limitDateTime>
      </WHERE>
    </TASK>
</EXECUTION>
```

JSON example how to use this command:

```
{
  "secId": "<secId>",
  "command": "History.ListStringHistory",

  "data": {},

  "where": {
    "projectName": "My Project",
    "limitDateTime": "2019-09-13",
    "limit": 1000
}
```

XML successful response Example:

```
<RESULTSET>
  <DATASETS>
      <DATASET>
        <dataId>493FFA0E-C1FC-D512-7594-1EF2EE1F007F</dataId>
        <eventType>Created</eventType>
        <dataType>String</dataType>
        <languageCode>se</languageCode>
        <countryCode>SV</countryCode>
        <parentDataId>195B2E5C-BB70-BDC8-22E8-EA35DE4CAE66</parentDataId>
        <Folder>Strings/Menu items/</Folder>
        <dataName>MNU_EXIT</dataName>
        <eventText>Avsluta</eventText>
        <userName>Admin</userName>
        <eventTime>2019-09-18 10:51:23</eventTime>
     </DATASET>
    </DATASETS>
  <RESULT />
</RESULTSET>
```

JSON successful response Example:

```
{
  "parentDataId": "628D7110-DBF7-DA3A-4626-841F4621F0B3",
  "updatedFields": {
      "identifierName": "STR_1",
      "Speed": "110",
      "Platform": "PS2"
  },
  "dataId": "576ED212-81E6-8836-AD3F-506E52E58026",
  "countryCode": "",
  "dataType": "String",
  "eventText": "String updated",
  "eventTime": "2019-09-13 14:47:57",
  "dataName": "STR_1",
  "eventType": "Changed",
  "languageCode": "",
  "Folder": "Strings/",
  "userName": "Admin"
}
```

GetHistory Command

With this command you can get history for a specific item.

XML request (Socket API or HTTP API)

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="GetHistory">
          <OBJECT name="History"/>
          <WHERE>
               <dataId/>
               [<limit>]
          </WHERE>
     </TASK>
</EXECUTION>
```

Example how to get item history:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="GetHistory">
          <OBJECT name="History"/>
          <WHERE>
               <dataId>E20F4D7D-021C-7E00-1DFC-E38092389F95</dataId>
          </WHERE>
     </TASK>
</EXECUTION>
```

JSON request (HTTP API)

```
{
  "secId": "<secId>",
  "command": "History.GetHistory",

  "data": {},

  "where": {
     (see xml example above)
  }
}
```

WHERE-fields	
dataId	Data Id (GUID) of the item to get history for. The data Id can be a string id, user id, group id, etc.
limit	Maximum number of rows (returning datasets) that this call should return. (E.g. if set to 10, then only the 10 latest history rows will be returned). If not specified then all rows will be returned.

RESULTSET structure

Successful Result Example:

```
<RESULTSET>
```

```
  <DATASETS>
    <DATASET datatype="History">
      <historyId>222860</historyId>
      <dataId>E20F4D7D-021C-7E00-1DFC-E38092389F95</dataId>
      <tag />
      <userId>E30F5D7D-011C-8E00-1DFC-E38092389F92</userId>
      <userName>Test User</userName>
      <languageCode />
      <countryCode />
      <eventText>Properties changed.</eventText>
      <eventType>3</eventType>
      <eventTime>2011-10-28 13:29:00.0</eventTime>
      <updatedFields/>
    </DATASET>
  </DATASETS>
  <RESULT />
</RESULTSET>
```

Dataset element description

DATASET element	
`historyId`	Unique id
dataId	GUID of the data that this history row was generated for (for example a String GUID)
tag	Used by data objects that have additional history information (The tag is never used by String data objects).
userId	GUID of the user responsible for this action/task.
userName	Name of the user responsible for this action/task.
languageCode	If this is a history row for a string or a translation then the two character language code is stored here. (This only applies to strings created or updated after upgrading to version 2.14.159).
countryCode	If this is a history row for a string or a translation then the two character country code is stored here.
eventText	Text that describes the event.
eventType	Type of event. The event type can be one of the following: 1 = Data added 2 = Data deleted 3 = Data changed
eventTime	Time when this event occurred.

Unsuccessful Result Example:

