maintenance

The Maintenance plugin can be used in combination with the solution import hooks to control the workflow when importing solutions into the Servoy Application Server.

The pre-import and post-import hooks i.c.w. the Maintenance plugin allow the automatic execution of administrative tasks on the Servoy Application Server while importing solutions, like:

- Send messages to connected clients, or even kill connected clients.
- Put the server into maintenance mode for the duration of the solution import.
- Perform operations on databases.

Solution Import Hooks
During solution import from the Servoy Admin Page, modules with certain name patterns are handled in a special way. Modules whose names start with "before_import" are handled as pre-import hooks. Their onOpen event handlers are executed before the solution is actually imported. Similarly, modules whose names start with "after_import" are handled as post-import hooks. Their onOpen event handlers are executed after the solution is actually imported.

> ⚠ **New SolutionType**
>
> In the next major version the pre_ & post_import hook modules will get their own SolutionType, instead of having to rely on SolutionName prefixes.

If multiple modules exists with a name that start with "before_import" or "after_import", they will be all handled in this way, but no handling order is guaranteed.

Pre & post import hook modules are to be self contained, meaning that they are not to rely on included modules.

> ⚠ **Pre & post import hooks containing modules**
>
> In the next major version, the inclusion of modules in the pre & post import hooks will not be allowed.

During the execution of the onOpen event handlers from the pre & post import hooks, any output printed with application.output(...) will be redirected to the import log, which you can see in the Servoy Admin page.

Typical usage scenario

- Before importing the solution, connected clients are notified that an import will take place, then they are disconnected and then the server is put into maintenance mode, so that no new clients can connect during the import.
- The import is performed as usual.
- After the import is over, the server is taken out of maintenance mode, so that clients can connect again.

All these steps can be automated by creating a pre-import and a post-import hook module. The pre-import hook module can have the following function set as its onOpen event handler:

```
function onBeforeImportSolutionOpen()
{
	application.output("Putting server into maintenance mode...");
	plugins.maintenance.setMaintenanceMode(true);
	application.output("Notifying and disconnecting connected clients...");
	plugins.maintenance.sendMessageToAllClients("A solution upgrade will take place on the server. You will
be disconnected in 2 minutes.");
	application.sleep(2 * 60 * 1000);
	plugins.maintenance.sendMessageToAllClients("A solution upgrade will take place on the server. You will
be disconnected NOW.");
	plugins.maintenance.shutDownAllClients();
	application.output("Proceeding to import...");
}
```

This function puts the servoy Application Server into maintenance mode and sends a notification to all connected clients, telling them that in two minutes they will be disconnected. The function waits for two minutes, then it sends another message to the clients, telling them that they will be immediately disconnected. Then all connected clients are killed. From this moment on, no new client will be able to connect to the Server.

The post-import hook module can have the following function as its onOpen event handler:

```
function onAfterImportSolutionOpen()
{
	application.output("Taking server out of maintenance mode.");
	plugins.maintenance.setMaintenanceMode(false);
	application.output("Clients can now connect to the server.");
}
```

This function takes the Server out of maintenance mode, so that clients can again connect to it.

Modifying Database Structure and Content
Using the Maintenance plugin, it is possible to programmatically modify the database structure. It allows for tables to be created and dropped or modify existing tables by creating and deleting columns.

On solution import, Servoy will automatically create all missing tables and columns, based on the metadata confined in the solution export file. When more control is required on how the columns are created and/or columns need to be removed or altered, the pre import hook i.c.w. the Maintenance plugin allows the developer full control.

For example the following function, set as handler for the onOpen even of a pre-import hook, will delete a no longer needed column from a table.

```
function onSolutionOpenBeforeImport()
{
	var server = plugins.maintenance.getServer("example_data")
	if (server) {
		var table = server.getTable("todo_list")
		if (table) {
			table.deleteColumn("todo_list_suggested_by")
			var result = server.synchronizeWithDB(table)
			if (result)
				application.output("'Suggested By' column removed.")
			else
				application.output("Something went wrong while deleting column.")
		}
		else {
			application.output("Table 'todo_list' cannot be found.")
		}
	}
	else {
		application.output("Server 'example_data' cannot be found.")
	}
}
```

The function tries to obtain a reference to the server where the table is stored. If successful, then it tries to obtain a reference to the table which holds the obsolete column. If this step is also successful, then it deletes the column. In order to make the change permanent, the table must be synchronized with the underlying database.

In pre-import and post-import hooks, besides changing the database structure, you can also change the database content. This is done in the classical way, by using the foundset API.

It's also possible to use the rawSQL plugin to modify both data and the datamodel in the database. If changes are made to the datamodel this way, after completing the changes, the function JSServer.reloadDataModel(...) needs to be called, to make the Servoy Application Server aware of the changes made.