```
<RESULTSET>
     <DATASETS />
     <RESULT>
          <MESSAGE id="0" type="Error">
            Reason for the failure message.
          </MESSAGE>
     </RESULT>
</RESULTSET>
```

GetCustomFields Command

With this command you can get the names and id's of the custom fields for a specific project.

XML request (Socket API or HTTP API)

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="GetCustomFields">
          <OBJECT name="String"/>
          <WHERE>
               [<projectName/>]
               [<projectId/>]
          </WHERE>
     </TASK>
</EXECUTION>
```

JSON request (HTTP API)

```
{
  "secId": "<secId>",
  "command": "String.GetCustomFields",

  "data": {},

  "where": {
     (see xml example above)
  }
}
```

WHERE-fields	
projectName	(Optional see Text) Project Name. Must be specified unless *projectId* is specified.
projectId	(Optional see Text) Unique Project Id (GUID). Must be specified unless *projectName* is specified.

Example command:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="GetCustomFields">
          <OBJECT name="String"/>
          <WHERE>
               <projectName>MyGame</projectName>
          </WHERE>
     </TASK>
</EXECUTION>
```

RESULTSET structure

This command will return the names and id's of the custom fields as semicolon separated lists as shown in the example below.

Successful Result Example:

```
<RESULTSET>
      <DATASETS>
        <DATASET datatype="dataSet">

<customFields>MyCustom1;MyCustom2;Test1;Test2;A_Field;New_Field;</
customFields>
          <customIds>84C6CDDA-901C-F59F-DCB0-1D350A8205DB;88FE8FEB-54F8-
5529-F2C0-6238CA3ED9AC;2A41B636-E36D-393C-E9B0-12F245F5EB7E;E4DEC38D-F42C-
4A3F-CDEC-1EA3516FE267;18875B87-0204-2BE7-3FE1-5FC31699562E;689E2B24-A04A-
5E68-83DE-E9D71A59D0F8;</customIds>
        </DATASET>
      </DATASETS>  <RESULT />
</RESULTSET>
```

GetCharacters Command

With this command you can get dialogue characters for a specific project.

XML request (Socket API or HTTP API)

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="GetCharacters">
          <OBJECT name="DialogueCharacter"/>
          <WHERE>
               [<projectName/>]
               [<projectId/>]
          </WHERE>
     </TASK>
</EXECUTION>
```

JSON request (HTTP API)

```
{
  "secId": "<secId>",
  "command": "DialogueCharacter.GetCharacters",

  "data": {
     (see xml example above)
  },

  "where": {
     (see xml example above)
  }
}
```

WHERE-fields	
projectName	(Optional see Text) Project Name. Must be specified unless *projectId* is specified.
projectId	(Optional see Text) Unique Project Id (GUID). Must be specified unless *projectName* is specified.

Example of how use this command:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
  <TASK name="GetCharacters">
     <OBJECT name="DialogueCharacter"/>
     <WHERE>
           <projectName>My Game</projectName>
     </WHERE>
  </TASK>
</EXECUTION>
```

RESULTSET structure

This command will return the all dialogue characters of the specified project, one dataset per character.

GetStringFolders Command

With this command you can get all String Folders in a project. Then folderId's and parentFolderId's shows the relation ships between the folders (parent and children). The command will return all folders in a project, both String and Dialogue folders. To determine if a folder is a string folder or a dialogue folder, check the field named "childrenDataTypeId", 0 = String and 1 = Dialogue.

XML request (Socket API or HTTP API)

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name=" GetStringFolders">
          <OBJECT name="Folder"/>
          <WHERE>
               [<projectName/>]
               [<projectId/>]
          </WHERE>
     </TASK>
</EXECUTION>
```

JSON request (HTTP API)

```
{
  "secId": "<secId>",
  "command": "Folder.GetStringFolders",

  "data": {},

  "where": {
    (see xml example above)
  }
}
```

WHERE-fields	
projectName	(Optional see Text) Project Name. Must be specified unless *projectId* is specified.
projectId	(Optional see Text) Unique Project Id (GUID). Must be specified unless *projectName* is specified.

Example command:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
  <TASK name="GetStringFolders">
     <OBJECT name="Folder"/>
     <WHERE>
           <projectName>My Game</projectName>
     </WHERE>
  </TASK>
</EXECUTION>
```

RESULTSET structure

This command will return the all string folders of the specified project.

Delete Folders Recursively **Command**

This command will delete string folders with all content recursively.

XML request (Socket API or HTTP API)

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="DeleteRecursively">
          <OBJECT name="Folder"/>
          <WHERE>
               [<projectName/>]
               [<projectId/>]
               [<folderName/>]
          </WHERE>
     </TASK>
</EXECUTION>
```

JSON request (HTTP API)

```
{
  "secId": "<secId>",
  "command": "Folder.DeleteRecursively",

  "data": {},

  "where": {
    (see xml example above)
  }
}
```

WHERE-fields	
projectName	(Optional see Text) Project Name. Must be specified unless *projectId* is specified.
projectId	(Optional see Text) Unique Project Id (GUID). Must be specified unless *projectName* is specified.
folderName	(Optional see Text) Path to the folder that should be deleted.

Example command:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
  <TASK name="DeleteRecursively">
     <OBJECT name="Folder"/>
     <WHERE>
           <projectName>My Game</projectName>
           <folderName>Strings/My_Folder/My_SubFolder</folderName>
     </WHERE>
  </TASK>
</EXECUTION>
```

RESULTSET structure

This command will return the RESULTSET structure with the result of this command.

Rename Folder **Command**

This command allow you to rename a folder.

XML request (Socket API or HTTP API)

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="RenameFolder">
          <OBJECT name="Folder">
               <folderName/>
          </OBJECT>
          <WHERE>
               [<projectName/>]
               [<projectId/>]
               [<folderPath/>]
               [<folderId/>]
          </WHERE>
     </TASK>
</EXECUTION>
```

JSON request (HTTP API)

```
{
  "secId": "<secId>",
  "command": "Folder.RenameFolder",

  "data": {
    (see xml above)
  },

  "where": {
    (see xml above)
  }
}
```

WHERE-fields	
projectName	(Optional see Text) Project Name. Must be specified unless *projectId* is specified.
projectId	(Optional see Text) Unique Project Id (GUID). Must be specified unless *projectName* is specified.
folderPath	(Optional see Text) Path of the folder that should be renamed. Must be specified unless *folderId* is specified.
folderId	(Optional see Text) Folder id of the folder that should be renamed. Must be specified unless *folderPath* is specified.

Example command:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
  <TASK name="DeleteRecursively">
    <OBJECT name="Folder">
      <folderName>New folder name </folderName>
    </OBJECT>
    <WHERE>
      <folderPath>Strings/My folder</folderPath>
      <projectName>My Game</projectName>
    </WHERE>
  </TASK>
</EXECUTION>
```

RESULTSET structure

This command will return the RESULTSET structure with the result of this command.

GetDomains Command

This command will list all domains that the user making the request have access to (typically an administrator account).

XML request (Socket API or HTTP API)

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="GetDomains">
          <OBJECT name="Domain"/>
          <WHERE/>
     </TASK>
</EXECUTION>
```

JSON request (HTTP API)

```
{
  "secId": "<secId>",
  "command": "Domain.GetDomains",
  "data": {},
  "where": {}
}
```

RESULTSET structure

This is an example of a response of this request:

```
    <RESULTSET>
     <DATASETS>
        <DATASET>
          <domainId>8A43BFA2-C85A-A88F-A04D-87934B680426</domainId>
          <domainName>Domain One</domainName>
          <description />
        </DATASET>
        <DATASET>
          <domainId>CDA90B5C-CEF3-861B-5AB3-0B022F99DC09</domainId>
          <domainName>Domain Two</domainName>
          <description />
        </DATASET>
      </DATASETS>
    </RESULTSET>
```

GetUserFolders Command

This command will list all user folders that the user making the request have access to (typically an administrator account).