Multi database SaaS

When deploying in a multi database SaaS model, any operation done on the Database Servers needs to be performed on each clone of the master Database Server. The getDataModelClonesFrom(serverName) function can be used to retrieve a list of all Database Servers marked as clone of a master Database Server (using the "Data model cloned from" property

Maintenance plugin & [Servoy Cluster]

All operations performed by the Maintenance plugin when executed on a Servoy Application Server that is part of a Servoy Cluster will operate on the entire cluster.

Maintenance plugin availablility

The Maintenance plugin is only available to the code being executed inside the pre and post import hooks. The plugin can not be used in normal clients

Return Types

JSClientInformation JSServer JSTableObject

Method Summery

	#getConnectedClients() Returns an array of JSClientInformation elements describing the clients connected to the server.
String[]	#getDataModelClonesFrom(serverName) Retrieves a list with names of all database servers that have property DataModelCloneFrom equal to the parameter.
JSServer	#getServer(serverName, [mustBeEnabled], [mustBeValid]) Retrieves an instance of JSServer corresponding to the server with the name specified through the "serverName" argument.
String[]	#getServerNames([mustBeEnabled], [mustBeValid], [sort], [includeDuplicates]) Retrieves a list with the names of all available database servers.
Boolean	#isInMaintenanceMode() Returns true if the server is in maintenance mode, false otherwise.
void	#sendMessageToAllClients(message) Sends a message to all connected clients.
void	#sendMessageToClient(clientId, message) Sends a message to a specific client, identified by its clientId.
void	#setMaintenanceMode(maintenanceMode) Puts the server into/out of maintenance mode, depending on the boolean parameter that is specified (if the parameter is true, then the server will be put into maintenance mode; if the parameter is false, then the server will be put out of maintenance mode).
void	#shutDownAllClients() Shuts down all connected clients.
void	#shutDownClient(clientId) Shuts down a specific client, identified by its clientId.

Method Details

getConnectedClients

getConnectedClients()

Returns an array of JSClientInformation elements describing the clients connected to the server.

Sample

```
// WARNING: maintenance plugin is only meant to run during solution import using before or after import hook(so
not from Smart/Web client)
//Returns an array of JSClientInformation elements describing the clients connected to the server.
var clients = plugins.maintenance.getConnectedClients();
application.output("There are " + clients.length + " connected clients.");
for (var i = 0; i < clients.length; i++)
        application.output("Client has clientId '" + clients[i].getClientId() + "' and has connected from host
'" + clients[i].getHostAddress() + "'.");
```

getDataModelClonesFrom

String[] **getDataModelClonesFrom**(serverName)

Retrieves a list with names of all database servers that have property DataModelCloneFrom equal to the parameter.

Parameters

serverName

Returns

String[]

Sample

```
// WARNING: maintenance plugin is only meant to run during solution import using before or after import hook(so
not from Smart/Web client)
//Retrieves a list with names of all database servers that have property DataModelCloneFrom equal to the
parameter.
var serverNames = plugins.maintenance.getDataModelClonesFrom('my_server');
for (var i=0; i<serverNames.length; i++)
        application.output("Process server " + i + ": " + serverNames[i]);
```

getServer

JSServer **getServer**(serverName, [mustBeEnabled], [mustBeValid])

Retrieves an instance of JSServer corresponding to the server with the name specified through the "serverName" argument.
If the optional argument "mustBeEnabled" is set to true, then the JSServer instance is returned only if the server is active.
Similarly, if the "mustBeValid" optional argument is set to true, then the JSServer instance is returned only if the server is valid.
If the specified server is not found, or if it does not meet the requirements imposed by the optional arguments, then null is returned.
By default both optional arguments have the value false.

Parameters

serverName
[mustBeEnabled]
[mustBeValid]

Returns

JSServer

Sample

```
// WARNING: maintenance plugin is only meant to run during solution import using before or after import hook(so
not from Smart/Web client)
//Retrieves an instance of JSServer corresponding to the server with the name specified through the
"serverName" argument.
//If the optional argument "mustBeEnabled" is set to true, then the JSServer instance is returned only if the
server is active.
//Similarly, if the "mustBeValid" optional argument is set to true, then the JSServer instance is returned only
if the server is valid.
//If the specified server is not found, or if it does not meet the requirements imposed by the optional
arguments, then null is returned.
//By default both optional arguments have the value false.
var server = plugins.maintenance.getServer("example_data");
if (server) {
        var tableNames = server.getTableNames();
        application.output("There are " + tableNames.length + " tables.");
        for (var i=0; i<tableNames.length; i++)
                application.output("Table " + i + ": " + tableNames[i]);
        }
else {
        plugins.dialogs.showInfoDialog("Attention","Server 'example_data' cannot be found.","OK");
}
```

getServerNames

String[] **getServerNames**([mustBeEnabled], [mustBeValid], [sort], [includeDuplicates])

Retrieves a list with the names of all available database servers. The returned list will contain only enabled servers if the "mustBeEnabled" optional argument is set to true. The list will contain only valid servers if the "mustBeValid" argument is set to true. If the "sort" optional argument is set to true, then the list will be sorted alphabetically. If the "includeDuplicates" optional argument is set to false, then duplicate servers will appear only once in the list. By default, the "mustBeEnabled" and the "mustBeValid" arguments have the value false, while the "sort" and "includeDuplicates" arguments have the value true.