XML request (Socket API or HTTP API)

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="GetUserFolders">
          <OBJECT name="Folder"/>
          <WHERE/>
     </TASK>
</EXECUTION>
```

JSON request (HTTP API)

```
{
  "secId": "<secId>",
  "command": "Folder.GetUserFolders",
  "data": {},
  "where": {}
}
```

RESULTSET structure

This is an example of a response of this request:

```
    <RESULTSET>
     <DATASETS>
        <DATASET>
          <folderId>91202D8C-75C4-0B73-93FA-6FA4A90E4599</folderId>
          <folderName>Domain Users</folderName>
          <description />
          <Folder>Domains/My Domain/</Folder>
        </DATASET>
     <DATASETS>
  </RESULTSET>
```

GetGroupFolders Command

This command will list all group folders that the user making the request have access to (typically an administrator account).

XML request (Socket API or HTTP API)

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="GetGroupFolders">
          <OBJECT name="Folder"/>
          <WHERE/>
     </TASK>
</EXECUTION>
```

JSON request (HTTP API)

```
{
  "secId": "<secId>",
  "command": "Folder.GetGroupFolders",
  "data": {},
  "where": {}
}
```

RESULTSET structure

This is an example of a response of this request:

```
    <RESULTSET>
     <DATASETS>
        <DATASET>
          <folderId>A2202D8C-75C4-0B73-93FA-6FA4A90E45AA</folderId>
          <folderName>Domain Groups</folderName>
          <description />
          <Folder>Domains/My Domain/</Folder>
        </DATASET>
     <DATASETS>
  </RESULTSET>
```

GetProjects Command

This command will list all projects that the user making the request have access to (typically an administrator account).

XML request (Socket API or HTTP API)

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="GetProjects">
          <OBJECT name="Project"/>
          <WHERE/>
     </TASK>
</EXECUTION>
```

JSON request (HTTP API)

```
{
  "secId": "<secId>",
  "command": "Project.GetProjects",
  "data": {},
  "where": {}
}
```

RESULTSET structure

This is an example of a response of this request:

```
    <RESULTSET>
     <DATASETS>
        <DATASET>
          <projectId>8A43BFA2-C85A-A88F-A04D-87934B680427</projectId>
          <projectName>Project One</projectName>
          <description />
        </DATASET>
        <DATASET>
          <projectId>CDA90B5C-CEF3-861B-5AB3-0B022F99DC0A</projectId>
          <projectName>Project Two</projectName>
          <description />
        </DATASET>
      </DATASETS>
    </RESULTSET>
```

GetProjectLanguages Command

This command will return all defined languages for a project.

XML request (Socket API or HTTP API)

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
  <TASK name="GetProjectLanguages">
     <OBJECT name="Language"/>
     <WHERE>
         [<projectName/>]
         [<projectId/>]
     </WHERE>
  </TASK>
</EXECUTION>
```

JSON request (HTTP API)

```
{
  "secId": "<secId>",
  "command": "Language.GetProjectLanguages",

  "data": {},

  "where": {
    (see xml example above)
  }
}
```

WHERE-fields	
projectName	(Optional see Text) Project Name. Must be specified unless *projectId* is specified.
projectId	(Optional see Text) Unique Project Id (GUID). Must be specified unless *projectName* is specified.

Example command:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
  <TASK name="GetProjectLanguages">
     <OBJECT name="Language"/>
     <WHERE>
           <projectName>My Game</projectName>
     </WHERE>
  </TASK>
</EXECUTION>
```

RESULTSET structure

This command will return the all languages of the specified project, one dataset per language.

SendEmail Command

If a SMTP server has been defined (project settings), then it's possible to use this API call to send emails via the LocDirect API. This command can be convenient if for example an email should be sent on a Webhook event. In that case this API command can be triggered on the Webhook event.

XML request (Socket API or HTTP API)

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
  <TASK name="SendEmail">
     <OBJECT name="DataObject">
          <senderName/>
          <senderEmail/>
          <recepientEmail/>
          <subject/>
          <emailText/>
     </OBJECT>
      <WHERE>
          [<projectName/>]
          [<projectId/>]
      </WHERE>
  </TASK>
</EXECUTION>
```

JSON request (HTTP API)

```
{
  "secId": "<secId>",
  "command": " DataObject.SendEmail",

  "data": {
     (see xml example above)
  },

  "where": {
     (see xml example above)
  }
}
```

OBJECT-fields	
senderName	Name of the email sender
senderEmail	Email address of the sender
recepientEmail	Email address of the recipient or a semicolon separated list of recipients.
subject	Email subject
emailText	Email text

WHERE-fields	
projectName	(Optional see Text) Project Name. Must be specified unless *projectId* is specified.
projectId	(Optional see Text) Unique Project Id (GUID). Must be specified

	unless *projectName* is specified.

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
  <TASK name="SendEmail">
     <OBJECT name="DataObject">
    <senderName>John Smith</senderName>
    <recepientEmail>all@tester.com</recepientEmail>
    <subject>Notification</subject>
    <senderEmail>john.smith@tester.com</senderEmail>
    <emailText>Hi all!

A new build file is now available!

Thanks,
John</emailText>
     </OBJECT>
     <WHERE>
           <projectName>My Game</projectName>
     </WHERE>
  </TASK>
</EXECUTION>
```

AddComment Command

Adds a comment to string. This command can be used as an incoming Webhook when a respond has been made in an external system on a comment that originally was created in LocDirect.

XML request (Socket API or HTTP API)

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
  <TASK name="AddComment">
     <OBJECT name="Comment">
          <commentText/>
     </OBJECT>
      <WHERE>
          <stringId/>
      </WHERE>
  </TASK>
</EXECUTION>
```

JSON request (HTTP API)

```
{
  "secId": "<secId>",
  "command": "Comment.AddComment",

  "data": {
    (see xml example above)
  },
  "where": {
    (see xml example above)
  }
}
```

OBJECT-fields	
commentText	Comment Text

WHERE-fields	
stringId	Id of the string to which the comment should be added.

Example JSON:

```
{
  "secId": "<secId>",
  "command": "Comment.AddComment",

  "data": {
    "commentText": "This is a comment! ☺"
  },
  "where": {
    "stringId": "A89B3DA8-9D08-E068-EF01-4ABD9034798A"
  }
}
```

CreateCharacter Command

Creates a new dialogue character under the specified folder.

XML request (Socket API or HTTP API)

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="CreateCharacter">
          <OBJECT name="DialogueCharacter">
               <dialogueCharacterName/>
               ...

               [The complete list of fields that can be used can
                be found under StringExport – List of Character Fields]

          </OBJECT>
          <WHERE>
               [<projectName/>]
               [<parentFolderId/>]
               <parentFolderName>
          </WHERE>
     </TASK>
</EXECUTION>
```

Example on how to create a new dialogue character:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="CreateCharacter">
          <OBJECT name="DialogueCharacter">
               <dialogueCharacterName>Anna</dialogueCharacterName>
               <dialogueCharacterType>0</ dialogueCharacterType>
               <dialogueCharacterAge>35</dialogueCharacterAge>
          </OBJECT>
          <WHERE>
               <projectName>My Game</projectName>
               <parentFolderName>Characters</parentFolderName>
          </WHERE>
     </TASK>
</EXECUTION>
```

OBJECT-fields	
dialogueCharacterName	Character name. Must be unique under its parent folder.
[Other fields]	The complete list of character fields that can be used, can be found under the StringExport command – *List of Character Fields*.