Parameters

[mustBeEnabled]
[mustBeValid]
[sort]
[includeDuplicates]

Returns

String[]

Sample

```
// WARNING: maintenance plugin is only meant to run during solution import using before or after import hook(so
not from Smart/Web client)
//Retrieves a list with the names of all available database servers. The returned list will contain only
enabled servers if the "mustBeEnabled"
//optional argument is set to true. The list will contain only valid servers if the "mustBeValid" argument is
set to true. If the "sort" optional
//argument is set to true, then the list will be sorted alphabetically. If the "includeDuplicates" optional
argument is set to false, then duplicate
//servers will appear only once in the list. By default, the "mustBeEnabled" and the "mustBeValid" arguments
have the value false, while the "sort"
//and "includeDuplicates" arguments have the value true.
var serverNames = plugins.maintenance.getServerNames();
application.output("There are " + serverNames.length + " servers.");
for (var i=0; i<serverNames.length; i++)
        application.output("Server " + i + ": " + serverNames[i]);
```

isInMaintenanceMode

Boolean **isInMaintenanceMode**()

Returns true if the server is in maintenance mode, false otherwise.

Returns

Boolean

Sample

```
// WARNING: maintenance plugin is only meant to run during solution import using before or after import hook(so
not from Smart/Web client)
//Returns true if the server is in maintenance mode, false otherwise.
if (plugins.maintenance.isInMaintenanceMode())
        application.output("Server is in maintenance mode.");
else
        application.output("Server is not in maintenance mode.");
```

sendMessageToAllClients

void **sendMessageToAllClients**(message)

Sends a message to all connected clients.

Parameters

message

Returns

void

Sample

```
// WARNING: maintenance plugin is only meant to run during solution import using before or after import hook(so
not from Smart/Web client)
//Sends a message to all connected clients.
plugins.maintenance.sendMessageToAllClients("Hello, all clients!");
```

sendMessageToClient

void **sendMessageToClient**(clientId, message)

Sends a message to a specific client, identified by its clientId. The clientIds are retrieved by calling the getConnectedClients method.

Parameters

clientId

message

Returns

void

Sample

```
// WARNING: maintenance plugin is only meant to run during solution import using before or after import hook(so
not from Smart/Web client)
//Sends a message to a specific client, identified by its clientId. The clientIds are retrieved by calling the
getConnectedClients method.
var clients = plugins.maintenance.getConnectedClients();
for (var i=0; i<clients.length; i++)
        plugins.maintenance.sendMessageToClient(clients[i].getClientId(), "Hello, client " + clients[i].
getClientId() + "!");
```

setMaintenanceMode

void **setMaintenanceMode**(maintenanceMode)

Puts the server into/out of maintenance mode, depending on the boolean parameter that is specified (if the parameter is true, then the server will be put into maintenance mode; if the parameter is false, then the server will be put out of maintenance mode).

Parameters

maintenanceMode

Returns

void

Sample

```
// WARNING: maintenance plugin is only meant to run during solution import using before or after import hook(so
not from Smart/Web client)
//Puts the server into/out of maintenance mode, depending on the boolean parameter that is specified (if the
parameter is true, then the server will be put into maintenance mode; if the parameter is false, then the
server will be put out of maintenance mode).
plugins.maintenance.setMaintenanceMode(!plugins.maintenance.isInMaintenanceMode());
```

shutDownAllClients

void **shutDownAllClients**()

Shuts down all connected clients. This method returns immediately, it does not wait until the client shuts down.

Returns

void

Sample

```
// WARNING: maintenance plugin is only meant to run during solution import using before or after import hook(so
not from Smart/Web client)
//Shuts down all connected clients. This method returns immediately, it does not wait until the client shuts
down.
plugins.maintenance.shutDownAllClients();
```

shutDownClient

void **shutDownClient**(clientId)

Shuts down a specific client, identified by its clientId. The clientIds are retrieved by calling the getConnectedClients method. This method returns immediately, it does not wait until the client shuts down.

Parameters

clientId

Returns

void

Sample

```
// WARNING: maintenance plugin is only meant to run during solution import using before or after import hook(so
not from Smart/Web client)
//Shuts down a specific client, identified by its clientId. The clientIds are retrieved by calling the
getConnectedClients method. This method returns immediately, it does not wait until the client shuts down.
var clients = plugins.maintenance.getConnectedClients();
for (var i=0; i<clients.length; i++)
        plugins.maintenance.shutDownClient(clients[i].getClientId());
```