WHERE-fields	
projectName	(Optional see Text) Project Name. Must be specified unless *projectId* is specified.

projectId	(Optional see Text) Unique Project Id (GUID). Must be specified unless *projectName* is specified.
parentFolderName	Name of the folder under which the character should be created.

RESULTSET structure

If successfully created, the result set will return the dialogueCharacterId of the created character:

Result Example:

```
<RESULTSET>
  <DATASETS>
    <DATASET datatype="DialogueCharacter">
      <dialogueCharacterId>6D7D37B3-BEE7-3392-6DE0-C59D35E3B8DE</dialogueCharacterId>
     </DATASET>
  </DATASETS>
  <RESULT />
</RESULTSET>
```

UpdateCharacter Command

Updates a dialogue character.

XML request (Socket API or HTTP API)

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="UpdateCharacter">
          <OBJECT name="DialogueCharacter">
               <dialogueCharacterName/>
               ...

               [The complete list of fields that can be used can
                be found under StringExport - List of Character Fields]

          </OBJECT>
          <WHERE>
               <dialogueCharacterId/>
          </WHERE>
     </TASK>
</EXECUTION>
```

Example on how to update a dialogue character:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="UpdateCharacter">
          <OBJECT name="DialogueCharacter">
               <dialogueCharacterAge>36</dialogueCharacterAge>
          </OBJECT>
          <WHERE>


<dialogueCharacterId>0F2030CD-7EC1-BE1E-DFCC-01D3AECB9777</dialogueCharacte
rId>                   </WHERE>
     </TASK>
</EXECUTION>
```

OBJECT-fields	
dialogueCharacterName	Character name. Must be unique under its parent folder.
[Other fields]	The complete list of character fields that can be used, can be found under the StringExport command – *List of Character Fields*.

WHERE-fields	
dialogueCharacterId	Id of the character to be updated.

DeleteCharacter Command

Deletes a dialogue character.

XML request (Socket API or HTTP API)

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="DeleteCharacter">
          <OBJECT name="DialogueCharacter"></OBJECT>
          <WHERE>
               <dialogueCharacterId/>
          </WHERE>
     </TASK>
</EXECUTION>
```

Example on how to delete a dialogue character:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="DeleteCharacter">
          <OBJECT name="DialogueCharacter"></OBJECT>
          <WHERE>


<dialogueCharacterId>0F2030CD-7EC1-BE1E-DFCC-01D3AECB9777</dialogueCharacte
rId>                    </WHERE>
     </TASK>
</EXECUTION>
```

WHERE-fields	
dialogueCharacterId	Id of the character to be deleted.

GetAudioEffects Command

With this command you can list the audio effects of a specific project.

XML request (Socket API or HTTP API)

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="GetAudioEffects">
          <OBJECT name="AudioEffect"/>
          <WHERE>
               [<projectName/>]
               [<projectId/>]
          </WHERE>
     </TASK>
</EXECUTION>
```

WHERE-fields	
projectName	(Optional see Text) Project Name. Must be specified unless *projectId* is specified.
projectId	(Optional see Text) Unique Project Id (GUID). Must be specified unless *projectName* is specified.

Example of how use this command:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
  <TASK name="GetAudioEffects">
     <OBJECT name="AudioEffect"/>
     <WHERE>
           <projectName>My Game</projectName>
     </WHERE>
  </TASK>
</EXECUTION>
```

RESULTSET structure

This command will return the all audio effects of the specified project, one dataset per audio effect.

CreateAudioEffect Command

Creates a new audio effect under the specified folder.

XML request (Socket API or HTTP API)

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="CreateAudioEffect">
          <OBJECT name="AudioEffect">
               <audioEffectName/>
               <audioEffectData/>
          </OBJECT>
          <WHERE>
               [<projectName/>]
               [<parentFolderId/>]
               <parentFolderName>
          </WHERE>
     </TASK>
</EXECUTION>
```

Example on how to create a new audio effect:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="CreateAudioEffect">
          <OBJECT name="AudioEffect">
               <audioEffectName>Special Effect 1</audioEffectName>
               <audioEffectData>Test data</audioEffectData>
          </OBJECT>
          <WHERE>
               <projectName>My Game</projectName>
               <parentFolderName>Effects</parentFolderName>
          </WHERE>
     </TASK>
</EXECUTION>
```

OBJECT-fields	
audioEffectName	Audio effect name. Must be unique under its parent folder.
audioEffectData	Audio Effect data field.

WHERE-fields	
projectName	(Optional see Text) Project Name. Must be specified unless *projectId* is specified.
projectId	(Optional see Text) Unique Project Id (GUID). Must be specified unless *projectName* is specified.
parentFolderName	Name of the folder under which the audio effect should be created.

RESULTSET structure

If successfully created, the result set will return the audioEffectId of the created audio effect:

Result Example:

```
<RESULTSET>
  <DATASETS>
    <DATASET datatype="AudioEffect">
      <audioEffectId>3D7D37B3-AEE7-3392-6DE0-C59D35E3B8DA</audioEffectId>
     </DATASET>
  </DATASETS>
  <RESULT />
</RESULTSET>
```

UpdateAudioEffect Command

Updates an audio effect.

XML request (Socket API or HTTP API)

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="UpdateAudioEffect">
          <OBJECT name="AudioEffect">
               <audioEffectName/>
               <audioEffectData/>
          </OBJECT>
          <WHERE>
               <audioEffectId/>
          </WHERE>
     </TASK>
</EXECUTION>
```

Example on how to update an audio effect:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="UpdateAudioEffect">
          <OBJECT name="AudioEffect">
               <audioEffectName>New name</audioEffectName>
          </OBJECT>
          <WHERE>

     <audioEffectId>0F2030CD-7EC1-BE1E-DFCC-01D3AECB9777</audioEffectId>
          </WHERE>
     </TASK>
</EXECUTION>
```

OBJECT-fields	
audioEffectName	Audio effect name. Must be unique under its parent folder.
audioEffectData	Audio Effect data field.

WHERE-fields	
audioEffectId	Id of the audio effect to be updated.

DeleteAudioEffect Command

Deletes an audio effect.

XML request (Socket API or HTTP API)

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="DeleteAudioEffect">
          <OBJECT name="AudioEffect"></OBJECT>
          <WHERE>
               <audioEffectId/>
          </WHERE>
     </TASK>
</EXECUTION>
```

Example on how to delete an audio effect:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="DeleteAudioEffect">
          <OBJECT name="AudioEffect"></OBJECT>
          <WHERE>

     <audioEffectId>1F2030CD-7EC1-BE1E-DFCC-01D3AECB9222</audioEffectId>
                         </WHERE>
     </TASK>
</EXECUTION>
```

WHERE-fields	
audioEffectId	Id of the audio effect to be deleted.

GetAudioEffectChains Command

With this command you can list the audio effect chains of a specific project.

XML request (Socket API or HTTP API)

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="GetAudioEffectChains">
          <OBJECT name="AudioEffectChain"/>
          <WHERE>
               [<projectName/>]
               [<projectId/>]
          </WHERE>
     </TASK>
</EXECUTION>
```

WHERE-fields	
projectName	(Optional see Text) Project Name. Must be specified unless *projectId* is specified.
projectId	(Optional see Text) Unique Project Id (GUID). Must be specified unless *projectName* is specified.

Example of how use this command:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
  <TASK name="GetAudioEffectChains">
     <OBJECT name="AudioEffectChain"/>
     <WHERE>
           <projectName>My Game</projectName>
     </WHERE>
  </TASK>
</EXECUTION>
```

RESULTSET structure

This command will return the all audio effect chains of the specified project, one dataset per audio effect. It will also return the audio effects of each chain.

CreateAudioEffectChain Command

Creates a new audio effect chain under the specified folder.

XML request (Socket API or HTTP API)

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="CreateAudioEffectChain">
          <OBJECT name="AudioEffectChain">
               <audioEffectChainName/>
               <audioEffectRelations/>
          </OBJECT>
          <WHERE>
               [<projectName/>]
               [<parentFolderId/>]
               <parentFolderName>
          </WHERE>
     </TASK>
</EXECUTION>
```

Example on how to create a new audio effect:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="CreateAudioEffectChain">
          <OBJECT name="AudioEffectChain">
               <audioEffectName>Effect Chain 1</audioEffectName>
               <audioEffectRelations>2370B0DE-94ED-4135-AE38-
613CEAE56EAE;F9FE634B-B97F-2388-C6E2-6E277B986418</audioEffectRelations>
          </OBJECT>
          <WHERE>
               <projectName>My Game</projectName>
               <parentFolderName>Effect Chains</parentFolderName>
          </WHERE>
     </TASK>
</EXECUTION>
```

OBJECT-fields	
audioEffectChainName	Audio effect chain name. Must be unique under its parent folder.
audioEffectRelations	A semicolon separated list of audio effect id's.

WHERE-fields	
projectName	(Optional see Text) Project Name. Must be specified unless *projectId* is specified.
projectId	(Optional see Text) Unique Project Id (GUID). Must be specified unless *projectName* is specified.
parentFolderName	Name of the folder under which the audio effect chain should be created.

RESULTSET structure

If successfully created, the result set will return the audioEffectChainId of the created audio effect chain:

Result Example:

```
<RESULTSET>
  <DATASETS>
    <DATASET datatype="AudioEffectChain">
      <audioEffectChainId>7214F143-6A10-2E92-FE79-D57D2E03F14C</audioEffectChainId>
     </DATASET>
  </DATASETS>
  <RESULT />
</RESULTSET>
```

UpdateAudioEffect Command

Updates an audio effect chain.

XML request (Socket API or HTTP API)

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="UpdateAudioEffectChain">
          <OBJECT name="AudioEffectChain">
               <audioEffectChainName/>
               <audioEffectRelations/>
          </OBJECT>
          <WHERE>
               <audioEffectChainId/>
          </WHERE>
     </TASK>
</EXECUTION>
```

Example on how to update an audio effect chain:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
     <TASK name="UpdateAudioEffectChain">
          <OBJECT name="AudioEffectChain">
               <audioEffectChainName>New name</audioEffectChainName>
          </OBJECT>
          <WHERE>


<audioEffectChainId>0F2030CD-7EC1-BE1E-DFCC-01D3AECB9777</audioEffectChainI
d>
          </WHERE>
     </TASK>
</EXECUTION>
```

OBJECT-fields	
audioEffectChainName	Audio effect chain name. Must be unique under its parent folder.
audioEffectRelations	A semicolon separated list of audio effect id’s.

WHERE-fields	
audioEffectChainId	Id of the audio effect chain to be updated.

DeleteAudioEffectChain Command

Deletes an audio effect chain.

XML request (Socket API or HTTP API)

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
    <TASK name="DeleteAudioEffectChain">
        <OBJECT name="AudioEffectChain"></OBJECT>
        <WHERE>
            <audioEffectChainId/>
        </WHERE>
    </TASK>
</EXECUTION>
```

Example on how to delete an audio effect chain:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTION>
    <TASK name="DeleteAudioEffectChain">
        <OBJECT name="AudioEffectChain"></OBJECT>
        <WHERE>


<audioEffectChainId>1F2030CD-7EC1-BE1E-DFCC-01D3AECB9222</audioEffectChainI
d>                  </WHERE>
    </TASK>
</EXECUTION>
```

WHERE-fields	
audioEffectChainId	Id of the audio effect chain to be deleted.